

Machine Learning

01 – Introduction

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Outline (Introduction)

1. What is Machine Learning?
2. Types of Machine Learning
3. Basic Concepts

Lessons learned

- Machine learning aims to learn from experience
 - ▶ Applications everywhere, top IT skill
- Supervised methods take examples and correct answers as input
 - ▶ Classification, regression, structured prediction
- Unsupervised methods use unlabeled examples
 - ▶ Clustering, representation learning, network analysis, ...
- Handling uncertainty is important
- Models can underfit or overfit → need for model selection
- No free lunch theorem implies that we need to study many models

Suggested reading

- Murphy, Ch. 1

Other:

- Mitchell, Ch. 1
- Goodfellow et al., Ch. 5.1–5.3

Machine Learning

01 – Introduction

Part 1: What is Machine Learning?

Prof. Dr. Rainer Gemulla

Universität Mannheim

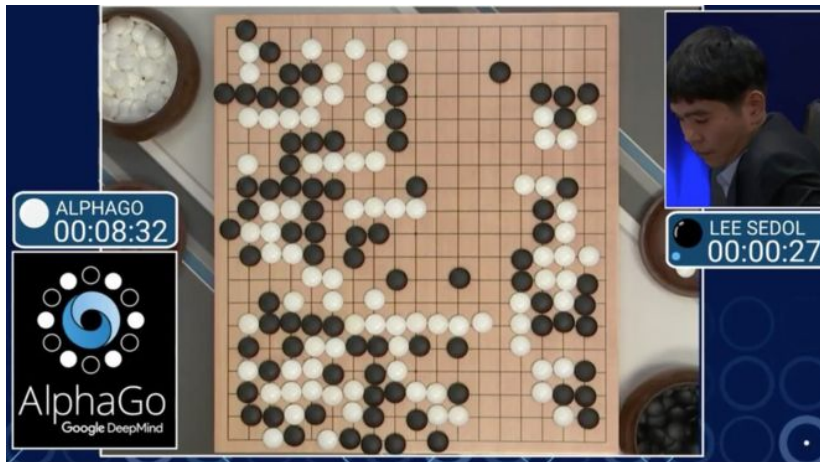
Version: 2022-1

What is machine learning?

Key question: How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?

- Easy to write good programs for certain tasks (e.g., shortest path)
- Hard to write good programs for other tasks (e.g., spam detection)
- Machine learning generally comprises
 - ▶ A **task** T (e.g., playing Checkers)
 - ▶ A **performance metric** P (e.g., percent of games won)
 - ▶ Training **experience** E (e.g., playing games)
 - ▶ Goal: machine learns to reliably improve performance P at task T , following experience E

In 2016, DeepMind's AlphaGo beat Go master Lee Se-dol



Let's learn!

- Rainer distinguishes good and bad triples
- Here is a good triple

2 4 8

- Ask me about triples being good or bad; then tell the rule I use to distinguish (you are allowed to guess only once!)
- Could you write a program that could guess such rules?

Training experience (1)

- Available experience can have a significant impact
- Consider task of playing Checkers
- Does experience provide **direct** or **indirect** feedback regarding performance? E.g.,
 - ▶ Direct: provide feedback on each move the system makes
 - ▶ Indirect: move sequences and final outcomes
 - ▶ Indirect typically harder due to **credit assignment problem**: How much does each move deserve credit/blame for the final outcome?
- To which degree can the learner **control** the experience? E.g.,
 - ▶ Informative examples are provided by a teacher (e.g., a Checkers expert)
 - ▶ Machine can propose board states and ask teacher for suitable move
 - ▶ No teacher present

Training experience (2)

- How **representative** is training experience for final system performance?
 - ▶ Learning is most reliable when training examples have similar distribution as future test examples
 - ▶ In practice, this must often be violated
 - ▶ E.g., machine plays only against itself → potentially not representative for a world-class tournament
- Example
 - ▶ Task: playing Checkers
 - ▶ Performance measure: percentage of games won in world tournament
 - ▶ Training experience: games played against itself

Target function

- Which function(s) do we want to learn?
 - ▶ ML task needs to be broken down into concrete **target functions**
 - ▶ E.g., a function $V : \text{Board} \rightarrow \mathbb{R}$ to “score” a given board
 - ▶ Score should be high when board is good and low otherwise
 - ▶ Can use function to determine next move
- Ideally, function allows for optimal moves
 - ▶ Assuming that the opponent plays optimally, score each board for which learner wins with 1, draw with 0, loses with -1
 - ▶ Not efficiently computable, so of little use in practice
- In machine learning, we search for an **operational** description of the ideal target function
 - ▶ I.e., an efficiently computable function that picks the next move
 - ▶ Function $\hat{V} : \text{Board} \rightarrow \mathbb{R}$, called **model**
 - ▶ Learning task now reduced to finding a suitable **approximation** of an ideal target function

Representation of the target function

- How do we represent the model?
 - ▶ Highly expressive? E.g., a table with each possible board and its score
 - ▶ Very simple? E.g., linear combination of “features” (such as number of black/white pieces/kings)
 - ▶ Anything in between?
- Inherent trade-off: a more expressive representation. . .
 - ▶ May allow better approximations of the target function
 - ▶ Typically requires more training data (often also: more computational resources) to learn a good approximation
- Candidate functions also called **hypothesis**
- Class of considered functions called **model class** or **hypothesis space**

Learning algorithm (1): Target values

- Sometimes values of target functions unknown: e.g., it's easy to score boards that define the end of a game, but not obvious how to score intermediate boards
 - ▶ Player may win even though an intermediate move was bad
 - ▶ Player may lose even though an intermediate move was good
- One simple approach uses target values

$$V_{\text{train}}(\text{board}) \leftarrow \hat{V}(\text{successor}(\text{board}))$$

- ▶ $\text{successor}(\text{board})$ = state of board after own and opponent's move
- ▶ \hat{V} is current approximation
 - Current approximation used to determine estimation target
- ▶ Intuition: if \hat{V} tends to be accurate towards the end of the game, learn to make it more accurate earlier
- ▶ Can be proven to converge to perfect scores under certain conditions

Learning algorithm (2): Adjust model

- Given a new game, we want to improve our model \hat{V} such that it fits the training examples $\{ (board, V_{\text{train}}(board)) \}$ as good as possible
- Need to define best fit; e.g. small error

$$E = \sum_{(board, V_{\text{train}}(board))} \underbrace{(V_{\text{train}}(board) - \hat{V}(board))^2}_{\text{squared error loss function}}$$

- And improve the model; e.g., gradient descent
 - ▶ Suppose \hat{V} is a linear combination of numerical board features
 - ▶ Each feature j has an associated real **weight** w_j
 - ▶ Function \hat{V} can be represented by a **weight vector** w
 - ▶ In each step, gradient descent slightly adjust the weights into a direction that reduces E , e.g.,

$$w \leftarrow w - \epsilon \nabla_w E$$

- Note: actual objective is *not* to minimize error on training data, but on future data (**generalization**)

Summary

- Well-defined learning problem needs well-specified task, performance metric, and source of training experience
- Many design choices exist
 - ▶ Type of training experience
 - ▶ Target function(s) to be learned
 - ▶ Representation of the target function
 - ▶ Learning algorithm
- Learning involves searching through a space of possible hypotheses
- Set of assumptions made by learner is known as **inductive bias**

Machine Learning

01 – Introduction

Part 2: Types of Machine Learning

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Main types of machine learning

	X			y	
x_1	2	4	8	Good	y_1
x_2	4	8	16	Good	y_2
x_3	1	2	1	Bad	y_3

- **Supervised learning** (*predictive*)
 - ▶ Given inputs and right answers
 - ▶ Learn mapping from inputs x to outputs y given a labeled set $\mathcal{D} = \{ (x_i, y_i) \}_{i=1}^N$ of input-output pairs
 - ▶ Use mapping on new (unlabeled) inputs x
- **Unsupervised learning** (*descriptive*)
 - ▶ Given only an unlabeled set $\mathcal{D} = \{ x_i \}_{i=1}^N$ of inputs
 - ▶ Find interesting patterns in the data, learn useful properties or representations, learn data distribution, ...
- **Reinforcement learning**
 - ▶ Learn how to act or behave when occasional reward or punishment signals are given

Supervised learning

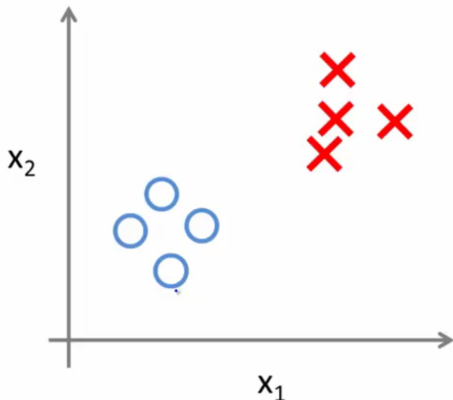
Learn mapping from inputs x to outputs y given a **training set** $\mathcal{D} = \{ (x_i, y_i) \}_{i=1}^N$ of **training examples**.

Common setting




































- Each training input x_i is a D -dimensional vector of numbers (**features**, *attributes*, *covariates*)
 - ▶ Often stored in a $N \times D$ **design matrix** X
 - ▶ Corresponding **labels** (*targets*) in N -dimensional vector y
- Each output (*response variable*) is
 - ▶ **Classification**: y_i is categorical
E.g., document classification, e-mail spam filtering, handwriting recognition, face detection and recognition
 - ▶ **Regression**: y_i is real-valued
E.g., predict stock market price, predict age of person
 - ▶ **Structured prediction**: y_i is more complex
E.g., sequences, trees, graphs

Classification

- Two classes → **binary classification** (aka **concept learning**)
- More than two classes → **multiclass classification**
- Example may belong to multiple classes → **multi-label classification**



Example: Learn to recognize hand-writing

20M-40M-60M-80M-100M-120M-150N												0.35		Label		
												9	←			
8 9	3 5	4 6	1 7	6 4	5 9	3 2	3 5	4 9	6 5	9 4	4 9	4 9	←	First prediction		
												4 9	←			
4 9	8 2	2 1	9 4	0 6	8 4	2 7	7 1	2 9	5 3	4 9	4 9	4 9	←	Second prediction		
																
2 7	3 5	1 4	6 1	6 1	9 4	2 7	5 8	4 1	6 5	6 9						
35 errors																

0.35% errors on validation

30 out of the 35 errors have correct second prediction

Example: Learn to recognize objects



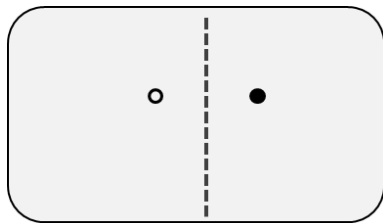
"I can see the cup on the table," interrupted Diogenes, "but I can't see the 'cupness'".

"That's because you have the eyes to see the cup," said Plato, "but", tapping his head with his forefinger, "you don't have the intellect with which to comprehend 'cupness'".

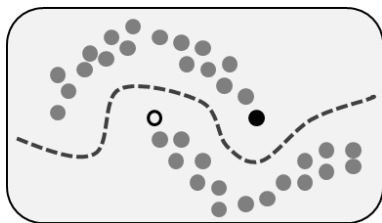
Teachings of Diogenes

Semi-supervised learning

In **semi-supervised learning** (SSL), we are given a **labeled** training set $\mathcal{D}_L = \{(\mathbf{x}_l, y_l)\}_{l=1}^L$ and an **unlabeled** training set $\mathcal{D}_U = \{\mathbf{x}_u\}_{u=1}^U$.



Supervised



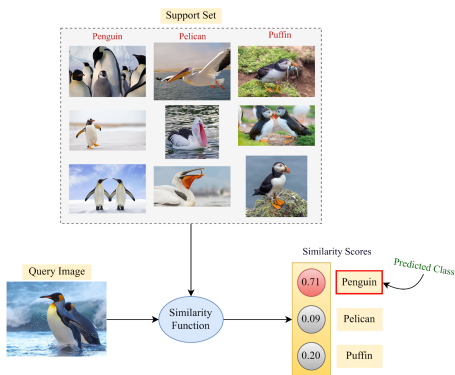
Semi-supervised

Variants:

- **Transductive learning**: infer labels of \mathcal{D}_u only
- **Inductive learning**: learn mapping from \mathbf{x} to y

Few-shot learning

k -shot learning means to learn from only k labeled examples per category (= **support set**).

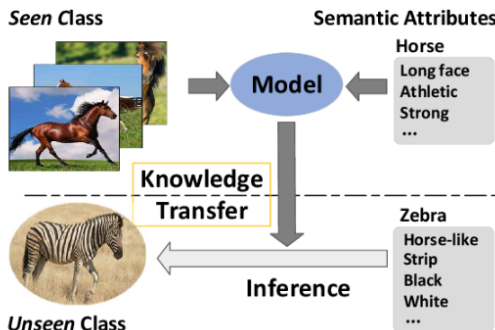


Special case: **one-shot learning** ($k = 1$)

Zero-shot learning

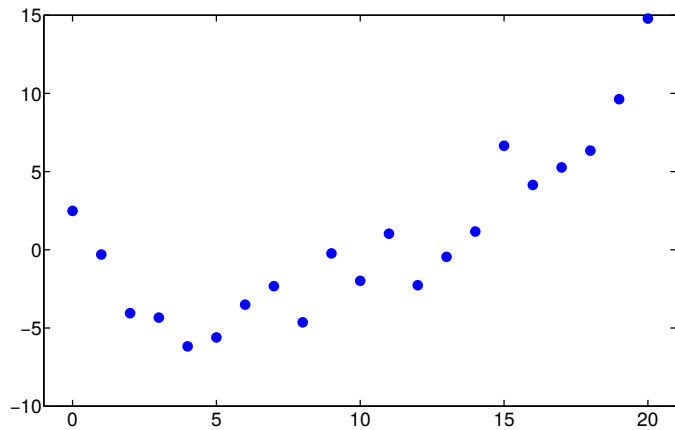
Zero-shot learning means to learn without any labeled examples.

- How is this even possible?
- Generally, based on *auxiliary information* (e.g., descriptions)

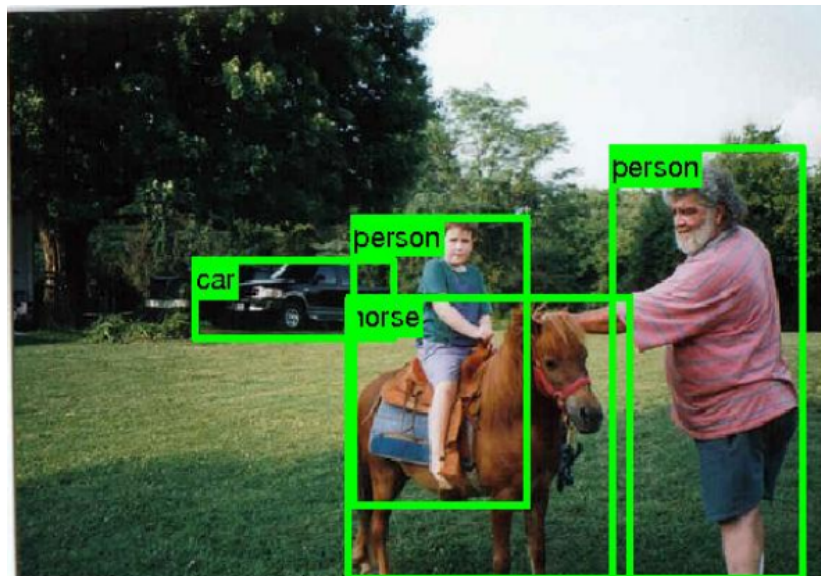


- [ChatGPT](#): “A new animal has been found in Mannheim. It’s very social. Suggest a name.” → [...] *Sociabilis mannheimensis* [...]
- Cf. survey of [Wang et al., 2019](#)

Regression



Example: Learn to detect objects, too



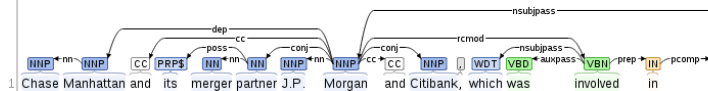
Example: Learn to read and translate

These are examples of **structured prediction** in NLP.

Named Entity Recognition:

1 Chase Manhattan and its merger partner J.P.Morgan and Citibank, which was involved in moving about \$100 million for Raul Salinas de Gortari, brother of a former Mexican president, to banks in Switzerland, are also expected to sign on.

Basic dependencies:



DeepL

Translate from **ENGLISH** (detected) ▾

What are hot topics in machine learning?

Translate into **GERMAN** ▾

Was sind aktuelle Themen des maschinellen Lernens?

Example: Learn to answer questions

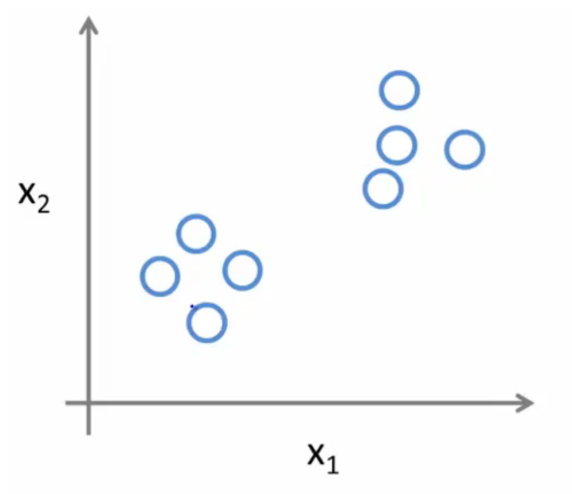


Unsupervised learning

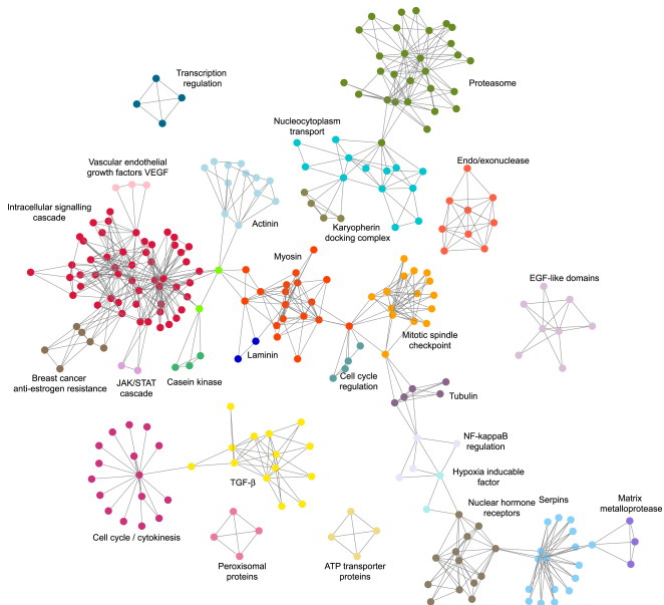
Find “interesting patterns” in the data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$.

- **Density estimation**: learn (properties of) data distribution
- **Clustering**: divide data into groups
 - ▶ E.g., customer segmentation, community detection, sequence analysis
- Unsupervised **representation learning**
 - ▶ Learn “useful” representations or features of the data
 - ▶ E.g., map (potentially complex) data points into a low-dimensional **latent space** that retains (or reveals) the data’s main “structure”
 - ▶ Goal: easier to work with than original data (e.g., facilitate learning, reduce cost, improve interpretability, ...)
 - ▶ Examples: latent variable models, autoencoders, self-supervised pretraining, graph embeddings
- Market basket analysis, sequential pattern mining, social network analysis, ...

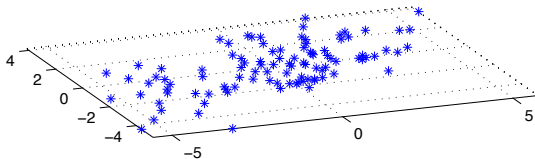
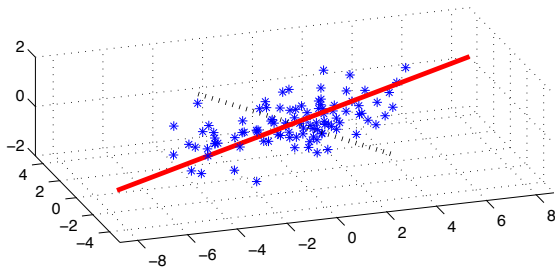
Clustering



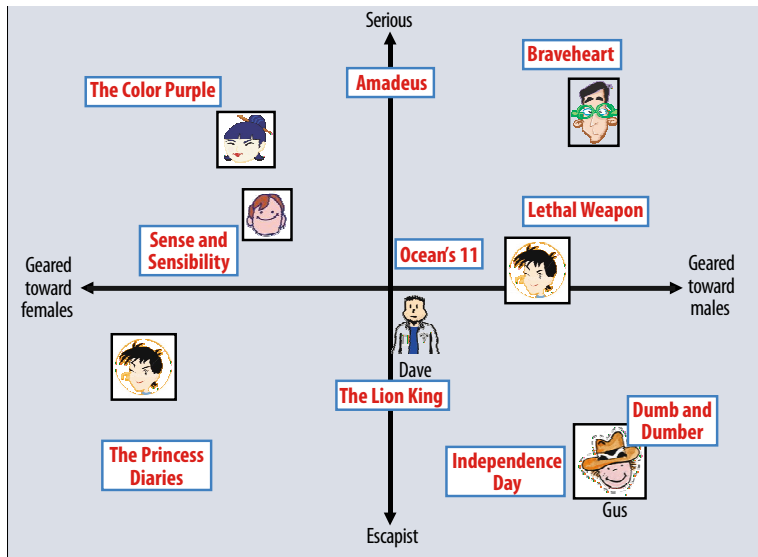
Example: Community detection



Dimensionality reduction

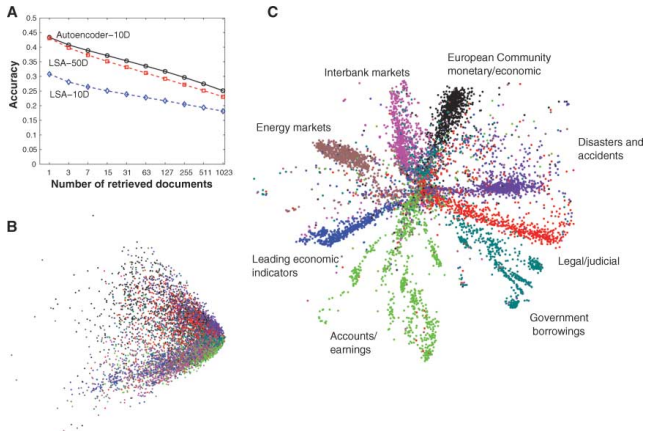


Example: Collaborative filtering

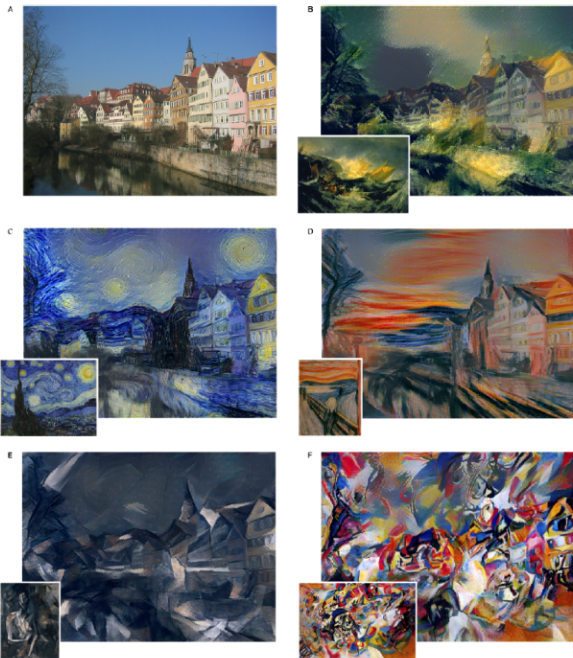


Example: Representing documents

Fig. 4. (A) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (B) The codes produced by two-dimensional LSA. (C) The codes produced by a 2000-500-250-125-2 autoencoder.



Example: Learn to paint



Machine Learning

01 – Introduction

Part 3: Basic Concepts

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Parametric vs. non-parametric models

- Machine learning often uses models that are parameterized
- E.g., in supervised learning
 - ▶ During training, we use examples and labels to learn parameter values
 - ▶ During prediction, we use those values to infer the label
- In a **parametric model**, the number of parameters is fixed
 - ▶ Ranging from a few parameters (e.g., linear/logistic regression on low-dimensional data) to billions of parameters (e.g., large language models)
 - ▶ Esp. when few: Can be faster to use, but may make stronger assumptions about nature of data
- In a **non-parametric model**, the number of parameters grows with the amount of training data
 - ▶ E.g., k -nearest neighbor classifier
 - ▶ More flexible, but can be computationally intractable

K -nearest neighbor classifier (KNN)

- Simple, non-parametric classifier
- Uses statistics about neighbors $N_K(\mathbf{x}, \mathcal{D})$, i.e., the K training points closest to classify test input \mathbf{x} :

$$p(y = c | \mathbf{x}, \mathcal{D}, K) = \frac{1}{K} \sum_{i \in N_K(\mathbf{x}, \mathcal{D})} \mathbb{I}(y_i = c),$$

where $\mathbb{I}(e)$ is the indicator function

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- Discussion
 - ▶ Makes probabilistic predictions
 - ▶ Example of **memory-based learning**
 - ▶ Key assumption: close points have similar labels
 - ▶ Requires a suitable distance function and sufficient data

Example: KNN

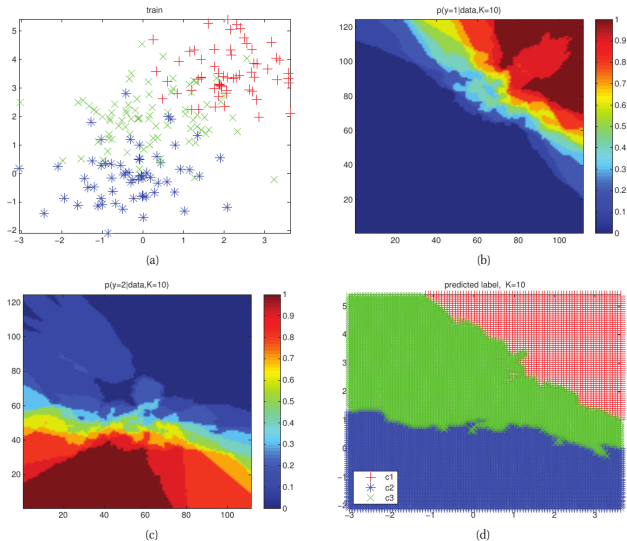
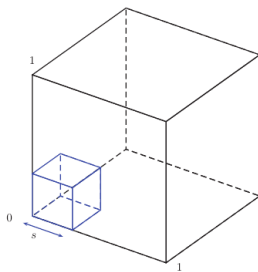


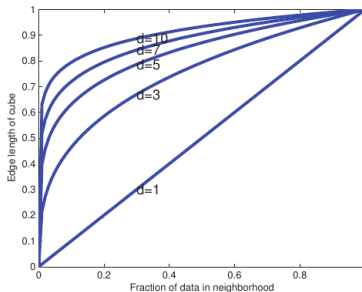
Figure 1.15 (a) Some synthetic 3-class training data in 2d. (b) Probability of class 1 for KNN with $K = 10$. (c) Probability of class 2. (d) MAP estimate of class label. Figure generated by `knnClassifyDemo`.

Curse of dimensionality

Methods such as K NN may not work well with high-dimensional inputs \rightarrow **curse of dimensionality**



(a)



(b)

Figure 1.16 Illustration of the curse of dimensionality. (a) We embed a small cube of side s inside a larger unit cube. (b) We plot the edge length of a cube needed to cover a given volume of the unit cube as a function of the number of dimensions. Based on Figure 2.6 from (Hastie et al. 2009). Figure generated by `curseDimensionality`.

Linear regression

- Main way to combat curse of dimensionality is to make assumptions, e.g., by using a parametric model
- **Linear regression** is a parametric regression model assuming that

$$y(\mathbf{x}) = \sum_{j=1}^D w_j x_j + \epsilon,$$

where the $\{w_j\}_{j=1}^D$ are real-valued **parameters** and ϵ is the **residual error** (on which further assumptions are being placed)

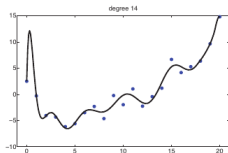
- We can also replace \mathbf{x} by a set of features $\{\phi_j(\mathbf{x})\}$ and assume

$$y(\mathbf{x}) = \sum_{j=1}^{D'} w_j \phi_j(\mathbf{x}) + \epsilon$$

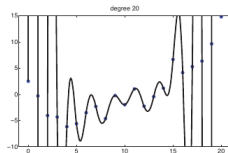
- ▶ Known as **basis function expansion**
- ▶ Example: **polynomial regression** $\phi_j(x) = x^{j-1}$ for $j = 1, \dots, D'$

Overfitting

If we use highly flexible models, we need to be careful that we do not **overfit** the training data.

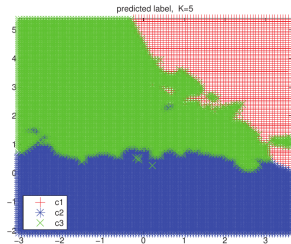
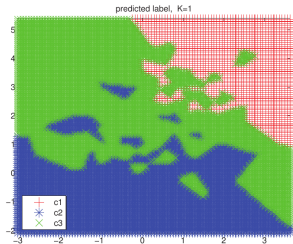


(a)



(b)

Figure 1.18 Polynomial of degrees 14 and 20 fit by least squares to 21 data points. Figure generated by `linregPolyVsDegree`.

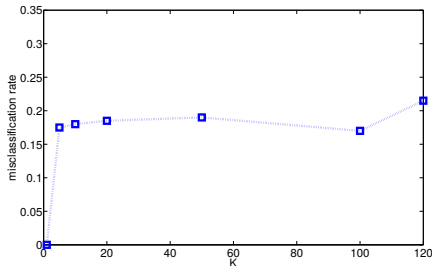


Model selection (1)

- Which model class? → **representational capacity**
 - ▶ E.g., degree of polynomials for polynomial regression
 - ▶ E.g., value of K for KNN classifiers
- Which learning algorithm? → **effective capacity**
- **Model selection**: How to find the most suitable model?
- Natural approach: minimize **misclassification rate**

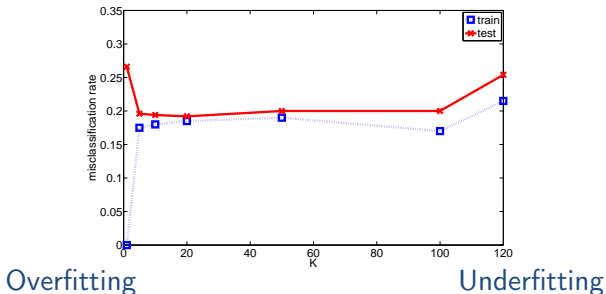
$$\text{err}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(f(\mathbf{x}_i) \neq y_i)$$

- Often does not work: e.g., KNN obtains best results with $k = 1$



Model selection (2)

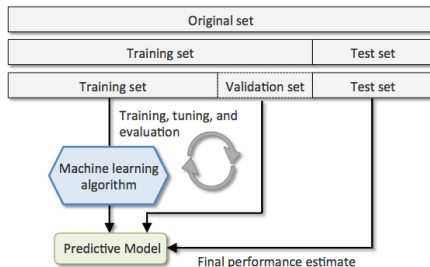
- We care about **generalization error** = expected misclassification rate over *future* data
- Can be approximated by computing misclassification rate on a sufficiently large, independent **test set**



- Problems
 - ▶ Usually we do not have access to a test set
 - ▶ Golden rule of machine learning: **test set should not influence the learning process in any way** → cannot be used for model selection

Model selection (3)

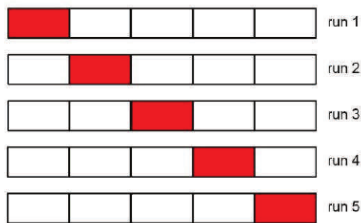
- We have multiple goals
 - ▶ Learn parameters
 - ▶ Perform model selection
 - ▶ Estimate generalization error
- Idea: partition the available data into
 - ▶ A part used for training the model (training set, e.g., 70%)
 - ▶ A part used for model selection (**validation set**, e.g., 20%)
 - ▶ A part used for estimating generalization error (test set, e.g., 10%)



- Problematic if little data available

Model selection (4)

- Improvement: **K -fold cross validation**
 - Split the training set into K **folds**
 - Keep test set separate (test set not shown below)



- Use for model selection
 - For $k = 1 \dots K$, train on all folds but k th, validate on k th
 - Average error over all folds to estimate model performance; then refit best model to entire data
- For $K = N$, called **leave-one-out cross validation**
- More in “IE500 Data Mining I”

No free lunch theorem

All models are wrong, but some models are useful.

— George Box

- There is no single best model that works optimally for all kinds of datasets
 - ▶ A set of assumptions suitable for one domain can be poor for another
- Consequences
 - ▶ Many different models
 - ▶ Many different learning algorithms
- Machine learning studies combination of data, models, and algorithms

Machine Learning

02 – Inference and Decision

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Outline (Inference and Decision)

1. Probability Refresher
2. Generative & Discriminative Models
3. Parameter Estimation
4. Decision

Lessons learned

- Probability refresher (in ILIAS)
 - ▶ Events, random variables, discrete/continuous, sum rule, product rule, conditional probability, Bayes' theorem, independence, expected values ...
 - ▶ Common distributions
 - ▶ Use of shortcut notation in ML
- Three approaches: generative models, discriminative models, discriminative functions
- Bayesian methods model degree of belief about parameter choices
 - ▶ From prior belief to posterior belief through data and model
 - ▶ Posterior \propto likelihood \times prior
 - ▶ Predictions obtained by marginalizing out parameters
- Parameter estimation
 - ▶ Empirical and regularized risk minimization
 - ▶ Maximum likelihood estimation (MLE)
 - ▶ Maximum a posteriori (MAP) estimation
 - ▶ Bayesian inference via posterior predictive
- Bayes estimator for decisions

Suggested reading

- Murphy, Ch. 2.1–2.3
- Murphy, Ch. 4.1–4.3
- Murhpy, Ch. 5.1.1, 5.1.2, 5.1.5
- Goodfellow et al., Ch. 5.4–5.6

More in

- Remaining parts of Murphy, Ch. 2/4/5

Machine Learning

02 – Inference and Decision

Part 1: Probability Refresher

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Probability Refresher

- Here we only refresh some important concepts; more details in
 - ▶ Murphy, Ch. 2, *Probability*
 - ▶ Goodfellow et al., Ch. 3: *Probability and Information Theory*
 - ▶ Supplementary material “A - Probability Refresher” (on ILIAS)
 - ▶ Supplementary Jupyter notebook “distributions” (on ILIAS)
- We will also introduce additional background throughout the course

Notation

We will work with many random variables.

- We write $X \sim \mathcal{N}(0, 1)$ to say that r.v. X has the specified distr.
- We write $p(X = x)$ to refer to the pmf (when X discrete) or pdf (continuous)
- We often drop the r.v. from our notation (when clear from text)
 - ▶ Write $p(x)$ instead of $p(X = x)$ (**marginal distribution**)
 - ▶ Write $p(x, y)$ instead of $p(X = x, Y = y)$ (**joint distribution**)
 - ▶ Write $p(x|y)$ instead of $p(X = x|Y = y)$ (**conditional distribution**)
- $p(x)$ can refer to a probability/density (x fixed) or a distribution (x variable)
- We write $X \perp Y$ if X and Y are independent
- We write $X \perp Y|Z$ if X and Y are conditionally independent given Z

Product rule

- Recall **conditional probability**

$$p(x|y) = \frac{p(x, y)}{p(y)} \quad \text{if } p(y) > 0$$

- Product rule** is

$$p(x, y) = p(x|y)p(y)$$

- Relates joint distribution $p(x, y)$, conditional distribution $p(x|y)$ and marginal distribution $p(y)$
- Generalizes to **chain rule**

$$p(x_{1:n}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_n|x_{1:n-1})$$

Sum rule

- **Sum rule** (*law of total probability*)

$$p(x) = \sum_y p(x, y) \quad (y \text{ discrete r.v.})$$

$$p(x) = \int p(x, y) \, dy \quad (y \text{ continuous r.v.})$$

- ▶ Determine the marginal distribution $p(x)$ from the joint distribution $p(x, y)$
- ▶ When we apply the sum rule in this way, we say that we **marginalize out** y

Example (two dependent coins)

$p(X, Y)$	H	T
$X=H$	0.1	0.2
$X=T$	0.3	0.4

$$p(X = H) = p(X = H, Y = H) + p(X = H, Y = T) = 0.3$$

Bayes' rule

- From the product rule, we can obtain **Bayes' rule** (*Bayes' theorem*)

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

- Combined with the sum rule, we further obtain

$$p(y|x) = \frac{p(y)p(x|y)}{\sum_{y'} p(x, y')}$$

- Many applications, foundation of Bayesian inference

Example: Medical diagnosis

- A mammogram is a test for breast cancer
- Suppose you are a woman in your 40s
- If you have cancer, test positive ($T = 1$) with probability 80%
- If you don't have cancer, test positive with prob. 10%
- About 0.4% of women in their 40s have breast cancer ($B = 1$)
- How likely is it that you have breast cancer if the test is positive?

$$\begin{aligned} p(B = 1|T = 1) &= \frac{p(B = 1)p(T = 1|B = 1)}{p(T = 1)} \\ &= \frac{0.004 \cdot 0.8}{0.004 \cdot 0.8 + 0.996 \times 0.1} \\ &= 0.031 \end{aligned}$$

Important properties

$$p(A \cup B) = p(A) + p(B) - p(A \cap B) \quad (\text{inclusion-exclusion})$$

$$p(\bar{A}) = 1 - p(A)$$

$$\text{If } B \supseteq A, \quad p(B) = p(A) + p(B \setminus A) \geq p(A)$$

$$p(X, Y) = p(Y|X)p(X) \quad (\text{product rule})$$

$$p(X) = \sum_y p(X, y) \quad (\text{sum rule, } y \text{ discrete})$$

$$p(X) = \int_y p(X, y) \, dy \quad (\text{sum rule, } y \text{ continuous})$$

$$p(X|Y) = \frac{p(Y|X)p(X)}{p(Y)} \quad (\text{Bayes theorem})$$

$$E[aX + b] = aE[X] + b \quad (\text{linearity of expectation})$$

$$E[X + Y] = E[X] + E[Y]$$

$$E_Y[E_X[X|Y]] = E[X] \quad (\text{law of total expectation})$$

Machine Learning

02 – Inference and Decision

Part 2: Generative & Discriminative Models

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Inference and decision

- In classification, we seek a classifier that outputs an estimate \hat{y} of the true label y for input x
- **Training** = determine a suitable classifier
 - ▶ E.g., in supervised learning, based on a training set $\mathcal{D} = \{ (x_i, y_i) \}_{i=1}^N$ of labeled examples
 - ▶ Three main approaches: generative, discriminative, function
- **Inference** = given a new input x_{new} , reason about output y_{new}
 - ▶ Often: determine **posterior class probabilities** $p(y_{\text{new}}|x_{\text{new}})$
 - ▶ Depending on model, can be trivial or very hard
- **Decision** = predict an output \hat{y}_{new}
 - ▶ E.g., **Bayes classifier**: $\hat{y}_{\text{new}} = \operatorname{argmax}_c p(y_{\text{new}} = c|x_{\text{new}})$
 - ▶ One may also refrain from a decision (**reject option**)
 - ▶ Decision is not always needed/desired (e.g., when composing multiple models)

Approaches (1)

- **Discriminative functions**

- ▶ Find a discriminative function $f(x)$ that maps each input x to a class label \hat{y}
- ▶ Probabilities play no role
- ▶ Inference and decision merged
- ▶ E.g., k -Nearest Neighbor with majority, (certain) feedforward NNs

- **Discriminative models**

- ▶ Model the posterior class probabilities $p(y|x)$ *directly*
- ▶ But: Inputs are not modeled
- ▶ E.g., logistic regression, k -Nearest Neighbor with probabilities, (other) feedforward NNs

Approaches (2)

Generative models

- Model joint distribution $p(\mathbf{x}, y)$ of inputs and outputs; often:
 - ▶ Model **class-conditional densities** $p(\mathbf{x}|y)$ for each class y *individually*
 - ▶ Model **prior class probabilities** $p(y)$
 - ▶ $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$
- For inference, use Bayes' theorem:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

- Called generative models because we can use the model to generate data
 - ▶ E.g., given an output, we can “generate” suitable inputs
- Examples: Naive Bayes, LDA/QDA, RBMs, GANs

Discussion (1)

- Summary

<i>Model</i>	Generative	Discriminative	Function
<i>Learns</i>	$p(\mathbf{x}, y)$	$p(y \mathbf{x})$	$f(\mathbf{x})$

- Discriminative functions

- ▶ Combine inference and decision
- ▶ No access to posterior class membership probabilities
- ▶ Easy to use
- ▶ But also risky to use since unsure how accurate result is
- ▶ Hard to combine models

- Discriminative models

- ▶ Avoids modeling the input, which can be complex
- ▶ Key advantage: often allows to use a richer feature set (because we do not need to model their distribution) or less stringent assumptions
 - May lead to better performance
- ▶ Addresses the above disadvantages of discriminative functions

Discussion (2)

- Generative models
 - ▶ Models inputs and outputs jointly → generally demanding
 - ▶ Some models make strong assumptions on data distribution
 - ▶ May need less training data / be more accurate than discriminative models if assumptions indeed hold
 - ▶ Probabilities $p(y|x)$ may not be **well-calibrated** when assumptions do not hold (e.g., Naive Bayes may be over-confident)
 - ▶ Useful for unsupervised learning (then: $p(x)$ only) or semi-supervised learning
 - ▶ Can handle **missing data** in a principled way
 - ▶ Also helpful for generating complex data (e.g., text/images), outlier detection, representation learning, ...

Machine Learning

02 – Inference and Decision

Part 3: Parameter Estimation

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-2

Parameter estimation

- Suppose we are given a model class with parameters θ
 - ▶ Generative: $p(\mathbf{x}, y|\theta)$
 - ▶ Discriminative: $p(y|\mathbf{x}, \theta)$
 - ▶ Discriminative function: $f(\mathbf{x}, \theta)$
- And some training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$
- **Parameter estimation** = estimate the value of θ using \mathcal{D}
 - ▶ Also called *training* or *learning*
 - ▶ Should fit the training data well
 - ▶ Should generalize well to new data (more importantly)
 - ▶ **Point estimate** $\hat{\theta}$
- We may or may not be interested in $\hat{\theta}$ itself
 - ▶ Our prime goal is to make predictions
 - ▶ To do so, we need some information about θ , but not necessarily a point estimate
 - ▶ Instead, can use a **posterior** $p(\theta|\mathcal{D})$

What do probabilities mean?

- **Frequentist interpretation**

- ▶ Probability of an event = relative frequency when repeated often
- ▶ Coin, n trials, n_H observed heads

$$\lim_{n \rightarrow \infty} \frac{n_H}{n} = \frac{1}{2} \implies p(H) = \frac{1}{2}$$

- ▶ Frequentist statistics: unknown parameters often assumed to have fixed but unknown value θ^* \rightarrow inappropriate to treat as random variable under frequentist interpretation
- ▶ Estimators analyzed w.r.t. data distribution p^* , which is determined by θ^*

- **Bayesian interpretation**

- ▶ Probability of an event = degree of belief that event holds
- ▶ Degree of belief depends on background knowledge and assumptions and is influenced by seeing data
- ▶ Bayesian statistics: probability distributions associated with unknown parameters
- ▶ Analysis w.r.t. posterior probabilities $p(\theta|\mathcal{D})$

- Both interpretations rely on the rules of probability

Properties of estimators (1)

- We first recap basic properties of (parameter) estimators focusing on a single parameter $\theta \in \mathbb{R}$
- Consider a frequentist setting
 - ▶ **True parameter** θ^* , corresponding **data distribution** p^*
 - ▶ Training data \mathcal{D} is given by N iid. samples from p^*
 - ▶ **Estimator** $\hat{\theta}$, computes point estimate $\hat{\theta}(\mathcal{D})$
 - ▶ E.g., sample mean $\hat{\theta}(\mathcal{D}) = \frac{1}{N} \sum x_i$ for $\mathcal{D} = \{x_i\}_{i=1}^N$
- Estimator $\hat{\theta}$ has **bias**

$$\text{bias}[\hat{\theta}] \stackrel{\text{def}}{=} E[\hat{\theta} - \theta^*]$$

- ▶ Expectation is w.r.t. the N samples of the data distribution p^* :
i.e., $E_{\mathcal{D} \sim p^*}[\hat{\theta}(\mathcal{D}) - \theta^*]$
- ▶ If $\text{bias}[\hat{\theta}] = 0$, the estimator is **unbiased** (correct in expectation),
else it is **biased**

Properties of estimators (2)

- Estimator $\hat{\theta}$ has **variance**

$$\text{var}[\hat{\theta}] = E[(\hat{\theta} - E[\hat{\theta}])^2]$$

- Estimator $\hat{\theta}$ has **mean squared error** (MSE)

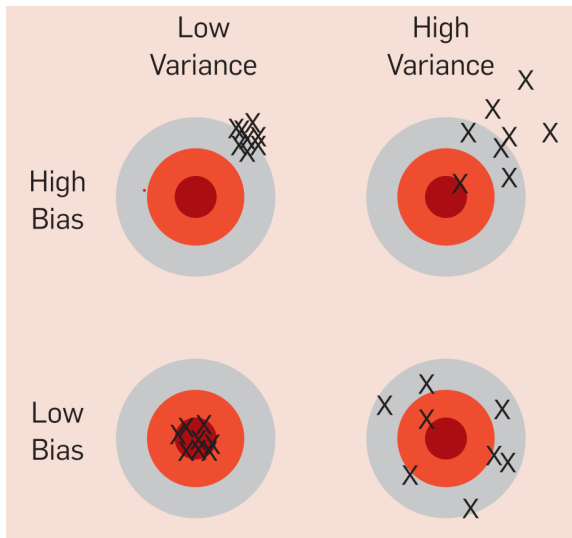
$$\text{mse}[\hat{\theta}] = E[(\hat{\theta} - \theta^*)^2]$$

- Bias-variance decomposition** states that

$$\text{mse}[\hat{\theta}] = \text{bias}[\hat{\theta}]^2 + \text{var}[\hat{\theta}]$$

- Gives rise to the **bias-variance tradeoff**, roughly:
 - ▶ Simple models \rightarrow high bias, low variance
 - ▶ Complex models \rightarrow low bias, high variance
 - ▶ Variance reduces with amount of available data
 - ▶ Bias may or may not reduce
 - ▶ Less data, simpler model (since otherwise variance high)
 - ▶ More data, more complex models (since otherwise bias high)

Bias and variance (illustration)



Top left: underfitting; bottom right: overfitting

Bias-variance tradeoff (example)

- Task: predict mean of Gaussian sampled from $\mathcal{N}(\theta^* = 1, \sigma^2)$
- Blue = sample mean: low bias, high variance
- Red = down-scaled sample mean (by $N/(N + 1)$): larger bias, less variance, lower MSE

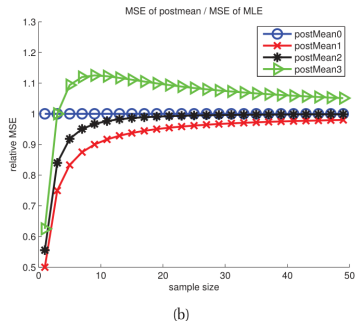
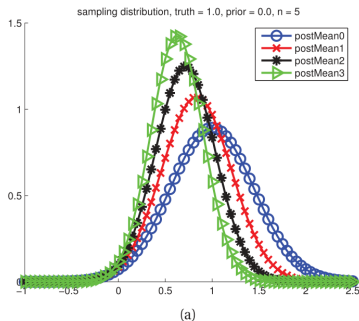


Figure 6.4 Left: Sampling distribution of the MAP estimate with different prior strengths κ_0 . (The MLE corresponds to $\kappa_0 = 0$.) Right: MSE relative to that of the MLE versus sample size. Based on Figure 5.6 of (Hoff 2009). Figure generated by `samplingDistGaussShrinkage`.

Likelihood

- Recall: we assumed that data distribution is $p^*(\mathcal{D}) = p(\mathcal{D}|\theta^*)$
 - ▶ p determined by assumed model class
(e.g., independent coin flips with success probability θ^*)
 - ▶ θ^* fixed, \mathcal{D} varies $\rightarrow p(\cdot|\theta^*)$ is probability distribution

- **Likelihood** of observed data

$$\mathcal{L}(\theta|\mathcal{D}) \stackrel{\text{def}}{=} \begin{cases} p(\mathcal{D}|\theta) & \text{for generative models} \\ p(\mathbf{y}|\mathbf{X}, \theta) & \text{for discriminative models} \end{cases}$$

- ▶ How likely is dataset \mathcal{D} if θ were the true parameter?
- ▶ Intuitively: the larger the likelihood, the more “consistent” the data is with parameter choice
- ▶ Now θ varies, \mathcal{D} fixed $\rightarrow \mathcal{L}(\cdot|\mathcal{D})$ *not* a probability distribution
- Example: Coin, 4 iid trials, $n_H = 4$ observed heads (= data \mathcal{D})
 - ▶ Unknown parameter: $\theta^* =$ true “probability” of heads
 - ▶ $\mathcal{L}(\theta = 0|n_H = 4) = 0$
 - ▶ $\mathcal{L}(\theta = 0.5|n_H = 4) = 0.0625$
 - ▶ $\mathcal{L}(\theta = 1|n_H = 4) = 1$
 - ▶ 16:1 **likelihood ratio** in favor of $\theta = 1$ vs. $\theta = 0.5$

Maximum likelihood estimation (MLE)

- **Maximum likelihood estimation (MLE)** chooses the value $\hat{\theta}$ that maximizes the (conditional) likelihood of the data

$$\hat{\theta}_{\text{MLE}} = \underset{\theta}{\operatorname{argmax}} \mathcal{L}(\theta|\mathcal{D})$$

- ▶ Probabilistic models only
- Good asymptotic properties (i.e., when $N \rightarrow \infty$); under mild conditions:
 - ▶ Consistent: converges to true value θ^* (in probability)
 - ▶ Efficient: no other estimator has lower asymptotic mean squared error
 - ▶ Asymptotically normally distributed
- In practice, we do not have $N = \infty$
 - ▶ Tendency to overfit training data

Empirical risk minimization (1)

- In supervised learning, we may not be interested in the parameters itself, but rather in the corresponding predictions
- Suppose we are given a non-negative, real-valued **loss function** $L(\hat{y}, y)$ that measures how different prediction \hat{y} is from true answer y
 - ▶ E.g., the **0-1 loss** $L(\hat{y}, y) = \mathbb{I}(y \neq \hat{y})$ for classification tasks
 - ▶ E.g., the **squared loss** $L(\hat{y}, y) = (\hat{y} - y)^2$ for regression tasks
- Assume that the data follows a distribution $p^*(\mathbf{x}, y)$
- The **risk** $R(h)$ associated with a hypothesis h is the expected loss over the data distribution

$$R(h) = E_{(\mathbf{x}, y) \sim p^*} [L(h(\mathbf{x}), y)] = \int \int L(h(\mathbf{x}), y) p^*(\mathbf{x}, y) \, d\mathbf{x} \, dy$$

- ▶ Misclassification rate for 0-1 loss
- ▶ Mean squared error (MSE) for squared loss
- ▶ Ideally, we want to choose h such that risk is minimized

Empirical risk minimization (2)

- But: risk cannot be computed since true data distribution p^* is unknown
- The **empirical risk** is the average loss on the training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$

$$R_{\text{emp}}(h) = \frac{1}{N} \sum_{i=1}^N L(h(\mathbf{x}_i), y_i),$$

which we can compute

- **Empirical risk minimization** chooses the estimator that minimizes $R_{\text{emp}}(h)$

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} R_{\text{emp}}(h)$$

- ▶ As we had discussed, typically results in overfitting
- ▶ Can be a difficult optimization problem

Regularized risk minimization

- **Regularized risk minimization** tries to avoid overfitting to the training data by adding a **penalty** term

$$R'_{\text{emp}}(h) = R_{\text{emp}}(h) + \lambda C(h)$$

- ▶ $C(h)$ measures the complexity of the model
- ▶ λ controls strength of penalty
- ▶ Ideally, $\lambda C(h)$ close to generalization gap $R(h) - R_{\text{emp}}(h)$
- ▶ Can be used for discriminative functions and probabilistic models
- Key issues
 - ▶ How to measure model complexity?
 - ▶ E.g., ℓ_2 regularization
 - ▶ How to pick λ ?
 - ▶ E.g., cross-validation
- We will revisit this during the course

Excursion: Learning theory (1)

- **Learning theory** uses formal methods to study learning tasks and learning algorithms
- E.g., consider a binary classification problem
 - ▶ Data distribution p^* such that each data point (x) is associated with a single class (y)
 - ▶ No noise
 - ▶ Hypothesis space is finite ($|H| < \infty$)
 - ▶ H contains a true hypothesis h (s.t. $R(h) = 0$)
- **Version space** = set of hypotheses consistent with training data
 - ▶ Consistent means here that all predictions are equal to true label
 - ▶ Implies that empirical risk R_{emp} is zero
 - ▶ Can still make errors on unseen data
 - ▶ In our example setting, version space always non-empty
 - ▶ If only one hypothesis left in version space, it's the right one
 - ▶ But what if there are many? We do not know which one to pick...

Excursion: Learning theory (2)

- Insight: with sufficient training data, version space unlikely to contain bad hypotheses
- Can show that

$$N \geq \frac{1}{\epsilon} \left(\ln(|H|) + \ln \frac{1}{\delta} \right)$$

suffice to ensure that ERM achieves a low generalization error ($\leq \epsilon$) with high probability ($1 - \delta$)

- Generally, things are more complicated
 - ▶ Noise
 - ▶ Infinite hypothesis spaces (e.g., a single real parameter)
 - ▶ Complex hypothesis spaces
 - ▶ Potentially high computational cost (e.g., not polynomial)
 - ▶ Key concepts: PAC learning, VC dimension

Prior and posterior

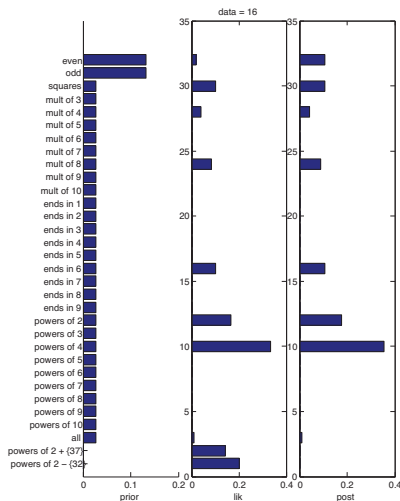
- Now let's look at Bayesian methods, using iid. coin flips as example
- **Prior** belief: $p(\theta)$
 - ▶ Incorporates prior knowledge
 - ▶ E.g., coin is likely to be fair, coin is likely to show heads, ...
 - ▶ Different priors may lead to different results → subjective aspect, controversial (but priors can be very useful)
 - ▶ Can be *uninformative* ("nothing" is known, also controversial)
- **Posterior** belief: $p(\theta|\mathcal{D})$
 - ▶ "Updated" belief after seeing the data
 - ▶ E.g., do you think the coin is fair after seeing 10 heads and 2 tails?
 - ▶ Depends on prior and likelihood via Bayes' theorem

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta)$$

- ▶ In words: **posterior** \propto **likelihood** \times **prior**

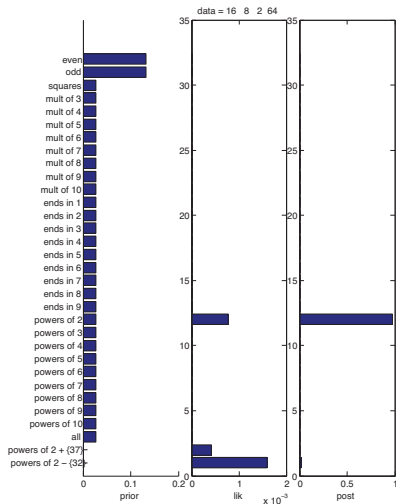
Example: Bayesian concept learning

- Numbers game
 - ▶ I envision a set $A \subseteq \{1, \dots, 100\}$
 - ▶ I give you some numbers from A
 - ▶ What is set A ?
- Need further assumptions and sufficient data
- Figure on the right shows
 - ▶ A hypothesis space of “simple” rules (θ = correct rule)
 - ▶ A prior that gives more weight to simpler explanations (*Occam's razor*)
 - ▶ The likelihood of $\{16\}$, assuming data is randomly sampled from A
 - ▶ The posterior, after seeing $\{16\}$



Example: Bayesian concept learning

- Numbers game
 - ▶ I envision a set $A \subseteq \{1, \dots, 100\}$
 - ▶ I give you some numbers from A
 - ▶ What is set A ?
- Need further assumptions and sufficient data
- Figure on the right shows
 - ▶ A hypothesis space of “simple” rules (θ = correct rule)
 - ▶ A prior that gives more weight to simpler explanations (*Occam's razor*)
 - ▶ The likelihood of $\{16, 8, 2, 64\}$, assuming data is randomly sampled from A
 - ▶ The posterior, after seeing $\{16, 8, 2, 64\}$



Maximum a posteriori estimation (MAP)

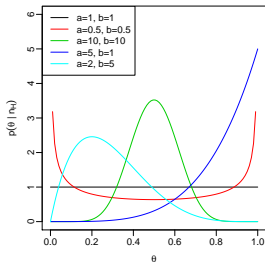
- The **maximum a posteriori (MAP) estimate** $\hat{\theta}_{\text{MAP}}$ is the point estimate that maximizes the posterior

$$\hat{\theta}_{\text{MAP}} = \underset{\theta}{\operatorname{argmax}} p(\theta|\mathcal{D}) = \underset{\theta}{\operatorname{argmax}} \mathcal{L}(\theta|\mathcal{D})p(\theta)$$

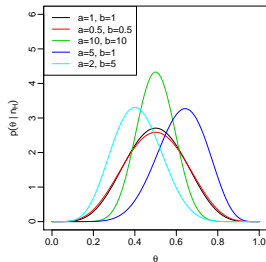
- Think: most probable parameter choice given data *and prior*
- Prior can help to avoid overfitting (see example of slide 16)
- We will see: MLE / MAP estimates sometimes correspond to certain empirical / regularized risk minimization formulations

Example: n iid coin flips, n_H heads

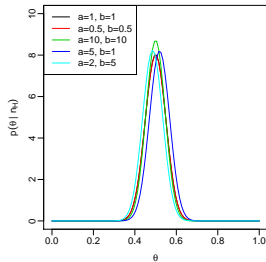
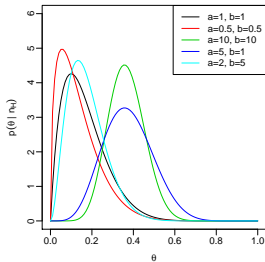
Priors



$$n = 10, n_H = 5 \rightarrow \hat{\theta}_{MLE} = 0.5$$



$$n = 10, n_H = 1 \rightarrow \hat{\theta}_{MLE} = 0.1 \quad n = 100, n_H = 50 \rightarrow \hat{\theta}_{MLE} = 0.5$$



Bayesian inference

- The fully Bayesian approach is to avoid parameter estimation all together
- Use **posterior predictive** distribution

$$p(\mathcal{D}_{\text{new}}|\mathcal{D}) = \int p(\mathcal{D}_{\text{new}}|\theta)p(\theta|\mathcal{D}) \, \mathrm{d}\theta$$

- ▶ E.g., prediction of outcome of n_{new} additional trials
 - ▶ Intuitively, combines predictions of every hypothesis (θ) weighted by its posterior probability (= marginalize out θ)
 - ▶ **posterior predictive** = \int_{θ} **new-data likelihood** \times **posterior** $\mathrm{d}\theta$
- Likewise, for discriminative models

$$p(\mathbf{y}_{\text{new}}|\mathcal{D}, \mathbf{X}_{\text{new}}) = \int p(\mathbf{y}_{\text{new}}|\theta, \mathbf{X}_{\text{new}})p(\theta|\mathcal{D}) \, \mathrm{d}\theta$$

Example: Bayesian linear regression

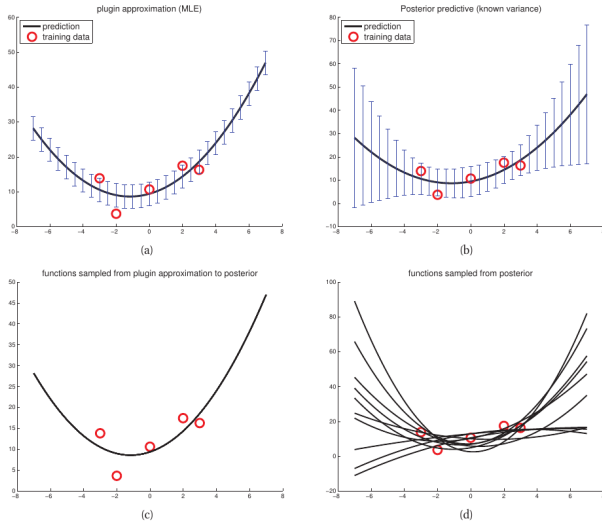


Figure 7.12 (a) Plug-in approximation to predictive density (we plug in the MLE of the parameters). (b) Posterior predictive density, obtained by integrating out the parameters. Black curve is posterior mean, error bars are 2 standard deviations of the posterior predictive. (c) 10 samples from the plugin approximation to posterior predictive. (d) 10 samples from the posterior predictive. Figure generated by `linregPostPredDemo`.

Discussion

- Empirical risk minimization, regularized risk minimization, MLE, and MAP all obtain point estimates
 - ▶ Empirical risk minimization and MLE prone to overfitting
 - ▶ Regularized risk minimization uses a model complexity penalty to avoid overfitting with too-complex models
 - ▶ MAP estimation uses prior to steer away from unlikely models (e.g., a prior may have low density for complex models)
 - ▶ Estimates obtained by solving an optimization problem
- A fully Bayesian approach weights every hypothesis by its posterior probability
 - ▶ Uncertainty in parameter estimates taken into account
 - ▶ Estimates obtained by performing probabilistic inference

Machine Learning

02 – Inference and Decision

Part 4: Decision

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Bayesian decision theory

- Modeling decisions
 - ▶ Known data: $x \in \mathcal{X}$
 - ▶ Unknown data: $y \in \mathcal{Y}$
 - ▶ A set \mathcal{A} of actions
 - ▶ Goal: pick a suitable **action** $a \in \mathcal{A}$
- Example: classification
 - ▶ x : training set + new example
 - ▶ y : label of new example
 - ▶ Action a : predict class $\hat{y} = a$
- We aim to derive a **policy** $\delta : \mathcal{X} \rightarrow \mathcal{A}$ to make a decision
- What is the optimal policy?

Bayes estimator

- Consider an arbitrary but fixed \mathbf{x}
- Loss function
 - ▶ Suppose that we can quantify how good an action a is if we know y
 - ▶ Via a **loss function** $L(a, y)$
 - ▶ Example: classification, $L(a, y) = \mathbb{I}(a \neq y)$ (0-1 loss)

- Since we do not know y , we consider the **expected loss**

$$E[L(y, a)] = \sum_y L(y, a)p(y|\mathbf{x}),$$

- ▶ Bayesian approach: expectation w.r.t. data seen so far
 - ▶ I.e., $p(y|\mathbf{x})$ is the posterior predictive
- The optimal policy minimizes the expected loss

$$\delta(\mathbf{x}) = \operatorname{argmin}_{a \in \mathcal{A}} E[L(y, a)]$$

- ▶ Called the **Bayes estimator**

Examples

- Classification with misclassification rate (0-1 loss)
 - ▶ Bayes estimator: pick most probable class

$$\hat{y} = \operatorname{argmax}_y p(y|\mathbf{x})$$

- ▶ Called **Bayes optimal classifier**
- Regression with MSE (ℓ_2 loss)
 - ▶ Bayes estimator: pick posterior mean

$$\hat{y} = E[y|\mathbf{x}] = \int y p(y|\mathbf{x}) \, dy$$

- Regression with mean absolute error (ℓ_1 loss)
 - ▶ Bayes estimator: pick posterior median
 - ▶ I.e., pick \hat{y} such that $p(y < \hat{y}|\mathbf{x}) = p(y \geq \hat{y}|\mathbf{x}) = 0.5$

Machine Learning

03 – Generative Models for Discrete Data

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Recap: Generative models

- Model joint distribution $p(\mathbf{x}, y)$ of inputs and outputs; often:
 - ▶ Model **class-conditional densities** $p(\mathbf{x}|y)$ for each class y *individually*
 - ▶ Model **prior class probabilities** $p(y)$
 - ▶ $p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y)$
- For inference, use Bayes' theorem:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

- Called generative models because we can use the model to generate data
 - ▶ E.g., given an output, we can “generate” suitable inputs
- Examples: Naive Bayes, LDA/QDA, RBMs, GANs

Let's count

Here is some data (\mathcal{D}):

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

- Generative models estimate $p(y)$ and $p(x|y)$
- A simple way: count to obtain relative frequencies (=MLE)

State	$p(\text{State})$
poor	2/3
rich	1/3

$p(x \text{poor})$	young	old	$p(x \text{rich})$	young	old
student	1/2	1/2	student	0	0
CEO	0	0	CEO	0	1

Let's generate

- The joint distribution is estimated as

$p(x, \text{poor})$	young	old
student	1/3	1/3
CEO	0	0

$p(x, \text{rich})$	young	old
student	0	0
CEO	0	1/3

- Sample from the various estimated distributions

From $p(x|\text{poor})$

young	student
young	student
old	student
old	student
...	

From $p(x|\text{rich})$

old	CEO
old	CEO
old	CEO
old	CEO
...	

From $p(x, y)$

old	student	poor
old	CEO	rich
young	student	poor
young	student	poor
...		

- Sampling is sometimes useful (e.g., to understand or debug a model)

Let's predict

Using Bayes theorem, we obtain estimates

$p(\text{poor} x)$	young	old
student	1	1
CEO	?	0

$p(\text{rich} x)$	young	old
student	0	0
CEO	?	1

Problems

- **Overfitting**: students can't be rich
- **Zero-count problem**: cannot predict for a young CEO since we have never seen one
- **Complexity** of model
 - ▶ D binary features, C classes
→ $O(C2^D)$ non-redundant parameters (probability table)
 - ▶ Infeasible to even store for moderately large D
- Possible solutions: use more data, use a prior, make additional assumptions, use unlabeled data (semi-supervised learning), ...

Outline (Generative Models for Discrete Data)

1. The Beta-Binomial Model
2. The Dirichlet-Multinomial Model
3. Naive Bayes

Also: worked out frequentist/Bayesian approaches to parameter estimation and prediction

Lessons learned

- Beta-binomial model for independent coin flips
- Dirichlet-multinomial model is generalization of Beta-binomial model to $K > 2$ categories
- In generative models, it is typically infeasible to model $p(\mathbf{x}|y)$ without additional assumptions
- Naive Bayes assumption: features cond. independent given y
 - ▶ Naive Bayes classifiers exploit this assumption
 - ▶ Prior class distribution: model fitting = compute histograms (or background knowledge)
 - ▶ Categorical features: model fitting = compute histograms
- Overfitting/zero-count problem may arise
 - ▶ Can be addressed by adding a prior
 - ▶ For categorical distributions, Dirichlet-multinomial model suitable (e.g., add-one smoothing)

Suggested reading

- Murphy, Ch. 4.6, *Bayesian Statistics*
- Murphy, Ch. 9.3, *Naive Bayes Classifiers*

Machine Learning

03 – Generative Models for Discrete Data

Part 1: The Beta-Binomial Model

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Beta-binomial model

- Coin flips
 - ▶ Assume we flip a coin n times and observe the outcome
 - ▶ What can you say about the coin?
 - ▶ E.g., is it fair? (parameter estimation/inference)
 - ▶ E.g., what will the next coin flip(s) show? (prediction)
- **Beta-binomial model** is a simple generative model for coin flips
 - ▶ Coin flips assumed i.i.d. with (unknown) success probability θ
 - ▶ Then suffices to record number n_H of heads and $n_T = n - n_H$ tails (*sufficient statistics*)
 - ▶ Beta prior
- Why study this model?
 - ▶ Fully **worked out example of Bayesian inference**
→ reinforce what we just learned
 - ▶ Forms **basis of other probabilistic models**

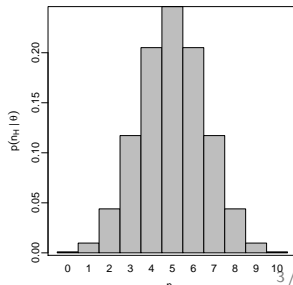
Recap: The binomial distribution (1)

- Pick a coin with probability of heads θ , toss it n times
 - ▶ Let $\{x_1, \dots, x_n\}$ be the outcome (0=tail, 1=head)
 - ▶ Let $n_H = \sum_i x_i \in \{0, \dots, n\}$ be number of heads (random variable)
- n_H follows **binomial distribution** $\text{Bin}(n, \theta)$ with probability mass function

$$\text{Bin}(n_H|n, \theta) = \binom{n}{n_H} \theta^{n_H} (1 - \theta)^{n - n_H}$$

- Expected value: θn
- n_H is a **sufficient statistics**, i.e., there is no additional information about the value of θ in the data (order does not matter)
 - ▶ Observed data is $\{x_1, \dots, x_n\}$
 - ▶ But we use $\mathcal{D} = n_H$ in what follows

$\theta = 0.5, n = 10$

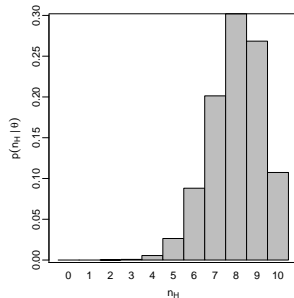
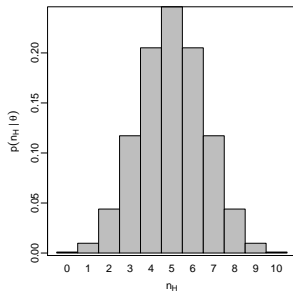


Recap: The binomial distribution (2)

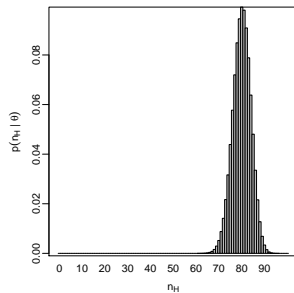
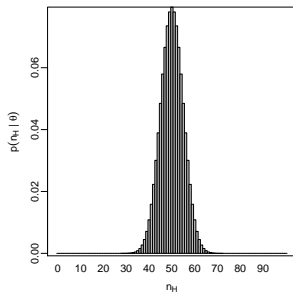
$\theta = 0.5$

$\theta = 0.8$

$n = 10$



$n = 100$



Likelihood and posterior

- Recall:

$$\underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} \propto \underbrace{p(\mathcal{D}|\theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}}$$

- For the beta-binomial model: $\mathcal{D} = n_H$ and $\theta \in [0, 1]$ is the success probability
- **Likelihood** of seeing n_H heads (= the data) and n_T tails after n i.i.d. trials (= the model)

$$p(n_H|\theta) = \text{Bin}(n_H|n, \theta) = \binom{n}{n_H} \theta^{n_H} (1 - \theta)^{n_T}$$

- **Posterior**: If we saw n_H heads, what is our belief about θ ?

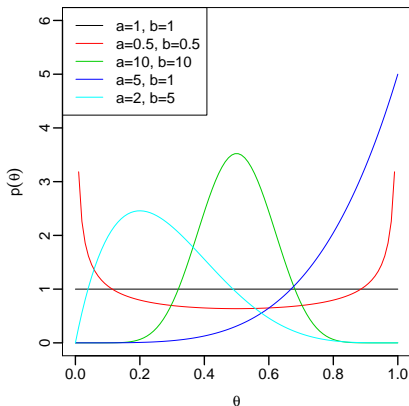
$$p(\theta|n_H) = \frac{p(n_H|\theta)p(\theta)}{p(n_H)} \propto p(n_H|\theta)p(\theta)$$

- $p(n_H) = \int_0^1 p(n_H|\theta)p(\theta) d\theta$ is a normalizing constant
- Posterior depends on **prior** $p(\theta)$; how do we express our prior belief?

Beta distribution (prior)

- Let's pick the $\text{Beta}(\alpha, \beta)$ distr. with **hyperparameters** α, β

$$p(\theta) = \text{Beta}(\theta|\alpha, \beta) = \theta^{\alpha-1}(1-\theta)^{\beta-1}/B(\alpha, \beta)$$



- We will see: $\alpha - 1$ and $\beta - 1$ can be interpreted as number of “prior” successes and failures (**pseudo-counts**)

Beta distribution (properties)

- Notation: $X \sim \text{Beta}(\alpha, \beta)$
- Parameters
 - ▶ Shape $\alpha \in \mathbb{R} > 0$ (prior successes)
 - ▶ Shape $\beta \in \mathbb{R} > 0$ (prior failures)
- Support: $X \in [0, 1]$
- Mean

$$E[X] = \frac{\alpha}{\alpha + \beta}$$

- Mode

$$\frac{\alpha - 1}{\alpha + \beta - 2} \quad \text{for } \alpha, \beta > 1 \text{ (then unimodal)}$$

Beta function

- The **Beta function** $B(x, y)$ arises as the normalizing constant of the Beta distribution
- Selected properties for real $x, y > 0$

$$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

$$B(x, y) = B(y, x)$$

$$B(x+1, y) = B(x, y) \frac{x}{x+y}$$

$$B(x, y+1) = B(x, y) \frac{y}{x+y}$$

$$B(x, y) = \frac{(x-1)!(y-1)!}{(x+y-1)!} \quad \text{for integers } x, y \geq 1$$

$$\binom{n}{k} = \frac{1}{(n+1)B(n-k+1, k+1)} \quad \text{for integers } n \geq k \geq 0$$

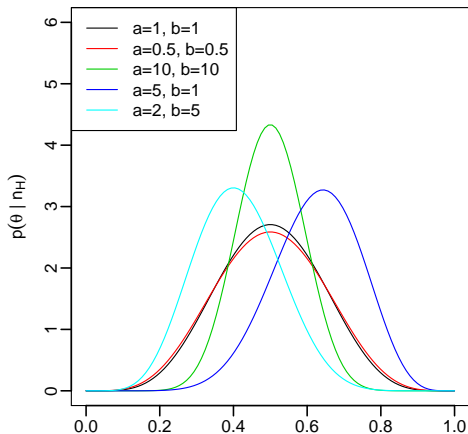
Beta distribution (posterior)

- The posterior distribution is another Beta distribution

$$p(\theta|n_H) \propto \theta^{n_H+\alpha-1}(1-\theta)^{n_T+\beta-1}$$

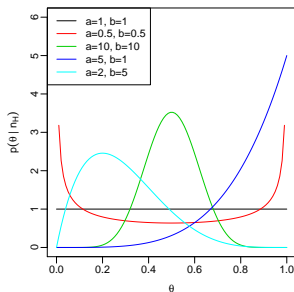
$$p(\theta|n_H) = \text{Beta}(\theta|n_H + \alpha, n_T + \beta)$$

After 10 flips with 5 heads

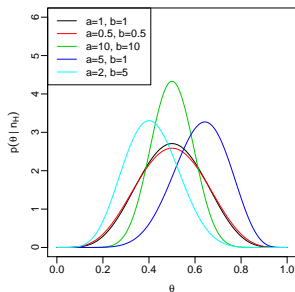


Examples

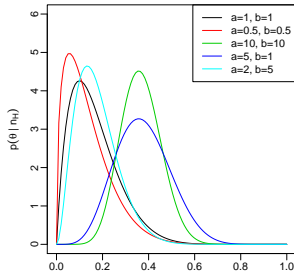
$n = 0$



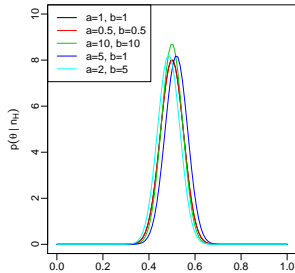
$n = 10, n_H = 5$



$n = 10, n_H = 1$

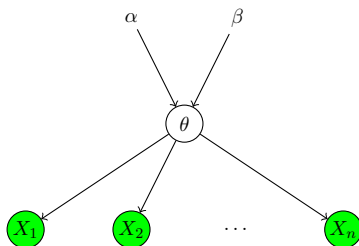


$n = 100, n_H = 50$



Discussion

- Prior and the posterior were from same family of distributions
 - ▶ Such priors are called **conjugate priors** for the likelihood
 - ▶ Convenient to use, often “natural” interpretation
- We made independence assumptions
 - ▶ E.g., all coin flips conditionally independent given θ
 - ▶ Can be represented using *graphical models*



Standard notation

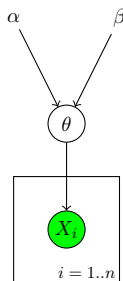


Plate notation

Posterior predictive distribution (1)

- Suppose we do another coin flip x_{n+1} . How likely is it to show heads based on what we learned?
 - ▶ More than one flip → exercise

- Recall:

$$\underbrace{p(\mathcal{D}_{\text{new}}|\mathcal{D})}_{\text{posterior predictive}} = \int \underbrace{p(\mathcal{D}_{\text{new}}|\theta)}_{\text{new-data likelihood}} \underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} d\theta,$$

where $\mathcal{D} = n_H$ and $\mathcal{D}_{\text{new}} = x_{n+1}$

- I.e., we seek **posterior predictive**

$$p(x_{n+1} | n_H) = \int_{\theta} p(x_{n+1}|\theta)p(\theta|n_H) d\theta$$

- ▶ We know the *likelihood* $p(x_{n+1} = 1|\theta) = \theta$, but we don't know θ
- ▶ We know the *posterior* $p(\theta|n_H) = \text{Beta}(\theta|n_H + \alpha, n_T + \beta)$
- To put things together, we start with the joint distribution $p(x_{n+1} = H, \theta|n_H)$ and then marginalize out θ

Posterior predictive distribution (2)

- Observe that $x_{n+1} \perp n_H \mid \theta$ under the independent coin flips assumption
- Implies: $p(x_{n+1} = 1 \mid \theta, n_H) = p(x_{n+1} = 1 \mid \theta) = \theta$
- Using the product rule, we obtain

$$\begin{aligned} p(x_{n+1} = 1, \theta \mid n_H) &= p(x_{n+1} = 1 \mid \theta, n_H) p(\theta \mid n_H) \\ &= p(x_{n+1} = 1 \mid \theta) p(\theta \mid n_H) \\ &= \theta \text{Beta}(\theta \mid n_H + \alpha, n_T + \beta) \\ &= \frac{\theta^{n_H + \alpha} (1 - \theta)^{n_T + \beta - 1}}{B(n_H + \alpha, n_T + \beta)} \end{aligned}$$

- This has a similar form as the posterior, where we “see” one head more

Posterior predictive distribution (2)

- Joint distribution: $p(x_{n+1} = 1, \theta | n_H) = \theta \text{Beta}(\theta | n_H + \alpha, n_T + \beta)$
- To obtain the posterior predictive $p(x_{n+1} = 1 | n_H)$, we marginalize out θ via sum rule

$$\begin{aligned} p(x_{n+1} = 1 | n_H) &= \int_0^1 p(x_{n+1} = 1, \theta | n_H) d\theta \\ &= \int_0^1 \theta^{n_H + \alpha} (1 - \theta)^{n_T + \beta - 1} d\theta / B(n_H + \alpha, n_T + \beta) \\ &= \frac{B(n_H + \alpha + 1, n_T + \beta)}{B(n_H + \alpha, n_T + \beta)} \\ &= \frac{B(n_H + \alpha, n_T + \beta)}{B(n_H + \alpha, n_T + \beta)} \frac{n_H + \alpha}{n + \alpha + \beta} \\ &= \frac{n_H + \alpha}{n + \alpha + \beta} \end{aligned}$$

- Here we used the fact that $B(x + 1, y) = B(x, y)x/(x + y)$

Posterior predictive distribution (3)

- Let's recap
 - ▶ We started with a prior and likelihood to obtain the posterior
 - ▶ We then determined the distribution of new data given the parameters (= likelihood)
 - ▶ We put all together to obtain the **posterior predictive distribution**

$$p(x_{n+1}|n_H) = \text{Ber}\left(x_{n+1} \middle| \frac{n_H + \alpha}{n + \alpha + \beta}\right)$$

- ▶ Simple form (here), interpretable
- **posterior predictive** = \int_{θ} **new-data likelihood** \times **posterior** $d\theta$
 - ▶ Assumes: new data conditionally independent of old data given parameters
- In ML, we use the posterior predictive to
 - ▶ Perform predictions
 - ▶ Evaluate models
 - ▶ Find outliers

Machine Learning

03 – Generative Models for Discrete Data

Part 2: The Dirichlet-Multinomial Model

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Recall: Let's predict

Using Bayes theorem, we obtain estimates

$p(\text{poor} x)$	young	old
student	1	1
CEO	?	0

$p(\text{rich} x)$	young	old
student	0	0
CEO	?	1

Problems

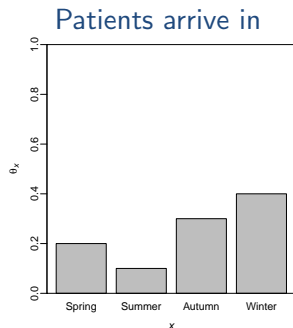
- **Overfitting**: students can't be rich
- **Zero-count problem**: cannot predict for a young CEO since we have never seen one
- ...

Coming up

Adress these problems using a suitable prior
→ the Dirichlet distribution (=conjugate prior)

The categorical distribution

- Consider a random variable $X \sim \text{Cat}(\boldsymbol{\theta})$ over $\{1, \dots, K\}$
 - ▶ K possible values
 - ▶ Probabilities of categories given by $\boldsymbol{\theta} = (\theta_1 \ \theta_2 \ \dots \ \theta_K)^T$
 - ▶ $\boldsymbol{\theta}$ is probability vector (non-negative, sums to one)
 - ▶ I.e., $\text{Cat}(k|\boldsymbol{\theta}) = \theta_k$



The multinomial distribution (1)

- Consider a random variable $X \sim \text{Cat}(\boldsymbol{\theta})$ over $\{1, \dots, K\}$
- How to estimate $\boldsymbol{\theta}$ from n observations of X ?
 - ▶ Independent observations x_1, \dots, x_n
 - ▶ Suppose we see n_k instances of category k
 - ▶ I.e., $n_k = \sum_i \mathbb{I}(x_i = k)$

x_i	k	n_k
Spring	Spring	1
Autumn	Summer	0
Winter	Autumn	2
Winter	Winter	2
Autumn		

The multinomial distribution (2)

- Let $\mathbf{n} = (n_1 \ n_2 \ \cdots \ n_K)^T$, where $\sum_k n_k = n$
 - ▶ As before: \mathbf{n} is sufficient statistics for $\boldsymbol{\theta}$ and we set $\mathcal{D} = \mathbf{n}$
- Given $\boldsymbol{\theta}$, \mathbf{n} follows the **multinomial distribution** $\text{Mu}(n, \boldsymbol{\theta})$

$$\underbrace{p(\mathcal{D}|\boldsymbol{\theta})}_{\text{likelihood}} = \text{Mu}(\mathbf{n}|n, \boldsymbol{\theta}) = \binom{n}{n_1 \cdots n_K} \prod_{k=1}^K \theta_k^{n_k}$$

- Here we use the **multinomial coefficient**

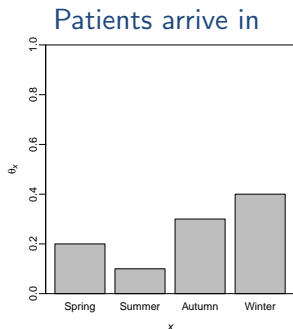
$$\binom{n}{n_1 \cdots n_K} = \frac{n!}{n_1! n_2! \cdots n_K!}$$

- Multinomial distribution is a generalization of the binomial distribution to $K > 2$ categories
 - ▶ $\prod_{k=1}^K \theta_k^{n_k}$ is probability of sequence $(X_i)_{i=1..n}$, in this order
 - ▶ Multinomial coefficient is number of distinct ways to reorder the sequence

The multinomial distribution (3)

- Suppose $\boldsymbol{\theta} = (0.2 \ 0.1 \ 0.3 \ 0.4)^T$
- Suppose $\mathbf{n} = (1 \ 0 \ 2 \ 2)^T$
- Then

$$\begin{aligned}\text{Mu}(\mathbf{n}|\boldsymbol{\theta}, n) &= \binom{5}{1, 0, 2, 2} 0.2^1 0.1^0 0.3^2 0.4^2 \\ &= 30 \cdot 0.0028 = 0.0864\end{aligned}$$



k	n_k
Spring	1
Summer	0
Autumn	2
Winter	2

Maximum likelihood estimation (MLE)

- Let $S_K = \{ \boldsymbol{\theta} \in \mathbb{R}^K : 0 \leq \theta_k \leq 1, \sum_k \theta_k = 1 \}$ be the **probability simplex**
 - ▶ Set of all possible categorical distributions over K values
- One way to estimate $\boldsymbol{\theta}$ is to use maximum likelihood estimation

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MLE}} &= \operatorname{argmax}_{\boldsymbol{\theta} \in S_K} \text{Mu}(\mathbf{n} | \boldsymbol{\theta}, n) = \operatorname{argmax}_{\boldsymbol{\theta} \in S_K} \binom{n}{n_1 \cdots n_K} \prod_{k=1}^K \theta_k^{n_k} \\ &= \operatorname{argmax}_{\boldsymbol{\theta} \in S_K} \prod_{k=1}^K \theta_k^{n_k} && \text{(take logs)} \\ &= \operatorname{argmax}_{\boldsymbol{\theta} \in S_K} \sum_{k=1}^K n_k \log \theta_k = \dots && \text{(exercise)} \\ &= \frac{\mathbf{n}}{n} = \left(\frac{n_1}{n} \quad \frac{n_2}{n} \quad \dots \quad \frac{n_K}{n} \right)^T\end{aligned}$$

- MLE estimate = **relative frequencies** of the categories

MLE on our example

- For $\mathbf{n} = (1 \ 0 \ 2 \ 2)^T$, we obtain $\hat{\boldsymbol{\theta}}_{\text{MLE}} = (0.2 \ 0 \ 0.4 \ 0.4)^T$
- In introductory example, we used MLE for $p(y)$ and $p(\mathbf{x}|y)$
 - ▶ For $p(y)$, we have two values {poor, rich} \rightarrow parameter π
 - ▶ For $p(\mathbf{x}|y)$, we have four “values” {(young, student), (young, CEO), (old, student), (old, CEO)} \rightarrow parameter θ_y

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

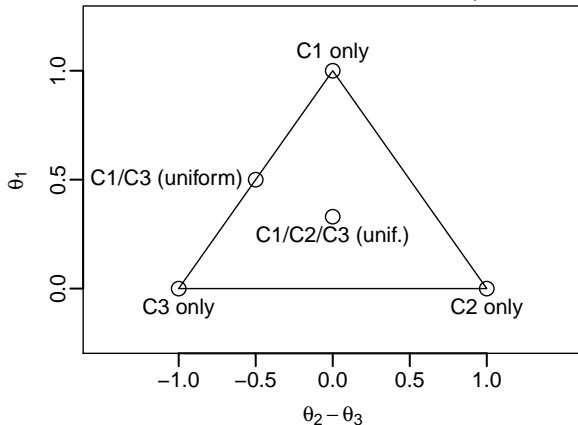
State	$\hat{\pi}$
poor	2/3
rich	1/3

(Age, Position)	$\hat{\theta}_{\text{poor}}$
(young, student)	1/2
(young, CEO)	0
(old, student)	1/2
(old, CEO)	0

(Age, Position)	$\hat{\theta}_{\text{rich}}$
(young, student)	0
(young, CEO)	0
(old, student)	0
(old, CEO)	1

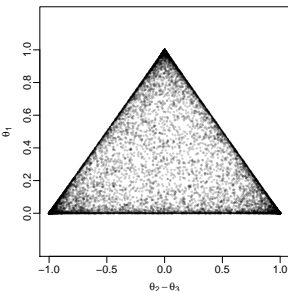
The Dirichlet distribution (1)

- One way to address overfitting/zero-count problem is to add prior on θ
- The conjugate prior of the multinomial is the **Dirichlet distribution** $\text{Dir}(\alpha)$
 - ▶ Distribution over probability vectors $\theta \in S_K$
 - ▶ Elements of θ are probabilities of categories (as before)

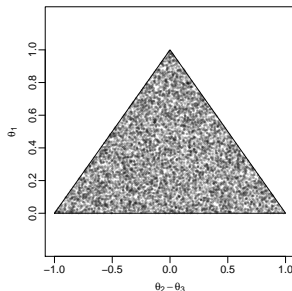


Dirichlet distribution (2)

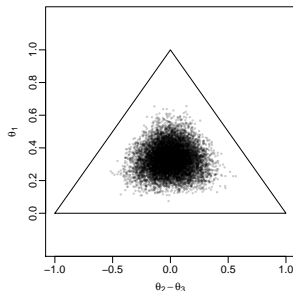
- Parameterized by vector $\alpha \in \mathbb{R}_+^K$ with $\alpha_k > 0$ (**concentration parameters**)
- $\text{Dir}(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^K \theta_k^{\alpha_k-1}$
- Special case: **symmetric Dirichlet distribution**
 - ▶ Single concentration parameter α ; set $\alpha_k = \alpha$
 - ▶ $\alpha \ll 1$: multinomials concentrate around single category (sparse)
 - ▶ $\alpha \gg 1$: multinomials spread uniformly over categories (dense)
 - ▶ $\alpha = 1$: uniform distribution over multinomials



$\alpha = .3$



$\alpha = 1$



$\alpha = 10$

Dirichlet distribution (properties)

- Notation: $\boldsymbol{\theta} \sim \text{Dir}(\boldsymbol{\alpha})$
- Parameters
 - ▶ Number of categories $K \in \mathbb{N}$
 - ▶ Concentration parameters $\boldsymbol{\alpha} \in \mathbb{R}^K > \mathbf{0}$
- Support: $\boldsymbol{\theta} \in S_K$
- Mean

$$E[\theta_k] = \frac{\alpha_k}{\sum_{k'} \alpha_{k'}}$$

- For $\alpha > 1$ (then unimodal), mode is $\boldsymbol{\theta}^{\text{mode}}$ with

$$\theta_k^{\text{mode}} = \frac{\alpha_k - 1}{\sum_{k'} \alpha_{k'} - K}$$

Maximum a posteriori estimation (MAP)

- Recall: posterior \propto likelihood \times prior
- Can show: posterior $p(\boldsymbol{\theta}|\mathbf{n}) = \text{Dir}(\boldsymbol{\theta}|n_1 + \alpha_1, \dots, n_K + \alpha_K)$
 - ▶ Thus the α_k can be interpreted as **pseudo-counts** (as in the beta-binomial model)
- The **maximum a posteriori estimate** is the categorical distribution ($\boldsymbol{\theta}$) that maximizes the posterior
- We have

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \underset{\boldsymbol{\theta} \in S_K}{\operatorname{argmax}} p(\boldsymbol{\theta}|\mathbf{n}) = \underset{\boldsymbol{\theta} \in S_K}{\operatorname{argmax}} \text{Mu}(\mathbf{n}|\mathbf{n}, \boldsymbol{\theta}) \text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) \\ &= \underset{\boldsymbol{\theta} \in S_K}{\operatorname{argmax}} \text{Dir}(\boldsymbol{\theta}|n_1 + \alpha_1, \dots, n_K + \alpha_K) \\ &= \underset{\boldsymbol{\theta} \in S_K}{\operatorname{argmax}} \prod_{k=1}^K \theta_k^{n_k + \alpha_k - 1} \\ &= \frac{(n_1 + \alpha_1 - 1 \quad n_2 + \alpha_2 - 1 \quad \cdots \quad n_K + \alpha_K - 1)^T}{n + \alpha_0 - K},\end{aligned}$$

where $\alpha_0 = \sum_{k=1}^K \alpha_k$

MAP on our example (with sym. Dirichlet)

- With $\alpha = 1$, we obtain the MLE estimate
- With $\alpha = 2$, we obtain **add-one smoothing**; let's fix $\alpha = 2$
- For $\mathbf{n} = (1 \ 0 \ 2 \ 2)^T$, we obtain $\hat{\boldsymbol{\theta}}_{\text{MAP}} = (2/9 \ 1/9 \ 3/9 \ 3/9)^T$
- And for our introductory example

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

State	$\hat{\pi}$
poor	3/5
rich	2/5

(Age, Position)	$\hat{\boldsymbol{\theta}}_{\text{poor}}$
(young, student)	2/6
(young, CEO)	1/6
(old, student)	2/6
(old, CEO)	1/6

(Age, Position)	$\hat{\boldsymbol{\theta}}_{\text{rich}}$
(young, student)	1/5
(young, CEO)	1/5
(old, student)	1/5
(old, CEO)	2/5

Let's predict with add-one smoothing

Using Bayes theorem, we obtain

$p(\text{poor} x, \theta)$	young	old	$p(\text{rich} x, \theta)$	young	old
student	5/7	5/7	student	2/7	2/7
CEO	5/9	5/13	CEO	4/9	8/13

Discussion

- For $\alpha > 1$, we reduce overfitting and avoid the zero-count problem
 - ▶ There can be rich students
 - ▶ Can predict for a young CEO, even though we have never seen one
- **Complexity** of model still present
 - ▶ D binary features, C classes
 - $O(C2^D)$ non-redundant parameters (probability table)
 - ▶ Infeasible to even store for moderately large D
- To combat complexity, let's look at Naive Bayes, the perhaps simplest possible model

Machine Learning

03 – Generative Models for Discrete Data

Part 3: Naive Bayes

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-2

Naive Bayes in a nutshell

- C discrete class labels
 - ▶ Prior class distribution $p(y)$ is a categorical distribution
 - ▶ $O(C)$ parameters for prior
 - ▶ Can often be accurately estimated from sample or domain knowledge
- The **Naive Bayes assumption**: features are conditionally independent given the class label

$$p(\mathbf{x}|y) = \prod_{j=1}^D p(x_j|y)$$

- If used to model $p(\mathbf{x}|y)$, we obtain a **Naive Bayes classifier**
- Called “naive” because features are usually not conditionally independent
 - ▶ But may nevertheless “work well”
 - ▶ Usually $O(CD)$ parameters for class-conditional densities
 - ▶ Relatively immune to overfitting

Naive Bayes classifiers

- Naive Bayes is not “one classifier,” but a family of classifiers
- How to model $p(x_j|y)$?
 - ▶ Depends on type of feature
 - ▶ E.g., Bernoulli distribution for binary features
 - ▶ E.g., categorical distribution for categorical features
 - ▶ E.g., Gaussian distribution for real-valued features
- Which priors to use (if any)?
- How to train and predict?
 - ▶ Point estimates: MLE, MAP
 - ▶ Bayesian inference

Naive Bayes for categorical data

- For the prior class distribution, we use a categorical distribution with parameter vector $\boldsymbol{\pi} \in S_C$ (as before)

$$p(y_i|\boldsymbol{\pi}) = \text{Cat}(y_i|\boldsymbol{\pi})$$

- Let's assume all features are discrete-valued, $x_{ij} \in \{1, \dots, K\}$
- Since features are independent under the naive Bayes assumption, each feature follows a (class-conditional) categorical distribution as well
 - ▶ Consider feature j and class c
 - ▶ Class-conditional distribution $p(X_j|y = c)$ is categorical
 - ▶ With parameter vector $\boldsymbol{\theta}_{cj} \in S_K$, we have

$$p(x_j|y = c, \boldsymbol{\theta}_{cj}) = \text{Cat}(x_j|\boldsymbol{\theta}_{cj})$$

MLE for Naive Bayes for categorical data (1)

- Denote by θ both π and all θ_{cj} 's
- Using the naive Bayes assumption, we obtain likelihood

$$p(\mathbf{x}_i, y_i | \theta) = \text{Cat}(y_i | \pi) \prod_{j=1}^D \text{Cat}(x_{ij} | y_i, \theta_{y_i j})$$

- Using the i.i.d. assumption, the likelihood of the training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ is given by

$$\mathcal{L}(\theta | \mathcal{D}) = \prod_i p(\mathbf{x}_i, y_i | \theta) = \prod_i \text{Cat}(y_i | \pi) \prod_{j=1}^D \text{Cat}(x_{ij} | y_i, \theta_{y_i j})$$

- To obtain the MLE estimate, we can alternatively maximize the **log-likelihood**

$$\ell(\theta | \mathcal{D}) \stackrel{\text{def}}{=} \log \mathcal{L}(\theta | \mathcal{D}) = \sum_{c=1}^C \sum_{i: y_i=c} \log \pi_c + \sum_{j=1}^D \sum_{c=1}^C \sum_{i: y_i=c} \log [\theta_{cj}]_{x_{ij}},$$

MLE for Naive Bayes for categorical data (2)

- Log-likelihood (rewritten)

$$\ell(\boldsymbol{\theta}|\mathcal{D}) \stackrel{\text{def}}{=} \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_{c=1}^C n_c \log \pi_c + \sum_{j=1}^D \sum_{c=1}^C \sum_{k=1}^K n_{cjk} \log [\boldsymbol{\theta}_{cj}]_k,$$

- ▶ $n_c = |\{i : y_i = c\}|$ is the number of examples for class c
- ▶ $n_{cjk} = |\{i : y_i = c, x_j = k\}|$ is the number of examples of class c for which feature j takes value k
- The log-likelihood decomposes into a part that depends on π and a part that depends on the $\boldsymbol{\theta}_{cj}$'s
- We can maximize each part separately and obtain the MLE estimates

$$\hat{\pi}_c = \frac{n_c}{n} \quad \text{and} \quad [\hat{\boldsymbol{\theta}}_{cj}]_k = \frac{n_{cjk}}{n_c}$$

Naive Bayes on our example, MLE

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

State	$\hat{\pi}$
poor	2/3
rich	1/3

Age	$\hat{\theta}_{\text{poor}, \text{Age}}$
young	1/2
old	1/2

Age	$\hat{\theta}_{\text{rich}, \text{Age}}$
young	0
old	1

Position	$\hat{\theta}_{\text{poor}, \text{Pos}}$
student	1
CEO	0

Position	$\hat{\theta}_{\text{rich}, \text{Pos}}$
student	0
CEO	1

Let's predict: MLE and Naive Bayes

Using Bayes theorem, we obtain

$p(\text{poor} x, \hat{\theta})$	young	old	$p(\text{rich} x, \hat{\theta})$	young	old
student	1	1	student	0	0
CEO	?	0	CEO	?	1

Discussion

- Same result as original approach (coincidentally)
- Overfitting is and zero-count problem still persists
 - ▶ Both problems are mildened: e.g., if we see each value at least once for every feature j and for every class, then the zero-count problem does not occur
- **Simple** model
 - ▶ D features, each with K possible values, C classes
→ $O(CDK)$ space
 - ▶ Training is simple and linear in data size: compute histograms

Naive Bayes on our example, MAP

- As before, we can put a Dirichlet prior on the parameters of each multinomial to avoid the zero-count problem
- As before, for MAP estimation this means adding pseudo-counts
- E.g., with $\alpha = 2$ for all multinomials and MAP estimates:

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

State	$\hat{\pi}$
poor	3/5
rich	2/5

Age	$\hat{\theta}_{\text{poor, Age}}$
young	2/4
old	2/4

Age	$\hat{\theta}_{\text{rich, Age}}$
young	1/3
old	2/3

Position	$\hat{\theta}_{\text{poor, Pos}}$
student	3/4
CEO	1/4

Position	$\hat{\theta}_{\text{rich, Pos}}$
student	1/3
CEO	2/3

Let's predict: MAP and Naive Bayes

Using Bayes theorem, we obtain

$p(\text{poor} \mathbf{x}, \hat{\theta})$	young	old	$p(\text{rich} \mathbf{x}, \hat{\theta})$	young	old
student	81/97	81/113	student	16/97	32/113
CEO	27/59	27/91	CEO	32/59	64/91

Discussion

- For $\alpha > 1$, we reduce overfitting and avoid the zero-count problem
- **Simple** model

Bayesian Naive Bayes

- The fully Bayesian approach to prediction is to marginalize out all parameters
- Posterior predictive is

$$p(y, \mathbf{x} | \mathcal{D}) = \left(\int \text{Cat}(y | \boldsymbol{\pi}) p(\boldsymbol{\pi} | \mathcal{D}) d\boldsymbol{\pi} \right) \prod_{j=1}^D \left(\int \text{Cat}(x_j | y, \boldsymbol{\theta}_{yj}) p(\boldsymbol{\theta}_{yj} | \mathcal{D}) d\boldsymbol{\theta}_{yj} \right)$$

- Suppose we use a Dirichlet prior. We obtain **posterior means**

$$\bar{\pi}_c = \frac{n_c + \alpha_c}{n + \sum_c \alpha_c} \quad \text{and} \quad [\bar{\theta}_{cj}]_k = \frac{n_{cjk} + \alpha_{cjk}}{n_c + \sum_k \alpha_{cjk}}$$

and posterior predictive cond. on new example \mathbf{x}

$$p(y = c | \mathbf{x}, \mathcal{D}) \propto \bar{\pi}_c \prod_{j=1}^D [\bar{\theta}_{cj}]_{x_j}$$

Naive Bayes on our example, Bayesian

- E.g., with $\alpha = 2$ on all multinomials:

Age	Position	State
young	student	poor
old	student	poor
old	CEO	rich

State	$\bar{\pi}$
poor	4/7
rich	3/7

Age	$\bar{\theta}_{\text{poor, Age}}$
young	3/6
old	3/6

Age	$\bar{\theta}_{\text{rich, Age}}$
young	2/5
old	3/5

Position	$\bar{\theta}_{\text{poor, Pos}}$
student	4/6
CEO	2/6

Position	$\bar{\theta}_{\text{rich, Pos}}$
student	2/5
CEO	3/5

Let's predict: Bayesian Naive Bayes

Using Bayes theorem, we obtain

$p(\text{poor} \mathbf{x}, \mathcal{D})$	young	old	$p(\text{rich} \mathbf{x}, \mathcal{D})$	young	old
student	25/34	50/77	student	9/34	27/77
CEO	25/52	50/131	CEO	27/52	81/131

Discussion

- Hyperparameters can be estimated as part of model selection
- Add-one smoothing is also commonly used
 - ▶ Can be interpreted as MAP estimate with Dirichlet prior, $\alpha = 2$
 - ▶ **Can be interpreted as fully Bayesian inference with uniform prior** (= Dirichlet prior, $\alpha = 1$)
- **Simple** model, cheap to compute, reduced overfitting, no zero-count problem
- But beware: Naive Bayes assumption is a strong assumption

Machine Learning

04 – Classifiers for Continuous Data

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Outline (Classifiers for Continuous Data)

1. Logistic Regression
2. Softmax regression
3. Gaussian Naive Bayes

Lessons learned

- Logistic regression is a simple discriminative binary classifier
 - ▶ Assumes that log odds is linear function of input features
 - ▶ A generalized linear model
 - ▶ Efficient to train, interpretable
- Softmax regression (multinomial logistic regression) is a generalization for multiclass classification
- Gaussian Naive Bayes is a simple generative classifier for continuous data
 - ▶ Naive Bayes assumption
 - ▶ Class-conditional densities $p(x_j|y)$ assumed Gaussian
 - ▶ Model fitting = estimate means and variances
 - ▶ Related to logistic regression
- Naive Bayes or logistic regression?
 - ▶ It depends...

Suggested readings

- Murphy, Ch. 10, *Logistic Regression*
- Tom Mitchell, [Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression](#) (draft), 2016

Machine Learning

04 – Classifiers for Continuous Data

Part 1: Logistic Regression

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Logistic regression

- **Logistic regression** (also: *logit regression*, *logit model*) is a discriminative classifier
 - ▶ Given: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$
 - ▶ Sought: $p(y|\mathbf{x})$
 - ▶ Dependent variable y is discrete
- Binary classifier, i.e., $y \in \{0, 1\}$ is binary
- In a nutshell
 - ▶ Assume y generated from a coin flip
 - ▶ Log odds of success are linear in \mathbf{x}
 - ▶ A generalized linear model
- Variants
 - ▶ **Softmax regression** (*multinomial logistic regression*):
 y categorical (with more than two classes)
 - ▶ **Ordinal logistic regression**: y ordered (not covered here)

Bernoulli model and odds

- Recall the **Bernoulli model** $\text{Ber}(\theta)$ for a single coin flip
 - ▶ i.e., $Y \sim \text{Ber}(\theta)$
 - ▶ $\theta \in [0, 1]$ is *success probability*
 - ▶ $p(y|\theta) = \text{Ber}(y|\theta) = \begin{cases} \theta & y = 1 \\ 1 - \theta & y = 0 \end{cases}$
 - ▶ $E[Y] = \theta$, $\text{var}[Y] = \theta(1 - \theta)$

- What are the **odds** of success?

- ▶ For success probability θ , define the **odds on**

$$\text{odds}(\theta) = \frac{\theta}{1 - \theta}$$

- ▶ Example: $\theta = 0.1$, $\text{odds}(\theta) = 1/9$ (also written 1:9)

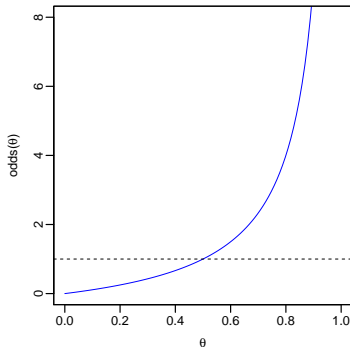
- $\text{odds}(1 - \theta)$ is called **odds against**

- ▶ 9:1 odds against = 1:9 odds on

- When $\text{odds}(\theta) = 1$, we say **odds even**

Odds decision function

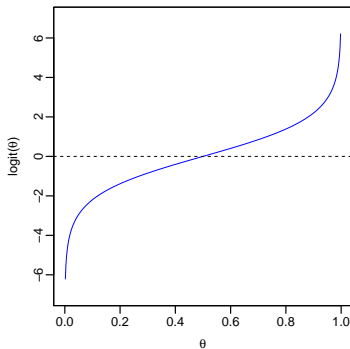
- Suppose you are asked whether you expect success
 - ▶ Say yes, when $\text{odds}(\theta) > 1$ (i.e., $\theta > 0.5$)
 - ▶ Say no, when $\text{odds}(\theta) \leq 1$ (i.e., $\theta \leq 0.5$)
 - ▶ Here we used **decision threshold** 1 (i.e., 0.5 on θ)



Note: $\text{odds}(\theta) \in [0, \infty]$

Logit

- $\log \text{odds}(\theta)$ is referred to as the *log odds* or **logit** function
- We have $\text{logit}(\theta) \stackrel{\text{def}}{=} \log \text{odds}(\theta) = \log \theta - \log(1 - \theta)$
- The decision function then becomes
 - ▶ Yes, when $\text{logit}(\theta) > 0$
 - ▶ No, when $\text{logit}(\theta) \leq 0$



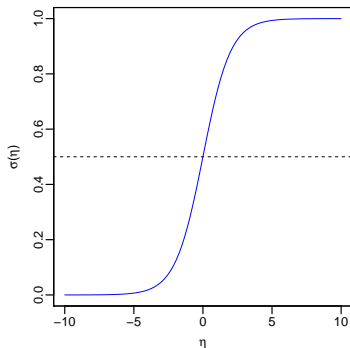
Note: $\text{logit}(\theta) \in [-\infty, \infty]$

The logistic function

- Given a logit of η , what is the success probability θ ?

$$\text{logit}(\theta) = \eta \quad \Longleftrightarrow \quad \theta = \sigma(\eta) \stackrel{\text{def}}{=} \frac{1}{1 + \exp(-\eta)}$$

- Here $\sigma(\eta)$ denotes the **logistic function**
 - ▶ Inverse of logit function: $\sigma(\text{logit}(\theta)) = \theta$
 - ▶ We also write $\sigma^{-1}(\theta)$ for $\text{logit}(\theta)$



Important properties

$$\sigma(\eta) = \frac{1}{1 + \exp(-\eta)} = \frac{\exp(\eta)}{1 + \exp(\eta)}$$

$$\frac{\partial}{\partial \eta} \sigma(\eta) = \sigma(\eta)(1 - \sigma(\eta))$$

$$1 - \sigma(\eta) = \sigma(-\eta)$$

$$\sigma^{-1}(\theta) = \log\left(\frac{\theta}{1 - \theta}\right) = \text{logit}(\theta)$$

Probability (θ)	Logit scores (η)
0.0001	-9.21
0.001	-6.90
0.01	-4.59
0.05	-2.94
0.1	-2.19
0.5	0.00
0.9	2.19
0.95	2.94
0.99	4.59
0.999	6.90
0.9999	9.21

Logistic regression

- Let $\mathbf{x} \in \mathbb{R}^D$ be an input
- Logistic regression then *assumes* that

log odds on y = linear function of \mathbf{x}

- Linear function can be described by parameters w_0, \dots, w_D

$$\eta = w_0 + w_1x_1 + w_2x_2 + \dots + w_Dx_D$$

- We have

$$\begin{aligned} p(y = 1 | \mathbf{x}, \{w\}_j) &= \sigma\left(w_0 + \sum_j w_j x_j\right) \\ &= \frac{1}{1 + \exp(-(w_0 + \sum_j w_j x_j))} \end{aligned}$$

$$\begin{aligned} p(y = 0 | \mathbf{x}, \{w\}_j) &= 1 - \sigma\left(w_0 + \sum_j w_j x_j\right) \\ &= \frac{1}{1 + \exp(w_0 + \sum_j w_j x_j)} \end{aligned}$$

Notation

- We collect the parameters into a **weight vector**

$$\mathbf{w} = (w_1 \quad w_2 \quad \cdots \quad w_D)^\top$$

and a **bias** term w_0

- We can then use inner products

$$\eta = w_0 + \sum_{j=1}^D w_j x_j = w_0 + \mathbf{w}^\top \mathbf{x} = w_0 + \langle \mathbf{w}, \mathbf{x} \rangle$$

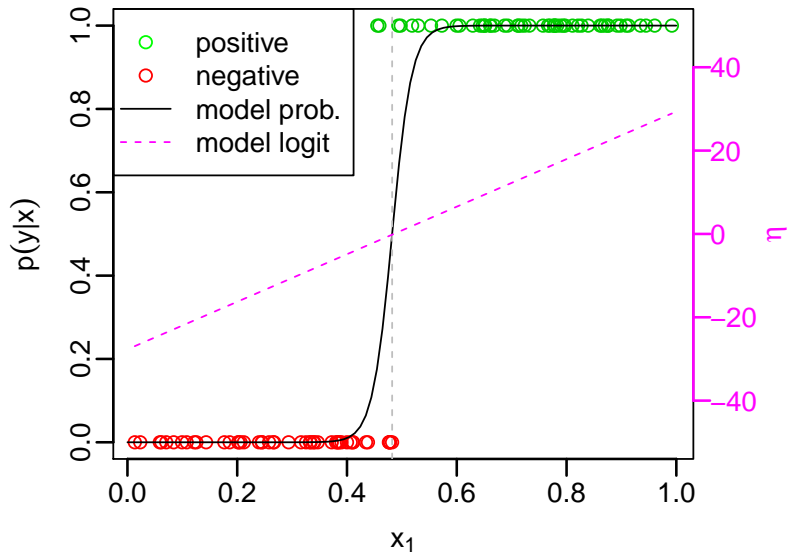
- and obtain

$$p(y = 1 | \mathbf{x}, w_0, \mathbf{w}) = \sigma(w_0 + \langle \mathbf{w}, \mathbf{x} \rangle)$$

$$p(y = 0 | \mathbf{x}, w_0, \mathbf{w}) = \sigma(-(w_0 + \langle \mathbf{w}, \mathbf{x} \rangle))$$

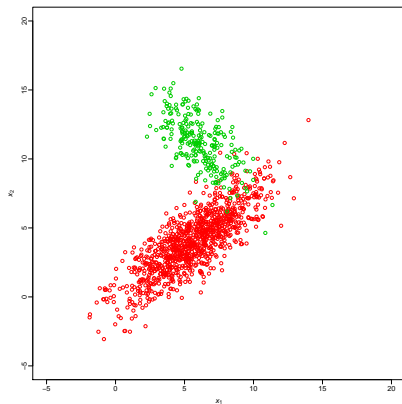
$$p(y | \mathbf{x}, w_0, \mathbf{w}) = \text{Ber}(y | \sigma(w_0 + \langle \mathbf{w}, \mathbf{x} \rangle))$$

Example (1D)

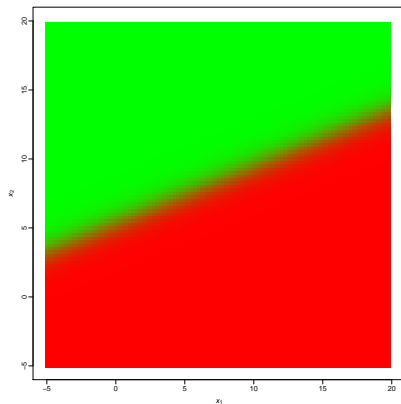


Example (2D)

Data



Prediction



Bias feature

- To simplify implementation or notation, we may omit the explicit bias term of logistic regression by adding a special **bias feature** $x_0 = 1$ to inputs

$$\begin{aligned}\mathbf{x} &= (1 \quad x_1 \quad x_2 \quad \cdots \quad x_D)^\top \\ \mathbf{w} &= (w_0 \quad w_1 \quad w_2 \quad \cdots \quad w_D)^\top \\ \eta &= \langle \mathbf{w}, \mathbf{x} \rangle\end{aligned}$$

- Then

$$\begin{aligned}p(y = 1 | \mathbf{x}, \mathbf{w}) &= \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \\ p(y = 0 | \mathbf{x}, \mathbf{w}) &= \sigma(-\langle \mathbf{w}, \mathbf{x} \rangle) \\ p(y | \mathbf{x}, \mathbf{w}) &= \text{Ber}(y | \sigma(\langle \mathbf{w}, \mathbf{x} \rangle))\end{aligned}$$

- We will mostly focus on the case without a bias term for simplicity

Perceptron and decision boundary

- Let $\mathbf{w} \in \mathbb{R}^D$. The **perceptron** is a classifier with decision rule

$$\hat{y} = \begin{cases} 0 & \langle \mathbf{w}, \mathbf{x} \rangle < 0 \\ 1 & \langle \mathbf{w}, \mathbf{x} \rangle > 0 \end{cases}$$

- If we use logistic regression and predict the most likely class, we obtain the same decision rule
- The **decision boundary of a classifier** is the set of data points for which the classifier is unsure in that multiple classes achieve the highest possible probability or score
- I.e., \mathbf{x} is on the decision boundary if there exist $c_1 \neq c_2$ s.t.

$$p(y = c_1 | \mathbf{x}) = p(y = c_2 | \mathbf{x}) = \max_c p(y = c | \mathbf{x})$$

- For logistic regression/the perceptron, the decision boundary is

$$\{ \mathbf{x} : \langle \mathbf{w}, \mathbf{x} \rangle = 0 \}$$

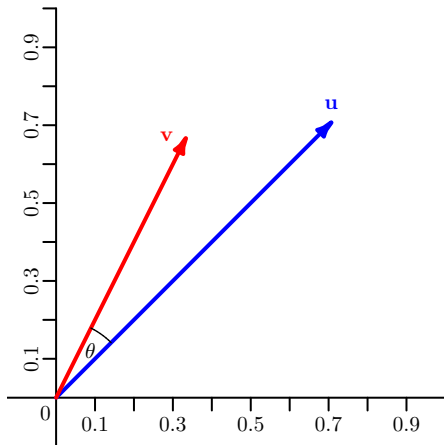
- What does this mean?

Inner product (geometric interpretation)

The geometric interpretation of the inner product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is given by

$$\langle \mathbf{u}, \mathbf{v} \rangle = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta,$$

where $-\pi \leq \theta \leq \pi$ denotes the angle between \mathbf{u} and \mathbf{v} .



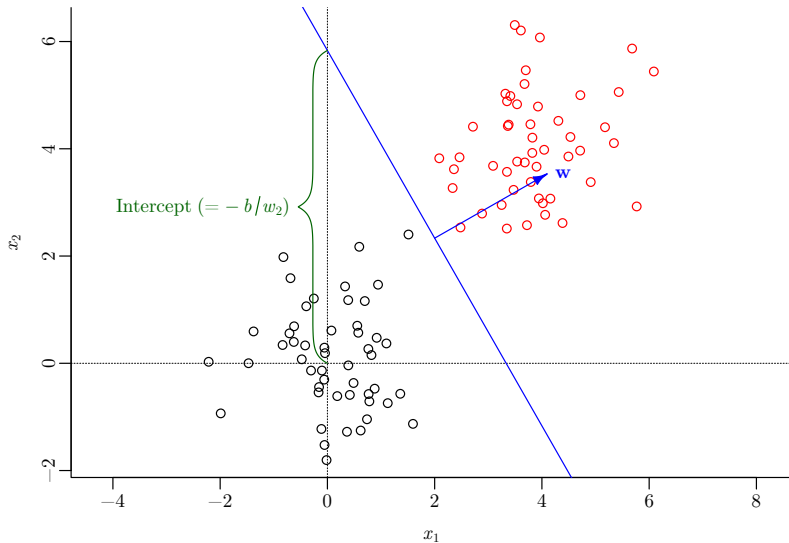
Understanding perceptrons (1)

- Consider the classification rule

$$\hat{y} = \begin{cases} 0 & \langle \mathbf{w}, \mathbf{x} \rangle < 0 \\ 1 & \langle \mathbf{w}, \mathbf{x} \rangle > 0 \end{cases} = \begin{cases} 0 & \|\mathbf{w}\| \|\mathbf{x}\| \cos \angle(\mathbf{w}, \mathbf{x}) < 0 \\ 1 & \|\mathbf{w}\| \|\mathbf{x}\| \cos \angle(\mathbf{w}, \mathbf{x}) > 0 \end{cases}$$

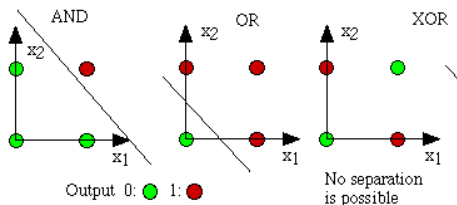
- Observe
 - ▶ Negative instances have angle $|\angle(\mathbf{w}, \mathbf{x})| > 90^\circ$
 - ▶ The decision boundary has angle $|\angle(\mathbf{w}, \mathbf{x})| = 90^\circ$ (includes origin)
 - ▶ Positive instances have angle $|\angle(\mathbf{w}, \mathbf{x})| < 90^\circ$
- Decision boundary defines a **hyperplane** with **normal vector** \mathbf{w}
 - ▶ = set of points orthogonal to \mathbf{w} (a $D - 1$ dimensional subspace of \mathbb{R}^D)
 - ▶ E.g., for two dimensions we have the line $w_1x_1 + w_2x_2 = 0$
(and consequently $x_2 = -\frac{w_1}{w_2}x_1$)
- If we add a bias term $b \neq 0$, we obtain an **affine hyperplane**
 - ▶ I.e., does not go through origin
 - ▶ For two dimensions, intercept is $-b/w_2$
- A classifier for which the decision boundary is (always) a hyperplane is called a **linear classifier**

Understanding perceptrons (2)



What can perceptrons learn?

- Perceptrons (with bias) can classify perfectly if there exists an affine hyperplane that separates the classes
 - ▶ We then say the data is **linearly separable**
- Otherwise, the perceptron must make errors on some inputs
- This is quite limited; e.g., perceptrons cannot learn the XOR function



Discussion (1)

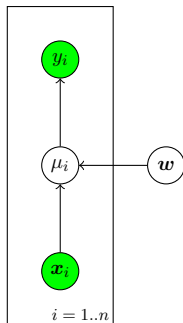
- Logistic regression is one of the most popular classifiers
- Provides probabilities
- Easy to fit (more later)
- Easy to interpret
 - ▶ Suppose we want to predict probability of getting lung cancer
 - ▶ Features: number of cigarettes per day (x_1), minutes of exercise per day (x_2)
 - ▶ Estimated weights: $\hat{\mathbf{w}} = (1.3 \quad -1.1)^\top$
 - ▶ Means: for every cigarette, odds on getting cancer increased by factor $\exp(1.3) \approx 3.7$
- Is a binary classifier, but can be extended to more than two classes (coming up next)
- Is a linear classifier, but can be extended to handle non-linear decision boundaries (*kernel logistic regression*, more later)

Discussion (2)

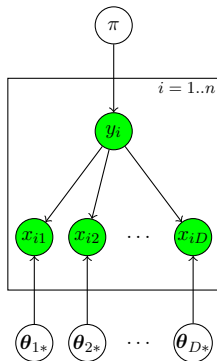
- Virtually any feature function can be used
- For categorical features, use **one-hot encoding**
 - ▶ Encode with binary vector with one element per possible value of the feature \rightarrow one weight per possible value
 - ▶ Entry that corresponds to actual value set to 1; rest 0
 - ▶ Example: $X \in \{\text{red}, \text{green}, \text{blue}\}$
 - ▶ Then $x = \text{green}$ becomes $x = (0 \ 1 \ 0)^\top$
- If features are linearly dependent, MLE weights underdetermined
 - ▶ Perform feature selection
 - ▶ Use a prior / regularization
- When examples are linearly separable, MLE weights “infinite”
 - ▶ Use a prior / regularization
- Under mild conditions, can handle imbalanced classes in the data to some extent

Graphical models

Here without explicitly showing priors.



Logistic regression
(training)



Naive Bayes
(training)

Excursion: Generalized linear models

- Logistic regression, where $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \mathbf{x}))$, is an example of a **generalized linear model**
- Generally, three components
 - ▶ An exponential family of **probability distributions**: Bernoulli
 - ▶ A **linear predictor**: $\eta = \mathbf{w}^\top \mathbf{x}$
 - ▶ A **link function** g that connects the mean μ of the cond. distribution of y with the linear predictor: $\eta = g(\mu)$
 - ▶ Here: $\mu = \theta$ and $\eta = g(\mu) = \sigma^{-1}(\mu) = \text{logit}(\mu)$
- We can pick other distributions and link functions
 - ▶ E.g., normal distribution + identity link \rightarrow linear regression
 - ▶ E.g., Poisson distribution + log link \rightarrow Poisson regression
 - ▶ See [Wikipedia](#) or lecture Cross Sectional Data Analysis (SoWi)
- More on exponential family in exercise

Machine Learning

04 – Classifiers for Continuous Data

Part 2: Softmax Regression

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Softmax regression

- Recall logistic regression
 - ▶ Log odds modeled as a linear function of the features
 - ▶ Technically, first compute logit score $\eta = \langle \mathbf{w}, \mathbf{x} \rangle$
 - ▶ Then apply the logistic function to obtain success probability
$$p(y = 1 | \mathbf{x}, \mathbf{w}) = \sigma(\eta)$$
- **Softmax regression** (or *multinomial logistic regression*)
generalizes logistic regression to multiple classes
- Consider a multiclass problem with C classes
 - ▶ Instead of one linear function, use C linear functions
$$\eta_1 = \langle \mathbf{w}_1, \mathbf{x} \rangle, \dots, \eta_C = \langle \mathbf{w}_C, \mathbf{x} \rangle$$
 (one per class)
 - ▶ Can also be interpreted as log odds under certain conditions (exercise)
 - ▶ Instead of the logistic function, use the softmax function to obtain
$$p(Y | \mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_C)$$

The softmax function (1)

- The **softmax function** $S(\boldsymbol{\eta})$
 - ▶ Takes a real vector $\boldsymbol{\eta} = (\eta_1, \dots, \eta_C)^\top \in \mathbb{R}^C$
 - ▶ And transforms it into an C -dimensional probability vector $S(\boldsymbol{\eta})$

$$S(\boldsymbol{\eta})_c = \frac{\exp(\eta_c)}{\sum_{c'=1}^C \exp(\eta_{c'})}$$

- ▶ Called this way because it exaggerates differences and acts somewhat like the max function (approximates indicator function of largest coefficient)

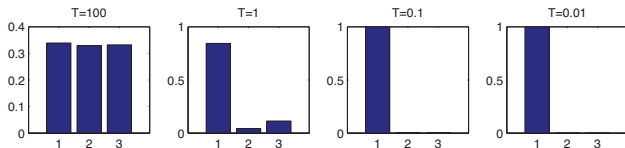
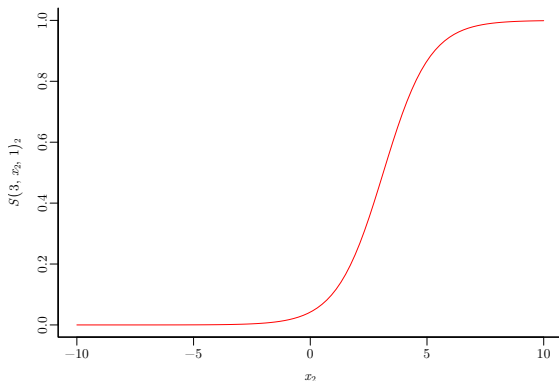


Figure 4.4 Softmax distribution $S(\boldsymbol{\eta}/T)$, where $\boldsymbol{\eta} = (3, 0, 1)$, at different temperatures T . When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is “spiky”, with all its mass on the largest element. Figure generated by `softmaxDemo2`.

The softmax function (2)

Here is a plot of $S(3, x_2, 1)_2$.



- When we fix all but one argument and look at the corresponding output, we obtain a shifted logistic function

Logistic regression and softmax

- Recall logistic regression model

$$p(y = 0|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(\langle \mathbf{w}, \mathbf{x} \rangle)}$$

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\langle \mathbf{w}, \mathbf{x} \rangle)} = \frac{\exp(\langle \mathbf{w}, \mathbf{x} \rangle)}{1 + \exp(\langle \mathbf{w}, \mathbf{x} \rangle)}$$

- We can express this with the softmax function

$$p(y = 0|\mathbf{x}, \mathbf{w}) = S(0, \langle \mathbf{w}, \mathbf{x} \rangle)_1$$

$$p(y = 1|\mathbf{x}, \mathbf{w}) = S(0, \langle \mathbf{w}, \mathbf{x} \rangle)_2$$

→ Can be seen as a generalization of the logistic function

Softmax regression

- Let $\mathbf{W} = (\mathbf{w}_1 \ \mathbf{w}_2 \ \cdots \ \mathbf{w}_C)$
- For C classes, softmax regression uses the model

$$p(y = c | \mathbf{x}, \mathbf{W}) = S(\langle \mathbf{w}_1, \mathbf{x} \rangle, \dots, \langle \mathbf{w}_C, \mathbf{x} \rangle)_c = S(\mathbf{W}^\top \mathbf{x})_c$$

- The weight vectors are redundant (exercise)
 - ▶ We get non-redundant parameters if we set $\mathbf{w}_C = \mathbf{0}$
- Maximum likelihood estimation
 - ▶ Let $\mathbf{p}_i = S(\mathbf{W}^\top \mathbf{x}_i) \in \mathcal{S}_C$ be the predicted probabilities for example i
 - ▶ Likelihood is given by

$$\mathcal{L}(\mathbf{W} | \mathbf{X}, \mathbf{y}) = \prod_{i=1}^N p(y_i | \mathbf{x}_i, \mathbf{W}) = \prod_{i=1}^N S(\mathbf{W}^\top \mathbf{x}_i)_{y_i} = \prod_{i=1}^N p_{iy_i}$$

- ▶ Gradient of neg. log-likelihood (lecture 05)

$$\nabla_{\mathbf{w}_c^\top} - \ell(\mathbf{W} | \mathbf{X}, \mathbf{y}) = \sum_i (p_{ic} - \mathbb{I}(y_i = c)) \mathbf{x}_i^\top$$

Machine Learning

04 – Classifiers for Continuous Data

Part 3: Gaussian Naive Bayes

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-2

Recall: Naive Bayes

- C discrete class labels
 - ▶ Prior class distribution $p(y)$ is a categorical distribution
 - ▶ $O(C)$ parameters for prior
 - ▶ Can often be accurately estimated from sample or domain knowledge
- The **Naive Bayes assumption**: features are conditionally independent given the class label

$$p(\mathbf{x}|y) = \prod_{j=1}^D p(x_j|y)$$

- If used to model $p(\mathbf{x}|y)$, we obtain a **Naive Bayes classifier**
- Called “naive” because features are usually not conditionally independent
 - ▶ But may nevertheless “work well”
 - ▶ Usually $O(CD)$ parameters for class-conditional densities
 - ▶ Relatively immune to overfitting

Gaussian Naive Bayes classifier

- When features are continuous, $p(x_j|y)$ is continuous
 - ▶ I.e., feature distributions are modeled with a continuous distribution
- **Gaussian Naive Bayes** (GNB): use Gaussian distribution
 - ▶ $p(x_j|y = c) \sim \mathcal{N}(\mu_{jc}, \sigma_{jc}^2)$
 - ▶ μ_{jc} is mean parameter of feature j in class c
 - ▶ σ_{jc}^2 is variance parameter of feature j in class c
 - ▶ Prior class probabilities as before
 - ▶ Total number of parameters: $O(DC)$
- Maximum likelihood estimate

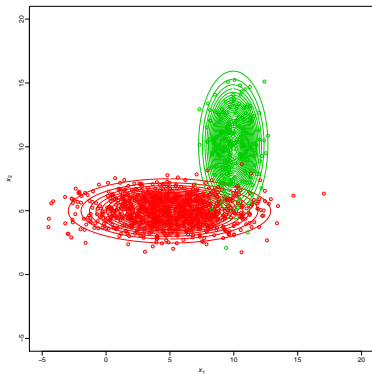
$$\hat{\mu}_{jc} = \frac{\sum_{i:y_i=c} x_{ij}}{n_c} \quad \text{and} \quad \hat{\sigma}_{jc}^2 = \frac{\sum_{i:y_i=c} (x_{ij} - \hat{\mu}_{jc})^2}{n_c},$$

where n_C refers to number of training examples of class c

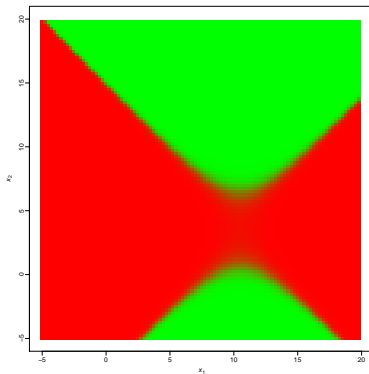
Example (GNB assumptions hold)

1000 red points, 250 green points

Data and MLE fit



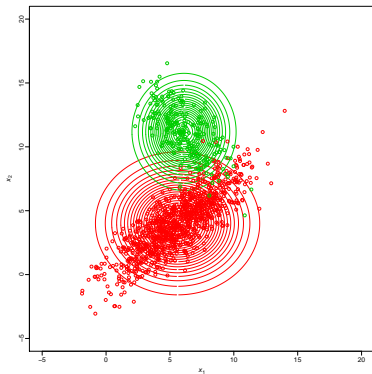
Posterior class probabilities



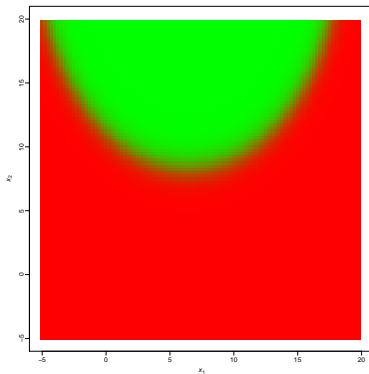
Example (GNB assumptions do not hold)

1000 red points, 250 green points

Data and MLE fit



Posterior class probabilities

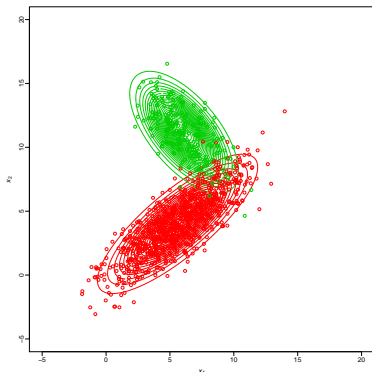


Excursion: Quadratic Discriminant Analysis (QDA)

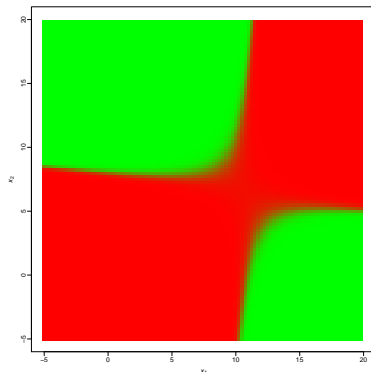
QDA models class-conditional density $p(\mathbf{x}|y)$ via multivariate Gaussian

- I.e., $p(\mathbf{x}|y = c) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
- Parameters can be fit separately per class
- Generative, but not Naive Bayes classifier

Data and MLE fit



Posterior class probabilities

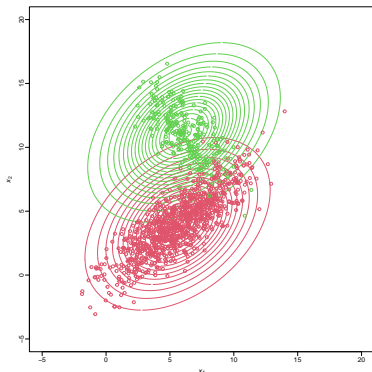


Excursion: Linear Discriminant Analysis (LDA)

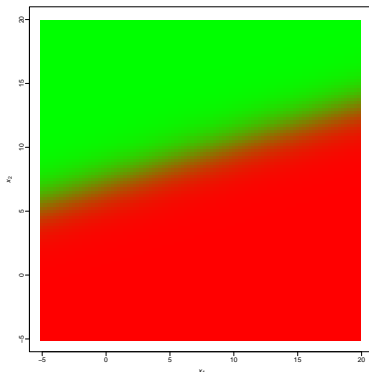
LDA is a variant of QDA that additionally assumes **homoscedasticity**.

- Means that covariances among classes are equal
- I.e., $p(\mathbf{x}|y = c) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma})$ (note: $\boldsymbol{\Sigma}$, not $\boldsymbol{\Sigma}_c$)
- Gives a linear classifier

Data and MLE fit



Posterior class probabilities



Gaussian Naive Bayes and logistic regression (1)

- We can also make the homoscedasticity assumption for Gaussian Naive Bayes
 - ▶ Then, $p(x_j|y) = \mathcal{N}(x_j|\mu_{jy}, \sigma_j)$
 - ▶ Let's call this classifier **GNB=**
- For binary classification, we will now show:
 - ▶ For every GNB= classifier, there is an equivalent logistic regression classifier
 - ▶ But not the other way around
 - ▶ Consequently, GNB= is a linear classifier
 - ▶ Consequently, logistic regression is “more powerful” than GNB=
- For multi-class classification
 - ▶ GNB= and softmax regression each contain unique classifiers
 - ▶ GNB= still linear
 - ▶ More in exercise

Gaussian Naive Bayes and logistic regression (2)

Let's look at the parametric form of $p(y = 1|\mathbf{x})$.

$$\begin{aligned} p(y = 1|\mathbf{x}) &= \frac{p(y = 1)p(\mathbf{x}|y = 1)}{\sum_{y'=0}^1 p(y')p(\mathbf{x}|y')} = \dots \\ &= \frac{1}{1 + \exp\left(\ln \frac{1-\pi}{\pi} + \sum_j \left[\frac{\mu_{j0} - \mu_{j1}}{\sigma_j^2} x_j + \frac{\mu_{j1}^2 - \mu_{j0}^2}{2\sigma_j^2} \right]\right)} \\ &= \frac{1}{1 + \exp(-(w_0 + \mathbf{w}^\top \mathbf{x}))}, \end{aligned}$$

where

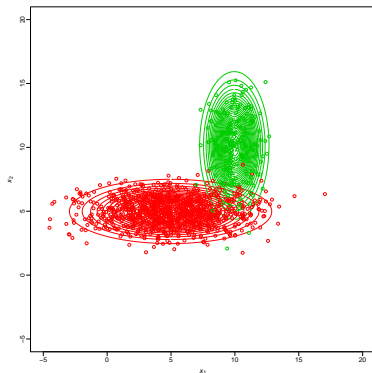
$$\begin{aligned} -w_0 &= \ln \frac{1-\pi}{\pi} + \sum_j \frac{\mu_{j1}^2 - \mu_{j0}^2}{2\sigma_j^2} \\ -w_j &= \frac{\mu_{j0} - \mu_{j1}}{\sigma_j^2} \end{aligned}$$

Thus, under our assumptions, we can express $p(y|\mathbf{x})$ in the parametric form of logistic regression.

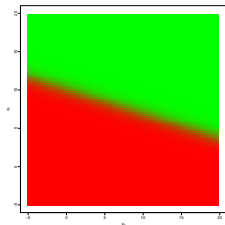
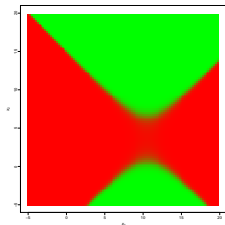
Example (GNB assumptions hold)

1000 red points, 250 green points

Data & MLE fit



Posterior class prob. (GNB)

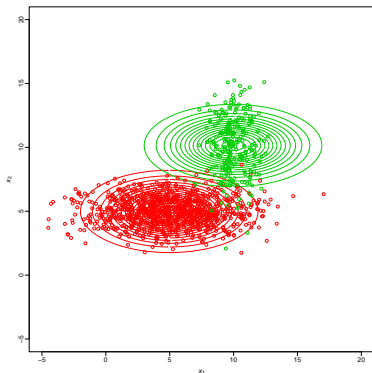


Posterior class prob. (LR)

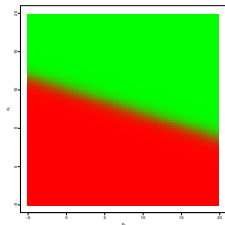
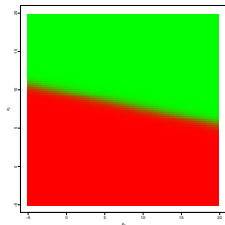
Same example with GNB=

1000 red points, 250 green points

Data & MLE fit



Posterior class prob. (GNB=)

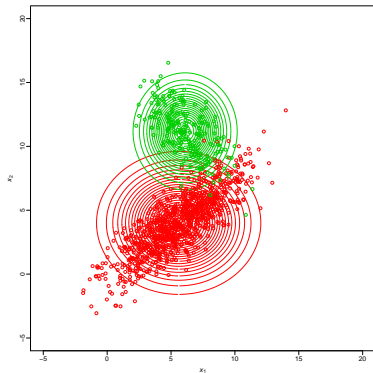


Posterior class prob. (LR)

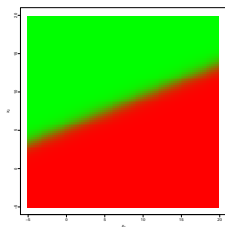
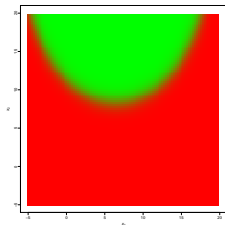
Example (GNB assumptions do not hold)

1000 red points, 250 green points

Data & MLE fit



Posterior class prob. (GNB)

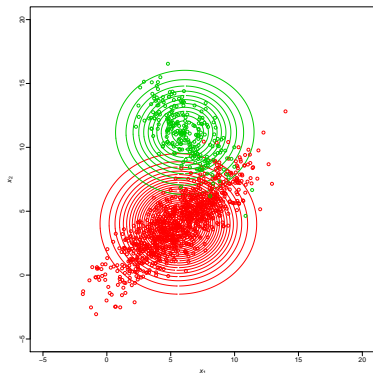


Posterior class prob. (LR)

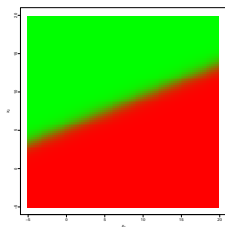
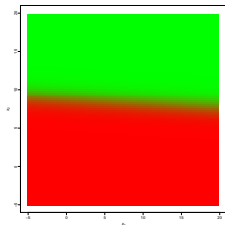
Same example with GNB=

1000 red points, 250 green points

Data & MLE fit



Posterior class prob. (GNB=)



Posterior class prob. (LR)

Gaussian Naive Bayes and logistic regression (3)

- GNB is generative, LR is discriminative
- For binary classification
 - ▶ When GNB= assumptions hold, GNB, GNB= and logistic regression asymptotically give identical classifiers
 - ▶ Otherwise, LR asymptotically often better than GNB=
 - ▶ Reason: LR consistent with but not rigidly tied to GNB= assumptions (and parameter estimates)
- Generally though, GNB and LR typically learn different classifiers
- Rates of convergence differ
 - ▶ $O(\log D)$ examples for GNB to “converge”
 - ▶ $O(D)$ examples for LR to “converge”
- Gaussian Naive Bayes most suitable if
 - ▶ GNB assumptions reasonable
 - ▶ As fallback when few examples available

Generalization error vs. number of examples (real data)

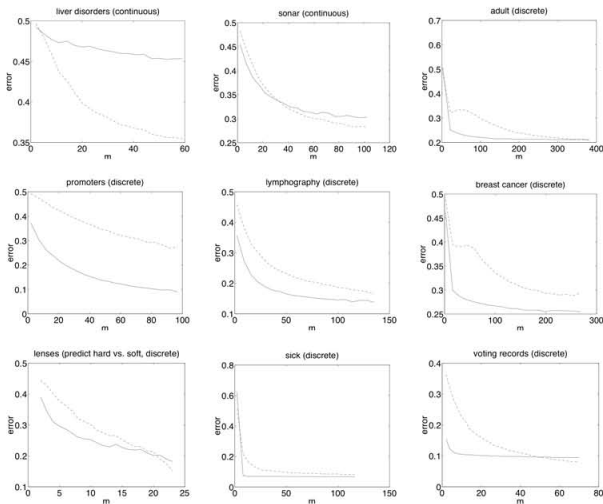


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learning repository. Plots are of generalization error vs. m (averaged over 1000 random train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

Machine Learning

05 – Point Estimation

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Outline (Point Estimation)

1. Maximum Likelihood Estimation & Empirical Risk Minimization
2. Gradient-Based Optimization
3. MAP Estimation & Regularized Risk Minimization

Lessons learned

- Point estimation often involves solving an optimization problem
 - ▶ Minimize a (non-linear) cost function
 - ▶ For differentiable cost functions, gradient-based learning dominant
- Gradient-based learning
 - ▶ First-order methods compute (GD) or estimate (SGD) gradient, move small step into opposite direction
 - ▶ Second-order methods also use (approximate) Hessian and can be superior when applicable
 - ▶ GD/SGD/variants (currently) often best choice for large data / many parameters / continuous cost functions
- MLE related to empirical risk minimization
 - ▶ Same cost functions for matching likelihood / loss
 - ▶ E.g., for discriminative classifiers: log loss, cross entropy loss, KL divergence loss
- MAP related to regularized risk minimization
 - ▶ Same cost function when priors / penalty additionally match
 - ▶ E.g., spherical Gaussian prior and ℓ_2 regularization

Suggested readings

- Murphy, Ch. 8, *Optimization*
- Also: Murphy, Ch. 10, *Logistic Regression*

Machine Learning

05 – Point Estimation

Part 1: Maximum Likelihood Estimation & Empirical Risk Minimization

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Recap: Maximum likelihood estimation (MLE)

- **Maximum likelihood estimation (MLE)** chooses the value $\hat{\theta}$ that maximizes the likelihood of the data

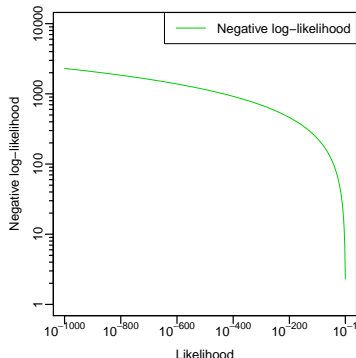
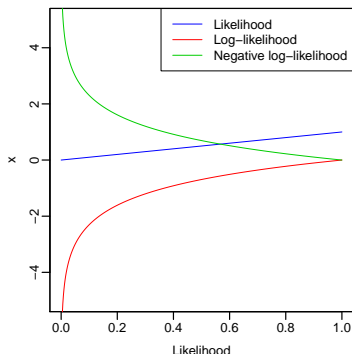
$$\hat{\theta}_{\text{MLE}} = \underset{\theta}{\operatorname{argmax}} \mathcal{L}(\theta|\mathcal{D}) = \begin{cases} p(\mathcal{D}|\theta) & \text{for generative models} \\ p(\mathbf{y}|\mathbf{X}, \theta) & \text{for discriminative models} \end{cases}$$

- ▶ Probabilistic models only
- Good properties when $N \rightarrow \infty$ (iid) samples; under mild conditions
 - ▶ Consistent: converges to true value θ^* (in probability)
 - ▶ Efficient: no other estimator has lower asymptotic mean squared error
 - ▶ Asymptotically normally distributed
- In practice, we do not have $N = \infty$
 - ▶ Tendency to overfit training data

Log-likelihood (1)

Instead of maximizing the likelihood, we can

- Maximize the **log-likelihood** $\ell(\boldsymbol{\theta}|\mathcal{D}) \stackrel{\text{def}}{=} \log \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$ or
- Minimize the **negative log-likelihood** $-\ell(\boldsymbol{\theta}|\mathcal{D})$



Log-likelihood (2)

- If training examples are iid, then (for discriminative models)

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \prod_{i=1}^N p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$

$$\ell(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \sum_{i=1}^N \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$

- ▶ **“Summation form” of log-likelihood facilitates gradient-based parameter estimation** (more later)

- For binary classification ($y_i \in \{0, 1\}$), we sometimes write

$$\ell(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \sum_{i=1}^N (y_i \log p_{i1} + (1 - y_i) \log p_{i0}),$$

where $p_{ic} = p(y_i = c | \mathbf{x}_i, \boldsymbol{\theta})$ refers to the predicted class probabilities

Recap: Empirical risk minimization (ERM)

- **Empirical risk minimization** chooses the estimator that minimizes $R_{\text{emp}}(h)$

$$\hat{h} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} R_{\text{emp}}(h) = \frac{1}{N} \sum_{i=1}^N L(h(\mathbf{x}_i), y_i),$$

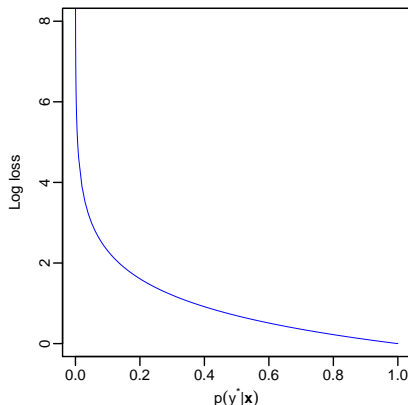
- ▶ h is hypothesis (corresponds to a choice of θ)
 - ▶ $L(\hat{y}, y)$ is **loss function** that measures how different prediction \hat{y} is from true answer y
 - ▶ Focus on prediction error, not parameters
 - ▶ Also applicable to non-probabilistic models
- Is there any relationship between MLE and ERM?
 - ▶ We will show: for discriminative models, MLE and ERM with certain choices of loss functions produce equivalent classifiers
→ **ERM is more general**
 - ▶ These loss functions include: log loss, cross entropy loss, KL divergence loss

Log loss

For probabilistic classifiers, the **log loss** is given by

$$L_{\log}(p(y|\mathbf{x}), y^*) = -\log p(y^*|\mathbf{x}),$$

where y^* is the true label and $p(y|\mathbf{x})$ is the probability distribution that the classifier outputs



Empirical risk minimization with log loss

- Empirical risk with log loss

$$R_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N -\log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}) \propto -\ell(\boldsymbol{\theta} | \mathbf{X}, \mathbf{y})$$

- For discriminative classifiers, MLE and empirical risk minimization with log loss
 - ▶ Use the same objective function (up to constants)
 - ▶ An MLE estimate also minimizes the empirical risk
 - ▶ A minimizer of empirical risk is also an MLE estimate
- No closed form solution
 - ▶ Numerical optimization often used
- Next: cross entropy loss and KL divergence loss
 - ▶ Close relationship to log loss
 - ▶ More general than log loss
 - ▶ Founded in information theory, with which we start

Background: Variable-Length Codes

- Consider a categorical distr. $p(x) = \text{Cat}(x|\theta)$ over $\{1, \dots, K\}$
- Anna samples a value X and wants to tell Bob the outcome
 - Anna and Bob can agree upfront on a codeword (bitstring) $c(x)$ for each category x
 - On average, how many bits does Anna have to send to Bob so that he can determine the value of X ?
- Answer: depends on code! Expected codeword length is

$$E_p[|c(x)|] = \sum_x p(x) |c(x)|$$

x	$p(x)$	$c(x)$	$ c(x) $
1	50%	00	2
2	25%	01	2
3	12.5%	10	2
4	12.5%	11	2

Expected length: 2

x	$p(x)$	$c(x)$	$ c(x) $
1	50%	0	1
2	25%	10	2
3	12.5%	110	3
4	12.5%	111	3

Expected length: 1.75

Entropy

- How small can the expected codeword length be?
 - ▶ Information theory (e.g., see [here](#)) tells us that *optimal code* c^* satisfies:

$$H(p) \leq E[|c^*(x)|] < H(p) + 1$$

- $H(p)$ is the **Shannon entropy** of p (above result: with base 2)

$$H(p) = \sum_x p(x) \underbrace{\log \frac{1}{p(x)}}_{I(x)}$$

- ▶ $p(x)$ = probability that Anna selects category x
 - ▶ **Information content** $I(x)$ = optimal length of codeword for x
 - ▶ Note: $I(x) = \log \frac{1}{p(x)} = -\log p(x)$
- Entropy is a **measure of uncertainty**
 - ▶ Ranges from 0 (no uncertainty, achieved by constant)
 - ▶ To $\log K$ (maximum uncertainty, achieved by uniform distribution)
 - ▶ Unit is bits with \log_2 / **nats** with \ln
(1 nat = $1/\ln 2$ bits ≈ 1.44 bits)

Entropy (example)

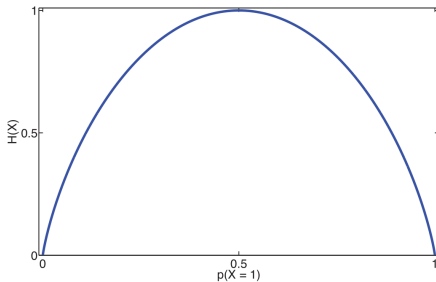


Figure 2.21 Entropy of a Bernoulli random variable as a function of θ . The maximum entropy is $\log_2 2 = 1$. Figure generated by `bernoulliEntropyFig`.

Cross entropy

- Let's modify the game a bit
 - ▶ Anna cheats and gives Bob the wrong distribution q
 - ▶ They use an optimal code for q , i.e., codeword for x has $\log \frac{1}{q(x)}$ bits
 - ▶ But Anna later selects values according to p , not q
 - ▶ How many bits are sent on average?
- Answer: **cross entropy** of q relative to p

$$H(p, q) = \sum_x p(x) \log \frac{1}{q(x)}$$

- ▶ $p(x)$ = probability that Anna selects category x according to p
- ▶ $\log \frac{1}{q(x)}$ = optimal size of codeword for x according to q
- ▶ $H(p, q) \geq H(p)$

Kullback-Leibler divergence

- On average, how many *additional* bits are now sent?
- Answer: **Kullback-Leibler divergence** of q w.r.t. p

$$D_{\text{KL}}(p||q) \stackrel{\text{def}}{=} H(p, q) - H(p) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

- KL divergence is a **measure of difference** between distributions
 - ▶ Interpretation: expected number of extra bits for encoding a value drawn from “true” distribution p using an optimum code for “estimated” distribution q
 - ▶ Aka **relative entropy** of q w.r.t. p
- Properties
 - ▶ $D_{\text{KL}}(p||q) \geq 0$
 - ▶ $D_{\text{KL}}(p||q) = 0$ iff $q = p$ (almost everywhere)
 - ▶ In general, $D_{\text{KL}}(p||q) \neq D_{\text{KL}}(q||p)$
- Concepts generalize to other (i.e., non-categorical) distributions:

$$H(p) = E_p[-\log p(x)] \quad H(p, q) = E_p[-\log q(x)]$$

Cross entropy loss

- Fix an example (\mathbf{x}, y^*)
- The **empirical distribution** $\tilde{p}(y)$ of y (for this example) is

$$\tilde{p}(y) = \mathbb{I}[y = y^*],$$

i.e., it puts all its mass on the correct label y^*

- Now consider the **model distribution** $q(y)$ of some classifier q (applied to this example; i.e., $q(y) = p(y|\mathbf{x})$)
- The **cross entropy loss** w.r.t. the empirical distribution is

$$\begin{aligned} L_{\text{CE}}(q(y), y^*) &\stackrel{\text{def}}{=} H(\tilde{p}, q) = \sum_y \tilde{p}(y)(-\log q(y)) \\ &= \sum_y \mathbb{I}[y = y^*](-\log q(y)) \\ &= -\log q(y^*) = L_{\log}(q(y), y^*) \end{aligned}$$

- ▶ For this reason, log loss is sometimes called **cross entropy loss**
- ▶ But cross entropy loss is more general (we may pick other distributions than \tilde{p} , e.g., to model noisy labels)

KL divergence loss

- We can also use KL divergence since:

$$\operatorname{argmin}_q H(\tilde{p}, q) = \operatorname{argmin}_q [H(\tilde{p}, q) \underbrace{- H(\tilde{p})}_{\text{indep. of } q}] = \operatorname{argmin}_q D_{\text{KL}}(\tilde{p} \| q)$$

- ▶ Generally, **KL divergence loss** differs from cross entropy loss by a model-independent constant
- ▶ For empirical distribution (as above), $H(\tilde{p}) = 0$
- Interpretation: we aim to minimize the (KL) divergence of the model distribution w.r.t. empirical distribution
- In summary, for classification, the following approaches are equivalent in that they share the same solutions
 - ▶ Maximum likelihood estimation
 - ▶ Empirical risk minimization with log loss
 - ▶ Empirical risk minimization with cross entropy loss
 - ▶ Empirical risk minimization with KL divergence loss

Direction of KL divergence

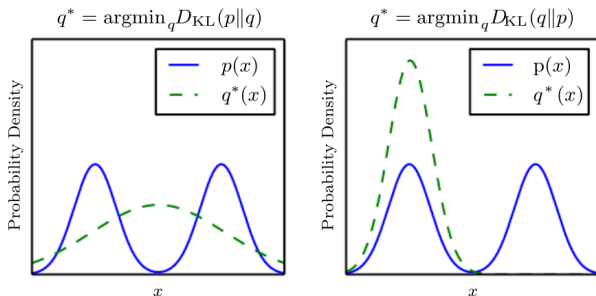


Figure 3.6: The KL divergence is asymmetric. Suppose we have a distribution $p(x)$ and wish to approximate it with another distribution $q(x)$. We have the choice of minimizing either $D_{\text{KL}}(p||q)$ or $D_{\text{KL}}(q||p)$. We illustrate the effect of this choice using a mixture of two Gaussians for p , and a single Gaussian for q . The choice of which direction of the KL divergence to use is problem-dependent.

- Think: p = data (blue); q^* = fitted model (green)
- Note: we used $D_{\text{KL}}(\tilde{p}||q)$ (left)

Machine Learning

05 – Point Estimation

Part 2: Gradient-Based Optimization

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Parameter estimation and optimization

- Often, parameter estimation requires the solution of a nonlinear optimization problem of form:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} f(\boldsymbol{\theta}).$$

- ▶ $\boldsymbol{\theta}$ corresponds to the model parameters
 - ▶ f called **cost function**
 - ▶ For MLE, f is the neg. log likelihood of the training data: $-\ell(\boldsymbol{\theta}|\mathcal{D})$
 - ▶ For ERM, f is the empirical risk: $R_{\text{emp}}(\boldsymbol{\theta})$
 - ▶ For MAP, f is the neg. log posterior: $-\log p(\boldsymbol{\theta}|\mathcal{D})$
 - ▶ For RRM, f is the regularized risk: $R'_{\text{emp}}(\boldsymbol{\theta})$
- Coming up: common optimization methods used in ML
 - Note: In ML, generalization error matters. The optimal solution (e.g., the ML estimate) may overfit.

Gradient-based methods

- We often (are forced to) use iterative methods
 - ▶ Pick a starting point
 - ▶ Repeatedly update the current point until some stopping criterion is met
 - ▶ Output the current point
- **First-order methods** use first-order partial derivatives
 - ▶ E.g.: gradient descent, stochastic gradient descent
 - ▶ Typically used for large datasets and/or models with many parameters
- **Second-order methods** use second-order information
 - ▶ Newton method or quasi-Newton methods (BFGS, L-BFGS)
 - ▶ Take curvature into account
 - ▶ Typically used for smaller datasets and/or models with few parameters

Outline

1. Gradient Descent
2. Stochastic Gradient Descent
3. Second-Order Methods

Continuous gradient descent

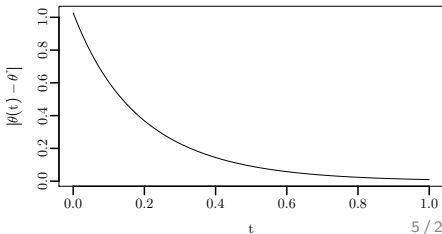
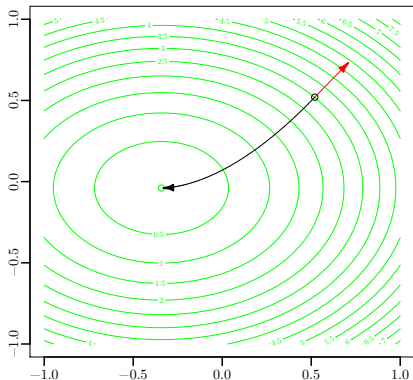
- Find minimum \mathbf{x}^* of function f
- Pick a starting point \mathbf{x}_0
- Compute gradient $\nabla f(\mathbf{x}_0)$
- Walk downhill
- Differential equation

$$\frac{\partial \mathbf{x}(t)}{\partial t} = -\nabla f(\mathbf{x}(t))$$

with boundary cond. $\mathbf{x}(0) = \mathbf{x}_0$

- Under certain conditions

$$\mathbf{x}(t) \rightarrow \mathbf{x}^*$$



Gradient descent

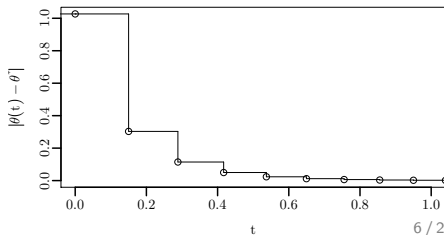
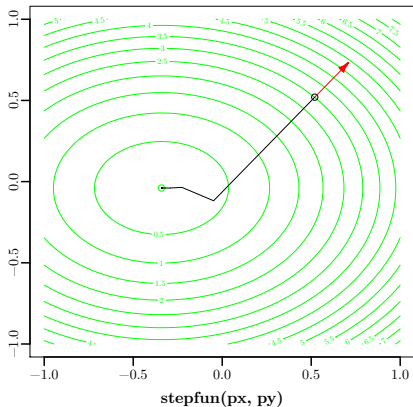
- Find minimum \mathbf{x}^* of function f
- Pick a starting point \mathbf{x}_0
- Compute gradient $\nabla f(\mathbf{x}_0)$
- Jump downhill
- Difference equation

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon_n \nabla f(\mathbf{x}_n)$$

- Under certain conditions, approximates CGD in that

$$\mathbf{x}^n(t) = \mathbf{x}_n + \text{"steps of size } t\text{"}$$

satisfies the PDE as $n \rightarrow \infty$



Derivatives for logistic regression

$$\frac{\partial}{\partial w_j} -\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \sum_{i=1}^N -x_{ij} [y_i - p_{i1}]$$

- Recall that $p_{i1} = \sigma(\mathbf{w}^\top \mathbf{x}_i)$ increases when we increase $\mathbf{w}^\top \mathbf{x}_i$
- Consider the contribution of any example i with $y_i = 1$ to the partial derivative w.r.t. w_j
 - ▶ Zero feature values $x_{ij} = 0$ do not contribute
 - ▶ Otherwise, contribution has opposite sign as feature x_{ij}
 - Contribution to gradient descent step towards increase of p_{i1}
 - ▶ More when difference between true label ($y_i = 1$) and prediction (p_{i1}) larger
 - ▶ More when feature x_{ij} larger (i.e., its absolute value)
- Similar arguments for $y_i = 0$ (consider the alternative formulation $\sum_{i=1}^N x_{ij} [(1 - y_i) - p_{i0}]$)

Gradient

- For functions with multiple inputs, there are multiple **partial derivatives**; e.g.,

$$f = x_1^2 + x_2^2$$

$$\frac{\partial}{\partial x_1} f = 2x_1$$

$$\frac{\partial}{\partial x_2} f = 2x_2$$

- We can gather them all in a single row vector (*numerator layout*), the **gradient** of f

$$\nabla_{\mathbf{x}^\top} f \stackrel{\text{def}}{=} \left(\frac{\partial}{\partial x_1} f \quad \frac{\partial}{\partial x_2} f \quad \cdots \quad \frac{\partial}{\partial x_n} f \right)$$

- For the example above, we obtain

$$\nabla_{\mathbf{x}^\top} f = 2\mathbf{x}^\top$$

Excursion: Matrix calculus

- Gradients can be computed element-by-element or directly using **matrix calculus**
 - ▶ Gives vectorized gradients (good for efficient gradient computation in software)
- For $\mathbf{x} \in \mathbb{R}^n$, the following rules hold

$$\begin{aligned}\nabla_{\mathbf{x}^\top} c &= \mathbf{0}_n^\top \\ \nabla_{\mathbf{x}^\top} \mathbf{c}^\top \mathbf{x} &= \mathbf{c}^\top \\ \nabla_{\mathbf{x}^\top} \mathbf{x}^\top \mathbf{c} &= \mathbf{c}^\top \\ \nabla_{\mathbf{x}^\top} \mathbf{x}^\top \mathbf{x} &= 2\mathbf{x}^\top \\ \nabla_{\mathbf{x}^\top} \mathbf{x}^\top \mathbf{A} \mathbf{x} &= \mathbf{x}^\top (\mathbf{A} + \mathbf{A}^\top),\end{aligned}$$

where constants $c \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^n$, and $\mathbf{A} \in \mathbb{R}^{n \times n}$ do not depend on \mathbf{x}

- Also: multiplicative rule, product rule, chain rule, ...

Gradient of logistic regression

$$\frac{\partial}{\partial w_j} \ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = - \sum_{i=1}^N x_{ij} [y_i - p_{i1}]$$

- Define the **error on example i** as $e_i = y_i - p_{i1}$ (a function of \mathbf{w})
- Define the **error vector** $\mathbf{e} = (e_1 \ e_2 \ \cdots \ e_N)^\top$
- Then

$$\frac{\partial}{\partial w_j} \ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = - \sum_{i=1}^N e_i x_{ij} = -\mathbf{e}^\top \mathbf{x}_j$$

- And therefore

$$\nabla_{\mathbf{w}^\top} \ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = -\mathbf{e}^\top \mathbf{X}$$

Gradient descent summary

- We aim to minimize an objective function such as the negative log-likelihood
- A single gradient computation and subsequent parameter update is called an **epoch**
- In the n -th epoch, we use **learning rule**

$$\theta_{n+1} \leftarrow \theta_n - \epsilon_n \nabla f(\theta)$$

for some **learning rate** $\epsilon_n > 0$

- For NLL and logistic regression
 - ▶ Update rule is $\mathbf{w}_{n+1}^\top \leftarrow \mathbf{w}_n^\top + \epsilon_n \mathbf{e}_n^\top \mathbf{X}$
 - ▶ Negative log-likelihood of logistic regression is convex; i.e., each local optimum is a global optimum
 - ▶ Under mild conditions on the step size sequence, GD converges to ML estimate $\hat{\mathbf{w}}_{\text{MLE}}$

Outline

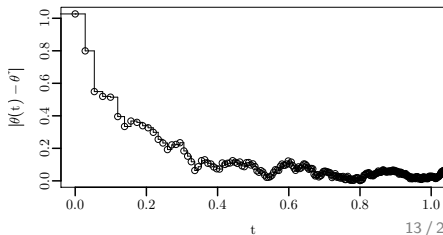
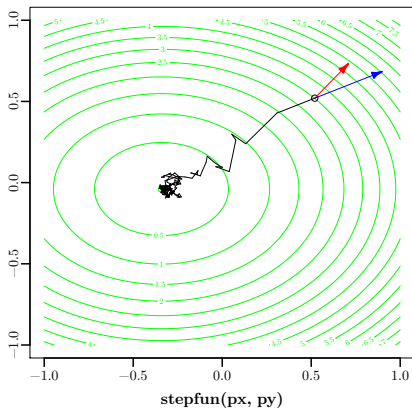
1. Gradient Descent
2. Stochastic Gradient Descent
3. Second-Order Methods

Stochastic gradient descent

- Find minimum \mathbf{x}^* of function f
- Pick a starting point \mathbf{x}_0
- Approximate gradient $\hat{\nabla} f(\mathbf{x}_0)$
- Jump “approximately” downhill
- Stochastic difference equation

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon_n \hat{\nabla} f(\mathbf{x}_n)$$

- Under certain conditions, asymptotically approximates (continuous) gradient descent



SGD for logistic regression

- We use

$$\nabla_{\mathbf{w}^\top} -\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = -\mathbf{e}^\top \mathbf{X} = -\sum_{i=1}^N e_i \mathbf{x}_i^\top$$

$$\hat{\nabla}_{\mathbf{w}^\top} -\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) = -N e_Z \mathbf{x}_Z^\top,$$

where $Z \in \{1, 2, \dots, N\}$ is a single training example chosen uniformly and at random from the N examples in the training set.

- SGD epoch (with or without replacement)
 1. Pick a random example z (with or without replacement)
 2. Compute approximate gradient $\hat{\nabla}_{\mathbf{w}^\top} -\ell(\mathbf{w}|\mathbf{X}, \mathbf{y})$
 3. Update parameters
$$\mathbf{w}_{n+1} = \mathbf{w}_n + \epsilon_n \hat{\nabla} \ell(\mathbf{w}|\mathbf{X}, \mathbf{y})$$
 4. Repeat N times
- Observe: only weights for non-zero features updated in each step

Comparison

- Per epoch, assuming $O(D)$ gradient computation per example

	GD	SGD
Algorithm	Deterministic	Randomized
Gradient computations	1	N
Gradient types	Exact	Approximate
Parameter updates	1	N
Time	$O(DN)$	$O(DN)$
Space	$O(D)$	$O(D)$

- Why stochastic? Empirically, for large datasets:
 - ▶ *Fast convergence* to vicinity of optimum
 - ▶ Randomization may help escape local minima
 - ▶ Exploitation of “repeated structure”

GD/SGD in practice (1)

Step size (or **learning rate**) sequence $\{\epsilon_n\}$ needs to be chosen carefully; widely studied, many options:

- Large \rightarrow good initially (move quickly), bad later on (juggle around optimum)
- Keep step size throughout or reduce it gradually
 - ▶ E.g., constant (useful for online learning)
 - ▶ E.g., $\epsilon_n = a/(b+n)$ for some constants a, b
 - ▶ E.g., pick $\epsilon_n \leq 1/L(\nabla f)$ if f has bounded gradient
- **Bold driver heuristic**: After every epoch
 - ▶ Increase step size slightly when objective decreased (by, say, 5%)
 - ▶ Decrease step size sharply when objective increased (by, say, 50%)
 - ▶ May (or should) use validation error, grace period, no increase, ...
- The above choices (most notably, the initial learning rate) are often treated as *hyperparameters* in machine learning
- See [pytorch's learning rate schedulers](#) for examples
- **Line search**: optimize the step size directly

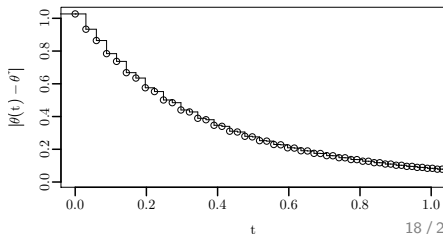
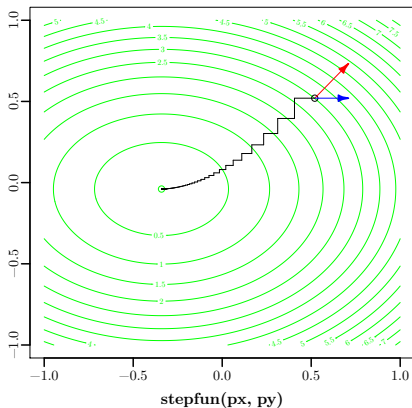
$$\epsilon_n = \underset{\epsilon}{\operatorname{argmin}} f(\theta_n - \epsilon \nabla f(\theta_n))$$

GD/SGD in practice (2)

- SGD is a common learning algorithm
 - ▶ E.g., training neural networks
 - ▶ Related to **incremental gradient descent** and **online learning**
- Many variants exist; e.g.,
 - ▶ Use more than one example per step (**mini-batch** GD)
 - ▶ [Polyak averaging](#)
 - ▶ [Momentum](#)
 - ▶ Adaptive, per-parameter step sizes ([AdaGrad](#), [RMSprop](#), [AdaDelta](#), [Adam](#))
 - ▶ More in Deep Learning lecture (spring term)
- And it can (often) be parallelized; e.g.,
 - ▶ Parallelizing mini-batch gradient computations
 - ▶ [Hogwild](#)
 - ▶ [Vowpal Wabbit](#)
 - ▶ [DSGD++](#) (for latent factor models)
 - ▶ ...

Excursion: Coordinate-Wise Gradient Descent

- Goal find $\min \mathbf{x}^*$ of function f
 - Pick starting point \mathbf{x}_0
 - Choose a coordinate $j \in \{1, \dots, D\}$
 - Compute gradient for coordinate $\frac{\partial}{\partial x_j} f$
 - Jump downhill along this coordinate
-
- Sometimes gradients simplify / can reuse computations
 - Variants
 - ▶ Coordinate-wise SGD
 - ▶ Block-coordinate (S)GD (choose more than one coordinate)



Outline

1. Gradient Descent
2. Stochastic Gradient Descent
3. Second-Order Methods

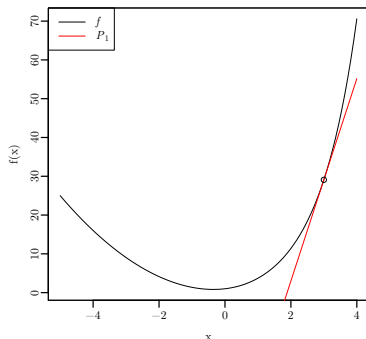
First-order Taylor polynomial

- Taylor's theorem tells us that if f is differentiable at some point a , then there exists function h_1 such that

$$f(x) = \underbrace{f(a) + f'(a)(x - a)}_{P_1(x)} + \underbrace{h_1(x)(x - a)}_{\text{error}}$$

and $\lim_{x \rightarrow a} h_1(x) = 0$

- $P_1(x)$ is the **first-order Taylor polynomial** of f at point a and the (asymptotically) best **linear approximation** of f at point a



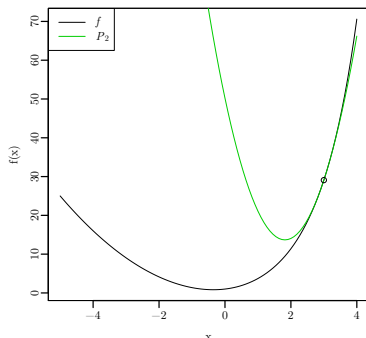
Second-order Taylor polynomial

- Taylor's theorem tells us that if f is twice differentiable at some point a , then there exists function h_2 such that

$$f(x) = \underbrace{f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2}_{P_2(x)} + \underbrace{h_2(x)(x-a)^2}_{\text{error}}$$

and $\lim_{x \rightarrow a} h_2(x) = 0$

- $P_2(x)$ is the **second-order Taylor polynomial** of f at point a and the (asymptotically) best **quadratic approximation** of f at point a



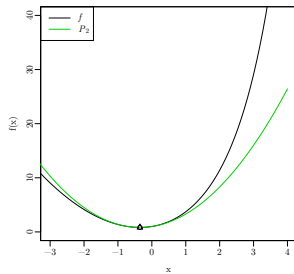
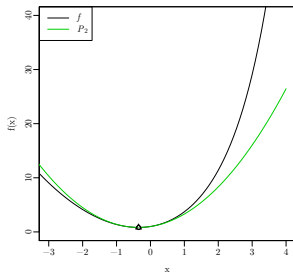
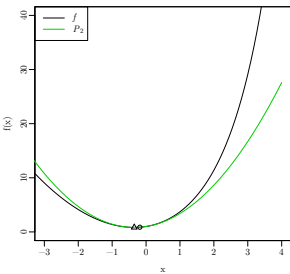
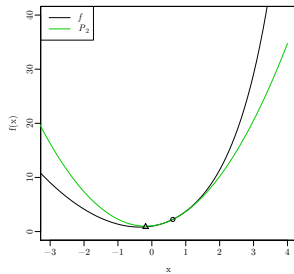
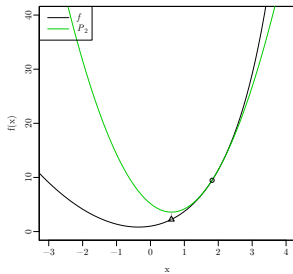
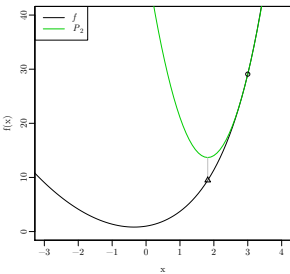
Newton's method

- If f is strictly convex (i.e., $f''(x) > 0$), then we can minimize f using Newton's method

$$x_{n+1} \leftarrow x_n - \frac{f'(x_n)}{f''(x_n)}$$

- ▶ rhs is stationary point of the 2nd-order Taylor polynomial of f at x_n
- ▶ If f is strictly convex, this stationary point is the minimum of P_2
 - Quadratic approximation of f is minimized in each step
- Discussion
 - ▶ Can converge in significantly less steps than gradient descent; e.g., if f is quadratic, only needs one step
 - ▶ Need to compute second derivative
 - ▶ If f is not strictly convex, may converge to any stationary point (including a local maximum!) or even diverge

Example



Second-order Taylor polynomial (multivariate)

- If $f : \mathbb{R}^D \rightarrow \mathbb{R}$ and twice differentiable at $\mathbf{x}_n \in \mathbb{R}^D$, then the second-order Taylor polynomial at \mathbf{x}_n is

$$P_2(\mathbf{x}) = f_n + \mathbf{g}_n^\top (\mathbf{x} - \mathbf{x}_n) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_n)^\top \mathbf{H}_n (\mathbf{x} - \mathbf{x}_n),$$

where

- ▶ $f_n = f(\mathbf{x}_n)$ denotes the function value,
 - ▶ $\mathbf{g}_n^\top = (\nabla_{\mathbf{x}^\top} f)(\mathbf{x}_n)$ denotes the gradient, and
 - ▶ $\mathbf{H}_n = (\nabla_{\mathbf{x}^\top} \nabla_{\mathbf{x}} f)(\mathbf{x}_n)$ denotes the Hessian matrix at \mathbf{x}_n .
- **Hessian** is $D \times D$ matrix of second-order partial derivatives

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2}{\partial x_1^2} f & \frac{\partial^2}{\partial x_1 \partial x_2} f & \cdots & \frac{\partial^2}{\partial x_1 \partial x_D} f \\ \frac{\partial^2}{\partial x_2 \partial x_1} f & \frac{\partial^2}{\partial x_2^2} f & \cdots & \frac{\partial^2}{\partial x_2 \partial x_D} f \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_D \partial x_1} f & \frac{\partial^2}{\partial x_D \partial x_2} f & \cdots & \frac{\partial^2}{\partial x_D^2} f \end{pmatrix}$$

Newton's method (multivariate)

- Newton's method then becomes

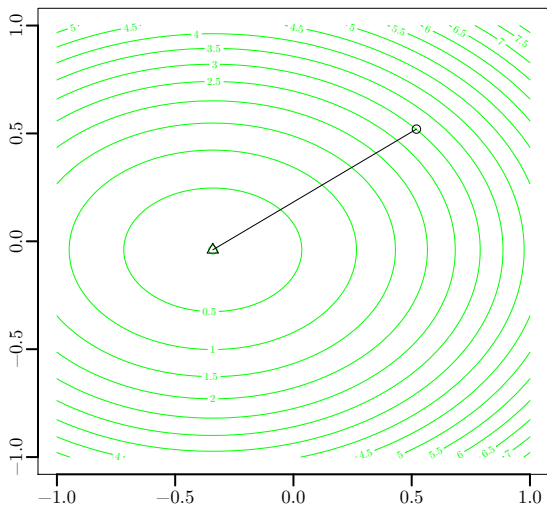
$$\mathbf{x}_{n+1} \leftarrow \mathbf{x}_n - \mathbf{H}_n^{-1} \mathbf{g}_n$$

- Discussion

- ▶ Newton's method is basic second-order method
- ▶ Can converge in significantly less steps than GD (quadratic convergence vs. linear convergence)
- ▶ Expensive for large D ; Hessian takes $O(D^2)$ space and its inversion $O(D^3)$ time
- ▶ Harder to make stochastic
- ▶ If f is strictly convex (i.e., \mathbf{H} positive definite; exercise), also converges to unique minimum
- ▶ Otherwise, $\mathbf{d}_n = -\mathbf{H}_n^{-1} \mathbf{g}_n$ may not be a descent direction
- ▶ More sophisticated methods can avoid this problem (e.g., use gradient descent step if \mathbf{d}_n is not a descent direction)

Example

Example function is quadratic \rightarrow one step



Newton's method for logistic regression

- Newton's method for MLE estimate of logistic regression
 - ▶ Hessian has simple form (exercise)
 - ▶ Newton's method known as **iteratively reweighted least squares** (IRLS)
 - ▶ IRLS very good optimization algorithm if D is not too large
- For large D , Newton's method becomes too costly
 - ▶ E.g., simple bag-of-words models in natural language processing tasks may have 10s of thousands of features (and thus parameters in logistic regression)
 - ▶ More complex models may have millions of parameters
- Alternatives include
 - ▶ Quasi-Newton methods such as BFGS or L-BFGS, which use gradient information only to build approximation of the (inverse) Hessian
 - ▶ Stochastic gradient descent
- More in MAC 507 Nonlinear Optimization and in [Optimization in Machine Learning](#)

Machine Learning

05 – Point Estimation

Part 3: MAP Estimation & Regularized Risk Minimization

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Potential problems with MLE

We use logistic regression as an example throughout, but the discussion applies generally. Potential problems with MLE:

1. Risk to overfit
 - ▶ E.g., when too many features (and thus weights)
 2. Weights may be non-unique or unstable
 - ▶ E.g., when training data linearly separable
 - ▶ E.g., when features (in training data) are linearly dependent
 3. Does not optimize for classification error
 - ▶ Fit with lower likelihood may actually produce less errors (even on training data)
 - ▶ Likelihood \neq error
- We now look closer at these problems
 - Using a prior is one option to mitigate 1 and 2
 - 3 corresponds to (empirical) risk minimization with 0-1 loss (NP hard, more in DL course)

Example: Weights not unique

- Consider the following training data

x_1	x_2	y
1	1	0
1	1	1

- Observe: Features are linearly dependent
- All these weight vectors are MLE estimates
 - ▶ $\mathbf{w}_1 = (1 \ -1)^\top$
 - ▶ $\mathbf{w}_2 = (0 \ 0)^\top$
 - ▶ $\mathbf{w}_3 = (-1 \ 1)^\top$
- But they give different estimates for new data
 - ▶ $\mathbf{x}_{\text{new}} = (1 \ 0)^\top$
 - ▶ $\sigma(\mathbf{w}_1^\top \mathbf{x}_{\text{new}}) \approx 0.73$
 - ▶ $\sigma(\mathbf{w}_2^\top \mathbf{x}_{\text{new}}) = 0.5$
 - ▶ $\sigma(\mathbf{w}_3^\top \mathbf{x}_{\text{new}}) \approx 0.27$
- Which weight vector is the “right” one?

Example: Weights unstable

- Recall: training data is **linearly separable** if there exists a weight vector \mathbf{w} such that for all i

$$\eta_i = \mathbf{w}^\top \mathbf{x}_i \begin{cases} > 0 & \text{if } y_i = 1 \\ < 0 & \text{if } y_i = 0 \end{cases}$$

- Implies correct decisions for all training examples
- If \mathbf{w} separates the training data, so does $\mathbf{w}' = c\mathbf{w}$ for some constant $c > 1$; i.e., for all $\mathbf{x} \in \mathcal{D}$
 - ▶ We have $\eta = \mathbf{w}^\top \mathbf{x}$ and $\eta' = (\mathbf{w}')^\top \mathbf{x} = c\mathbf{w}^\top \mathbf{x} = c\eta$
 - ▶ η and η' have same sign \rightarrow same decision
 - ▶ $|\eta'| > |\eta| \rightarrow$ higher confidence in decision
 - ▶ Putting both together, we conclude that \mathbf{w}' corresponds to higher training-data likelihood
- We can repeat this process ad infinitum
 - \rightarrow weights increase without bounds

Adding a prior

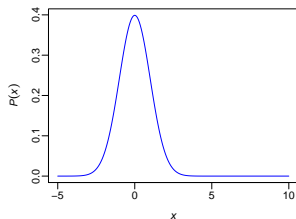
- To mitigate these problems, we can make use of a prior
 - ▶ Recall: $\text{posterior} \propto \text{likelihood} \times \text{prior}$
 - ▶ Recall: prior = apriori belief over distribution of parameters
 - ▶ Prior also allows to incorporate expert knowledge
- Which prior?
 - ▶ Beta-binomial model not applicable (why?)
 - ▶ Prior should not contradict available expert knowledge
 - ▶ Prior should not unduly “overrule” data (=too strong)
- We discuss: spherical Gaussian prior
 - ▶ Common, simple choice
 - ▶ See [Stan's recommendations](#) for discussion

Refresher: Gaussian distribution

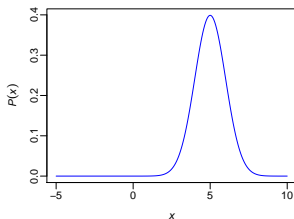
- Mean $\mu \in \mathbb{R}$, variance $\sigma^2 \in \mathbb{R}^+$ (or **precision** $\lambda = 1/\sigma^2$)
- Denoted $\mathcal{N}(\mu, \sigma^2)$
- $\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(x - \mu)^2\right]$

$$\sigma^2 = 1$$
$$\lambda = 1$$

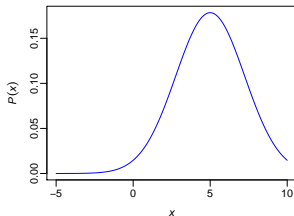
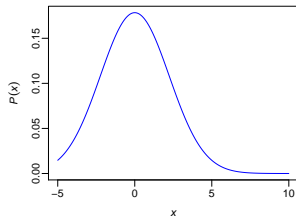
$$\mu = 0$$



$$\mu = 5$$



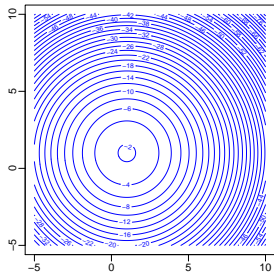
$$\sigma^2 = 5$$
$$\lambda = 0.2$$



Multivariate Gaussian distribution

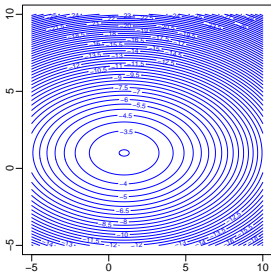
- Mean $\mu \in \mathbb{R}^D$, covariance $\Sigma \in \mathbb{R}^{D \times D}$ (or precision $\Lambda = \Sigma^{-1}$)
- Denoted $\mathcal{N}(\mu, \Sigma)$
- Let $|\Sigma|$ be the determinant of Σ . If Σ is positive definite:

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp \left[-\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu) \right]$$



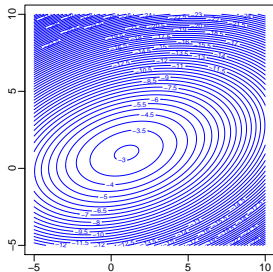
$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \sigma^2 I$$

(spherical)



$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

(diagonal)



$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 5 & 1 \\ 1 & 2 \end{pmatrix}$$

Spherical Gaussian prior

- $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ for some variance hyperparameter $\sigma^2 > 0$
- Equivalently, $w_j \sim \mathcal{N}(0, \sigma^2)$ and weights independent
 - ▶ Prior gives highest density to zero weights
 - ▶ Prior density of nonzero weights decreases with distance to zero
 - ▶ σ^2 controls how fast
 - ▶ Intuitively: posterior keeps weights close to zero unless data suggests otherwise

- Posterior

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}, \sigma^2) \propto \mathcal{L}(\mathbf{w}|\mathbf{X}, \mathbf{y}) \mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma^2 \mathbf{I})$$

- ▶ Gaussian is not a conjugate prior, posterior not Gaussian
 - ▶ No closed-form solution
 - ▶ Bayesian inference often via approximate methods
- This is for logistic regression
 - ▶ Generally, replace \mathbf{w} by $\boldsymbol{\theta}$

Maximum a posteriori estimation

- Recall: The **maximum a posteriori (MAP) estimate** $\hat{\mathbf{w}}_{\text{MAP}}$ is the point estimate that maximizes the posterior

$$\hat{\mathbf{w}}_{\text{MAP}} = \underset{\mathbf{w}}{\operatorname{argmax}} \mathcal{L}(\mathbf{w}|\mathbf{X}, \mathbf{y}) \mathcal{N}(\mathbf{w}|\mathbf{0}, \sigma^2 \mathbf{I})$$

► Think: most likely weight vector given data *and prior*

- Taking logs, we obtain

$$\begin{aligned}\hat{\mathbf{w}}_{\text{MAP}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) + \log \prod_j \mathcal{N}(w_j|0, \sigma^2) \right] \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) - \sum_j \frac{1}{2\sigma^2} w_j^2 \right] \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\ell(\mathbf{w}|\mathbf{X}, \mathbf{y}) - \frac{\lambda}{2} \|\mathbf{w}\|^2 \right],\end{aligned}$$

where $\lambda = 1/\sigma^2$

Discussion

- Recall our example

x_1	x_2	y
1	1	0
1	1	1

- When $\lambda > 0$, $\mathbf{w}_2 = (0 \ 0)^\top$ is *unique* MAP estimate
 - New-data predictions do not depend on arbitrary choices anymore
- When data is linearly separable and $\lambda > 0$, weights remain bounded (why?)
- With $\lambda = 0$, prior has no effect and we retain MLE
- Data scale matters
 - With MLE (and a bias term), shifting or changing the scale of parameters does not affect predicted probabilities (assignment 2)
 - E.g., when we scale a parameter by factor 10 and its weight by factor 1/10, we obtain the same predictions
 - With MAP, this does not hold: rescaled weight vector has higher prior density (and thus a lower “penalty” for MAP objective)
 - Scale needs to be taken into account (e.g., use z -scores)

Regularized risk minimization (1)

- Recall that for iid data, MLE estimates for classifiers can be viewed as *empirical* risk minimization with log loss in that

$$\hat{\mathbf{w}}_{\text{MLE}} = \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{\frac{1}{N} \sum_{i=1}^N -\log p(y_i | \mathbf{x}_i, \boldsymbol{\theta})}_{R_{\text{emp}}(\boldsymbol{\theta})}$$

- MAP estimation with spherical Gaussian priors can be viewed as *regularized* risk minimization with **ℓ_2 regularization**

$$\hat{\mathbf{w}}_{\text{MAP}} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=1}^N -\log p(y_i | \mathbf{x}_i, \boldsymbol{\theta}) + \frac{\lambda'}{2} \|\boldsymbol{\theta}\|^2 \right]$$

- ▶ Spherical Gaussian parameterized by variance hyperparameter σ^2
- ▶ L2 regularization parameterized by regularization coefficient hyperparameter $\lambda' = \lambda/N = 1/(N\sigma^2)$
- ▶ “Large” weight vectors penalized by their squared L2 norm

Regularized risk minimization (2)

- ℓ_2 regularization leads to GD learning rules with **weight decay**

$$\begin{aligned}\boldsymbol{\theta}_{n+1} &\leftarrow \boldsymbol{\theta}_n - \epsilon_n \nabla R_{\text{emp}}(\boldsymbol{\theta}_n) - \epsilon_n \lambda' \boldsymbol{\theta}_n \\ &= (1 - \epsilon_n \lambda') \boldsymbol{\theta}_n - \epsilon_n \nabla R_{\text{emp}}(\boldsymbol{\theta}_n)\end{aligned}$$

- Likewise for SGD; with loss L and example z , we obtain

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \epsilon_n \nabla L(p(Y|\mathbf{x}_z, \boldsymbol{\theta}_n), y_z) - \epsilon_n \lambda' \boldsymbol{\theta}_n$$

- ▶ E.g., for penalized logistic reg.: $\mathbf{w}_{n+1} \leftarrow \mathbf{w}_n + \epsilon_n e_z \mathbf{x}_z - \epsilon_n \lambda' \mathbf{w}_n$

- Notes

- ▶ ℓ_2 regularization / weight decay generally applicable (model does not need to be probabilistic)
- ▶ For iid data, MAP estimation with diagonal Gaussian prior can be expressed as regularized risk minimization with **weighted ℓ_2 regularization** (different weight for each parameter)
- ▶ And with a general Gaussian prior as RRM with **Tikhonov regularization** (penalty is $\|\Lambda' \boldsymbol{\theta}\|^2$)

Machine Learning

06 – Dimensionality Reduction

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Overview

- So far we have looked at basic generative and discriminative models for supervised learning
- Coming up: dimensionality reduction (unsupervised learning)
- Focus: the **singular value decomposition** (SVD)
 - ▶ A basic unsupervised model
 - ▶ Useful for dimensionality reduction, data compression, and denoising data
 - ▶ Best “low rank” approximation to the data
- By studying the SVD, we also learn about key approaches in ML
 - ▶ SVD is a matrix factorization model
 - ▶ SVD is closely related to the principal component analysis (PCA)
 - ▶ SVD is a latent linear model
 - ▶ SVD is a linear autoencoder
 - ▶ SVD is important workhorse of linear algebra
 - ▶ ...

Outline (Dimensionality Reduction)

1. Matrix Decompositions
2. Singular Value Decomposition
3. Interpreting the SVD
4. Using the SVD
5. Latent Linear Models

Lessons learned

- SVD is the Swiss Army knife of (numerical) linear algebra
 - Ranks, kernels, norms, inverses,
- SVD is also very useful in data analysis
 - Dimensionality reduction, noise removal, visualization, ...
- Truncated SVD is best low-rank factorization of the data in terms of Frobenius norm
- Selecting the right size for truncated SVD can be difficult
- Interpretation of results can be challenging
- Close relationship to
 - ▶ PCA (center data)
 - ▶ Probabilistic PCA (generative model)
 - ▶ Latent linear models as a general framework

Suggested reading

- Murphy, Ch. 20: *Dimensionality Reduction*
- David Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*
Ch. 3: Singular Value Decomposition
Chapman and Hall, 2007
- Carl Meyer. *Matrix Analysis and Applied Linear Algebra*
Ch. 5.12: Singular Value Decomposition
Society for Industrial and Applied Mathematics, 2000
<http://www.matrixanalysis.com>
- Bishop, Ch. 12: *Continuous Latent Variables*
- Gene H. Golub & Charles F. Van Loan: *Matrix Computations*,
3rd ed. Johns Hopkins University Press, 1996
 - ▶ Excellent source for the algorithms and theory, but very dense

Machine Learning

06 – Dimensionality Reduction

Part 1: Matrix Decompositions

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Matrix decompositions

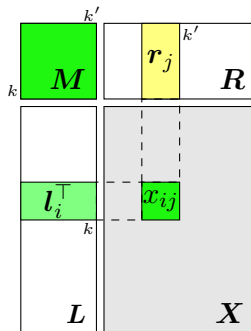
A **matrix decomposition** of a data matrix X is given by three matrices L , M , R such that

$$X = LMR,$$

where

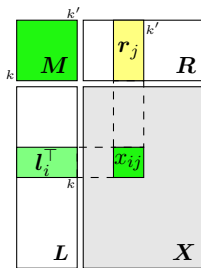
- X is an $m \times n$ data matrix,
- L is an $m \times r_1$ matrix,
- M is an $r_1 \times r_2$ matrix,
- R is an $r_2 \times n$ matrix, and
- r_1 and r_2 are integers ≥ 1 .
- Often $r_1 = r_2 = r \geq 1$

$$x_{ij} = \sum_{k,k'} l_{ik} m_{kk'} r_{k'j}$$



Matrix decompositions and constraints

- Decompositions as just defined are not really helpful
 - ▶ Suppose we set $r = r_1 = r_2 = n$, $L = X$, $M = R = I_n$ (the $n \times n$ identity matrix)
 - ▶ Then $X = LMR = XI_nI_n = X$
- To make decompositions useful, they need to satisfy certain (carefully chosen) properties or constraints
- For example: small r
 - ▶ Each example is represented by n numbers in X
 - ▶ Each example is represented by r numbers in L
 - ▶ If $r < n$, we performed some form of compression
- For example: constraints on factor matrices
 - ▶ Compare: integer factorization
 - ▶ $391 = 17 \cdot 13$



Approximate matrix decompositions

An **approximate matrix decomposition** of data matrix \mathbf{X} is given by three matrices \mathbf{L} , \mathbf{M} , \mathbf{R} such that




$$\mathbf{X} \approx \mathbf{LMR} = \hat{\mathbf{X}},$$

where each matrix has conforming dimensions (as before).

- Approximation is important because
 - ▶ Approximate decompositions can be much smaller than exact decompositions (small r)
 - ▶ **Reconstruction** $\hat{\mathbf{X}}$ can be viewed as a denoised version of \mathbf{X}
 - ▶ Can lead to more insightful/interpretable decompositions
 - ▶ More efficient to compute
 - ▶ Generally, trade-off between approximation error and “usefulness”
- \approx can be defined via a **reconstruction error** $E(\mathbf{X}, \hat{\mathbf{X}})$
 - ▶ E is an **error function**; e.g., root mean squared error (RMSE)
 - ▶ Low means good approximation, high means bad
 - ▶ Finding the best approximation (smallest error) can be hard
- Often: “matrix decomposition” said instead “approximate matrix decomposition”

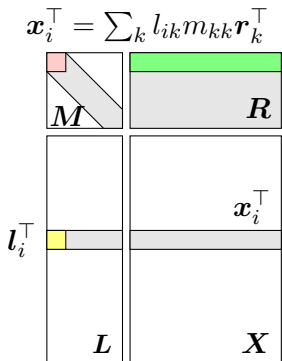
Factor interpretation of matrix decompositions

Assume that M is $r \times r$ and diagonal. Consider example i .

-  Row of R = part (or piece), called **latent factor** (“latent object”)
-  Entry of M = weight of corresponding part
- Row of MR = weighted part
-  Row of L = representation of object via weighted parts, called **embedding**, **code**, **scores**, **distributed representation**, ...
- **Size** r controls “compactness” (often $r < n$)

Each object can be viewed as a combination of r (weighted) “latent objects” (or “prototypical objects”). Similarly, each feature can be viewed as a combination of r (weighted) “latent features.”

(e.g., latent feature = “body size”; latent object relates body size to real features such as “height”, “weight”, “shoe size”)



Example: Weather data ($r = 1$)

1.00

9.05

16.55

26.73

18.75

17.81

0.62

0.69

0.83

0.90

0.88

0.98

1.09

1.14

1.16

1.24

1.21

1.27

	Jan	Apr	Jul	Oct	Year
Stockholm	-0.70	8.60	21.90	9.90	10.00
Minsk	-2.10	12.20	23.60	10.20	10.60
London	7.90	13.30	22.80	15.20	14.80
Budapest	1.20	16.30	26.50	16.10	15.00
Paris	6.90	14.70	24.40	15.80	15.50
Bucharest	1.50	18.00	28.80	18.00	16.50
Barcelona	12.40	17.60	27.50	21.50	20.00
Rome	11.90	17.70	30.30	21.40	20.40
Lisbon	14.80	19.80	27.90	22.50	21.50
Athens	12.90	20.30	32.60	23.10	22.30
Valencia	16.10	20.20	29.10	23.60	22.30
Malta	16.10	20.00	31.50	25.20	23.20

X

Example: Weather data ($r = 1$), reconstruction

1.00

9.05

16.55

26.73

18.75

17.81

		Jan	Apr	Jul	Oct	Year
0.62	Stockholm	5.65	10.33	16.68	11.70	11.11
0.69	Minsk	6.21	11.36	18.35	12.87	12.23
0.83	London	7.55	13.80	22.28	15.63	14.85
0.90	Budapest	8.11	14.83	23.94	16.80	15.96
0.88	Paris	7.96	14.56	23.52	16.50	15.67
0.98	Bucharest	8.91	16.30	26.32	18.47	17.54
1.09	Barcelona	9.88	18.06	29.17	20.46	19.44
1.14	Rome	10.28	18.80	30.35	21.30	20.23
1.16	Lisbon	10.47	19.15	30.92	21.70	20.61
1.24	Athens	11.21	20.50	33.11	23.23	22.07
1.21	Valencia	10.92	19.96	32.24	22.62	21.48
1.27	Malta	11.47	20.98	33.88	23.77	22.58

$$\overline{X}$$

(RMSE: 2.66)

Example: Weather data ($r = 2$), reconstruction

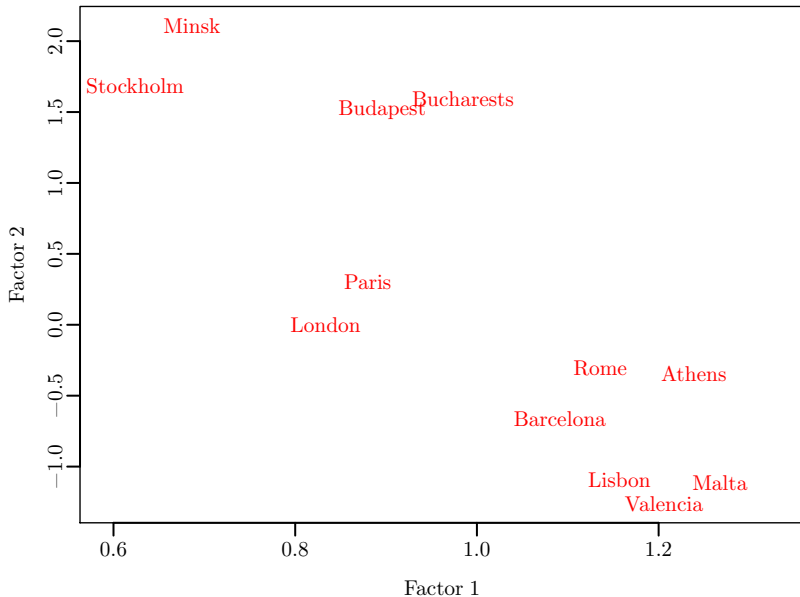
1.00		9.05	16.55	26.73	18.75	17.81
	1.00	-4.14	0.27	2.32	-0.89	-0.69

			Jan	Apr	Jul	Oct	Year
0.62	1.69	Stockholm	-1.34	10.79	20.59	10.20	9.95
0.69	2.11	Minsk	-2.52	11.94	23.23	10.99	10.77
0.83	0.00	London	7.54	13.80	22.28	15.63	14.85
0.90	1.52	Budapest	1.82	15.24	27.46	15.45	14.91
0.88	0.30	Paris	6.71	14.65	24.22	16.23	15.46
0.98	1.59	Bucharest	2.31	16.74	30.02	17.05	16.44
1.09	-0.66	Barcelona	12.61	17.88	27.64	21.05	19.90
1.14	-0.31	Rome	11.55	18.71	29.64	21.57	20.44
1.16	-1.09	Lisbon	15.00	18.85	28.39	22.67	21.36
1.24	-0.35	Athens	12.65	20.41	32.31	23.54	22.31
1.21	-1.26	Valencia	16.14	19.62	29.31	23.74	22.36
1.27	-1.12	Malta	16.10	20.67	31.29	24.76	23.35

 $\hat{\mathbf{X}}$

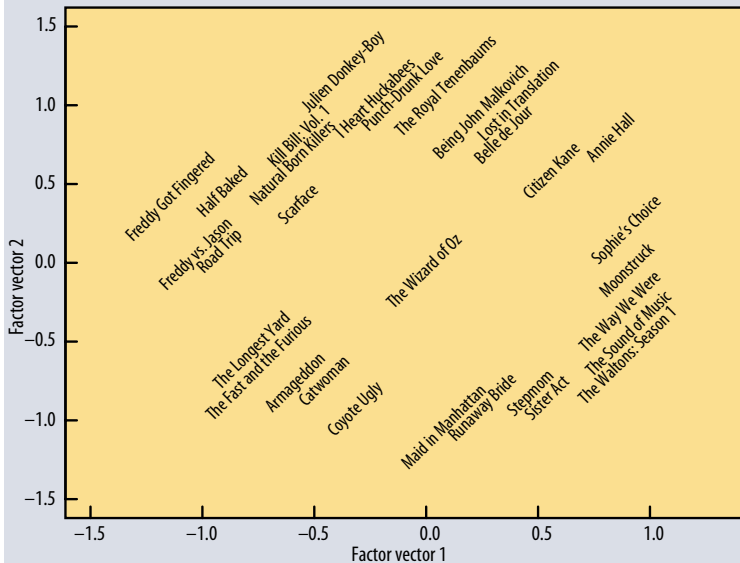
(RMSE: 0.60)

Example: Weather data ($r = 2$), plot



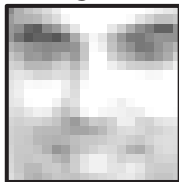
Example: Netflix prize data

($\approx 500k$ users, $\approx 17k$ movies, $\approx 100M$ ratings)

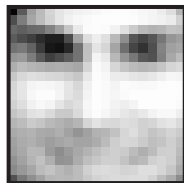


Example: k-Means / vector quantization

Original



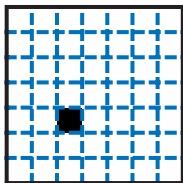
x_i^\top (original)



\hat{x}_i^\top

=

l_i^\top



VQ

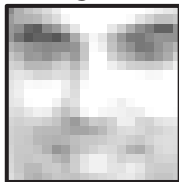


R

k -means factors correspond to prototypical faces.

Example: Non-negative matrix factorization

Original

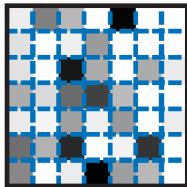


x_i^\top (original)

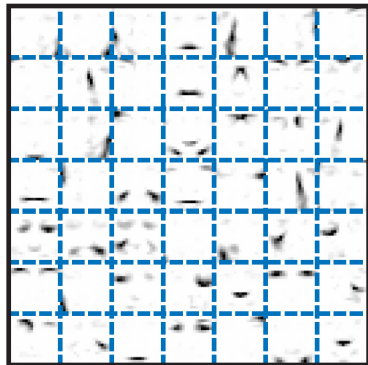


\hat{x}_i^\top

=



l_i^\top



R

NMF factors correspond to parts of faces.

Example: Latent Dirichlet allocation

“Arts”	“Budgets”	“Children”	“Education”
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

R
(topic \times word)

The William Randolph Hearst Foundation will give \$1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be \$200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive \$400,000 each. The Juilliard School, where music and the performing arts are taught, will get \$250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual \$100,000 donation, too.

L
(doc \times topic)

Machine Learning

06 – Dimensionality Reduction

Part 2: Singular Value Decomposition

Prof. Dr. Rainer Gemulla

Universität Mannheim

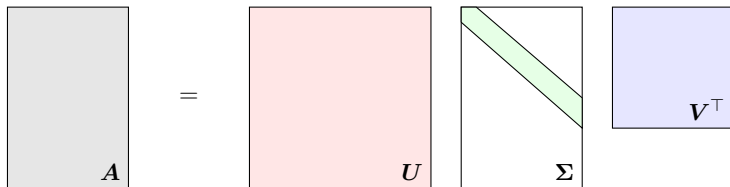
Version: 2023-2

Definition

Theorem

For each $A \in \mathbb{R}^{m \times n}$, there are orthogonal matrices $U_{m \times m}$, $V_{n \times n}$, and a diagonal matrix $\Sigma_{m \times n}$ with values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$ on the main diagonal such that $A = U\Sigma V^\top$.

- $U\Sigma V^\top$ is called the **singular value decomposition** (SVD) of A
- Values σ_i are the **singular values** of A
- Columns of U are the **left singular vectors** of A
- Columns of V are the **right singular vectors** of A



Characterization of the four fundamental subspaces

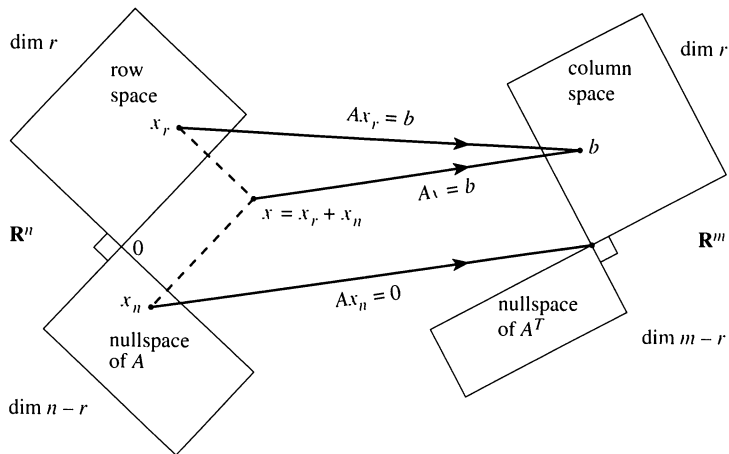
The **fundamental theorem of linear algebra** states that every matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ induces four fundamental subspaces:

- The **column space** (range, image) of dimension $\text{rank}(\mathbf{A}) = r$
 - ▶ All $\mathbf{x} \in \mathbb{R}^m$ that are linear combinations of columns of \mathbf{A}
- The **left nullspace** (cokernel) of dimension $m - r$
 - ▶ Left null space is orthogonal to column space
 - ▶ Set of all vectors $\mathbf{x} \in \mathbb{R}^m$ for which $\mathbf{x}^\top \mathbf{A} = \mathbf{0}^\top$
- The **row space** (coimage) of dimension r
- The **nullspace** (kernel) of dimension $n - r$

Explicit bases for these subspaces can be obtained from the SVD:

- Column space: the first r columns of \mathbf{U}
- Left null space: the last $(m - r)$ columns of \mathbf{U}
- Row space: the first r columns of \mathbf{V}
- Null space: the last $(n - r)$ columns of \mathbf{V}

The four fundamental subspaces



The action of A . Row space to column space, nullspace to zero.

Pseudo-inverse

Problem (least squares, linear regression).

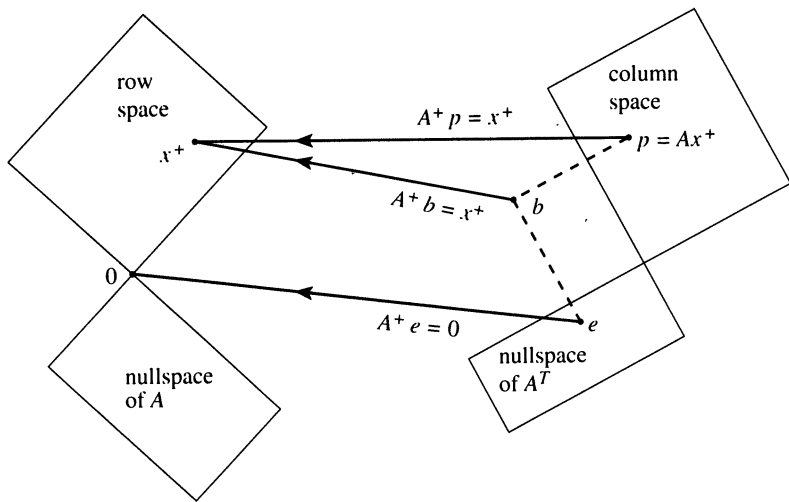
Given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, find $x \in \mathbb{R}^n$ minimizing $\|Ax - b\|_2$.

- If A is invertible, the solution is $A^{-1}Ax = A^{-1}b \Leftrightarrow x = A^{-1}b$
- A **pseudo-inverse** A^+ captures some properties of inverse A^{-1}
- The **Moore–Penrose pseudo-inverse** of A is a matrix $A^+ \in \mathbb{R}^{n \times m}$ satisfying the following criteria
 - ▶ $AA^+A = A$ (but it is possible that $AA^+ \neq I$)
 - ▶ $A^+AA^+ = A^+$ (cf. above)
 - ▶ $(AA^+)^\top = AA^+$ (AA^+ is symmetric)
 - ▶ $(A^+A)^\top = A^+A$ (as is A^+A)
- If $A = U\Sigma V^\top$ is the SVD of A , then $A^+ = V\Sigma^+U^\top$
 - ▶ Σ^+ replaces each $\sigma_i > 0$ by $1/\sigma_i$ and transposes

Theorem.

An optimum solution for the above problem is $x = A^+b$.

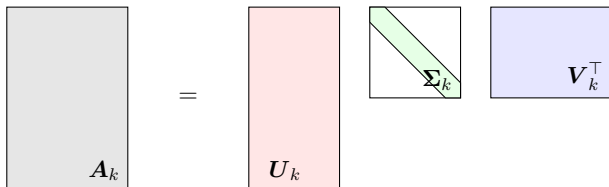
Pseudo inverse (illustration)



The inverse of A is (where possible) the pseudo-inverse A^+ .

Truncated SVD

- The rank of a matrix is number of its non-zero singular values
 - ▶ Easy to see by writing $\mathbf{A} = \sum_{j=1}^{\min\{n,m\}} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top$
- A **truncated SVD** only takes the first k columns of \mathbf{U} and \mathbf{V} and the main $k \times k$ submatrix of $\mathbf{\Sigma}$, where k called **size**
 - ▶ $\mathbf{A}_k = \sum_{j=1}^k \sigma_j \mathbf{u}_j \mathbf{v}_j^\top = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$
 - ▶ $\text{rank}(\mathbf{A}_k) = k$ (if $\sigma_k > 0$)
 - ▶ \mathbf{U}_k and \mathbf{V}_k are not orthogonal anymore, but they are **column-orthogonal**
- If $k = \min\{m, n\}$, then $\mathbf{A}_k = \mathbf{A}$; called **thin SVD** (*economy-sized*)
- If $k = \text{rank}(\mathbf{A})$, then $\mathbf{A}_k = \mathbf{A}$; called **compact SVD**
- If $k < \text{rank}(\mathbf{A})$, then \mathbf{A}_k is **low-rank approximation** of \mathbf{A}



SVD is best low-rank approximation

Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be the SVD of \mathbf{A} . Then

- $\|\mathbf{A}\|_F^2 = \sum_{i=1}^{\min\{n,m\}} \sigma_i^2$
- $\|\mathbf{A}\|_2 = \sigma_1$
 - ▶ Remember: $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min\{n,m\}} \geq 0$
- Therefore $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{n} \|\mathbf{A}\|_2$
- Sq. Frobenius norm of truncated SVD is $\|\mathbf{A}_k\|_F^2 = \sum_{i=1}^k \sigma_i^2$
 - ▶ And of the approximation error $\|\mathbf{A} - \mathbf{A}_k\|_F^2 = \sum_{i=k+1}^{\min\{n,m\}} \sigma_i^2$

The Eckart–Young theorem

Let $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$ be the size- k truncated SVD of \mathbf{A} with $k \leq \text{rank}(\mathbf{A})$. Then \mathbf{A}_k is best rank- k approximation to \mathbf{A} in terms of Frobenius norm. That is

$$\|\mathbf{A} - \mathbf{A}_k\|_F \leq \|\mathbf{A} - \mathbf{B}\|_F \quad \text{for all rank-}k \text{ matrices } \mathbf{B}.$$

Relationships to eigendecomposition

- An **eigenvector** of a square matrix A is a vector v such that A only changes the magnitude of v
 - ▶ I.e. $Av = \lambda v$ for some $\lambda \in \mathbb{R}$
 - ▶ Such λ is an **eigenvalue** of A
 - ▶ [Try it!](#)
- An **eigendecomposition** of A is $A = Q\Delta Q^{-1}$
 - ▶ The columns of Q are the eigenvectors of A
 - ▶ Matrix Δ is a diagonal matrix with the eigenvalues
- Not every (square) matrix has eigendecomposition
 - ▶ If A is of form BB^T , it always has eigendecomposition
- The SVD of A is closely related to the eigendecompositions of AA^T and $A^T A$
 - ▶ The left singular vectors are the eigenvectors of AA^T
 - ▶ The right singular vectors are the eigenvectors of $A^T A$
 - ▶ The singular values are the square roots of the eigenvalues of both AA^T and $A^T A$

Machine Learning

06 – Dimensionality Reduction

Part 3: Interpreting the SVD

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Factor interpretation

- The most common way to interpret the SVD is to consider the columns of U (or V)
 - ▶ Let X be examples-by-features and $U\Sigma V^\top$ its SVD
 - ▶ If two rows have similar values in a column of U , these examples are somehow similar
 - ▶ If two columns have similar values in a row of V^\top , these attributes are somehow similar
 - ▶ In both cases, first entries often matter most \rightarrow truncated SVD

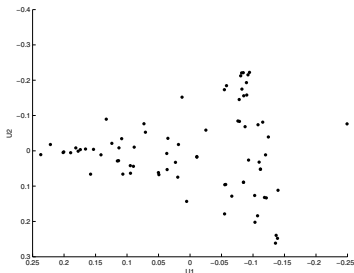


Figure 3.2. The first two factors for a dataset ranking wines.

- Example: people's ratings of different wines
- Scatterplot of first and second column of U
 - ▶ left: likes wine
 - ▶ right: doesn't like
 - ▶ up: prefers red wine
 - ▶ bottom: prefers white wine
- Conclusion: winelovers like red and white, others care

Example: Weather data ($k = 2$)

1.00		9.05	16.55	26.73	18.75	17.81
	1.00	-4.14	0.27	2.32	-0.89	-0.69

			Jan	Apr	Jul	Oct	Year
0.62	1.69	Stockholm	-0.70	8.60	21.90	9.90	10.00
0.69	2.11	Minsk	-2.10	12.20	23.60	10.20	10.60
0.83	0.00	London	7.90	13.30	22.80	15.20	14.80
0.90	1.52	Budapest	1.20	16.30	26.50	16.10	15.00
0.88	0.30	Paris	6.90	14.70	24.40	15.80	15.50
0.98	1.59	Bucharest	1.50	18.00	28.80	18.00	16.50
1.09	-0.66	Barcelona	12.40	17.60	27.50	21.50	20.00
1.14	-0.31	Rome	11.90	17.70	30.30	21.40	20.40
1.16	-1.09	Lisbon	14.80	19.80	27.90	22.50	21.50
1.24	-0.35	Athens	12.90	20.30	32.60	23.10	22.30
1.21	-1.26	Valencia	16.10	20.20	29.10	23.60	22.30
1.27	-1.12	Malta	16.10	20.00	31.50	25.20	23.20

X

(RMSE: 0.60)

Example: Weather data ($k = 2$), truncated SVD

147.54		0.22	0.40	0.64	0.45	0.43
	20.09	-0.85	0.06	0.47	-0.18	-0.14

			Jan	Apr	Jul	Oct	Year
0.18	0.41	Stockholm	-0.70	8.60	21.90	9.90	10.00
0.19	0.51	Minsk	-2.10	12.20	23.60	10.20	10.60
0.24	0.00	London	7.90	13.30	22.80	15.20	14.80
0.25	0.37	Budapest	1.20	16.30	26.50	16.10	15.00
0.25	0.07	Paris	6.90	14.70	24.40	15.80	15.50
0.28	0.39	Bucharest	1.50	18.00	28.80	18.00	16.50
0.31	-0.16	Barcelona	12.40	17.60	27.50	21.50	20.00
0.32	-0.07	Rome	11.90	17.70	30.30	21.40	20.40
0.33	-0.27	Lisbon	14.80	19.80	27.90	22.50	21.50
0.35	-0.08	Athens	12.90	20.30	32.60	23.10	22.30
0.34	-0.31	Valencia	16.10	20.20	29.10	23.60	22.30
0.36	-0.27	Malta	16.10	20.00	31.50	25.20	23.20

X

(RMSE: 0.60)

Thin SVD of Weather data (U)

$$U_5 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} Stockholm \\ Minsk \\ London \\ Budapest \\ Paris \\ Bucharest \\ Barcelona \\ Rome \\ Lisbon \\ Athens \\ Valencia \\ Malta \end{matrix} & \begin{pmatrix} 0.18 & 0.41 & 0.61 & 0.28 & -0.32 \\ 0.19 & 0.51 & 0.08 & -0.54 & 0.40 \\ 0.24 & 0.00 & 0.20 & -0.15 & 0.04 \\ 0.25 & 0.37 & -0.39 & 0.18 & -0.10 \\ 0.25 & 0.07 & 0.05 & -0.25 & -0.22 \\ 0.28 & 0.39 & -0.49 & 0.30 & 0.08 \\ 0.31 & -0.16 & -0.01 & 0.33 & -0.26 \\ 0.32 & -0.07 & 0.30 & 0.10 & 0.07 \\ 0.33 & -0.27 & -0.23 & -0.27 & -0.46 \\ 0.35 & -0.08 & 0.10 & -0.23 & -0.09 \\ 0.34 & -0.31 & -0.12 & -0.21 & 0.17 \\ 0.36 & -0.27 & 0.12 & 0.37 & 0.59 \end{pmatrix} \end{matrix}$$

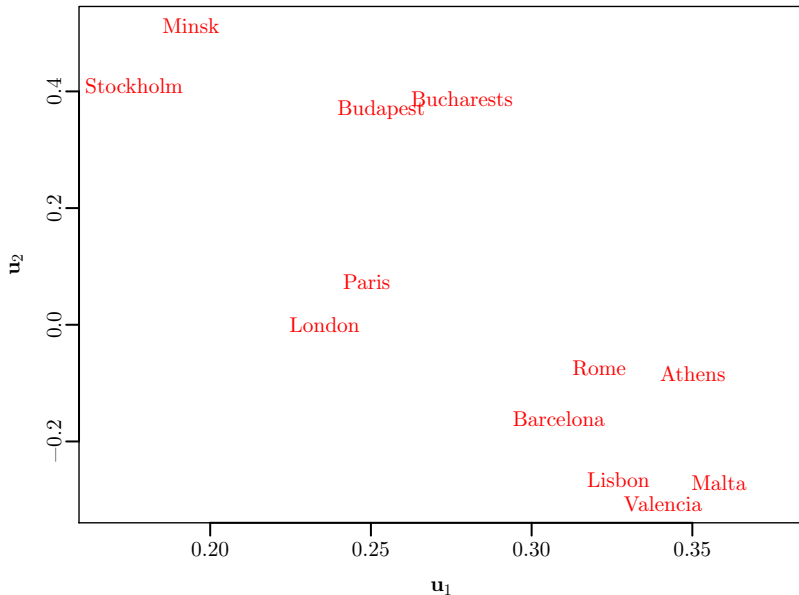
Thin SVD of Weather data (Σ)

$$\Sigma_5 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{pmatrix} 147.55 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 20.09 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 4.25 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.77 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.32 \end{pmatrix} \end{matrix}$$

Thin SVD of Weather data (V)

$$V_5 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} Jan \\ Apr \\ Jul \\ Oct \\ Year \end{matrix} & \begin{pmatrix} 0.22 & -0.85 & 0.31 & -0.30 & 0.21 \\ 0.40 & 0.06 & -0.74 & -0.52 & 0.17 \\ 0.64 & 0.47 & 0.54 & -0.16 & 0.21 \\ 0.45 & -0.18 & -0.25 & 0.78 & 0.30 \\ 0.43 & -0.14 & -0.03 & 0.05 & -0.8 \end{pmatrix} \end{matrix}$$

Example: Weather data ($r = 2$), SVD plot



Orthogonal matrices and rotations

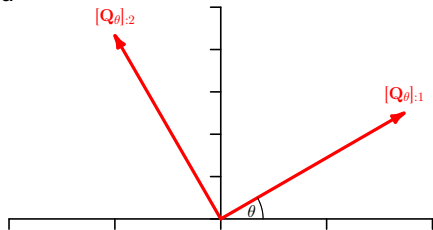
- Orthogonal matrices are **rotation matrices**
- Consider orthogonal matrix Q
- Inner products are retained: $(Qx)^\top (Qy) = x^\top Q^\top Qy = x^\top y$
- Thus Euclidean norms also retained: $\|Qx\| = \|x\|$
- Implies that all angles are retained

- In 2D: $Q_\theta = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$

- ▶ Consider vector

$$\begin{pmatrix} x \\ y \end{pmatrix} = xe_1 + ye_2$$

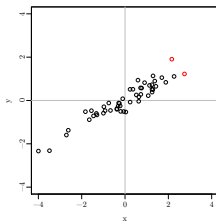
- ▶ $Q_\theta \begin{pmatrix} x \\ y \end{pmatrix} = x[Q_\theta]_{:1} + y[Q_\theta]_{:2}$



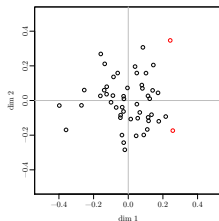
- Thus: the columns of Q form “new axes” for rotation Qv (and also $v^\top Q^\top$)
- Similarly: rows of Q form “new axes” for rotation $Q^\top v$ (and also $v^\top Q$; rotates backwards)

Geometric interpretation (1)

$$\mathbf{X}_{50 \times 2}$$



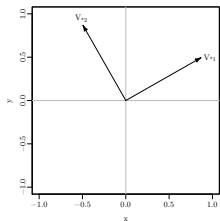
$$\mathbf{U}_2$$



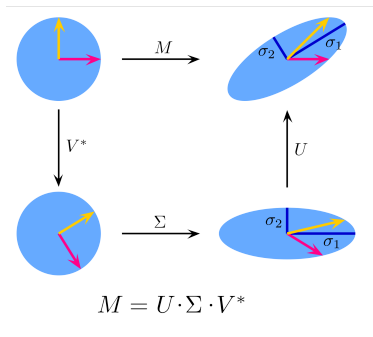
$$\Sigma_2$$

$$\begin{pmatrix} 11.64 & 0 \\ 0 & 1.69 \end{pmatrix}$$

$$\mathbf{V}_2^\top$$

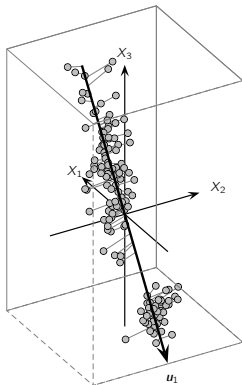


Geometric interpretation (2)



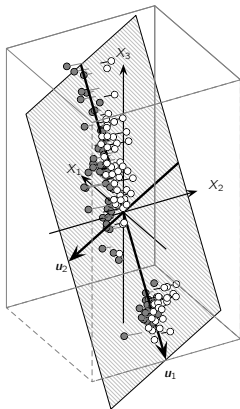
- Let $U\Sigma V^\top$ be the SVD of M
- SVD shows that every linear mapping $y = Mx$ can be considered as a series of rotation, stretching, and rotation operations
 - ▶ Matrix V^\top performs the first rotation $y_1 = V^\top x$
 - ▶ Matrix Σ performs the stretching $y_2 = \Sigma y_1$
 - ▶ Matrix U performs the second rotation $y = Uy_2$

Direction of largest variation (1)



- The right singular vectors give the directions of the variation in the data
 - ▶ The first right singular vector is the direction of the largest variation

Direction of largest variation (2)

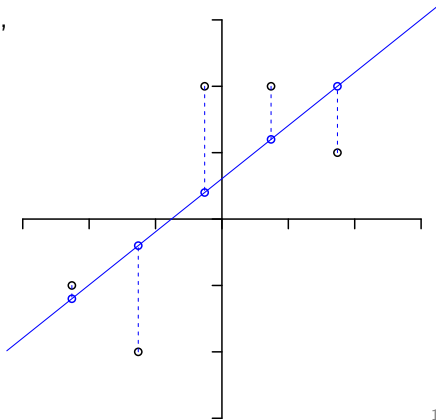


- The right singular vectors give the directions of the variation in the data
 - ▶ The first right singular vector is the direction of the largest variation
 - ▶ The second right singular vector is orthogonal to the first one and gives the direction of the second-largest variation
 - ▶ First two directions span a hyperplane
- From Eckart–Young we know that if we project the data to the spanned hyperplanes, the (sq.) distance of the projection is minimized

SVD and linear regression (1)

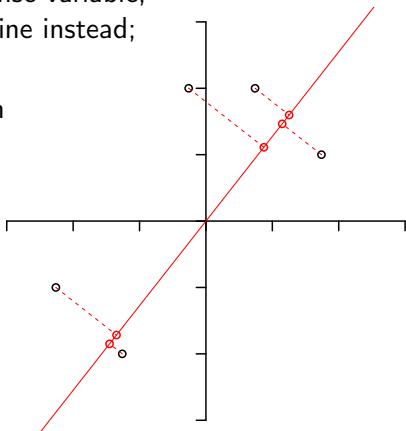
Consider the 1-dimensional case.

- Recall: linear regression models response $y \in \mathbb{R}$ as a linear function of input $x \in \mathbb{R}$
- Parameterized by a weight vector $\mathbf{w} = (w_0 \ w_1)^\top \in \mathbb{R}^2$, consisting of bias w_0 and slope w_1
- Prediction is $\hat{y} = \mathbf{w} \cdot (1 \ x)^\top$, the error is $(y - \hat{y})^2$ (least squares)
- Goal is to minimize this error, i.e., $\|\mathbf{y} - \hat{\mathbf{y}}\|_2$
- Generally, set $\mathbf{X}_1 = (\mathbf{1} \ \mathbf{X})$, then $\mathbf{w} = \mathbf{X}_1^+ \mathbf{y}$

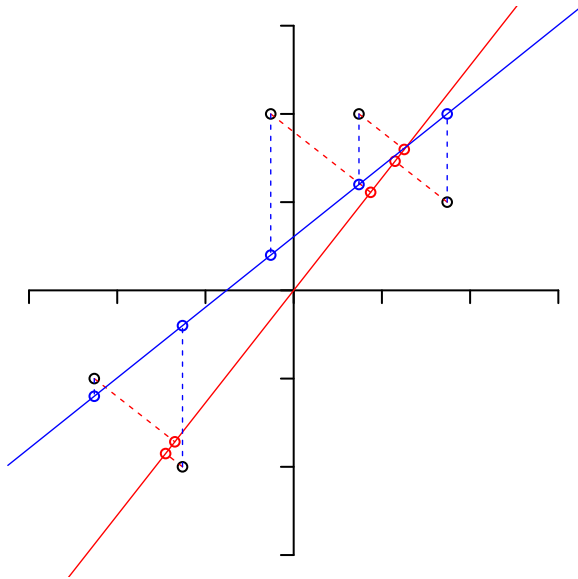


SVD and linear regression (2)

- Contrast this to the size-1 truncated SVD of $\mathbf{X} = (\mathbf{x} \ \mathbf{y})$
- We obtain a vector \mathbf{u}_1 , a scaling coefficient σ_1 , and a vector \mathbf{v}_1
- Let's look at the line described by \mathbf{v}_1 (roughly corresponds to \mathbf{w})
- Reconstructed data is $\mathbf{X}_1 = \mathbf{u}_1 \sigma_1 \mathbf{v}_1^\top$; all points lie on the line
- There is no distinguished response variable; we minimize (sq.) distance to line instead; i.e., $\|\mathbf{X} - \mathbf{X}_1\|_F$.
- This is different from regression



SVD and linear regression (3)



Component interpretation

- Recall that we can write

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \sum_{i=1}^r \mathbf{X}_{(i)},$$

where $\mathbf{X}_{(i)} = \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$

- This explains the data as a sums of (rank-1) layers
 - ▶ The first layer explains the most
 - ▶ The second corrects that by adding and removing smaller values
 - ▶ The third corrects that by adding and removing even smaller values
 - ▶ ...
- The layers don't have to be very intuitive

Machine Learning

06 – Dimensionality Reduction

Part 4: Using the SVD

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Outline

1. How many factors?
2. Data preprocessing and PCA
3. Other Uses
4. Computing the SVD

Problem

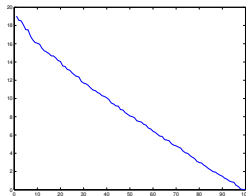
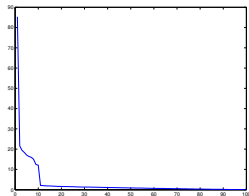
- Most applications do not use full SVD, but truncated SVD
 - ▶ To concentrate on “the most important parts”
- But how to select the size k of the truncated SVD?
 - ▶ What is important, what is unimportant?
 - ▶ What is structure, what is noise?
 - ▶ Too small: all subtlety is lost
 - ▶ Too big: all smoothing is lost
- Typical methods rely on singular values in a way or another
 - ▶ Neither of these methods is fully convincing
 - ▶ Problem addressed by factor analysis (discussed later)

Guttman–Kaiser criterion and captured energy

- Perhaps the oldest method is the Guttman–Kaiser criterion:
 - ▶ Select k so that for all $i > k$, $\sigma_i < 1$
 - ▶ Motivation: all components with singular value less than unit are uninteresting
- Another common method is to select enough singular values such that the sum of their squares is 90% of the total sum of the squared singular values
 - ▶ The exact percentage can be different (80%, 95%)
 - ▶ Motivation: The resulting matrix “explains” 90% of the (sq.) Frobenius norm of the matrix
- **Problem:** Both of these methods are based on arbitrary thresholds and do not consider the “shape” of the data

Cattell's Scree test

- **Scree plot** shows squared singular values in decreasing order
 - ▶ The plot (hopefully) looks like debris in front of a hill, hence the name
- The scree test is a subjective decision on the size based on the shape of the scree plot
- The size should be set to a point where
 - ▶ there is a clear drop in the magnitudes of the values; or
 - ▶ the values start to even out
- **Problem:** Scree test is subjective, and many data don't have any clear shapes to use (or have many)
 - ▶ Automated methods have been developed to detect the shapes from the scree plot



Entropy-based method

- Consider the relative contribution of each singular value to the overall (sq.) Frobenius norm
 - ▶ Relative contribution of σ_k is $f_k = \sigma_k^2 / \sum_i \sigma_i^2$
- We can treat these as probabilities and define the (normalized) **entropy** of the singular values as

$$E = -\frac{1}{\log(\min\{n, m\})} \sum_{i=1}^{\min\{n, m\}} f_i \log f_i$$

- ▶ The basis of the logarithm doesn't matter
- ▶ We assume that $0 \cdot \infty = 0$
- ▶ Low entropy (close to 0): the first singular value has almost all mass
- ▶ High entropy (close to 1): the singular values are almost equal
- The size is selected to be the smallest k such that $\sum_{i=1}^k f_i \geq E$
- **Problem:** Why entropy?

Random flip of signs

- Multiply every element of the matrix \mathbf{A} randomly with either 1 or -1 to get $\tilde{\mathbf{A}}$
 - ▶ The Frobenius norm doesn't change ($\|\mathbf{A}\|_F = \|\tilde{\mathbf{A}}\|_F$)
 - ▶ The spectral norm does change ($\|\mathbf{A}\|_2 \neq \|\tilde{\mathbf{A}}\|_2$)
 - ▶ The change is the larger the more “structure” \mathbf{A} has
- Idea: select k such that the residual matrix contains only noise
 - ▶ $\mathbf{X}_{-k} = \mathbf{X} - \mathbf{X}_k$ is the residual matrix after size- k truncated SVD
 - ▶ \mathbf{X}_{-k} is based on the last $m - k$ columns of \mathbf{U} , $\min\{n, m\} - k$ singular values, and last $n - k$ rows of \mathbf{V}^\top
 - ▶ Construct $\tilde{\mathbf{X}}_{-k}$ from \mathbf{X}_{-k} by randomly flipping signs
 - ▶ Select size k to be such that

$$\frac{\|\mathbf{X}_{-k}\|_2 - \|\tilde{\mathbf{X}}_{-k}\|_2}{\|\mathbf{X}_{-k}\|_F}$$

is small

- **Problem:** How small is small?

Outline

1. How many factors?
2. Data preprocessing and PCA
3. Other Uses
4. Computing the SVD

Normalization

- Consider data normalization before SVD is applied
 - ▶ SVD is sensitive to data scaling
 - ▶ If one attribute is height in meters and another weights in grams, weight seems to carry much more importance
 - ▶ If data is all positive, the first singular vector just explains where in the positive quadrant the data is
- The ***z*-scores** are attributes whose values are transformed by
 - ▶ Center each attribute (subtract mean)
 - ▶ Normalize each attributes to unit variance (divide by standard deviation)
 - ▶ Implicit assumption: attributes normally distributed (so that centering and rescaling to unit variance is meaningful)
 - ▶ Implicit assumption: attributes have equal importance
- Values that have larger magnitude than importance can also be normalized by first taking logarithms (from positive values) or cubic roots or ...

Relationship to PCA (1)

- Truncated SVD can also be used to battle the *curse of dimensionality*
 - ▶ All points are far from each other in very high dimensional spaces
 - ▶ High dimensionality slows down data mining algorithms
 - ▶ If we use the truncated SVD, every example is represented by its row in U_k (k values instead of n)
 - ▶ If $k \ll n$, we performed **dimensionality reduction**
- SVD is closely related to **principal components analysis** (PCA)
 - ▶ Technically, PCA works as follows:
 1. Center each attribute of X to obtain M
 2. Compute the **sample covariance matrix** $S = M^T M / (m - 1)$
 3. Compute the eigendecomposition $S = Q \Lambda Q^T$ s.t. Q orthogonal
 - ▶ The columns of Q are called **principal components**
 - ▶ The corresponding eigenvalues in Λ are the component weights
 - ▶ [Try it!](#)
 - ▶ We now show: **PCA \approx SVD on centered data**

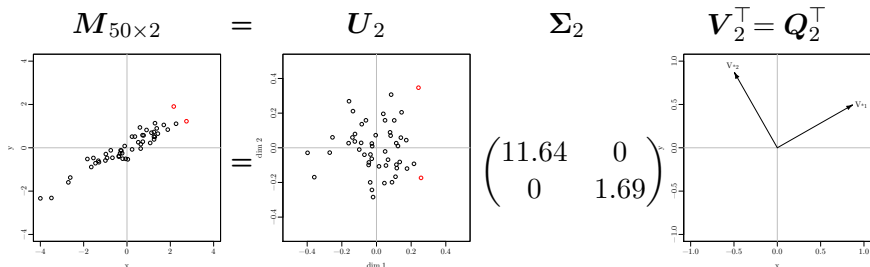
Relationship to PCA (2)

- Relationship between SVD and PCA
 - ▶ M : centered data, $S = M^T M / (m - 1)$: sample covariance
 - ▶ $S = Q \Lambda Q^T$: eigendecomposition of S (computed by PCA)
 - ▶ $M = U \Sigma V^T$: SVD of M
 - ▶ Observe: SVD of $M / \sqrt{m - 1}$ is then $U(\Sigma / \sqrt{m - 1})V^T$
 - ▶ From slide 06-2/9, we know that $Q = V$
→ **Principal components = right singular vectors of M**
 - ▶ From slide 06-2/9, we know that $\Sigma^2 / (m - 1) = \Lambda$
→ **Components weights = scaled roots of singular values of M**
- PCA associates each data example with a set of **scores**
 - ▶ One per principal component
 - ▶ $m \times n$ “score matrix” given by $Z = MQ$
 - ▶ We have: $Z = MQ = (U \Sigma V^T)V = U \Sigma$
 - ▶ Known as the **Karhunen–Loève transform** (KLT) of rows of M
 - ▶ For dimensionality reduction, we only take the first k components:

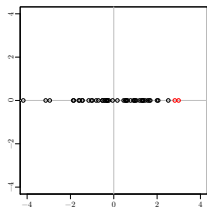
$$Z_k = M Q_k = U_k \Sigma_k$$

- More later when we talk about latent linear models

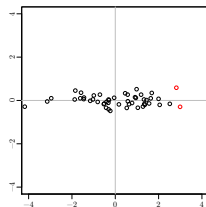
Relationship to PCA (3)



$$Z_1 = MQ_1 = U_1 \Sigma_1$$



$$Z_2 = MQ_2 = U_2 \Sigma_2$$



What is the difference between the PCA and the SVD of $X (\neq M)$?

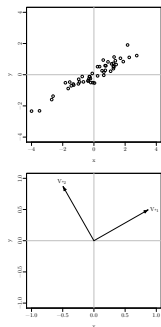
Outline

1. How many factors?
2. Data preprocessing and PCA
3. Other Uses
4. Computing the SVD

Denoising data

- Common application of SVD is to remove noise from data
 - ▶ Perturbations with random noise do not significantly affect good low-rank approximations (see, e.g., [Achlioptas and Mcsherry, 2007](#))
 - ▶ Assume $\mathbf{X} = \mathbf{A} + \mathbf{E}$, where \mathbf{A} is low-rank data and \mathbf{E} is noise
 - ▶ If noise is iid (mean 0) and not too large, we can approximately recover \mathbf{A} by taking the truncated SVD of \mathbf{X}
 - ▶ As before, key problem is to select k

- Example



- Original data
 - ▶ Looks like 1-dimensional with some noise
- The right singular vectors show the directions
 - ▶ The first looks like the data direction
 - ▶ The second looks like the noise direction
- The singular values confirm this (large drop)
 - ▶ $\sigma_1 = 11.64$
 - ▶ $\sigma_2 = 1.69$

Visualization

- Truncated SVD with $k = 2, 3$ allows us to visualize the data
 - ▶ We can plot the projected data points after 2D or 3D PCA
 - ▶ Or we can scatter plot the entries of two or three singular vectors
 - ▶ Or we color data points based on their entries in a singular vector
 - ▶ ...

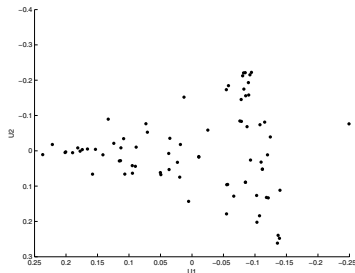
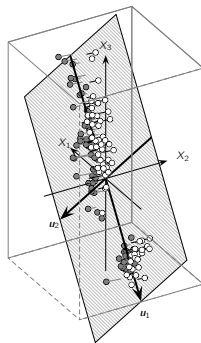


Figure 3.2. *The first two factors for a dataset ranking wines.*



Latent semantic analysis

- The **latent semantic analysis** (LSA) is an information retrieval method that uses SVD
- The data: a document-term matrix \mathbf{X}
 - ▶ Values are (weighted) term frequencies
 - ▶ Typically tf-idf values (the frequency of the term in the document divided by the global frequency of the term)
- The truncated SVD $\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$ of \mathbf{X} is computed
 - ▶ Matrix \mathbf{U}_k associates documents to “topics”
 - ▶ Matrix \mathbf{V}_k associates “topics” to terms
 - ▶ If two rows of \mathbf{U}_k are similar, the corresponding documents talk about the “same topics”
- A query q can be answered by considering its term vector q
 - ▶ q is projected to $q_k = (q^\top \mathbf{V}_k \mathbf{\Sigma}_k^+)^{\top}$ (called: **fold in**)
 - ▶ q_k is compared to rows of \mathbf{U}_k and most similar rows are returned

Outline

1. How many factors?
2. Data preprocessing and PCA
3. Other Uses
4. Computing the SVD

Algorithms for SVD

- In principle, the SVD of \mathbf{X} can be computed by computing the eigendecomposition of $\mathbf{X}^\top \mathbf{X}$
 - ▶ This gives us right singular vectors and squares of singular values
 - ▶ Left singular vectors can be solved: $\mathbf{U} = \mathbf{X}\mathbf{V}\mathbf{\Sigma}^+$
 - ▶ Bad for [numerical stability](#)
- Full SVD can be computed in time $O(nm \min\{n, m\})$
 - ▶ Matrix \mathbf{X} is first reduced to a bidiagonal matrix
 - ▶ The SVD of the bidiagonal matrix is computed using iterative methods (similar to eigendecompositions)
- Methods that are fast in practice exist
 - ▶ Especially for truncated SVD
- Efficient implementation of an SVD algorithm requires considerable work and knowledge
 - ▶ Luckily (almost) all numerical computation packages and programs implement SVD

Machine Learning

06 – Dimensionality Reduction

Part 5: Latent Linear Models

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Latent variable models

- **Latent variable models** (LVM) are models that assume that the data is explained by a set of unobserved **latent variables**
- Example: Weather data of Slide 06-1/6
 - ▶ One latent variable z_1 may correspond to mean temperature (cold or warm?)
 - ▶ Another latent variable z_2 may correspond to temperature variance (balanced or hot summers/icy winters)
 - ▶ LVMs then assume that a city's temperature data (x_i) can be explained by its values of the latent variables ($z_i = (z_{i1} \ z_{i2})^T$)
- LVMs are conceptually very general
 - ▶ Latent variables can be used to model dependencies (cf. graphical models)
 - ▶ E.g., dependencies between features
 - ▶ E.g., dependencies between examples (non-iid data)
 - ▶ More in IE678 Deep Learning course

Latent linear models

- In *latent linear models* (LLM) the relationship between the example x and the respective latent variables z is linear

$$x = Wz + \mu + \epsilon$$

- ▶ $x \in \mathbb{R}^D$ refers to data point
 - ▶ $z \in \mathbb{R}^L$ refers latent variables for this data point
 - ▶ Parameter $\mu \in \mathbb{R}^D$ is a bias term
 - ▶ Parameter $W \in \mathbb{R}^{D \times L}$ describes how to “transform” z (**factor loading matrix**)
 - ▶ $\epsilon \in \mathbb{R}^D$ is a noise term (independent, mean 0)
- We can interpret SVD and PCA as LLMs
 - ▶ SVD: $\mu = 0$, $W = V$, $z_i = \Sigma^T u_i$ (where $u_i = U_{i:}^\top$)
 - ▶ PCA: $\mu = \frac{1}{m} \sum_i x_i$, $W = Q$, $z_i = Q^T(x_i - \mu)$
 - ▶ For dimensionality reduction, we only keep the first $L < D$ components (then generally not exact, $\epsilon_i \neq 0$)
 - ▶ **Neither model is generative**

Factor analysis

- **Factor analysis** is an LLM where both latent variables and noise is assumed to be Gaussian
- We obtain the *generative* model

$$z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$x|z, \theta \sim \mathcal{N}(\mathbf{W}z + \mu, \Psi)$$

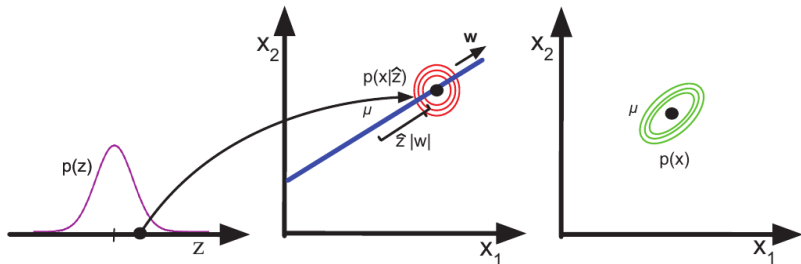
- ▶ $\Psi \in \mathbb{R}^{D \times D}$ is a *diagonal* covariance matrix of the noise
 - ▶ Diagonal since z_i should explain the correlation in the data, not the noise term
 - ▶ Parameters are $\theta = \{ \mathbf{W}, \mu, \Psi \}$
- One can show that the *marginal distribution* is Gaussian, too

$$p(x|\theta) = \mathcal{N}(x|\mu, \mathbf{W}\mathbf{W}^\top + \Psi)$$

Probabilistic PCA

- When $\Psi = \sigma^2 \mathbf{I}$ (isotropic Gaussian noise), the resulting model is called **probabilistic PCA** (PPCA)
- In PPCA, data is generated by transforming a standard multivariate Gaussian r.v. (z) into data (x)
 1. Map point to mean $\mathbf{W}z + \mu$
 2. Add Gaussian $\mathcal{N}(\mathbf{0}, \Psi)$ noise

Example: 2D data ($D = 2$), 1D latent variable ($L = 1$)



Why factor analysis?

- If data is assumed multi-variate Gaussian (MVN) in factor analysis, why not directly estimate the mean vector and covariance matrix?
- Reason 1: PPCA defines a density on \mathbf{x} with less parameters
 - ▶ PPCA: D for mean, DL for \mathbf{W}
 - ▶ MVN: D for mean, D^2 for Σ
- Reason 2: Hope that \mathbf{z} reveal interesting properties / are useful
 - ▶ Posterior of latent variables

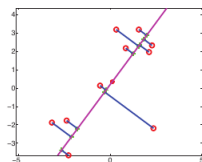
$$p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}|\mathbf{m}, \Sigma)$$

$$\Sigma = (\mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W})^{-1}$$

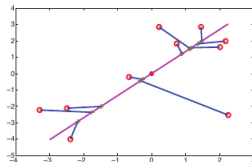
$$\mathbf{m} = \Sigma \mathbf{W}^\top \Psi^{-1} (\mathbf{x} - \boldsymbol{\mu})$$

- ▶ \mathbf{m} are called **scores** (= cond. expectation of latent variable)
- ▶ Dimensionality reduction when $L < D$: $\mathbf{z}, \mathbf{m} \in \mathbb{R}^L$
- ▶ Latent variables then serve as a **bottleneck**, i.e., a small, compressed representation of the data

Reconstruction of PCA vs. PPCA



(a)



(b)

Figure 12.5 An illustration of PCA and PPCA where $D = 2$ and $L = 1$. Circles are the original data points, crosses are the reconstructions. The red star is the data mean. (a) PCA. The points are orthogonally projected onto the line. Figure generated by `pcaDemo2d`. (b) PPCA. The projection is no longer orthogonal: the reconstructions are shrunk towards the data mean (red star). Based on Figure 7.6 of (Nabney 2001). Figure generated by `ppcaDemo2d`.

Unidentifiability

- Parameters of FA are **unidentifiable**, i.e., multiple different parameter choices correspond to the same data distribution
 - ▶ E.g., if we rotate \mathbf{W} via an orthogonal matrix \mathbf{R} , we obtain an equivalent marginal distribution (cf. slide 4), but different scores (rotated too)
- Unidentifiability implies that “true” parameters cannot be found
 - ▶ Even with infinite data
 - ▶ Does not affect predictive performance of the model (we can find an equivalent parameterization)
 - ▶ Does affect interpretation of the factors
- Common solutions
 - ▶ Force \mathbf{W} 's columns to be orthogonal, order by norm (as in PCA)
 - ▶ Force \mathbf{W} to be lower triangular
 - ▶ Use a (sparsity-promoting) prior on weights
 - ▶ Use a non-Gaussian prior on the latent factors (e.g., ICA)

Discussion (1)

- Parameters of FA models can be fit using the EM algorithm (more later)
- For PPCA, MLE estimate (assuming centered data) is

$$\hat{\mathbf{W}}_{\text{MLE}} = \mathbf{Q}_L(\mathbf{\Lambda}_L - \sigma^2 \mathbf{I})^{1/2}$$

$$\hat{\sigma}_{\text{MLE}}^2 = \frac{1}{D - L} \sum_{j=L+1}^D \lambda_j$$

- ▶ \mathbf{Q}_L are first L eigenvectors of data covariance matrix (as in PCA)
- ▶ $\mathbf{\Lambda}_L$ contains the corresponding eigenvalues (as in PCA); they are subsequently “reduced” by σ^2 (towards 0)
- ▶ When noise variance $\sigma^2 \rightarrow 0$ (no noise) in PPCA, we obtain PCA
- ▶ MLE $\hat{\sigma}_{\text{MLE}}^2$ of noise (or error) is average variance of discarded dimensions

Discussion (2)

- Choosing number of latent dimensions
 - ▶ Since PPCA is probabilistic, model selection is more principled (e.g., use L that maximizes the likelihood of validation data)
 - ▶ PCA can reconstruct the better the more components
 - ▶ PPCA model gets punished if it uses too many components (and thus puts probability mass on regions with little data)
- Can be extended in multiple ways; e.g.,
 - ▶ In **supervised PCA** (*Bayesian factor regression*), additionally model target $y|z, \theta \sim \mathcal{N}(\mathbf{w}_t^\top \mathbf{z} + \mu_t, \sigma_t^2)$
 - ▶ Also possible for other label distributions (e.g., classification)
 - ▶ In **independent component analysis** (ICA), use a non-Gaussian prior distribution on the \mathbf{z} , which can give more interpretable (and unique) results

Reconstruction error of PCA vs. PPCA

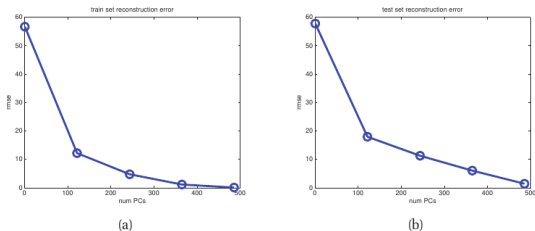


Figure 12.14 Reconstruction error on MNIST vs number of latent dimensions used by PCA. (a) Training set. (b) Test set. Figure generated by `pcaOverfitDemo`.

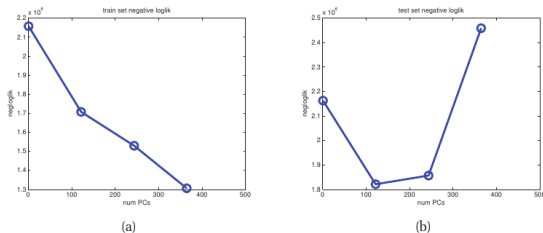
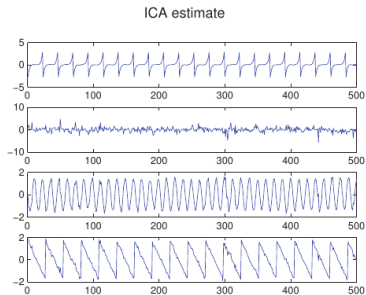
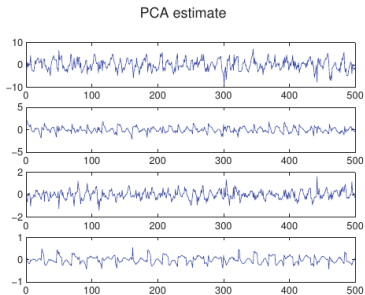
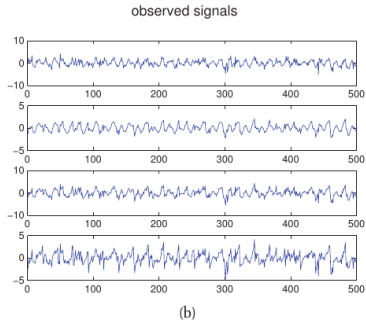
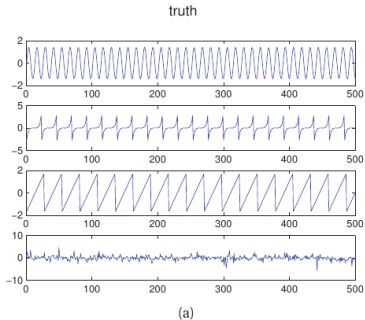


Figure 12.15 Negative log likelihood on MNIST vs number of latent dimensions used by PPCA. (a) Training set. (b) Test set. Figure generated by `pcaOverfitDemo`.

Example: ICA vs. PCA



Machine Learning

07 – EM Algorithm & Mixture Models

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Overview

- So far we assumed that all relevant variables are observed
 - ▶ In supervised learning: $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ during training (and \mathbf{x}_{test} during prediction)
 - ▶ In unsupervised learning: $\{\mathbf{x}_i\}$
- Coming up: How can we fit parameters if data is missing?
 - ▶ Training data is incomplete
 - ▶ Model contains latent variables (such as PPCA)
- In this lecture
 - ▶ The EM algorithm for ML/MAP parameter estimation
 - ▶ Mixture models, a powerful and useful class of models that can be fit with EM

Outline (The EM Algorithm)

1. Introduction
2. The EM Algorithm
3. Mixture Models

Summary

- EM algorithm
 - ▶ Framework for ML or MAP parameter estimation with missing data
 - ▶ E.g., partially observed data or LVMs
 - ▶ Iterates E(xpectation) and M(aximization) steps
 - ▶ E step infers missing-data distribution using current parameters
 - ▶ M step updates parameters using current missing-data distribution
- Mixture models
 - ▶ LVM with categorical latent variable
 - ▶ Density modeling, clustering, mixture of experts
 - ▶ For clustering, provides soft clustering (cluster membership probabilities instead for hard assignment)
 - ▶ In GMMs, each component distribution is a multivariate Gaussian
 - ▶ Parameter estimation via EM

Literature

- Murphy, Ch. 8.7 *Bound Optimization*, Ch. 3.5 *Mixture Models*, Ch. 21.4 *Clustering using mixture models*
- Mohammed J. Zaki, Wagner Meira Jr
Data Mining and Analysis: Fundamental Concepts and Algorithms ([Chapter 13.3](#))
2nd edition, Cambridge University Press, 2020
- Geoffrey McLachlan, Thriyambakam Krishnan
The EM Algorithm and Extensions
2nd edition, Wiley-Interscience, 2008

Machine Learning

07 – EM Algorithm & Mixture Models

Part 1: Introduction

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

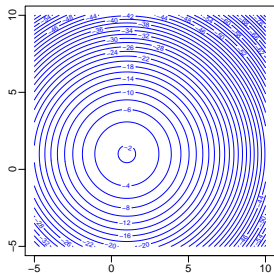
The EM Algorithm

- The **EM algorithm** is a framework to estimate model parameters with missing data
 - ▶ Due to [Dempster, Laird, Rubin \(1977\)](#)
 - ▶ Rather a framework than an algorithm
 - ▶ [Entire books](#) can be written about it
- Useful when
 - ▶ Observed data and missing data jointly modeled
 - ▶ MLE or MAP estimation desired, but direct methods are involved
 - ▶ But: parameter estimation would be “easy”, when all data were known (M step)
 - ▶ But: handling of missing values would be “easy”, when all parameters were known (E step)
- Key idea
 - ▶ Iterative method
 - ▶ Alternate between E step (using current parameter estimates) and M step (using “filled in” data)

Recall: Multivariate Gaussian distribution

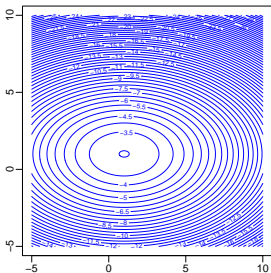
- Mean $\mu \in \mathbb{R}^D$, covariance $\Sigma \in \mathbb{R}^{D \times D}$ (or precision $\Lambda = \Sigma^{-1}$)
- Denoted $\mathcal{N}(\mu, \Sigma)$
- Let $|\Sigma|$ be the determinant of Σ . If Σ is positive definite:

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp \left[-\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu) \right]$$



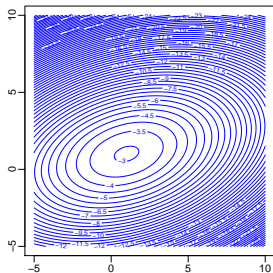
$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \sigma^2 I$$

(spherical)



$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

(diagonal)

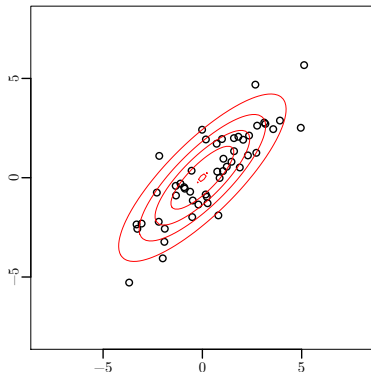


$$\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma = \begin{pmatrix} 5 & 1 \\ 1 & 2 \end{pmatrix}$$

MLE for Multivariate Gaussian

- Generative model: $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- With N iid. observations, $\mathbf{x}_1, \dots, \mathbf{x}_N$, ML estimate is given by sample mean and (uncorrected) sample covariance:

$$\hat{\boldsymbol{\mu}}_{\text{MLE}} = \frac{1}{N} \sum_i \mathbf{x}_i \qquad \hat{\boldsymbol{\Sigma}}_{\text{MLE}} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^\top$$



Missing data mechanisms (1)

- Some reasons for missing data
 - ▶ High cost of (complete) data acquisition
 - ▶ Errors in data acquisition
 - ▶ Non-response in surveys
 - ▶ Latent variable models
- **Missing data mechanism** is important for data analysis
 - ▶ Relationship between the complete data and the event that a data item is missing
- **Missing Completely At Random (MCAR)**
 - ▶ Event that data item is missing is independent of observed and missing data, i.e., occurs completely at random
 - ▶ No systematic reason for why data is missing
 - ▶ Rare in practice
 - ▶ Example: some questions only asked to random subset of persons in a survey

Missing data mechanisms (2)

- **Missing At Random (MAR)**

- ▶ Event that data item is missing depends on observed data but not on missing data
- ▶ Reason for missing data is systematic, but can be explained by observed data
- ▶ Example: students write 4 assignments (4 observed variables per student), only the ones who passed 3 assignments write the exam (1 variable per student)

- **Missing Not At Random (MNAR)** (*non-ignorable*)

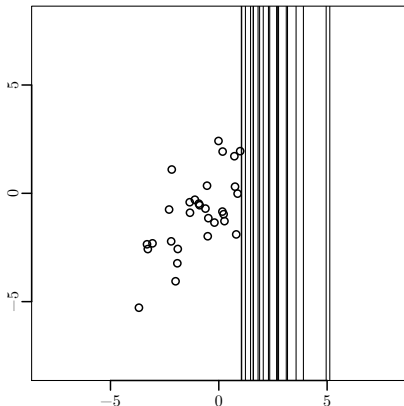
- ▶ Event that data item is missing can depend on observed and missing data
- ▶ Example: persons with high-income may respond to questions about income with lower probability

- Mechanism often cannot be determined

- ▶ Here we are mainly interested in latent variable models (= MCAR)
- ▶ Generally, we subsequently assume: MCAR oder MAR

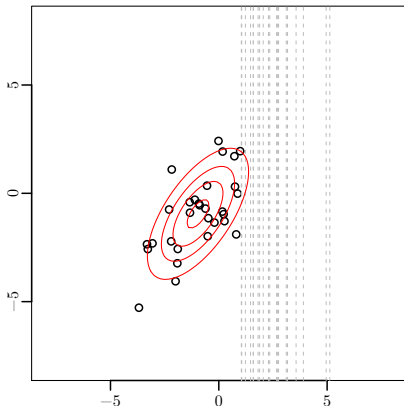
Example: Multivariate Gaussian with missing values

- Generative model: $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, 2D
- MAR mechanism: in each example (x_1, x_2) , x_2 is likely to be missing when $x_1 > 1$ (otherwise observed)
- Each data point with missing data lies on a line (given by x_1), but we do not know where



Complete case analysis

- **Complete case analysis** (*listwise deletion*) is simplest method
 - ▶ Ignores all data points with missing data
 - ▶ Can lead to biased estimates when mechanism is not MCAR
 - ▶ Does not use all available data
 - ▶ Not useful for LVMs



Marginals and conditionals of Gaussian models

Theorem

Suppose $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is jointly Gaussian with parameters

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}.$$

Then the marginals are given by

$$p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \quad p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}),$$

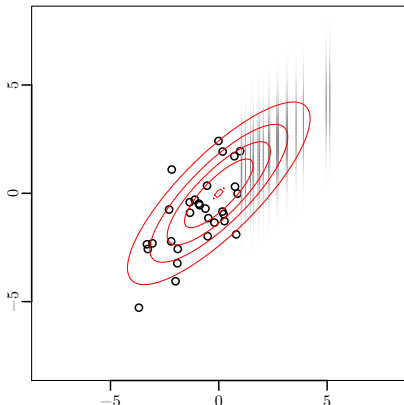
and the conditional distribution by

$$\begin{aligned} p(\mathbf{x}_1 | \mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1} \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2} (\boldsymbol{\Lambda}_{11} \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2)) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned}$$

Missing-data distribution

If we know the model parameters, we can use this result to determine the distribution of missing data. E.g., for our 2D example with $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}$, we obtain

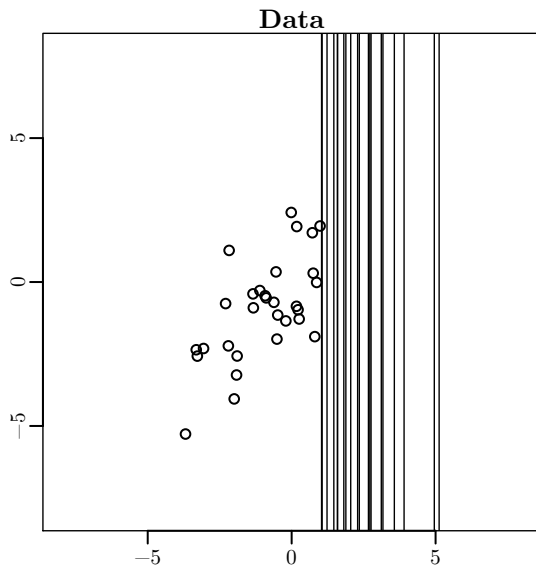
$$p(x_2|x_1) = \mathcal{N}(x_2|\mu_2 + \sigma_{21}\sigma_{11}^{-1}(x_1 - \mu_1), \sigma_{22} - \sigma_{21}\sigma_{11}^{-1}\sigma_{12}).$$



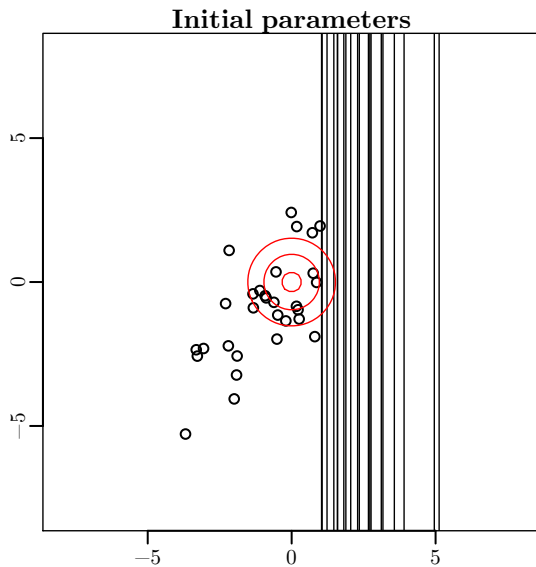
Intuition of EM algorithm

- Given the model parameters, we can determine the distribution of missing data
 - ▶ But we do not know the model parameters
- Given the distribution of missing data, we can determine the model parameters
 - ▶ But we do not know the distribution of missing data
- The **EM algorithm** exploits these observations
 1. Start with initial parameter estimate (e.g., random)
 2. Determine distribution of missing data based on current parameter estimate → E step
 3. Estimate parameters based on this distribution of missing data → M step
 4. Iterate steps 2 and 3 until a stopping criterion is satisfied

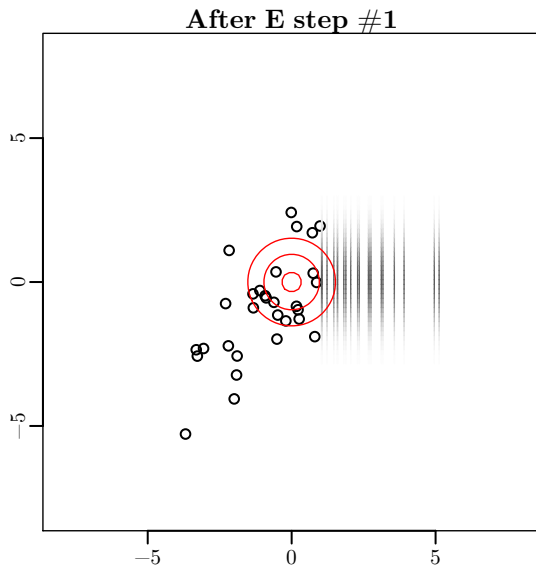
Example



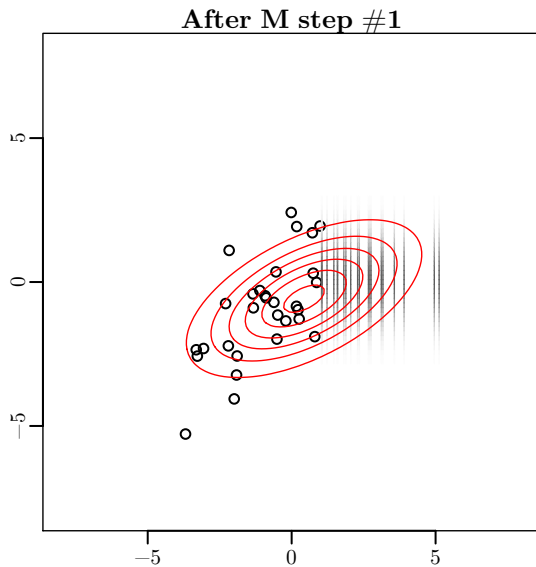
Example



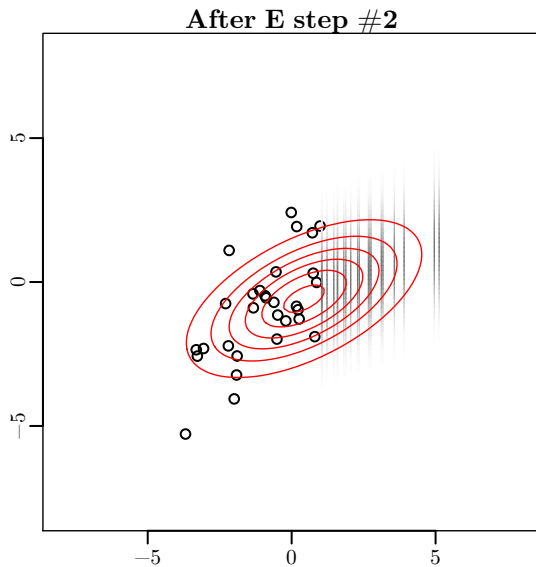
Example



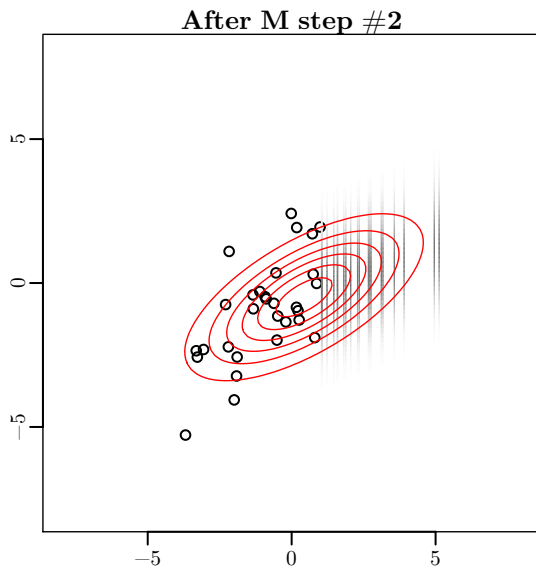
Example



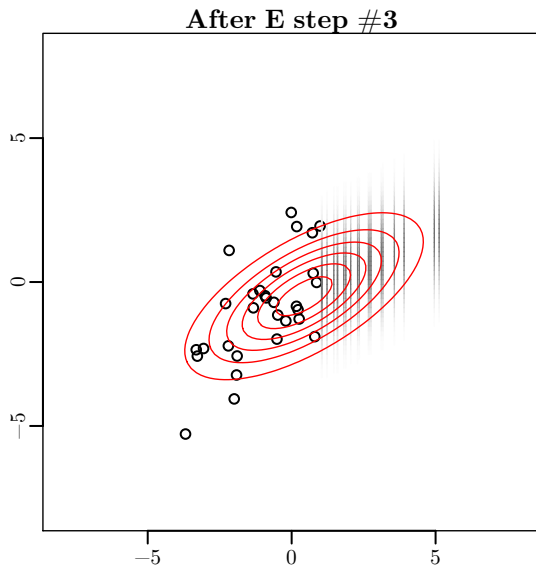
Example



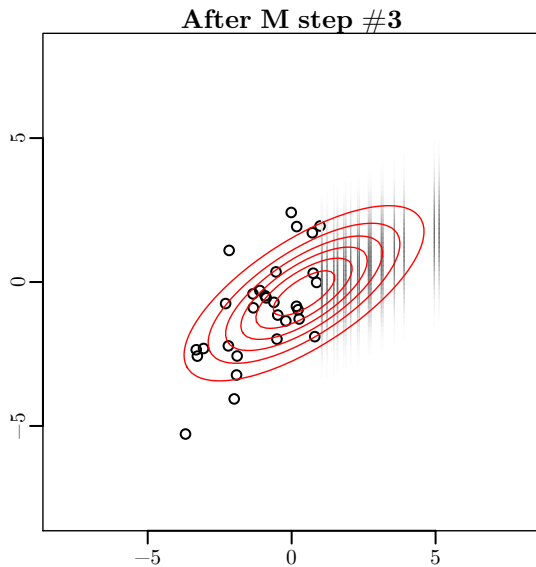
Example



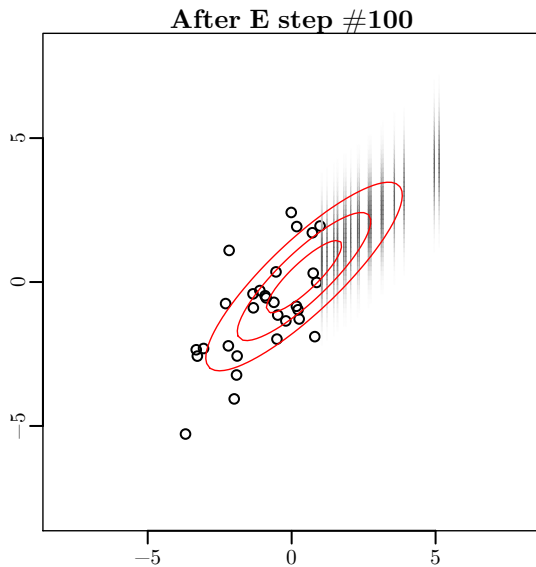
Example



Example



Example



Machine Learning

07 – EM Algorithm & Mixture Models

Part 2: The EM Algorithm

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-2

Terminology

- We split all available data into
 - ▶ $x =$ **observed data**
 - ▶ $z =$ **missing data**
 - ▶ $d = (x, z) =$ **complete data**
 - ▶ Generally, x , z , and d are sets of variables
- E.g., for iid data with missing values for each data point
 - ▶ $x = \{x_i\}_{i=1}^N =$ observed values for each example
 - ▶ $z = \{z_i\}_{i=1}^N =$ missing values for each example
 - ▶ $d = \{d_i\}_{i=1}^N$ with $d_i = (x_i, z_i)$
 - ▶ Set of observed and missing values may differ for each example
- Our focus
 - ▶ Given a generative model class $p(x, z|\theta)$,
 - ▶ Determine the observed-data ML estimate $\hat{\theta}_{\text{MLE}} = \operatorname{argmax}_{\theta} p(x|\theta)$
- EM also applicable (but not discussed here) for
 - ▶ MAP estimation $\hat{\theta}_{\text{MAP}} = \operatorname{argmax}_{\theta} p(x|\theta)p(\theta)$
 - ▶ Discriminative models of form $p(y, z|x)$ when explanatory variables x fully observed

Observed-data log-likelihood

- **Complete-data log-likelihood**

$$\ell_c(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \log p(\boldsymbol{x}, \boldsymbol{z} | \boldsymbol{\theta})$$

cannot be determined (and thus maximized), since \boldsymbol{z} unknown

- Instead, we maximize the **observed-data log-likelihood**

$$\ell_o(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \log p(\boldsymbol{x} | \boldsymbol{\theta}) = \log \int_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z} | \boldsymbol{\theta}) \, d\boldsymbol{z}$$

- But how? E.g., may use a gradient-based optimizer
 - ▶ Enforcing constraints on $\boldsymbol{\theta}$ can be tricky
 - ▶ Integral can be tricky to evaluate; e.g., $\boldsymbol{\theta}$ influences both distribution of missing values and likelihood of observed values

$$\ell_o(\boldsymbol{\theta}) = \log \int_{\boldsymbol{z}} p(\boldsymbol{x} | \boldsymbol{z}, \boldsymbol{\theta}) p(\boldsymbol{z} | \boldsymbol{\theta}) \, d\boldsymbol{z}$$

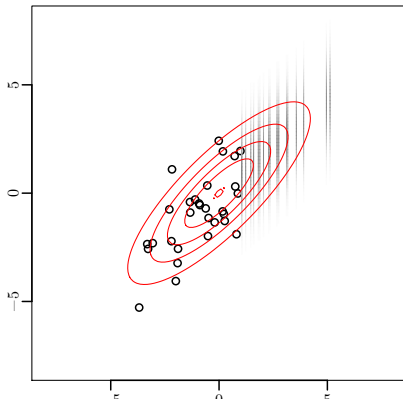
- EM is often much simpler (but not always faster)

Missing-data distribution

Given a parameter estimate $\theta^{(t)}$, we can determine the distribution of the missing data z

$$p(z|x, \theta^{(t)}) = \frac{p(x, z|\theta^{(t)})}{p(x|\theta^{(t)})}.$$

In the examples so far, we visualized $p(z|x, \theta^{(t)})$.



E step

- In the **E step**, the EM algorithm “computes” the **Q function** (*auxiliary function*)

$$\begin{aligned} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &\stackrel{\text{def}}{=} E_{z|\boldsymbol{x},\boldsymbol{\theta}^{(t)}}[\ell_c(\boldsymbol{\theta})] \\ &= \int_z \underbrace{p(z|\boldsymbol{x},\boldsymbol{\theta}^{(t)})}_{\text{depends on } \boldsymbol{\theta}^{(t)}} \underbrace{\log p(\boldsymbol{x}, z|\boldsymbol{\theta})}_{\text{depends on } \boldsymbol{\theta}} dz \end{aligned}$$

- ▶ $\boldsymbol{\theta}^{(t)}$ determines missing-data distribution (old parameters, fixed)
 - ▶ $\boldsymbol{\theta}$ determines complete-data log-likelihood (new parameters)
 - ▶ Q function corresponds to **expected complete-data log-likelihood** for a **fixed** missing-data distribution
- “Compute” means to determine quantities that can be used to evaluate $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ efficiently (with $\boldsymbol{\theta}^{(t)}$ fixed)
 - ▶ E.g., determine missing-data distribution $p(z|\boldsymbol{x},\boldsymbol{\theta}^{(t)})$
 - ▶ E.g., express Q function in closed form

M step

In the **M step**, the EM algorithm computes new parameter values by maximizing the Q function

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$$

After the M-step, the estimated missing-data distribution of z changes since the parameter changed. This is not accounted for in the Q function, thus: E step, M step, E step, M step, ...

Theorem

The EM algorithm monotonically increases the observed-data log-likelihood in that

$$\ell_o(\boldsymbol{\theta}^{(t+1)}) \geq \ell_o(\boldsymbol{\theta}^{(t)})$$

and

$$\ell_o(\boldsymbol{\theta}^{(t+1)}) > \ell_o(\boldsymbol{\theta}^{(t)}) \quad \text{if } \boldsymbol{\theta}^{(t)} \text{ is not a stationary point of } \ell_o.$$

Proof $\geq (1)$

1. Rewrite the observed data log-likelihood as

$$\ell_o(\boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta}) = \log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) - \log p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$$

2. Take conditional expectations on both sides

$$\begin{aligned}\ell_o(\boldsymbol{\theta}) &= E_{\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t)}}[\log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})] - E_{\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t)}}[\log p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})] \\ &= Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) + H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})\end{aligned}$$

- $H(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \stackrel{\text{def}}{=} E_{\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t)}}[-\log p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})]$ is **cross entropy**¹ of $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$ (new parameters) relative to $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t)})$ (old parameters, used in Q function)
- **Observed-data log-likelihood = Q function + cross entropy**

¹Note: We write $H(q|p) = E_p[-\log q]$ in this lecture for consistency with the Q function. In lecture 05-1, we wrote $H(p, q)$ instead.

Proof $\geq (2)$

3. Express improvement in terms of Q and H

$$\begin{aligned} \ell_o(\boldsymbol{\theta}^{(t+1)}) - \ell_o(\boldsymbol{\theta}^{(t)}) \\ = \underbrace{Q(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})}_{\geq 0 \text{ (M step)}} + \underbrace{H(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) - H(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)})}_{\geq 0 \text{ (see below)}} \end{aligned}$$

4. Q function non-decreasing since that is the objective of M step
5. Cross entropy also non-decreasing since $H(q|p) \geq H(p|p)$ (**Gibbs' inequality**). Step by step:

$$\begin{aligned} H(q|p) - H(p|p) &= E_p[-\log q/p] && \text{(Jensen's inequality)} \\ &\geq -\log E_p[q/p] && \text{(def. expectation)} \\ &= -\log \int p(\mathbf{z}) \frac{q(\mathbf{z})}{p(\mathbf{z})} d\mathbf{z} \\ &= -\log 1 = 0, \end{aligned}$$

where $p = p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t)})$ and $q = p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(t+1)})$

Discussion

- Convergence to local maximum or stationary point of $\ell_o(\theta)$
 - ▶ May not find global maximum
 - ▶ May run multiple times with different initializations (e.g., **random restarts**)
- In practice, use stopping criterion
 - ▶ E.g., small improvement in observed-data log-likelihood or Q function
 - ▶ E.g., small change of parameters
 - ▶ E.g., limit of total number of iterations or time budget
- Depending on model, E and M steps may still be complicated
- Many variants
 - ▶ Goals include better convergence, support for complex models, parallelizability
 - ▶ Generalized EM (don't maximize but improve Q)
 - ▶ Stochastic or batch EM (use subset of examples)
 - ▶ Monte-Carlo EM (approximate E step)
 - ▶ Hard EM (impute missing values)

Machine Learning

07 – EM Algorithm & Mixture Models

Part 3: Mixture Models

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Mixture models

- A **mixture model** (MM) is a latent variable model with a single categorical latent variable z per example \mathbf{x}

$$z \sim \text{Cat}(\boldsymbol{\pi})$$

$$\mathbf{x} \sim \text{some distribution } p(\mathbf{x}|z, \boldsymbol{\theta})$$

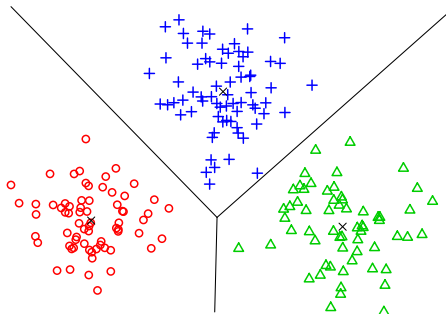
- Equivalently,

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K \pi_k p_k(\mathbf{x}|\boldsymbol{\theta}) \quad 0 \leq \pi_k \leq 1, \sum_k \pi_k = 1$$

- ▶ Data modeled as a mixture of K **base distributions** p_k (*mixture components*), where $p_k(\mathbf{x}|\boldsymbol{\theta}) \stackrel{\text{def}}{=} p(\mathbf{x}|z = k, \boldsymbol{\theta})$
- ▶ **Mixing weights** π_k , where $\pi_k = p(z = k|\boldsymbol{\theta})$
- ▶ Convex combination ($\boldsymbol{\pi} \in \mathcal{S}^K$)
- Interpretation: each data point generated from one base distribution \rightarrow latent variable z
- Main applications: density modeling, clustering

K -means clustering

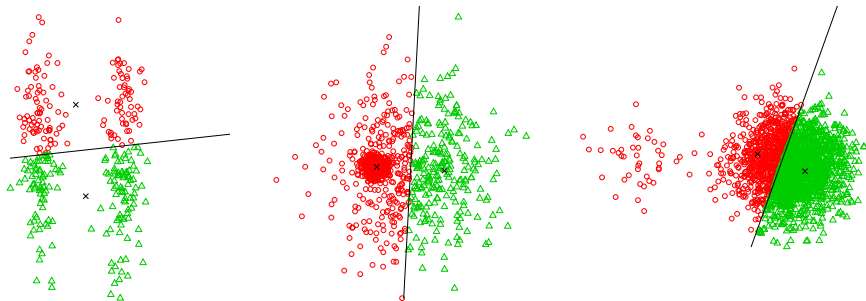
- K -means is perhaps most popular clustering objective
- Given a distance function, **partition** examples into K clusters
- Each cluster represented by a **centroid** (= cluster mean)
- Goal is to minimize sum of sq. distances between each data point and its closest centroid
- Space partitioned by Voronoi diagram of centroids
- We will see: closely related to (Gaussian) mixture models



Assumptions of K -means clustering

K -means clustering inherently assumes that

1. Clusters are spherical,
2. Clusters are non-overlapping,
3. Clusters have similar sizes.



Gaussian mixture models (GMM)

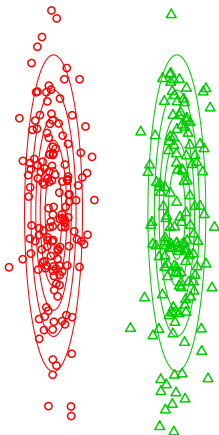
Gaussian mixture models (GMMs) are MM in which the base distributions are multivariate Gaussians.

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Generative model
- Note: $p(\mathbf{x}|\boldsymbol{\theta})$ is not Gaussian (e.g., may have multiple modes)
- Can be viewed as **soft clustering** variant of K -means
 - ▶ z corresponds to cluster identifier (as in K -means)
 - ▶ Data points within cluster k assumed to be Gaussian with “centroid” $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$
 - ▶ π_k models size of cluster (since $\pi_k = p(z = k|\boldsymbol{\pi})$)
 - ▶ Determine **cluster membership probabilities** $p(z = k|\mathbf{x}, \boldsymbol{\theta})$ to obtain soft clustering
 - ▶ Any MM can be interpreted as a soft clustering

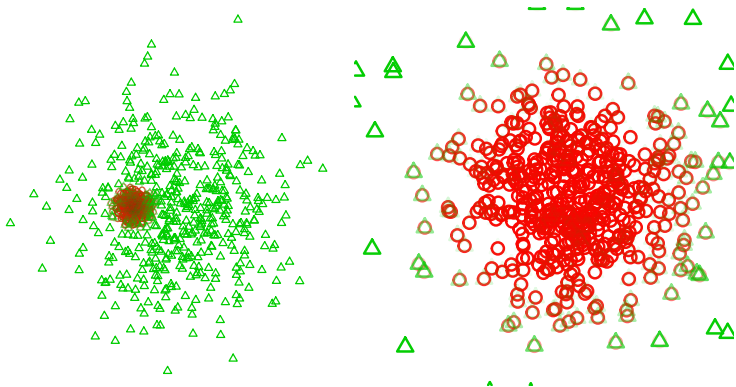
Example: Shape of clusters

- In K -means, shape of a cluster (ultimately) determined by centroids of other clusters
- In GMMs, shape of clusters is modeled explicitly



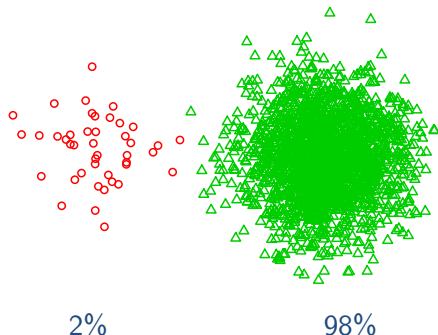
Example: Soft clustering

- In K -means, each data point is assigned to exactly one cluster (**hard clustering**)
 - ▶ Overlapping clusters cannot be handled appropriately
- In GMMs, cluster membership probabilities are explicitly modeled



Example: Cluster sizes

- In K -means, cluster sizes are not modeled
 - ▶ Large clusters contribute many “distances” to the objective, but small clusters just a few
 - ▶ This penalizes small clusters
- GMMs model cluster sizes explicitly



EM for GMMs: Likelihood, responsibility

- N independent data points $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$
- Model parameters $\boldsymbol{\theta} = \cup_k \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}$
- Likelihood of \mathbf{x}_i in k -th cluster:

$$f_k(\mathbf{x}_i) \stackrel{\text{def}}{=} p(\mathbf{x}_i | z_i = k, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Complete-data likelihood of (\mathbf{x}_i, z_i)

$$\begin{aligned} p(\mathbf{x}_i, z_i = k | \boldsymbol{\theta}) &= p(z_i = k | \boldsymbol{\theta}) p(\mathbf{x}_i | z_i = k, \boldsymbol{\theta}) \\ &= \pi_k f_k(\mathbf{x}_i) \end{aligned}$$

- **Cluster membership probabilities** of \mathbf{x}_i in k -th cluster

$$w_{ik} \stackrel{\text{def}}{=} p(z_i = k | \mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(\mathbf{x}_i, z_i = k | \boldsymbol{\theta})}{p(\mathbf{x}_i | \boldsymbol{\theta})} = \frac{\pi_k f_k(\mathbf{x}_i)}{\sum_{k'} \pi_{k'} f_{k'}(\mathbf{x}_i)}$$

- ▶ Also referred to as **responsibility** of the k -th mixture component for data point \mathbf{x}_i

EM for GMMs: E step

- Compute cluster membership probabilities $\{w_{ik}^{(t)}\}$ based on current parameters $\boldsymbol{\theta}^{(t)}$

$$w_{ik}^{(t)} \leftarrow p(z_i = k | \mathbf{x}_i, \boldsymbol{\theta}^{(t)})$$

- Let $\mathbf{z} = (z_1 \ \cdots \ z_N)^\top$. Q function is

$$\begin{aligned} Q(\boldsymbol{\theta} | \boldsymbol{\theta}^{(t)}) &= E_{\mathbf{z} | \mathbf{x}, \boldsymbol{\theta}^{(t)}} [\log p(\mathbf{X}, \mathbf{z} | \boldsymbol{\theta})] \\ &= \sum_i \sum_k w_{ik}^{(t)} \log [\pi_k f_k(\mathbf{x}_i)] \\ &= \sum_i \sum_k w_{ik}^{(t)} \log \pi_k + \sum_i \sum_k w_{ik}^{(t)} \log f_k(\mathbf{x}_i) \end{aligned}$$

- Q function uses “old” cluster memberships probabilities
→ That’s what we compute in the E step

EM for GMMs: M step

- M step now sets

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}} Q(\boldsymbol{\theta} | \boldsymbol{\theta}^{(t)})$$

- One can show that

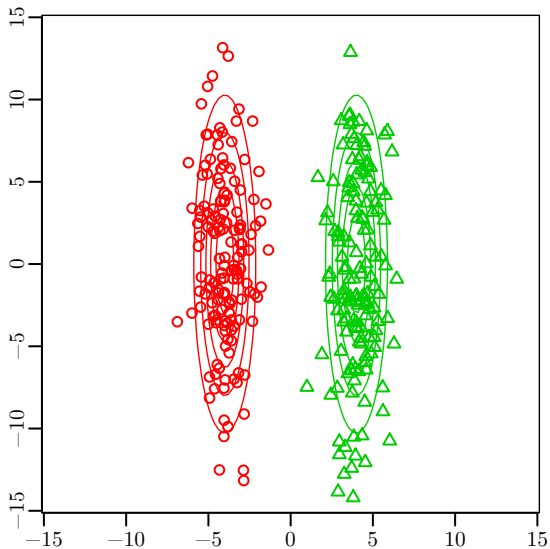
$$\pi_k^{(t+1)} \leftarrow \frac{1}{N} \sum_i w_{ik}^{(t)}$$

$$\boldsymbol{\mu}_k^{(t+1)} \leftarrow \frac{1}{\sum_i w_{ik}^{(t)}} \sum_i w_{ik}^{(t)} \mathbf{x}_i$$

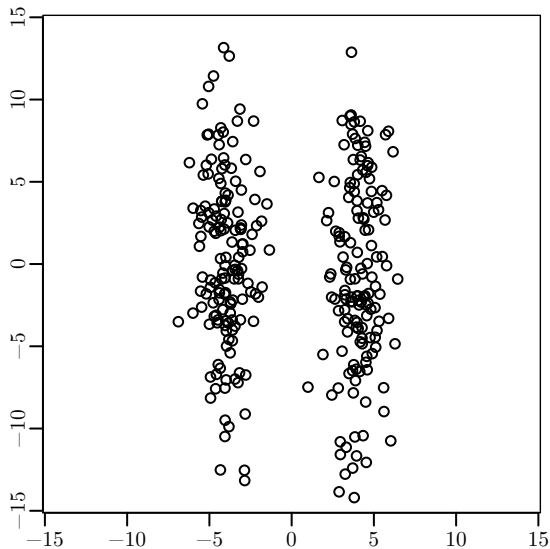
$$\boldsymbol{\Sigma}_k^{(t+1)} \leftarrow \frac{1}{\sum_i w_{ik}^{(t)}} \sum_i w_{ik}^{(t)} (\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^\top$$

- Observe: Similar to ML estimates for MVNs (07-1- slide 4), but data points are now weighted by cluster membership probabilities

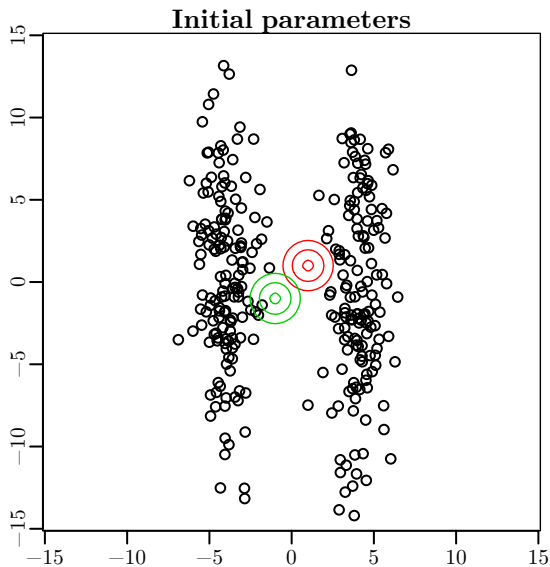
Example run of EM algorithm



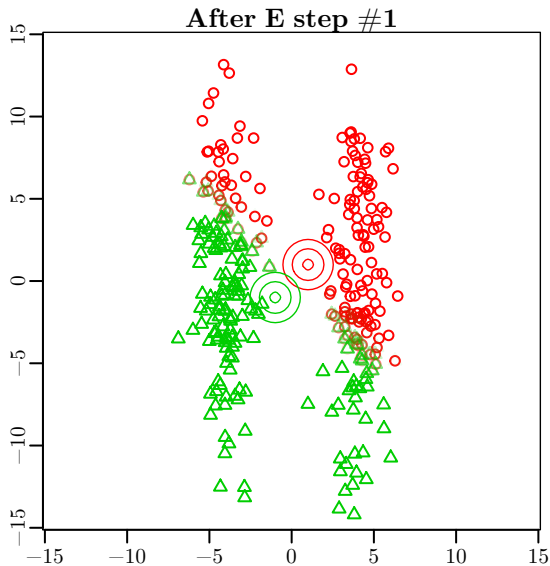
Example run of EM algorithm



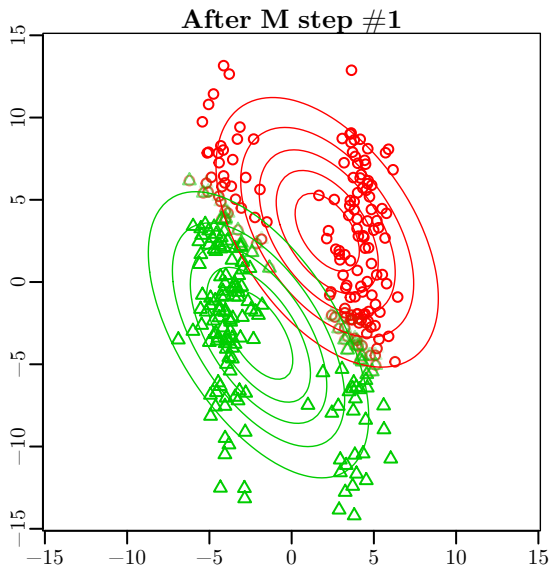
Example run of EM algorithm



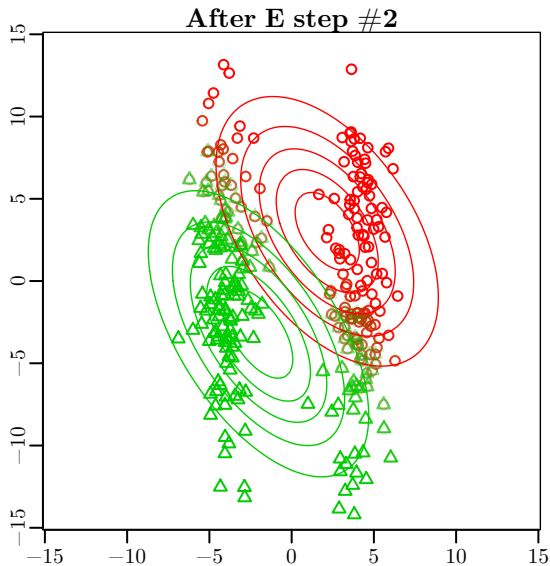
Example run of EM algorithm



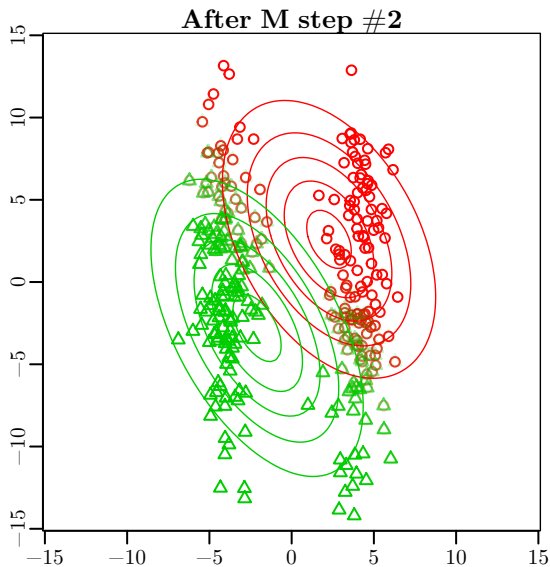
Example run of EM algorithm



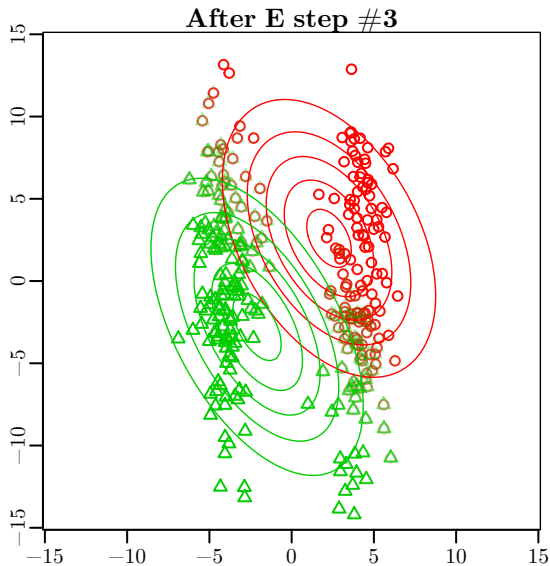
Example run of EM algorithm



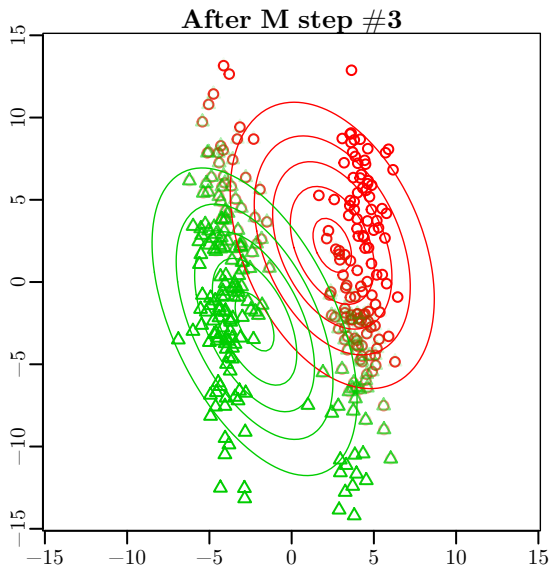
Example run of EM algorithm



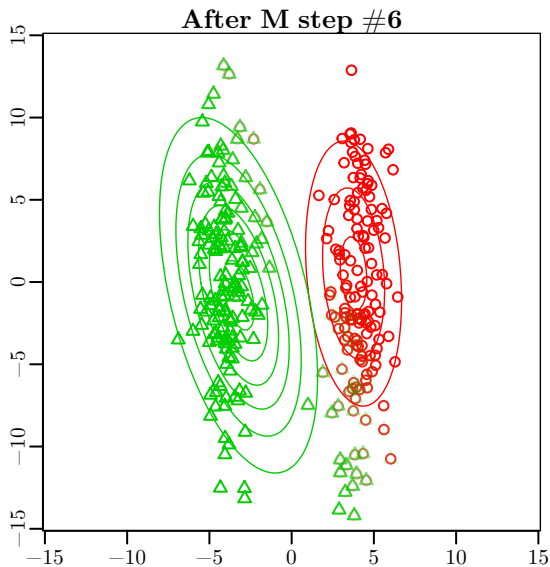
Example run of EM algorithm



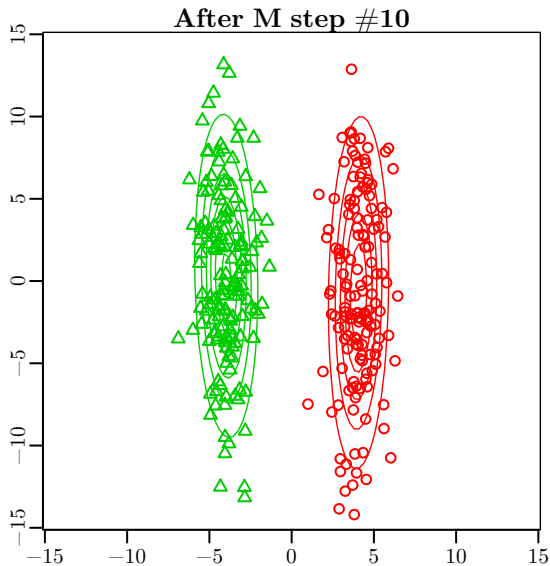
Example run of EM algorithm



Example run of EM algorithm



Example run of EM algorithm



GMMs and K -means

- **Lloyd's algorithm** is popular method for K -means clustering
 1. Start with initial centroids
(e.g., random, farthest point clustering / K -means++)
 2. Assign each data point to its closest centroid
 3. Set each centroid to the mean of the data points assigned to it
 4. Repeat steps 2+3 until some stopping criterion is met
- Equivalent to certain GMM with *hard EM*; i.e.,
 - ▶ Use (a priori) equally-likely clusters ($\pi_k = \frac{1}{K}$)
 - ▶ Use spherical Gaussians ($\Sigma_k = \mathbf{I}$)
 - ▶ Use hard cluster assignments in E step
($w_{ik} = 1$ for most likely cluster; zero for all other clusters)
 - ▶ Only centroids μ_k need to be computed in M step

Discussion

- EM for GMMs determines cluster membership probabilities, but no hard clustering
 - ▶ At the very end, may assign each point to its most likely cluster
- Large number of parameters when data high-dimensional
 - ▶ Covariance matrices have $O(D^2)$ parameters
→ $O(KD^2)$ parameters in total
 - ▶ Can be reduced, for example, by using diagonal covariances matrices or by combination with factor analysis
 - ▶ Overfitting (e.g., singularities) may arise → use MAP estimation
- Discriminative variant of MMs called **mixture of experts**

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \sum_k p(z = k|\mathbf{x}, \boldsymbol{\theta})p(y|\mathbf{x}, z = k, \boldsymbol{\theta})$$

- ▶ Each component model considered an **expert**
- ▶ **Gating function** $p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$ decides which expert to use

Example: Mixture of experts

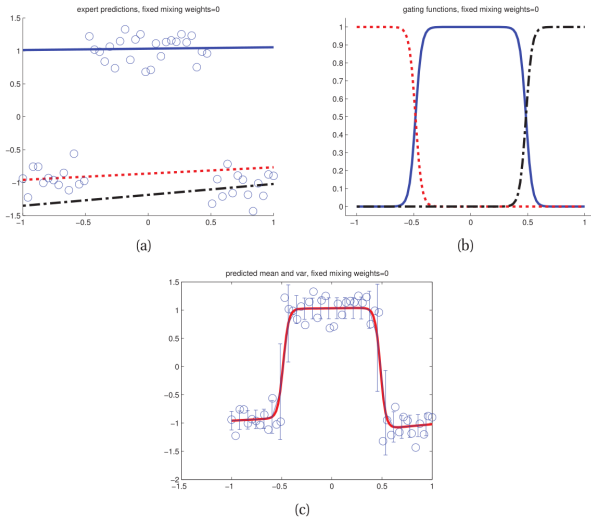


Figure 11.6 (a) Some data fit with three separate regression lines. (b) Gating functions for three different “experts”. (c) The conditionally weighted average of the three expert predictions. Figure generated by `mixexpDemo`.

Some other LVMs

$p(\mathbf{x}_i \mathbf{z}_i)$	$p(\mathbf{z}_i)$	Name	Section
MVN	Discrete	Mixture of Gaussians	11.2.1
Prod. Discrete	Discrete	Mixture of multinomials	11.2.2
Prod. Gaussian	Prod. Gaussian	Factor analysis/ probabilistic PCA	12.1.5
Prod. Gaussian	Prod. Laplace	Probabilistic ICA/ sparse coding	12.6
Prod. Discrete	Prod. Gaussian	Multinomial PCA	27.2.3
Prod. Discrete	Dirichlet	Latent Dirichlet allocation	27.3
Prod. Noisy-OR	Prod. Bernoulli	BN20/ QMR	10.2.3
Prod. Bernoulli	Prod. Bernoulli	Sigmoid belief net	27.7

Table 11.1 Summary of some popular directed latent variable models. Here “Prod” means product, so “Prod. Discrete” in the likelihood means a factored distribution of the form $\prod_j \text{Cat}(x_{ij}|\mathbf{z}_i)$, and “Prod. Gaussian” means a factored distribution of the form $\prod_j \mathcal{N}(x_{ij}|\mathbf{z}_i)$. “PCA” stands for “principal components analysis”. “ICA” stands for “independent components analysis”.

Machine Learning

08 – Kernels and Vector Machines

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

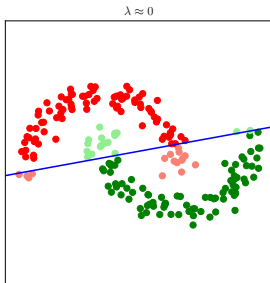
Version: 2023-1

Overview

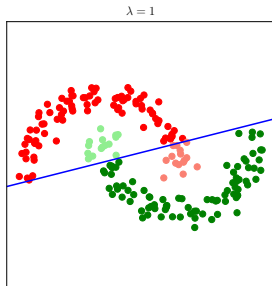
- So far: inputs represented by a fixed-size feature vector $\mathbf{x}_i \in \mathbb{R}^D$
- For some inputs, not clear how to do this
 - E.g., sequences of variable length (such as text documents or protein sequences)
 - E.g., graphs (such as molecular structure)
- Kernel approach: from features to similarity
 - Kernel approaches make use of a **kernel function** $\kappa(\mathbf{x}, \mathbf{x}')$ that can be interpreted as “similarity” between objects \mathbf{x} and \mathbf{x}'
 - Supervised learning: model output based on similar inputs
 - Unsupervised learning: use custom notions of similarity (e.g., for clustering)
 - Also useful when data represented as fixed-size feature vectors
- **Kernel trick** = modify learning algorithms to work solely with kernel functions → can use arbitrary inputs
 - E.g., linear regression, logistic regression, support vector machines, K NN, K -medoids, PCA, ...

Example: Logistic regression

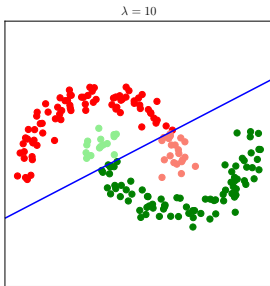
Very high complexity



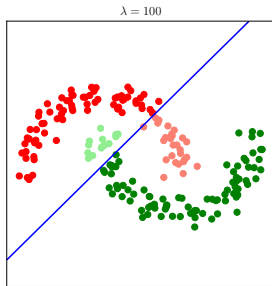
High complexity



Medium complexity

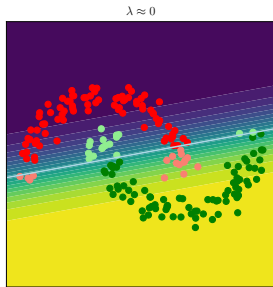


Low complexity

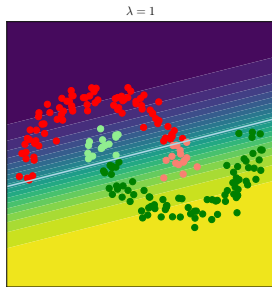


Example: Logistic regression

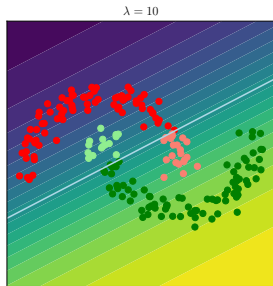
Very high complexity



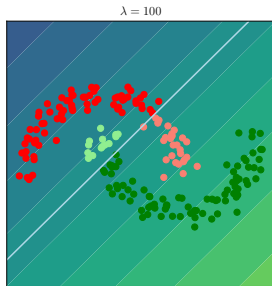
High complexity



Medium complexity

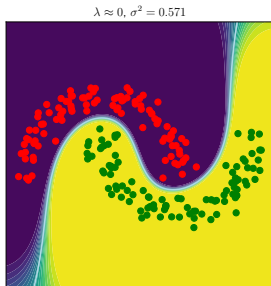


Low complexity

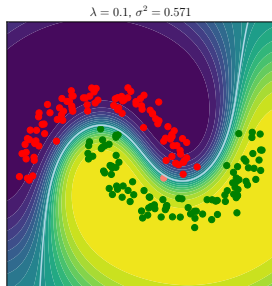


Example: Logistic regression (Gaussian kernel)

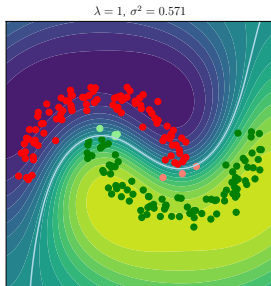
Very high complexity



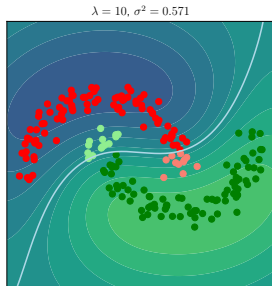
High complexity



Medium complexity

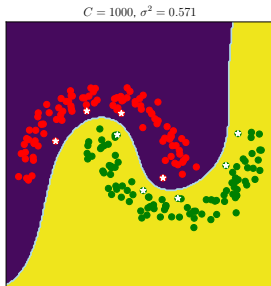


Low complexity

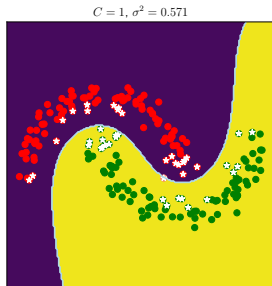


Example: Support vector machine (Gaussian kernel)

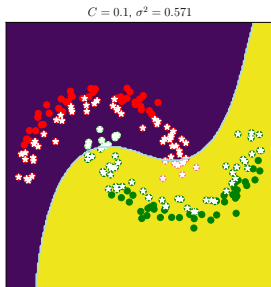
Very high complexity



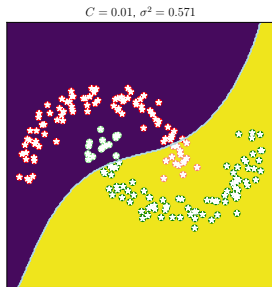
High complexity



Medium complexity



Low complexity



Outline (Kernels and Vector Machines)

1. Kernels
2. Kernel Machines and Vector Machines
3. The Kernel Trick
4. Sparse Vector Machines

Summary

- Kernel approaches make use of a **kernel function** $\kappa(\mathbf{x}, \mathbf{x}')$
 - Can be interpreted as “similarity” between objects \mathbf{x} and \mathbf{x}'
 - Allows to use structured objects
 - Can help to prevent underfitting (e.g., of a linear model)
- Kernel machines use similarity to K prototypes as features
- Vector machines use similarity to other data points as features
- Kernel trick: directly work in kernel's feature space
 - Mercer kernels \equiv inner product in kernel feature space
 - Modify learning algorithms to work solely with kernel function calls
 - Many methods have been be kernelized: e.g., linear regression, logistic regression, SVM, K NN, K -Means, PCA, ...
- Sparsity via prior/regularization (L1VM) or objective (SVM)
 - Reduces prediction costs and (hopefully) generalization error

Literature

- Murphy, Ch. 17, *Kernel Methods*
- John Shawe-Taylor, Nello Cristianini
Kernel Methods for Pattern Analysis
Cambridge University Press, 2004

Machine Learning

08 – Kernels and Vector Machines

Part 1: Kernels

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Kernel functions

- A **kernel function** is a real-valued function $\kappa(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$
 - ▶ Inputs $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ from some (arbitrary) input space
 - ▶ Typically symmetric: $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$
 - ▶ Sometimes non-negative: $\kappa(\mathbf{x}, \mathbf{x}') \geq 0$
 - ▶ Then: may be interpreted as measure of similarity
- Sometimes: term *kernel* used to refer to **Mercer kernels**
 - ▶ Positive semi-definite kernels
 - ▶ Can be expressed using an inner product on “transformed” inputs
 - ▶ Kernelization of learning algorithms using the kernel trick requires kernel function to be a Mercer kernels
- Examples: Gaussian kernel, linear kernel, polynomial kernel, cosine similarity kernel, string kernels, graph kernels, Laplacian kernel, probability product kernel, Fisher kernel, ...

Gaussian kernel (1)

- **Gaussian kernel** given by

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{x}') \right)$$

- ▶ Takes values in $[0, 1]$; in particular, $\kappa(\mathbf{x}, \mathbf{x}) = 1$
- ▶ Decreases exponentially in sq. **Mahalanobis distance**

- Special case: diagonal covariance

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{1}{2} \sum_{j=1}^D \frac{1}{\sigma_j^2} (x_j - x'_j)^2 \right)$$

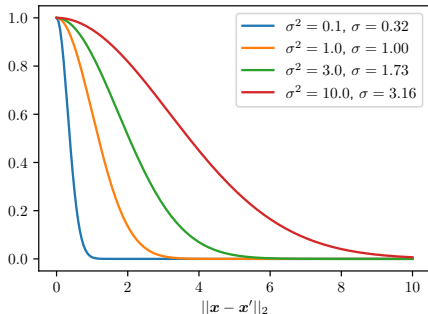
- ▶ σ_j can be interpreted as **length scale** of dimension j
- ▶ When $\sigma_j \rightarrow \infty$, dimension j is ignored
- ▶ Known as **ARD kernel** or **squared exponential kernel**

Gaussian kernel (2)

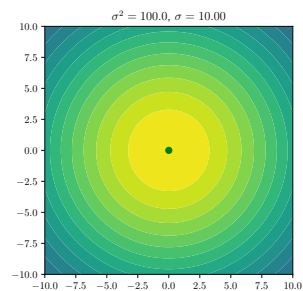
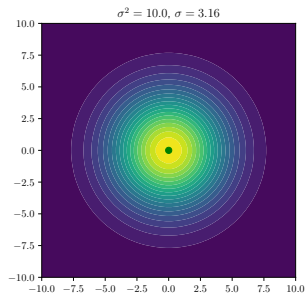
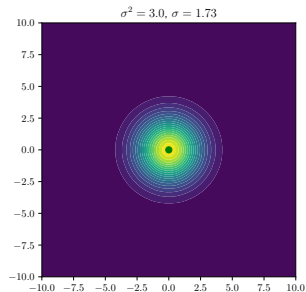
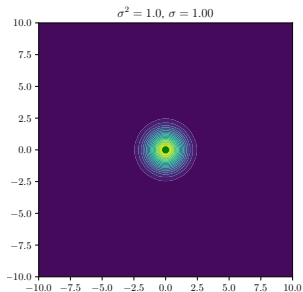
- Special case: isotropic Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{x}'\|_2^2\right)$$

- ▶ σ^2 known as **bandwidth**
- ▶ Example of an **radial basis function (RBF)** kernel: only a function of $\|\mathbf{x} - \mathbf{x}'\|$
- ▶ Rough interpretation: objects with $\|\mathbf{x} - \mathbf{x}'\|_2 \geq 3\sigma$ are dissimilar



Gaussian kernel (3)



Some other kernels

- Linear kernel: $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- Linear kernel with basis function expansion: $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$
- Polynomial kernel: $\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^\top \mathbf{x}' + r)^M$ with $\gamma, r \geq 0$
 - ▶ Corresponds to (certain) basis function expansion with all interaction terms up to degree M (when $\gamma, r \neq 0$; see slide 8)
- Cosine similarity kernel: $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' / (\|\mathbf{x}\| \|\mathbf{x}'\|)$
 - ▶ E.g., useful for comparing documents (\mathbf{x} is TF-IDF representation)
- String kernel: $\kappa(\mathbf{x}, \mathbf{x}') = \sum_{s \in \mathcal{A}^*} w_s n_s(\mathbf{x}) n_s(\mathbf{x}')$
 - ▶ \mathbf{x} and \mathbf{x}' are strings over alphabet \mathcal{A}
 - ▶ $n_s(\mathbf{x})$ is number of times substring s occurs in \mathbf{x}
 - ▶ E.g., large when many common substring (of high weight)
 - ▶ E.g., zero when no common substring (of nonzero weight)
 - ▶ Special cases: bag-of-characters kernel ($w_s = 0$ when $|s| \neq 1$), k -spectrum kernel ($w_s = 0$ when $|s| \neq k$), bag-of-words kernel ($w_s = 0$ if s not bordered by word boundaries)

Mercer kernel

- A kernel is a **Mercer kernel** if the **Gram matrix** (*kernel matrix*)

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \cdots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

is positive semi-definite (psd) for all inputs $\{\mathbf{x}_i\}_{i=1}^N$

- Recall: $\mathbf{A} \in \mathbb{R}^{n \times n}$ is psd iff symmetric and $\mathbf{v}^\top \mathbf{A} \mathbf{v} \geq 0$ for all $\mathbf{v} \in \mathbb{R}^n$
- Implies that eigendecomposition exists and all $\lambda_i \geq 0$:

$$\mathbf{K} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^\top = (\mathbf{Q} \mathbf{\Lambda}^{1/2})(\mathbf{Q} \mathbf{\Lambda}^{1/2})^\top$$

- ▶ Set $\phi(\mathbf{x}_i) = [\mathbf{Q} \mathbf{\Lambda}^{1/2}]_{i:}^\top$
- ▶ Then $\kappa_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$
- **Mercer's theorem**: κ is psd iff. there exists a “feature map” $\phi_\kappa : \mathcal{X} \rightarrow \mathbb{R}^D$ s.t. $\kappa(\mathbf{x}, \mathbf{x}') = \phi_\kappa(\mathbf{x})^\top \phi_\kappa(\mathbf{x}')$ for *all* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$
 - ▶ D may be a finite (e.g., polynomial kernel) or infinite (e.g., Gaussian kernel)

Example: Polynomial kernel

- Consider the polynomial kernel $\kappa(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^2$
- For each $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$, we have

$$\begin{aligned}(1 + \mathbf{x}^\top \mathbf{x}')^2 &= (1 + x_1x'_1 + x_2x'_2)^2 \\ &= 1 + 2x_1x'_1 + 2x_2x'_2 + (x_1x'_1)^2 + (x_2x'_2)^2 + 2x_1x'_1x_2x'_2\end{aligned}$$

- Can be written as $\phi_\kappa(\mathbf{x})^\top \phi_\kappa(\mathbf{x}')$ with

$$\phi_\kappa(\mathbf{x}) = \begin{pmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & x_2^2 & \sqrt{2}x_1x_2 \end{pmatrix}^\top \in \mathbb{R}^D$$

for $D = 6$

- Generally, feature map of polynomial kernel contains all interaction terms up to degree M
- We will see: using this kernel is equivalent to working in the above 6-dimensional feature space
 - ▶ With kernel trick, only the kernel function is evaluated, however
 - ▶ For this reason, we can handle infinite-dimensional features spaces (such as the one of the Gaussian kernel)

Discussion

- Mercer kernels determine an implicit feature map
 - ▶ $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ for some ϕ
 - ▶ Thus: κ computes inner product of mapped features
 - ▶ But: κ is not an inner product on \mathcal{X}
(e.g., $\kappa(a\mathbf{x}, \mathbf{x}')$ may not be equal to $a\kappa(\mathbf{x}, \mathbf{x}')$)
- Showing that a kernel function is Mercer can be complicated
 - ▶ If $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ for some ϕ , then κ is Mercer
 - ▶ $\kappa(\mathbf{x}, \mathbf{x}') = c$ is a Mercer kernel for any $c \in \mathbb{R}_{\geq 0}$
 - ▶ Sums and products of Mercer kernels are Mercer kernels, as is multiplication by non-negative scalar and exponentiation of a Mercer kernel
 - ▶ Non-Mercer kernels can also be useful (e.g., in kernel machines)
- Kernel matrix can be large: N^2 entries

Machine Learning

08 – Kernels and Vector Machines

Part 2: Kernel Machines and Vector Machines

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Kernel machines

- A **kernel machine** is a generalized linear model which uses a **kernelized feature vector** for its inputs:

$$\phi_{\text{km}}(\mathbf{x}) = \left(\kappa(\mathbf{x}, \boldsymbol{\mu}_1) \quad \kappa(\mathbf{x}, \boldsymbol{\mu}_2) \quad \dots \quad \kappa(\mathbf{x}, \boldsymbol{\mu}_K) \right)^\top$$

- ▶ The $\boldsymbol{\mu}_k \in \mathcal{X}$ are a set of K **centroids** (or *prototypes*)
- ▶ If κ is an RBF kernel, called **RBF network**
- Linear predictor specified by weight vector $\mathbf{w} \in \mathbb{R}^K$ predictor

$$\eta(\mathbf{x}) = \mathbf{w}^\top \phi_{\text{km}}(\mathbf{x}) = \sum_k w_k \kappa(\mathbf{x}, \boldsymbol{\mu}_k)$$

- ▶ Consider a centroid $\boldsymbol{\mu}_k$ and suppose κ measures similarity
- ▶ Contribution the larger the more similar \mathbf{x} is to $\boldsymbol{\mu}_k$
- ▶ Contribution the larger the higher the centroid's weight $|w_k|$
- ▶ Contribution can be positive ($w_k > 0$) or negative ($w_k < 0$)
- Examples
 - ▶ Linear regression: $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{w}^\top \phi_{\text{km}}(\mathbf{x}), \sigma^2)$
 - ▶ Logistic regression: $p(y|\mathbf{x}, \mathbf{w}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \phi_{\text{km}}(\mathbf{x})))$

Example: Linear regression, RBF centroids

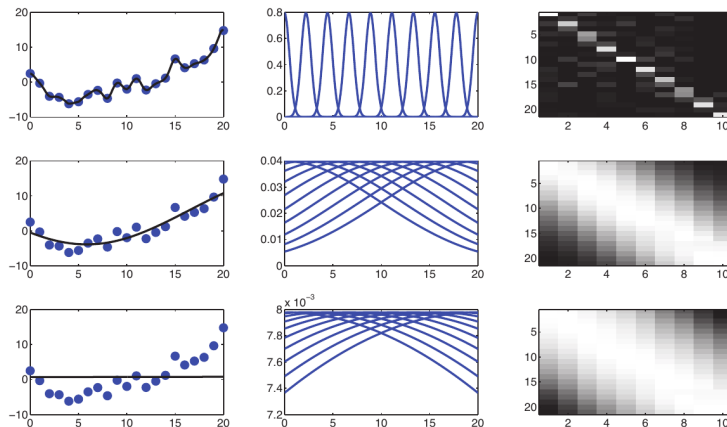
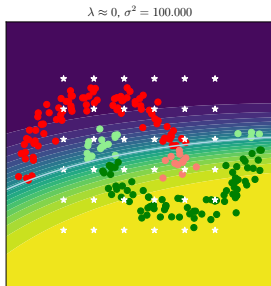


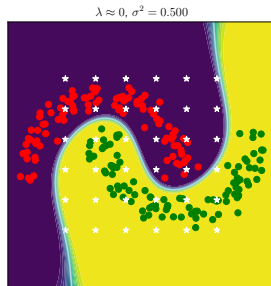
Figure 14.3 RBF basis in 1d. Left column: fitted function. Middle column: basis functions evaluated on a grid. Right column: design matrix. Top to bottom we show different bandwidths: $\tau = 0.1$, $\tau = 0.5$, $\tau = 50$. Figure generated by `linregRbfDemo`.

Example: Logistic regression, RBF centroids (1)

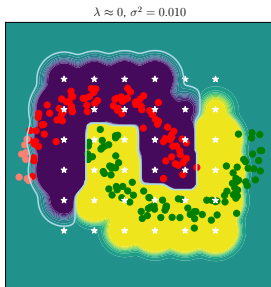
Large bandwidth



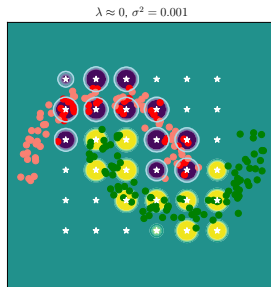
Medium bandwidth



Small bandwidth

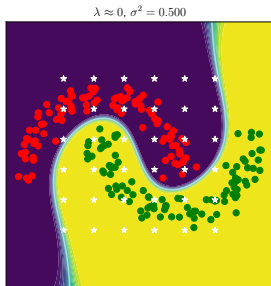


Tiny bandwidth

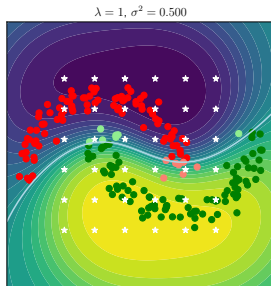


Example: Logistic regression, RBF centroids (2)

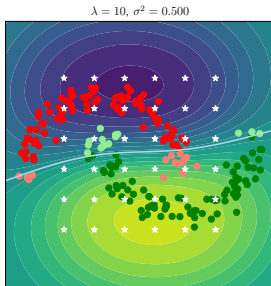
Very high complexity



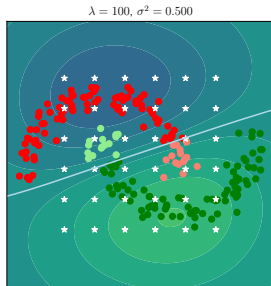
High complexity



Medium complexity



Low complexity



Which centroids?

- On low-dimensional inputs, may use a grid
 - But: breaks down on high dimensionality (curse of dimensionality)
- If $\mu_k \in \mathbb{R}^D$, may use numerical optimization
 - But: kernels are most useful for structured input spaces
- Cluster the data and use cluster centers?
 - Cluster centers are generally dense regions in input space, but may not be most useful for predicting outputs

Vector machines

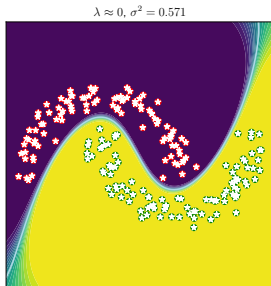
- **Vector machines** use the data points as centroids

$$\eta(\mathbf{x}) = \boldsymbol{\alpha}^\top \boldsymbol{\phi}_{\text{vm}}(\mathbf{x}) = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

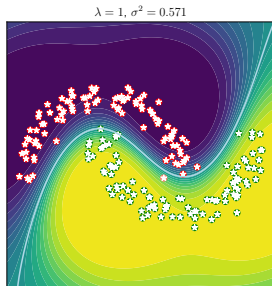
- Model is parameterized by a weight α_i for each data point
 - ▶ To predict, need access to all data points \mathbf{x}_i (assuming all $\alpha_i \neq 0$)
 - ▶ N weights, N data points, $O(N)$ cost to predict
 - ▶ Expensive for large datasets
- Various approaches exist to select a subset of the data points
 - ▶ These methods “encourage” choices of $\alpha_i = 0$
 - Corresponding data points ignored
 - ▶ Can use a sparsity-promoting prior; e.g., ℓ_1 (L1VM)
 - ▶ Can use customized loss function → support vector machines (SVM)
 - ▶ Pro: lower computational cost, lower space consumption
 - ▶ Cons: also affects model complexity (and thus generalization performance)

Example: Logistic regression, RBF centroids, L2

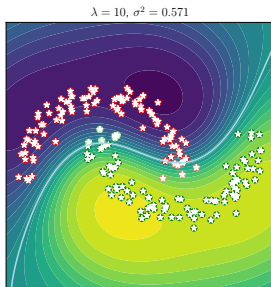
Very high complexity



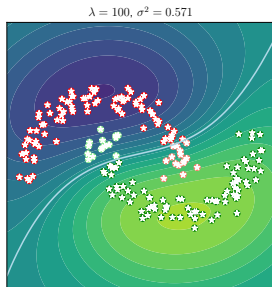
High complexity



Medium complexity

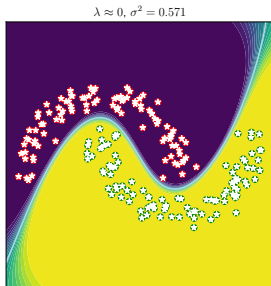


Low complexity

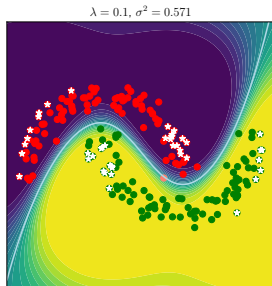


Example: Logistic regression, RBF centroids, L1

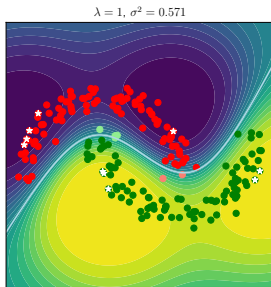
Very high complexity



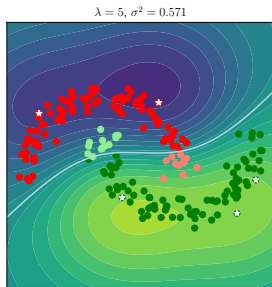
High complexity



Medium complexity

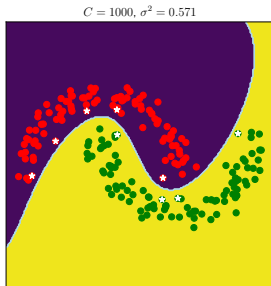


Low complexity

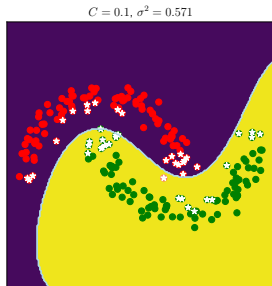


Example: Linear SVM, RBF centroids

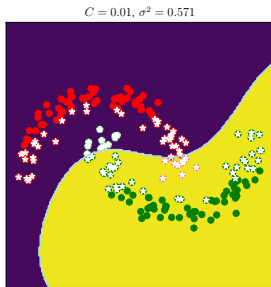
Very high complexity



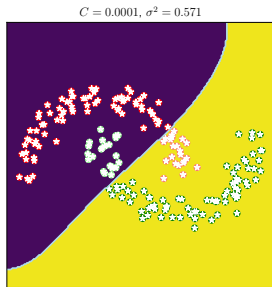
High complexity



Medium complexity

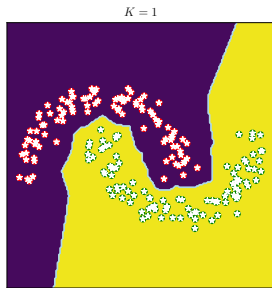


Low complexity

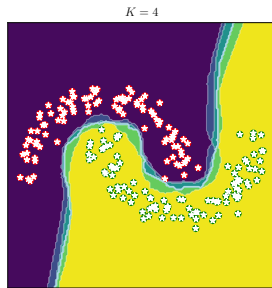


Example: KNN (for reference)

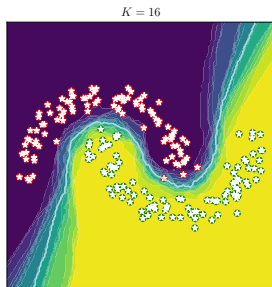
Very high complexity



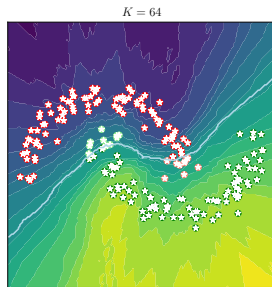
High complexity



Medium complexity



Low complexity



Machine Learning

08 – Kernels and Vector Machines

Part 3: The Kernel Trick

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Kernel trick

- Rather than working on kernelized feature vectors

$$\phi_{\text{vm}}(\mathbf{x}) = \left(\kappa(\mathbf{x}, \mathbf{x}_1) \quad \kappa(\mathbf{x}, \mathbf{x}_2) \quad \dots \quad \kappa(\mathbf{x}, \mathbf{x}_N) \right)^\top,$$

the kernel trick modifies the learning algorithm directly

- Key idea of the **kernel trick**
 - ▶ Express the learning algorithm in a way that accesses the data *only* in terms of inner products of form $\langle \mathbf{x}, \mathbf{x}' \rangle$
 - ▶ Then replace all inner products of form $\langle \mathbf{x}, \mathbf{x}' \rangle$ by calls to the kernel function $\kappa(\mathbf{x}, \mathbf{x}')$
- Discussion
 - ▶ Requires the kernel to be a Mercer kernel
 - ▶ Then approach is equivalent to working on the implicit feature representation ϕ_κ corresponding to κ

Recall: K -nearest neighbor classifier (KNN)

- Simple, non-parametric classifier
- Uses statistics about neighbors $N_K(\mathbf{x}, \mathcal{D})$, i.e., the K training points closest to classify test input \mathbf{x} :

$$p(y = c | \mathbf{x}, \mathcal{D}, K) = \frac{1}{K} \sum_{i \in N_K(\mathbf{x}, \mathcal{D})} \mathbb{I}(y_i = c),$$

where $\mathbb{I}(e)$ is the indicator function

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

- Discussion
 - ▶ Makes probabilistic predictions
 - ▶ Example of **memory-based learning**
 - ▶ Key assumption: close points have similar labels
 - ▶ Requires a suitable distance function and sufficient data

Kernelized KNN

- KNN accesses the training data to obtain the K -closest neighbors $N_K(\mathbf{x}, \mathcal{D})$ of \mathbf{x} in \mathcal{D}
- With squared Euclidean distance, we have

$$\begin{aligned}d(\mathbf{x}_i, \mathbf{x}) &= \|\mathbf{x}_i - \mathbf{x}\|_2^2 = \langle \mathbf{x}_i - \mathbf{x}, \mathbf{x}_i - \mathbf{x} \rangle \\ &= \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}_i, \mathbf{x} \rangle\end{aligned}$$

- After replacing inner products by kernel function calls, we obtain

$$d_\kappa(\mathbf{x}_i, \mathbf{x}) = \kappa(\mathbf{x}_i, \mathbf{x}_i) + \kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}_i, \mathbf{x})$$

- For example, with an Gaussian RBF kernel, this simplifies to

$$d_{\text{rbf}}(\mathbf{x}_i, \mathbf{x}) = 2 - 2\kappa(\mathbf{x}_i, \mathbf{x})$$

- ▶ Distance d_{rbf} the smaller the more similar \mathbf{x}_i and \mathbf{x} are
- ▶ For this choice of kernel, we obtain the same neighbors as KNN and hence the same classifier
- Key advantage: can use KNN on structured objects

Kernelized ridge regression (1)

- Kernel trick harder to apply to parametric models
- Here we outline how to kernelize **ridge regression** (= linear regression with ℓ_2 regularization)
- Regularized risk formulation

$$R'_{\text{emp}}(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

- Can show: optimal solution given by

$$\mathbf{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

- ▶ Since $[\mathbf{X}\mathbf{X}^\top]_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, we can kernelize this subexpression by using the kernel matrix \mathbf{K} instead (recall: $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$)
- ▶ But what about the leading \mathbf{X}^\top term?

Kernelized ridge regression (2)

- So far, we have

$$\mathbf{w} = \mathbf{X}^\top (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

- ▶ Called **primal variables** (one per feature)

- Define

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

- ▶ Called **dual variables** (one per example)
- ▶ Can be computed solely via kernel calls (= is kernelized)

- Rewrite the primal variables using the dual ones

$$\mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{i=1}^N \alpha_i \mathbf{x}_i$$

- ▶ Optimal weight vector is thus linear combination of the data points
- ▶ Cannot compute \mathbf{w} without accessing \mathbf{X} though (not kernelized)

Kernelized ridge regression (3)

- Let's plugin $\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i$ to predict

$$\hat{f}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \left(\sum_{i=1}^N \alpha_i \mathbf{x}_i^\top \right) \mathbf{x} = \sum_{i=1}^N \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle$$

- ▶ This we can kernelize!

- Final predictor is

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^N \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

- ▶ That's the predictor we have seen on slide 07-2/??
- ▶ But: with kernel trick, ℓ_2 regularization is applied to \mathbf{w} (not $\boldsymbol{\alpha}$)
 - ▶ \mathbf{w} = implicit weight vector for transformed features ($\phi_\kappa(\mathbf{x})$)
 - ▶ $\boldsymbol{\alpha}$ = explicit weight vector for data points ($\phi_{\text{vm}}(\mathbf{x})$)

Machine Learning

08 – Kernels and Vector Machines

Part 4: Sparse Vector Machines

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Sparse Vector Machines

- Recall: vector machines (VM) use linear predictor

$$\eta(\mathbf{x}) = w_0 + \sum_{i=1}^N \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

- Sparse VM:** few α_i 's non-zero \rightarrow few data points matter
- Sparsity reduces overfitting and computational cost
- Important esp. when N is large
- Non-kernelized: explicitly use features $\phi_{\text{vm}}(\mathbf{x}) = (\kappa(\mathbf{x}_i, \mathbf{x}))_{i=1}^N$
 - Directly learn α , can be used with any kernel
 - Feature space determined by training data
 - Regularization/prior on α
 - Sparsity achievable via feature selection / prior / regularization
- Kernelized: implicitly use transformed features $\phi_{\kappa}(\mathbf{x})$
 - Use Mercer kernel and apply kernel trick
 - Feature space determined by kernel κ
 - Regularization/prior on (implicit) weights for transformed features
 - Sparsity achievable via modified loss \rightarrow *support vector machine*

Sparsity via ℓ_0 regularization

- In the regularized risk minimization framework, we may directly encourage sparsity using the ℓ_0 pseudo-norm

$$J_0(\boldsymbol{\theta}) = R_{\text{emp}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_0$$

- ▶ Recall: $\|\boldsymbol{\theta}\|_0 = \text{no. nonzero entries in } \boldsymbol{\theta}$
- Penalty $\|\boldsymbol{\theta}\|_0$ corresponds to number of relevant parameters
→ variable selection
- Coefficient λ trades off fit (λ small) and sparsity (λ large)
- Hard optimization problem

Sparsity via ℓ_1 regularization

- We may use the ℓ_1 **regularization** instead

$$J_1(\boldsymbol{\theta}) = R_{\text{emp}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

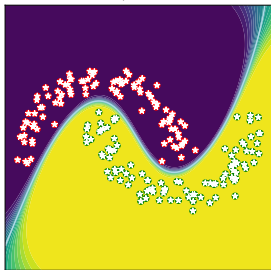
- ▶ Recall: $\|\boldsymbol{\theta}\|_1$ = sum of absolute values of entries in $\boldsymbol{\theta}$
- ▶ Agrees with $\|\boldsymbol{\theta}\|_0$ if $\boldsymbol{\theta} \in \{-1, 0, 1\}^D$
- Encourages sparsity
 - ▶ In contrast to ℓ_2 (see next slide)
- Leads to **shrinkage**
 - ▶ Larger nonzero parameter values penalized more than smaller ones
→ biased estimator
- Easier to optimize, e.g., via subgradient-based methods
- With squared loss, known as **LASSO** (*least absolute shrinkage and selection operator*)
- Can be combined with additional regularizers (e.g., $\ell_2 \rightarrow$ elastic net)

Example: L1VM (logistic regression, RBF features)

Objective: $\operatorname{argmin}_{w_0, \alpha} \left(\sum_i -\log \operatorname{Ber}(y_i | \sigma(w_0 + \alpha^\top \phi_{\text{vm}}(\mathbf{x}_i))) \right) + \lambda \|\alpha\|_1$

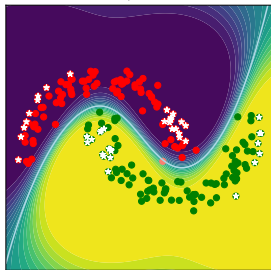
Very high complexity

$\lambda \approx 0, \sigma^2 = 0.571$



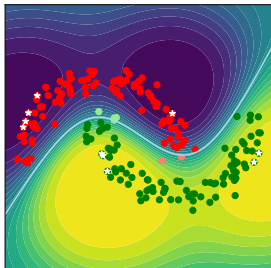
High complexity

$\lambda = 0.1, \sigma^2 = 0.571$



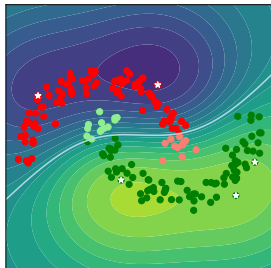
Medium complexity

$\lambda = 1, \sigma^2 = 0.571$



Low complexity

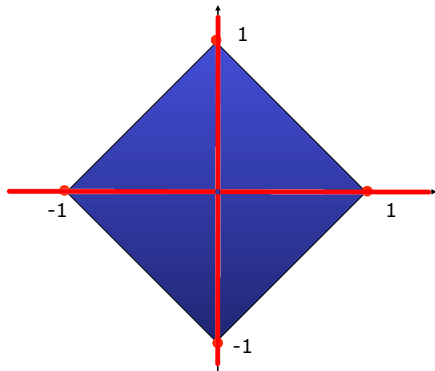
$\lambda = 5, \sigma^2 = 0.571$



Why ℓ_1 regularization? (1)

Let's look at a two-dimensional parameter space $\theta \in \mathbb{R}^2$.

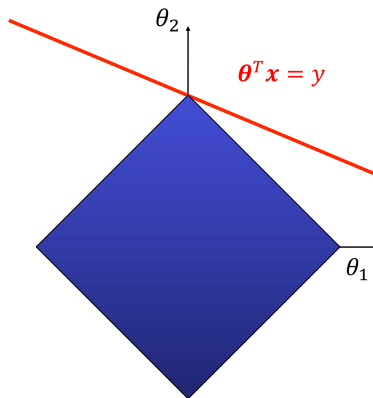
- Border of blue area = ℓ_1 unit ball
- Red lines = ℓ_0 "unit ball" \rightarrow axes except 0
- ℓ_0 unit ball intersects ℓ_1 unit ball at extreme points



Why ℓ_1 regularization? (2)

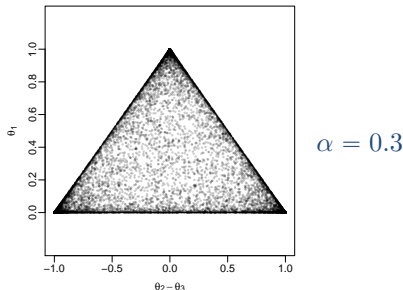
Let's find a solution to the problem $\theta^\top x = y$ w.r.t. θ .

- Underdetermined system with infinitely many solutions
- Sparse solution: $\min_{\theta} \|\theta\|_0$ s.t. $\theta^\top x = y$
 - ▶ Example: $\theta_1 = 0, \theta_2 = y/x_2 \rightarrow \ell_0 \text{ norm} = 1$
- Using ℓ_1 instead: $\min_{\theta} \|\theta\|_1$ s.t. $\theta^\top x = y$
 - ▶ Increasing ℓ_1 norm can be seen as “inflating” the ℓ_1 unit ball
 - ▶ Minimum ℓ_1 norm = minimum inflation
 - ▶ Achieved at intersection with θ_1 or θ_2 axis (whatever is smaller)
- ℓ_1 tends to select extreme points \rightarrow variable selection



Sparsity via prior

- Consider probabilistic model $p(\cdot|\boldsymbol{\theta})$ and a prior $p(\boldsymbol{\theta})$
 - ▶ Prior is **sparsity-promoting** if substantial probability mass at / density around regions where $\boldsymbol{\theta}$ contains zeros
 - ▶ Encourages sparse MAP estimates
- Example: Latent Dirichlet Allocation
 - ▶ Goal: given a document collection, find overall topics and topic distribution of each document
 - ▶ Makes sparsity assumptions between topics and words / documents
 - ▶ Uses Dirichlet prior with $\alpha \ll 1$, which is sparsity-promoting



Example: Latent Dirichlet allocation

“Arts”	“Budgets”	“Children”	“Education”
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

R
(topic \times word),
few words per
topic

The William Randolph Hearst Foundation will give \$1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be \$200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive \$400,000 each. The Juilliard School, where music and the performing arts are taught, will get \$250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual \$100,000 donation, too.

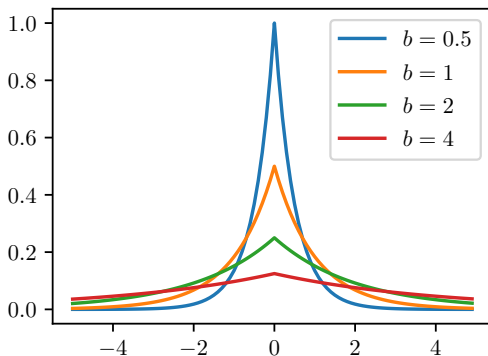
L (doc \times topic),
few topics per
doc

Example: Laplace distribution

- Recall: RRM with ℓ_2 regularization \triangleq MAP with Gaussian prior
- Likewise, RRM with ℓ_1 regularization \triangleq MAP with **Laplace prior**

$$\text{Lap}(\theta|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|\theta - \mu|}{b}\right)$$

- ▶ For sparsity, we use location $\mu = 0$ (= mean/median/mode)
- ▶ Scale parameter b controls strength of prior (variance is $2b^2$)



Support vector machines

- Recall: kernel trick
 - ▶ Mercer kernel κ
 - ▶ Rewrite learning/prediction algorithm in terms of inner products
 - ▶ Replace inner product by kernel calls
 - ▶ Equivalent to working in the implicit feature space of the kernel
- Recall: kernelized ridge regression
 - ▶ Squared loss, ℓ_2 regularization (on weights for impl. kernel features)
 - ▶ Train: $\alpha = (K + \lambda I_N)^{-1} \mathbf{y}$
 - ▶ Predict: $\eta(\mathbf{x}) = w_0 + \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$
 - ▶ α generally not sparse
 - ▶ Natural probabilistic interpretation
- **Support vector machines (SVMs)** modify loss for sparsity
 - ▶ \approx linear predictor + kernel trick + sparsity/large margin
 - ▶ Predict: $\eta(\mathbf{x}) = w_0 + \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$ as well
 - ▶ Few α_i 's nonzero, corresponding \mathbf{x}_i called **support vectors**
 - ▶ No natural probabilistic interpretation

ϵ -insensitive loss

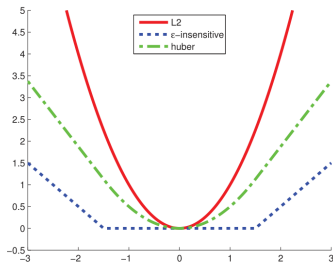
- Key idea of SVMs: no loss on examples predicted “well enough”
 - ▶ Such examples do not affect the cost at the solution
 - ▶ And they do not affect predictions ($\alpha_i = 0$)
- For regression, Vapnik proposed ϵ -insensitive loss

$$\begin{aligned} L_{\epsilon}(y, \hat{y}) &= \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases} \\ &= (|y - \hat{y}| - \epsilon)_+, \end{aligned}$$

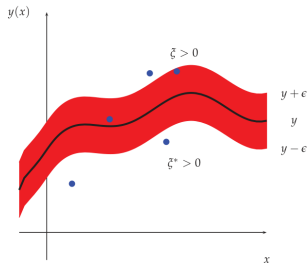
where $(x)_+ = \max(0, x)$.

- ▶ Data points with labels within an ϵ -tube around the prediction incur no loss, i.e., when $y \in (\hat{y} - \epsilon, \hat{y} + \epsilon)$

Illustration



(a)



(b)

Figure 14.10 (a) Illustration of ℓ_2 , Huber and ϵ -insensitive loss functions, where $\epsilon = 1.5$. Figure generated by `huberLossDemo`. (b) Illustration of the ϵ -tube used in SVM regression. Points above the tube have $\xi_i > 0$ and $\xi_i^* = 0$. Points below the tube have $\xi_i = 0$ and $\xi_i^* > 0$. Points inside the tube have $\xi_i = \xi_i^* = 0$. Based on Figure 7.7 of (Bishop 2006a).

SVMs for regression (SVR)

- Using the ϵ -insensitive loss with a linear predictor, we obtain **support vector regression (SVR)**:

$$J = C \sum_i L_\epsilon(y_i, \hat{y}(\mathbf{x}_i)) + \frac{1}{2} \|\mathbf{w}\|^2$$

- ▶ $\hat{y}(\mathbf{x}_i)$ linear in feature space of kernel
- ▶ With linear kernel, called **linear SVR**
- ▶ ℓ_2 regularization on weight as in ridge regression
- ▶ $C = 1/\lambda$ is a regularization constant
- Convex problem, can be solved via quadratic programming
 - ▶ Solution has form $\hat{y}(\mathbf{x}) = w_0 + \sum_i \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$
 - ▶ $\alpha_i = 0$ iff \hat{y}_i lies strictly within ϵ -tube around label y_i , i.e., $\hat{y}_i \notin (y_i - \epsilon, y_i + \epsilon)$
 - ▶ Support vectors ($\alpha_i > 0$) lie on or outside of ϵ -tube
- Hyperparameters: ϵ , C , kernel parameters

Hinge loss

- SVM are popular method for classification (**SVC**)
- Let $y \in \{-1, 1\}$ and consider the decision rule

$$\hat{y} = \text{sgn}(\eta(\mathbf{x})),$$

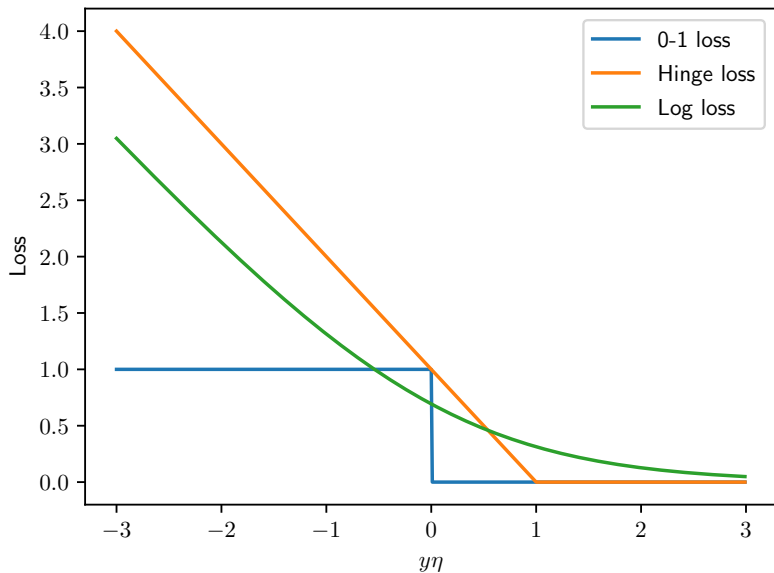
where $\eta(\mathbf{x})$ is a score/predictor produced by the classifier

- In SVC, we use a linear predictor and the **hinge loss**

$$\begin{aligned} L_{\text{hinge}}(y, \eta) &= \begin{cases} 0 & \text{if } y\eta > 1 \\ 1 - y\eta & \text{otherwise} \end{cases} \\ &= (1 - y\eta)_+ \end{aligned}$$

- ▶ $y\eta$ called **margin**
- ▶ Observe: $y\eta > 1$ iff prediction correct ($y = \text{sgn}(\eta)$) and predictor “confident” in that score large ($|\eta| > 1$)
- ▶ Data points with large margin do not incur loss

Hinge loss, illustration



SVMs for classification (SVC)

- Using the hinge loss with a linear predictor η , we obtain **support vector classification (SVC)**:

$$J = C \sum_i L_{\text{hinge}}(y_i, \eta(\mathbf{x}_i)) + \frac{1}{2} \|\mathbf{w}\|^2$$

- ▶ $\eta(\mathbf{x}_i)$ linear in feature space of kernel
- ▶ With linear kernel, called **linear SVC**
- ▶ ℓ_2 regularization on weight as in ridge regression
- ▶ $C = 1/\lambda$ is a regularization constant
- Convex problem, can be solved via quadratic programming
 - ▶ Standard solvers $O(N^3)$, SMO often faster, linear SVC $O(N)$
 - ▶ Solution has form $\eta(\mathbf{x}) = \hat{w}_0 + \sum_i \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$
 - ▶ $\alpha_i = 0$ iff margin large in that $\eta y > 1$
 - ▶ Support vectors ($\alpha_i > 0$) have small margin (misclassified and/or $\eta < 1$)
- Hyperparameters: C , kernel parameters

The large-margin principle

- **Large-margin principle:** select a decision boundary with a large margin (= distance to the closest training point)
- Goal is to reduce generalization error; e.g.,
 - ▶ Consider a perfect classifier (e.g., linearly separable training data)
 - ▶ Observe: All points “within margin” of a training point will be classified just as the training point
 - ▶ The larger the margin, the more consistent the output of the classifier is around the training data

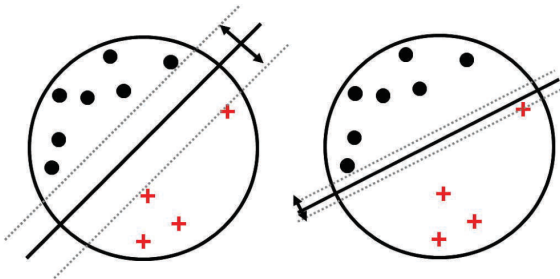


Figure 14.11 Illustration of the large margin principle. Left: a separating hyper-plane with large margin. Right: a separating hyper-plane with small margin.

SVMs are large-margin classifiers (1)

- Consider linear SVM with $\eta(\mathbf{x}) = w_0 + \mathbf{w}^\top \mathbf{x}$
- Distance of \mathbf{x} to decision boundary is $r = \frac{\eta(\mathbf{x})}{\|\mathbf{w}\|}$
 - ▶ Why? Let \mathbf{x}_\perp be orthogonal projection of \mathbf{x} to decision boundary
 - ▶ Write $\mathbf{x} = \mathbf{x}_\perp + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$
 - ▶ Observe that

$$\eta(\mathbf{x}) = \underbrace{w_0 + \mathbf{w}^\top \mathbf{x}_\perp}_{=0} + \underbrace{\mathbf{w}^\top \left(r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right)}_{=r\|\mathbf{w}\|}$$

- ▶ Note: $r > 0$ for positive predictions, $r < 0$ for negatives
- For lin. sep. training data, **maximum margin classifier** given by

$$\operatorname{argmax}_{\mathbf{w}, w_0} \min_i y_i r_i = \operatorname{argmax}_{\mathbf{w}, w_0} \min_i y_i \frac{\eta_i}{\|\mathbf{w}\|},$$

where $\eta_i = \eta(\mathbf{x}_i) = w_0 + \mathbf{w}^\top \mathbf{x}_i$ and $r_i = \eta_i / \|\mathbf{w}\|$

- ▶ At solution, all points correctly classified ($y_i r_i > 0$)
- ▶ And margin $y_i r_i = |r_i|$ as large as possible

SVMs are large-margin classifiers (2)

- $\operatorname{argmax}_{\mathbf{w}, w_0} \min_i y_i \eta_i / \|\mathbf{w}\|$
- Observe: rescaling \mathbf{w} and w_0 by constant $c > 0$
 - ▶ Scales $y_i \eta_i$ by factor c
 - ▶ But does not change margin $r_i = \eta_i / \|\mathbf{w}\|$
 - ▶ Thus: can add constraints $y_i \eta_i \geq 1$ without changing the solution
 - ▶ Inner minimum then $1 / \|\mathbf{w}\|$
- Rewritten optimization problem

$$\operatorname{argmin}_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i \eta_i \geq 1 \text{ for } i = 1, \dots, N$$

- ▶ Constraint = correct prediction, margin at least 1

SVMs are large-margin classifiers (3)

- $\operatorname{argmin}_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i \eta_i \geq 1 \text{ for } i = 1, \dots, N$
- What if data not linearly separable (in kernel's feature space)?
 - ▶ No feasible solution with $y_i \eta_i \geq 1$ for all i exists
 - ▶ Idea: introduce *slack variable* $\xi_i \geq 0$ for each data point
 - ▶ $\xi_i = 0 \rightarrow$ correctly classified, large margin (≥ 1)
 - ▶ $0 < \xi_i < 1 \rightarrow$ correctly classified, small margin ($\in (0, 1)$)
 - ▶ $\xi_i \geq 1 \rightarrow$ incorrectly classified
- Objective with **soft margin constraints**

$$\operatorname{argmin}_{\mathbf{w}, w_0} C \sum_i \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad \xi_i \geq 0, y_i \eta_i \geq 1 - \xi_i$$

- ▶ $\frac{1}{2} \|\mathbf{w}\|^2$ for large margin
- ▶ $\sum_i \xi_i$ for few errors (too-small margin or misclassified)
- ▶ C is hyperparameter that controls trade-off

SVMs are large-margin classifiers (4)

- $\operatorname{argmin}_{\mathbf{w}, w_0} C \sum_i \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad \xi_i \geq 0, y_i \eta_i \geq 1 - \xi_i$
- Observe: we have $\xi_i \geq 1 - y_i \eta_i$ and $\xi_i \geq 0$, hence at optimum

$$\xi_i = \max(0, 1 - y_i \eta_i) = (1 - y_i \eta_i)_+ = L_{\text{hinge}}(y_i, \eta_i)$$

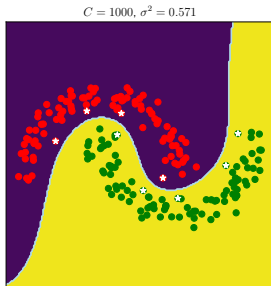
- We obtain the SVM objective (slide 17)

$$\operatorname{argmin}_{\mathbf{w}, w_0} C \sum_i L_{\text{hinge}}(y_i, \eta_i) + \frac{1}{2} \|\mathbf{w}\|^2$$

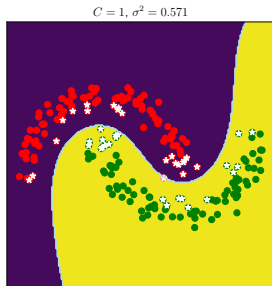
- ▶ C controls allowed training errors and affects generalization error
→ Often set via cross-validation
- ▶ Sometimes: parameterized using $C = 1/(\nu N)$, where ν roughly corresponds to the fraction of misclassified training examples

Example: Support vector machine (Gaussian kernel)

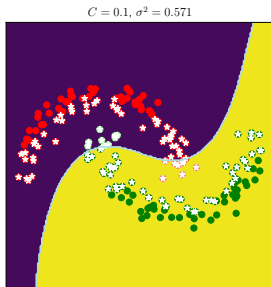
Very high complexity



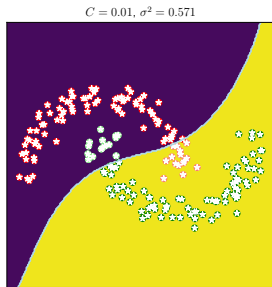
High complexity



Medium complexity

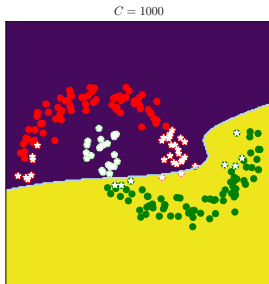


Low complexity

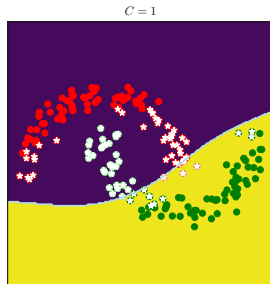


Example: Support vector machine (poly3 kernel)

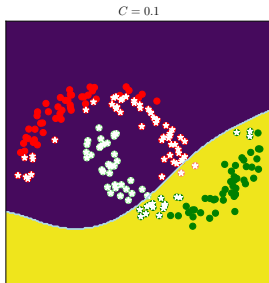
Very high complexity



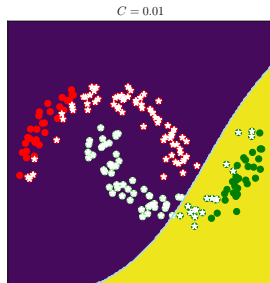
High complexity



Medium complexity



Low complexity



Discussion

- Key ingredients to SVM
 1. Kernel trick \rightarrow prevents underfitting with linear classifier
 2. Sparsity / large margin \rightarrow prevents overfitting
- Key ingredients to L1VM
 - ▶ Use kernel (solely) to generate explicit features ϕ_{vm}
 - ▶ ℓ_1 regularization for sparsity
- SVM or L1VM?
 - ▶ Both are discriminative kernel methods
 - ▶ Often similar performance in practice
 - ▶ L1VMs probabilistic, SVMs not
 - ▶ L1VMs fast to train, SVMs slow (unless linear)
 - ▶ L1VMs handle multiclass classification naturally, SVMs with difficulties
 - ▶ L1VMs any kernel, SVMs Mercer kernel
 - ▶ Both: kernel parameters / regularization weights need tuning

Machine Learning

09 – Hyperparameter Optimization

Part 0: Overview

Prof. Dr. Rainer Gemulla

Universität Mannheim

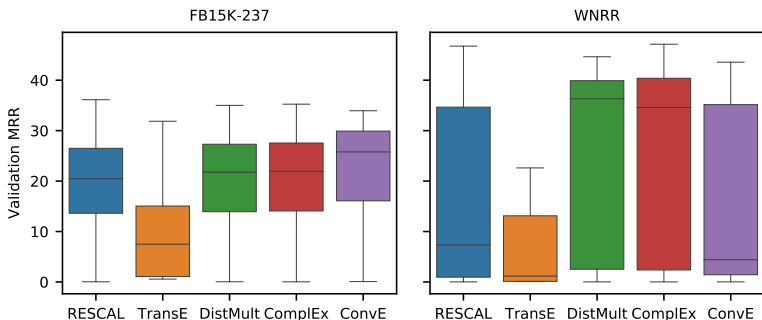
Version: 2023-1

Design decisions in machine learning

- Using machine learning in a application involves many design decisions; e.g.:
 - ▶ Data collection and preprocessing
 - ▶ Feature engineering
 - ▶ Model class & architecture engineering
 - ▶ Learning algorithm
 - ▶ Training objective (e.g., loss, regularization)
 - ▶ Hyperparameters of these components
 - ▶ ...
- How to choose?
 - ▶ Choices matter since they may heavily impact performance
 - ▶ Space of possible choices is large and difficult to navigate

Example: Link Prediction for Knowledge Graphs (1)

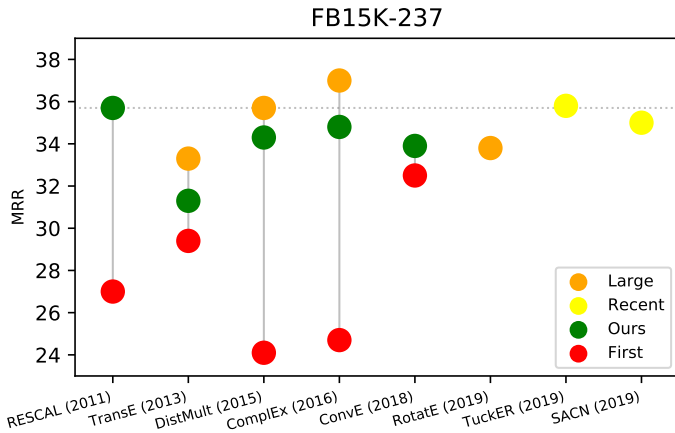
Task: Given a knowledge graph, predict missing links.



Performance over quasi-random hyperparameter configurations
(validation data, higher is better)

Large influence of design choices.

Example: Link Prediction for Knowledge Graphs (2)



Reported performance of various KGE models
(test data, higher is better)

Good choices key for model selection and comparative studies.

Automated Machine Learning (AutoML)

- Vision of **Automated Machine Learning (AutoML)**
 - ▶ Decide in a data-driven, objective, and automated way
 - ▶ User simply provides data, system does the rest
 - ▶ Democratization: make ML accessible to “everyone”
- In practice also: semi-automate
 - ▶ Reduce need for human in the loop
 - ▶ Make practice of ML more systematic and efficient
 - ▶ Improve performance
 - ▶ Improve reproducibility and fairness of scientific studies
- Many approaches, many systems, active research area
- Key directions
 - ▶ Hyperparameter optimization (*our focus*)
 - ▶ Meta-learning (learn how to learn)
 - ▶ Neural architecture search

Outline (Hyperparameter Optimization)

- 0. Overview
- 1. The Hyperparameter Optimization Problem
- 2. Blackbox Optimization
- 3. Multi-Fidelity Optimization
- 4. HPO in Practice

Summary

- Hyperparameter tuning important part of ML pipeline
 - ▶ Choices matter and are difficult to make
- Automated hyperparameter optimization
 - ▶ Explore the hyperparameter configuration space automatically
 - ▶ Use validation protocol to assess configurations
- Blackbox optimization methods
 - ▶ Only based on results of evaluations
 - ▶ E.g., grid search, random search, ES, CMA-ES, Bayesian optimization
- Multi-fidelity methods
 - ▶ Open the blackbox to use cheaper low-fidelity “approximation”
 - ▶ Especially useful for expensive tasks
 - ▶ E.g., successive halving, Hyperband, BOHB
- In practice, HPO method needs to be selected and parameterized
 - ▶ E.g., hyperparameter bounds and transformations very important

Suggested reading

- [Automated Machine Learning: Methods, Systems, Challenges](#) (Ch. 1)
Editors: Frank Hutter, Lars Kotthoff, Joaquin Vanschoren
Springer, 2019
- AutoML tutorial @ NIPS18 ([slides](#), [videos](#))
Frank Hutter, Joaquin Vanschoren

Additional resources

- [Taking the Human Out of the Loop: A Review of Bayesian Optimization](#)
Shahriari et al., Proc. of the IEEE, 2016

Machine Learning

09 – Hyperparameter Optimization

Part 1: The Hyperparameter Optimization Problem

Prof. Dr. Rainer Gemulla

Universität Mannheim

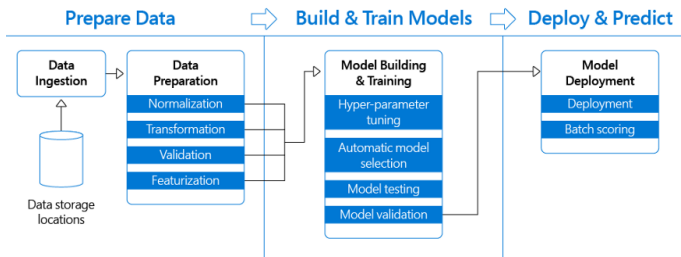
Version: 2023-1

Types of hyperparameters

- **Hyperparameters** influence the learning process
 - ▶ In contrast, parameters are learned
 - ▶ In practice, can have many hyperparameters (e.g., 10s or 100s)
- Discrete hyperparameters; e.g.,
 - ▶ k in a k NN classifier
 - ▶ Number of units in a hidden layer of an FNN
- Continuous hyperparameters; e.g.,
 - ▶ Learning rate
 - ▶ Regularization weight
- Categorical hyperparameters; e.g.,
 - ▶ Use early stopping (yes/no)
 - ▶ Learning algorithm (SGD, Adagrad, Adam)
 - ▶ Type of regularization (L1, L2, L3)
 - ▶ Activation function (tanh, ReLu)
 - ▶ Operator (convolution layer, max pooling layer)

Hyperparameter optimization

- **Hyperparameter optimization (HPO)**: automatically select values for the hyperparameters
- Important part of ML pipeline



- This is challenging!
 - ▶ Hyperparameter configuration space is complex and high-dimensional
 - ▶ Model training/evaluation can be expensive
 - ▶ Cannot directly optimize generalization performance due to limited training data
 - ▶ Often neither gradient nor useful properties such as convexity

Hyperparameter configuration space

- L hyperparameters
- l -th hyperparameter has domain Λ_l
- Hyperparameter **configuration space**: $\Lambda = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_L$
 - ▶ When hyperparameters for preprocessing/algorithms are included, HPO also referred to as full model selection (FMS) or combined algorithm selection and hyperparameter optimization (CASH)
- Space may contain **conditional** hyperparameters
 - ▶ Only active for certain choices of other hyperparameters
 - ▶ Structure can be modeled using a *directed acyclic graph*
 - ▶ Example: algorithm (kNN, LogReg) and algorithm hyperparameters (k only active when kNN is chosen)
 - ▶ Example: number of layers in an FNN and configuration of layer i (only active when at least i layers)

Definition

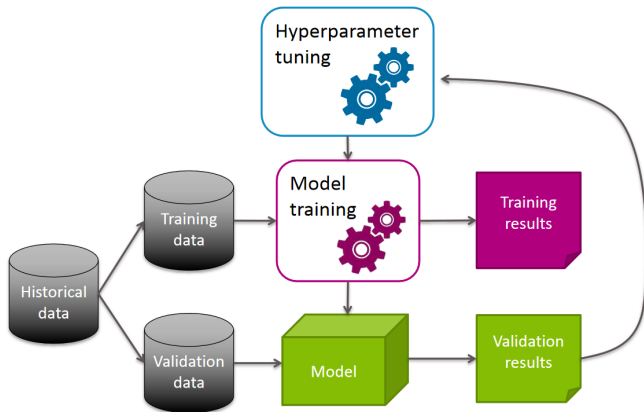
Definition: Hyperparameter Optimization

Given data \mathcal{D} , the HPO problem is to find

$$\boldsymbol{\lambda}^* = \operatorname{argmin}_{\boldsymbol{\lambda} \in \Lambda} V(\boldsymbol{\lambda}, \mathcal{D}).$$

- V is a **validation protocol**
 - ▶ Measures the performance of using hyperparameter configuration $\boldsymbol{\lambda}$ for a particular task based on data \mathcal{D}
 - ▶ Often needs to be approximated since only finite dataset available
- Example: holdout validation or cross validation with a user-defined loss (e.g., misclassification rate)
- Alternatives/variants
 - ▶ Multiple objectives (e.g., consider resource consumption)
 - ▶ Ensembling: combine multiple good hyperparameter configurations
 - ▶ Bayesian model selection: integrate out hyperparameters

Example: Holdout validation



- Holdout validation produces model as side product
- In general, validation protocol can be expensive to evaluate
 - ▶ Several strategies to reduce cost exists

Machine Learning

09 – Hyperparameter Optimization

Part 2: Blackbox Optimization

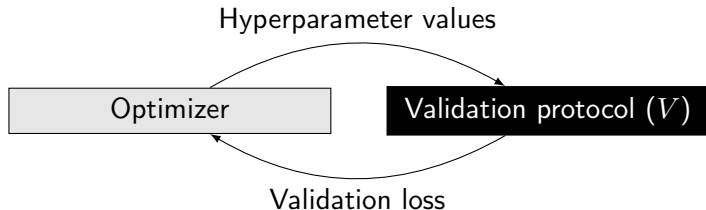
Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-2

Blackbox optimization

- Any **blackbox optimization** method can be used for HPO
 - ▶ Goal is to find $\min_{\lambda} V(\lambda)$
 - ▶ $V(\lambda)$ can be evaluated (or approximated)
 - ▶ No knowledge about what happens “inside” V
 - ▶ No access to gradient information such as $\nabla_{\lambda} V(\lambda)$



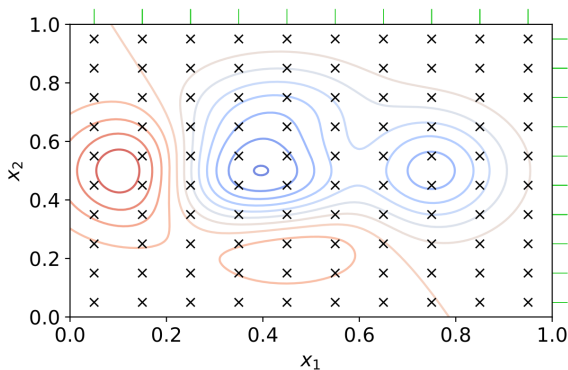
- Generally, use history $\{(\lambda_i, V(\lambda_i))\}_{i=1}^n$ to decide where to evaluate next (λ_{n+1})
 - ▶ Explored hyperparameter values often called **trials**
 - ▶ Note: closely related to learning

Approaches to blackbox optimization

- **Model-free methods:** do not use history
 - ▶ Example: grid search, random search
- **Stochastic search:** maintain probability distribution $p(\boldsymbol{\lambda})$
 - ▶ Used to sample new values of $\boldsymbol{\lambda}$ to evaluate
 - ▶ Updated with new observations
 - ▶ Example: population-based methods, simulated annealing
- **Global optimization:** maintain model $\hat{V}(\boldsymbol{\lambda})$ plus confidence
 - ▶ Use model to choose where to evaluate next
 - ▶ Exploration/exploitation trade-off
 - ▶ Example: Bayesian optimization

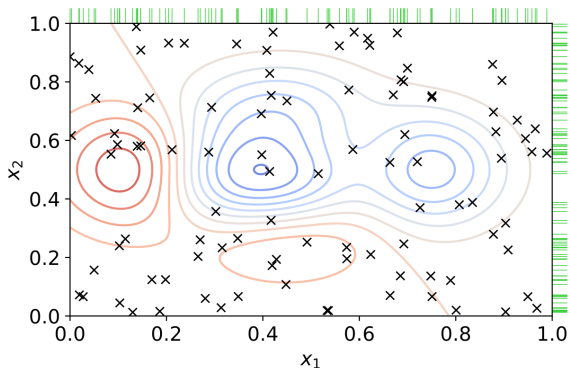
Grid search

- **Grid search** is most basic HPO method
 - ▶ Specify a finite set of choices for each hyperparameter
 - ▶ Evaluate all combinations of these choices (Cartesian product)
- Example: 2 real-valued hyperparameters and a 10×10 grid



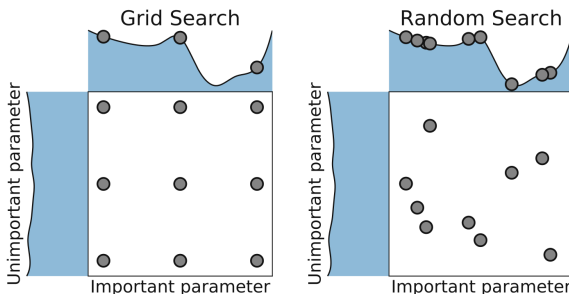
Random search

- **Random search** samples configurations from Λ at random
- Example: 2 real-valued hyperparameters, 100 trials (using uniform sampling)



Discussion (1)

- Grid search suffers from the curse of dimensionality
 - ▶ Number of trials grows exponentially with dimensionality of configuration space
 - ▶ E.g., 30 hyperparameters with 4 choices each $\rightarrow > 10^{18}$ trials
- Random search explores configuration space better
 - ▶ E.g., consider L real hyperparameters and n trials
 - ▶ Random search explores n values per hyperparameters
 - ▶ Grid search explores only $n^{1/L}$ values per hyperparameter
 - ▶ Especially problematic if some hyperparameters are unimportant



Discussion (2)

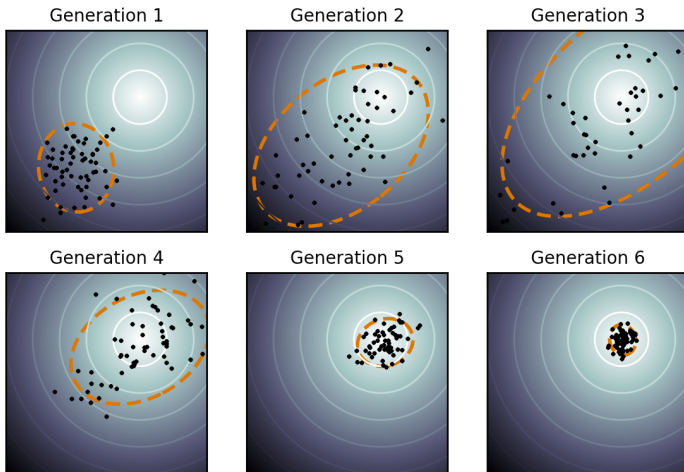
- Choice of grid points problematic
 - ▶ May be non-trivial
 - ▶ May significantly affect (or bias) results
- Both methods are parallelizable across a set of workers
 - ▶ Easier to do for random search (no synchronization needed)
- Random search easy to shrink/expand number of trials
- Random search generally preferable over grid search
 - ▶ More efficient empirically and theoretically
 - ▶ Hence useful baseline
- May take far longer than “guided” search method
 - ▶ E.g., HPO on L binary features *without interactions* require $O(2^L)$ trials with random search but $O(L)$ with suitable guided search
 - ▶ Still, random search often a component of such methods (e.g., for initialization and exploration)

Population-based methods (1)

- **Population-based methods** maintain a population of configurations
 - ▶ Special case of stochastic search
 - ▶ Population used to determine which trials to perform next
 - ▶ Population updated based on results (e.g., *mutation*, *cross-over*)
- Example: **evolution strategy** (ES)
 - ▶ Maintain an isotropic Gaussian $\mathcal{N}(\boldsymbol{\theta}^{(t)}, \sigma^2 \mathbf{I})$
 - ▶ In t -th iteration,
 1. Sample λ configurations from $\mathcal{N}(\boldsymbol{\theta}^{(t)}, \sigma^2 \mathbf{I})$ (**offspring**) and evaluate
 2. Keep best μ results
 3. Use their mean as $\boldsymbol{\theta}^{(t+1)}$
 - ▶ (μ, λ) -**ES**: select best results from offspring only
 - ▶ $(\mu + \lambda)$ -**ES**: include prior trials (**parents**)
 - ▶ Example: $(1 + 1)$ -ES = **hill climbing**

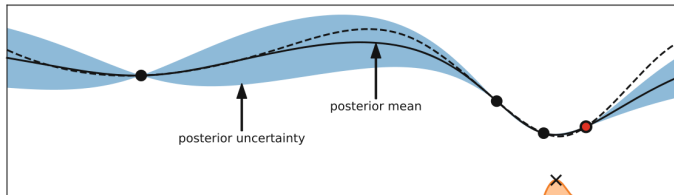
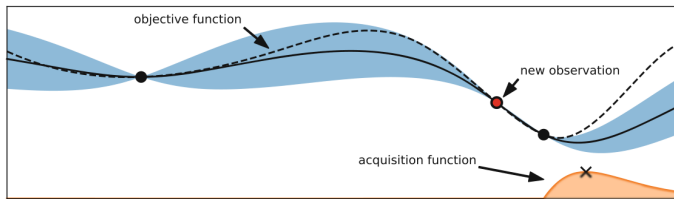
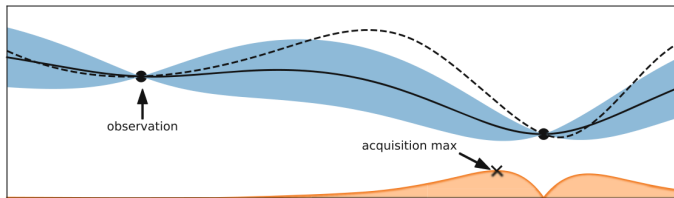
Population-based methods (2)

- Example: **covariance matrix adaptation** (CMA-ES)
 - ▶ Improves on ES by modeling covariance as well
 - ▶ Very competitive in [benchmarks](#)
 - ▶ Easy to use and parallelize



Bayesian Optimization

- **Bayesian optimization**
 - ▶ State-of-the-art global optimization framework
 - ▶ Very good results for HPO empirically
- Approach
 1. Maintain a *probabilistic* **surrogate model** of $V(\lambda)$
 2. Optimize an **acquisition function** using the surrogate model to select next configuration
 3. Update surrogate model and repeat
- Surrogate model cheap to evaluate
- Acquisition function trades off exploration and exploitation
 - ▶ Determines “utility” of evaluating a each configuration
 - ▶ Takes uncertainty provided by surrogate model into account



Surrogate models

- Used to model current knowledge about $V(\boldsymbol{\lambda})$
- At its heart, a regression problem
 - ▶ Input $\boldsymbol{x} = \boldsymbol{\lambda}$ corresponds to hyperparameter configuration
 - ▶ Output $y = V(\boldsymbol{\lambda})$ corresponds to validation loss
 - ▶ Data is given by trials so far: $\mathcal{D} = \{(\boldsymbol{\lambda}_i, V(\boldsymbol{\lambda}_i))\}_{i=1}^n$
- Parametric models
 - ▶ Parameter vector $\boldsymbol{\theta}$ with prior $p(\boldsymbol{\theta})$
 - ▶ Used to derive posterior predictive $p(\hat{V}|\boldsymbol{\lambda}, \mathcal{D}) = \int p(\hat{V}|\boldsymbol{\lambda}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D}) \, d\boldsymbol{\theta}$, which accounts for uncertainty about parameters $\boldsymbol{\theta}$
 - ▶ Example: **Bayesian linear regression**
 - ▶ $p(\hat{V}|\boldsymbol{\lambda}, \boldsymbol{\theta}, \sigma^2) = \mathcal{N}(\boldsymbol{\theta}^\top \boldsymbol{\lambda}, \sigma^2)$
 - ▶ $p(\boldsymbol{\theta})$ is normal inverse gamma (=conjugate prior)
 - ▶ Posterior predictive also Gaussian & can be computed in closed form
 - ▶ But: linearity assumption not suitable for HPO
 - ▶ Example: **Gaussian process regression** (coming up)
 - ▶ Think: Bayesian + kernel + linear regression
 - ▶ Commonly used for HPO, but also when little data and/or no gradients available

Recall: Bayesian linear regression (from 02-3)

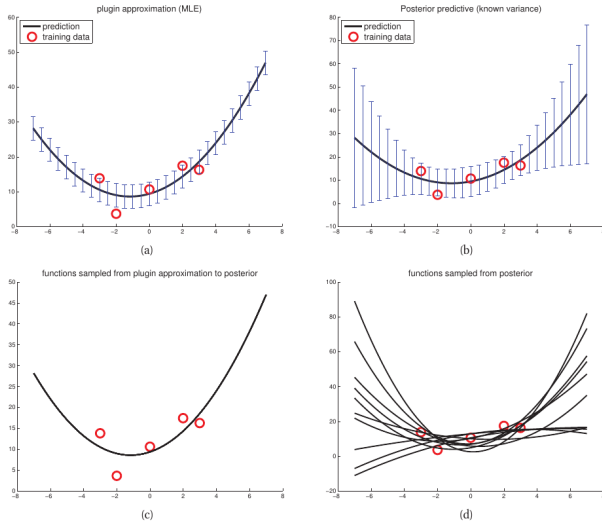


Figure 7.12 (a) Plug-in approximation to predictive density (we plug in the MLE of the parameters). (b) Posterior predictive density, obtained by integrating out the parameters. Black curve is posterior mean, error bars are 2 standard deviations of the posterior predictive density. (c) 10 samples from the plugin approximation to posterior predictive. (d) 10 samples from the posterior predictive. Figure generated by `linregPostPredDemo`.

Gaussian process regression

- **Gaussian process regression (GPR)** often used in HPO with BO
 - ▶ **Gaussian process** $\text{GP}(\mu_0, \kappa)$ represents a distribution over possible functions; for us: validation functions (from Λ to \mathbb{R})
 - ▶ **GPR** is a non-parametric Bayesian approach that uses a GP as a prior to build a regression model
 - ▶ Parameters: inputs λ_i and corresponding outputs $f_i (= V(\lambda_i))$
 - ▶ Hyperparameters: **kernel** κ , **prior mean function** $\mu_0 : \Lambda \rightarrow \mathbb{R}$
→ need to be chosen adequately
- Obtained by kernelizing Bayesian linear regression
 - ▶ Recall: lecture 08 on kernels
 - ▶ Use kernel $\kappa(\lambda_1, \lambda_2)$ to measure similarity between configurations
 - ▶ “Replace” design matrix $\mathbf{X} = (\lambda_i^T)_{i=1}^n$ with kernel matrix \mathbf{K}
(where $\mathbf{K}_{ij} = \kappa(\lambda_i, \lambda_j)$)
 - ▶ Use kernel trick
- Intuitively, prior ensures smoothness, i.e., that **similar inputs (according to kernel) produce similar outputs**

Illustration (1)

Sample of functions from a Gaussian process

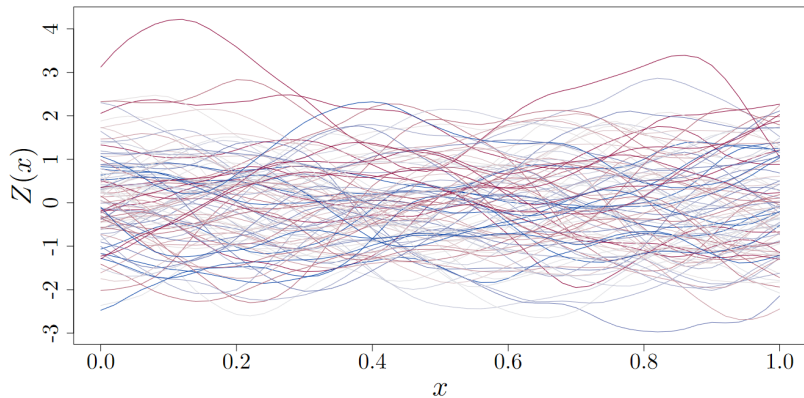


Illustration (2)

Observations (1-dimensional)

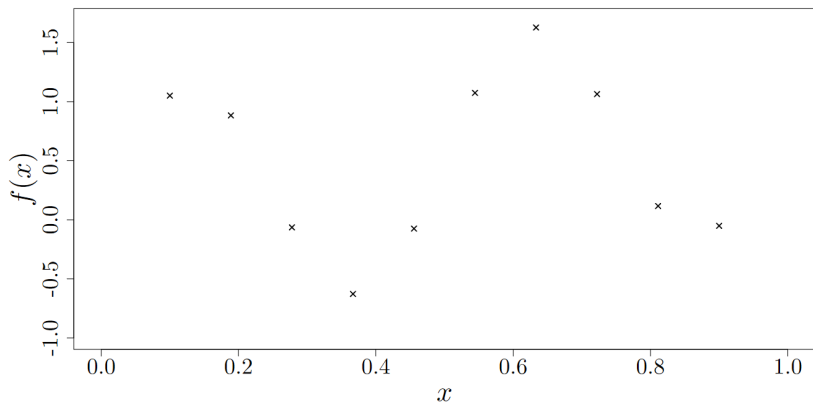


Illustration (3)

Samples from Gaussian process conditioned on observations
(=samples from GPR model)

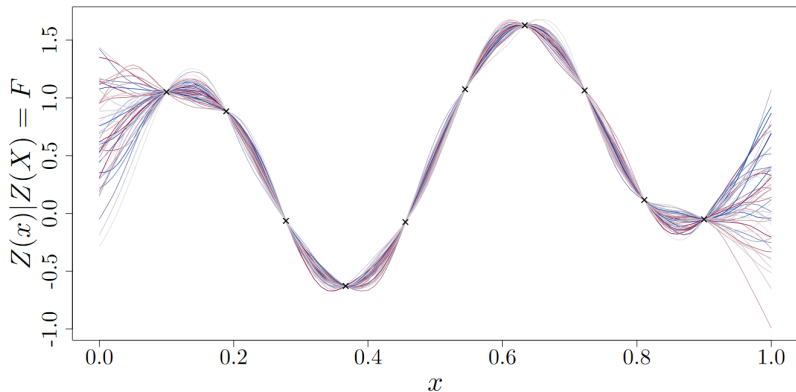
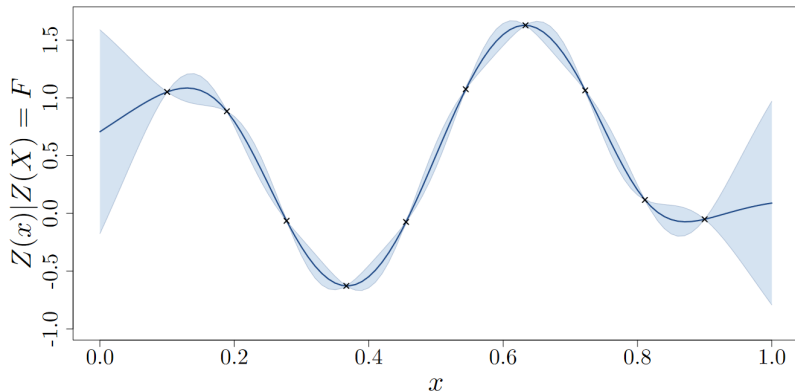


Illustration (4)

Conditional distribution of functions cond. on observations
(=GPR model)



Gaussian process regression (prediction)

- Consider new input λ_+ and data $\mathcal{D} = \{(\lambda_i, f_i)\}_{i=1}^n$
- For clarity, let's write F_i for unknown values (random variable) and f_i for observed values (constants)
 - ▶ We want to reason about F_+ , the unknown value at λ_+
- GPR model
 1. GP gives us joint distribution $p(F_1, \dots, F_n, F_+)$
 2. GPR determines cond. distribution

$$p_+(F_+ = f) \stackrel{\text{def}}{=} p(F_+ = f | F_1 = f_1, \dots, F_n = f_n)$$

- Using a GPR model
 - ▶ Prediction = **posterior mean** $\hat{f}_+ = E_{p_+}[f]$
 - ▶ Uncertainty = **posterior variance** $\sigma_+^2 = \text{var}_{p_+}[f]$
 - ▶ Sampling = sample value from p_+
 - ▶ \hat{f}_+ and σ_+^2 are shown on slide 18 for various choices of $\lambda_+ \in [0, 1]$
 - ▶ Samples are shown on slides 15 and 17 (each function sampled sequentially and by conditioning on previous values)

Details (Step 1)

Step 1: GP gives us joint distribution $p(F_1, \dots, F_n, F_+)$

- Let $\mathbf{F} = (F_1 \ \dots \ F_n \ F_+)^{\top}$
 - ▶ \mathbf{F} is an $(n + 1)$ -dimensional random variable
 - ▶ Elements are unknown function values at $\lambda_1, \dots, \lambda_n, \lambda_+$
- GP(μ_0, κ) **assumes** that F_1, \dots, F_n, F_+ are jointly Gaussian
 - ▶ In particular, that \mathbf{F} is a multivariate Gaussian

$$\mathbf{F} \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$

- ▶ \mathbf{m} is an $(n + 1)$ -dimensional mean vector with $m_i = \mu_0(\lambda_i)$
 - ▶ \mathbf{K} is a $(n + 1) \times (n + 1)$ -dimensional covariance matrix with $k_{ij} = \kappa(\lambda_i, \lambda_j)$
- Observe: distribution of \mathbf{F} completely determined by μ_0 and κ (for our choices of $\lambda_1, \dots, \lambda_n, \lambda_+$)
 - ▶ GP thus serves as a prior on function values
 - ▶ Hyperparameters μ_0 and κ define this prior

Details (Step 2)

Step 2: GPR determines cond. distribution

$$p_+(F_+ = f) \stackrel{\text{def}}{=} p(F_+ = f | F_1 = f_1, \dots, F_n = f_n)$$

- We assumed $p(\mathbf{F}) = \mathcal{N}(\mathbf{F} | \mathbf{m}, \mathbf{K})$ (a multivariate Gaussian)
- We know f_1, \dots, f_n , but not f_+
- p_+ is obtained by conditioning $p(\mathbf{F})$ on observed values f_1, \dots, f_n
 - ▶ Need to compute a conditional of a Gaussian (cf. 07-1/9)
 - ▶ Partition into observed (o) and new (+) parts

$$\mathbf{f} = \begin{pmatrix} \mathbf{f}_o \\ f_+ \end{pmatrix} \quad \mathbf{m} = \begin{pmatrix} \mathbf{m}_o \\ m_+ \end{pmatrix} \quad \mathbf{K} = \begin{pmatrix} \mathbf{K}_{oo} & \mathbf{k}_{o+} \\ \mathbf{k}_{+o}^\top & k_{++} \end{pmatrix}$$

- ▶ where $f^+, m^+, k_{++} \in \mathbb{R}$; $\mathbf{f}_o, \mathbf{m}_o, \mathbf{k}_{o+}, \mathbf{k}_{+o} \in \mathbb{R}^n$; $\mathbf{K}_{oo} \in \mathbb{R}^{n \times n}$
- ▶ All values but f^+ are known
- ▶ We obtain: $p_+(f) = \mathcal{N}(f | \mu_+, \sigma_+^2)$ (that's another Gaussian)
 - ▶ $\mu_+ = m_+ + \mathbf{k}_{+o}^\top \mathbf{K}_{oo}^{-1} (\mathbf{m}_o - \mathbf{f}_o)$
 - ▶ $\sigma_+^2 = k_{++} - \mathbf{k}_{+o}^\top \mathbf{K}_{oo}^{-1} \mathbf{k}_{o+}$

GPR for BO

- GPR can be used as a surrogate model in BO
 - ▶ But: $V(\boldsymbol{\lambda}_i)$ is often a random variable itself (since we often perform random choices during model fitting)
 - ▶ This can be modeled by observation noise
 - ▶ With isotropic Gaussian noise (as in linear regression), we assume $V(\boldsymbol{\lambda}_i) \sim \mathcal{N}(f_i, \sigma^2)$, i.e., we observed only a noisy variant of f_i
 - ▶ σ^2 is an additional hyperparameter
- Cost: $O(n^3)$ to fit (compute \mathbf{K}_{oo}^{-1}), then $O(n^2)$ per prediction
→ Expensive for large number of observations
- Predictions used to maximize acquisition function
 - ▶ Acquisition function has form $\alpha(\boldsymbol{\lambda}^+) \in \mathbb{R}$
 - ▶ Given $\boldsymbol{\lambda}^+$, $\alpha(\boldsymbol{\lambda}^+)$ computed based on \hat{f}_+ (prediction) and σ_+^2 (uncertainty)

Matérn kernels

- Common choice: a **Matérn kernel**
 - ▶ Stationary kernel (value only depends on difference of in inputs)
 - ▶ Smoothness parameter ν (kernel differentiable $\lfloor \nu - 1 \rfloor$ times)
 - ▶ $\nu \rightarrow \infty \rightarrow$ Equivalent to squared exponential kernel (Gaussian kernel with diagonal covariance)
 - ▶ $\nu = 1/2 \rightarrow$ Equivalent to exponential kernel
 - ▶ Parameterized by length scales σ_j (one per hyperparameter) and bandwidth σ^2

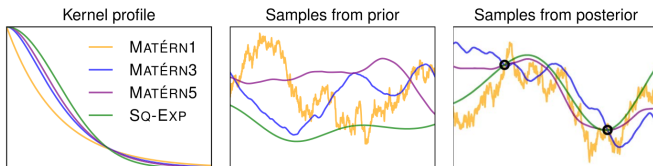


Fig. 3. (Left): Visualization of various kernel profiles. The horizontal axis represents the distance $r > 0$. (Middle): Samples from GP priors with the corresponding kernels. (Right): Samples from GP posteriors given two data points (black circles). Note the sharper drop in the Matérn1 kernel leads to rough features in the associated samples, while samples from a GP with the Matérn3 and Matérn5 kernels are increasingly smooth.

- Choice of kernels can have large impact (!!!)

Discussion and alternatives

- GP regression in Bayesian optimization
 - ▶ Traditional choice: expressive, smooth, well-calibrated uncertainty estimates, closed-form predictive distribution
 - ▶ Cubic cost → feasible only when limited function evaluations
 - ▶ Complex configuration spaces problematic: high dimensionality, discrete/categorical variables, conditional hyperparameters
 - ▶ Strong assumptions on noise
 - ▶ Harder to parallelize

→ Many approaches/variants proposed in the literature
- Alternative: Random forests
 - ▶ Natively handle larger and also conditional spaces
 - ▶ E.g., [SMAC framework](#)
- Alternative: [Tree-Structured Parzen Estimators](#) (**TPE**)
 - ▶ Model densities of good ($p(\boldsymbol{\lambda}|V < \alpha)$) and bad ($p(\boldsymbol{\lambda}|V > \alpha)$) choices using a kernel density estimator (Parzen estimator)
 - ▶ Multiple such estimators arranged in a tree for conditional spaces
 - ▶ Good empirical performance on structured HPO tasks

Acquisition function

- Many acquisition functions α exist, no single best one
- **Improvement-based methods** try to improve over a target τ
 - ▶ Ideally, value close to minimum
 - ▶ Heuristically, best value so far
- **Probability of improvement (PI)**: $\alpha_{\text{PI}}(\boldsymbol{\lambda}) = p(\hat{V}(\boldsymbol{\lambda}) < \tau)$
 - ▶ Can work well when target close to minimum
 - ▶ Otherwise may exploit too aggressively
- **Expected improvement (EI)**: $\alpha_{\text{EI}}(\boldsymbol{\lambda}) = E[(\tau - \hat{V}(\boldsymbol{\lambda}))^+]$
 - ▶ Incorporates amount of improvement (if any)
- Alternative: **upper confidence bound (UCB)**
 - ▶ E.g., set $\alpha_{\text{UCB}}(\boldsymbol{\lambda})$ such that $p(\hat{V}(\boldsymbol{\lambda}) < \alpha_{\text{UCB}}(\boldsymbol{\lambda})) = 0.05$
 - ▶ Optimistic method
- Alternative: **information-based methods** use posterior distribution $p(\boldsymbol{\lambda}^* | \hat{V})$ of best configuration

Acquisition function (example)

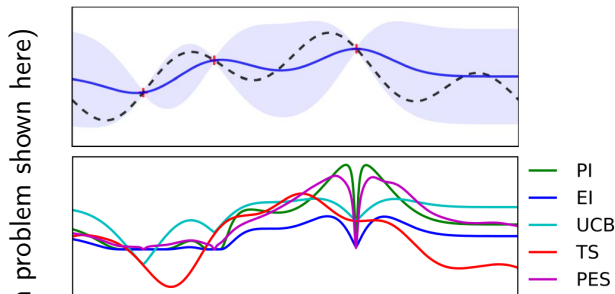


Fig. 5. Visualization of the surrogate regression model and various acquisition functions. (Top) The true objective function is shown as a dashed line and the probabilistic regression model is shown as a blue line with a shaded region delimiting the $2\sigma_n$ credible intervals. Finally, the observations are shown as red crosses. (Bottom) Four acquisition functions are shown. In the case of PI, the optimal mode is much closer to the best observation as in the alternative methods, which explains its greedy behavior. In contrast, the randomization in TS allows it to explore more aggressively.

- Note: Acquisition function needs to be maximized
→ Auxiliary optimization techniques

Practical considerations

- Description of configuration space
 - ▶ Often box constrained
 - ▶ Sometimes logarithm of hyperparameter optimized (e.g., learning rate, regularization weight)
 - ▶ As so often, choices matter
- Practical constraints
 - ▶ In practice, often additional constraints: memory consumption, training time, prediction time, energy usage, ...
 - ▶ E.g., a single slow configuration should not consume all of the available time budget
 - ▶ Can usually only be observed afterwards (e.g., training time)
 - ▶ Simplest approach: add penalty term to $V(\boldsymbol{\lambda})$ for constraint violations

Machine Learning

09 – Hyperparameter Optimization

Part 3: Multi-Fidelity Optimization

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Low-fidelity approximations

- Blackbox optimization can be expensive
 - ▶ Increasing dataset sizes
 - ▶ Increasingly complex models
 - ▶ Even training a single HP configuration may take a long time
 - ▶ Example: BERT training [took 4 days](#)
- Idea: “probe” a configuration in a less costly manner using **low-fidelity approximations**
 - ▶ Often done when performing manual HP tuning
 - ▶ Use a subset of the data
 - ▶ Use reduced inputs (e.g., feature subset, lower-resolution images)
 - ▶ Use reduced models (e.g., used by [GPT-4](#))
 - ▶ Train for a few iterations
 - ▶ Use a subset of the cross-validation folds
- Tradeoff between quality of approximation and runtime
 - ▶ Lower fidelity → faster but lower quality
 - ▶ Higher fidelity → slower but higher quality

Multi-fidelity hyperparameter optimization

- **Multi-fidelity optimization** methods

- ▶ Systematically use low-fidelity approximations for HPO
- ▶ Usually involve multiple fidelities
- ▶ Allows to explore larger parts of HP configuration space within a given computational budget
- ▶ For expensive learning tasks, runtime gains often outperform approximation error in practice

1. Predictive termination methods

- ▶ Successively increase fidelity for a single configuration
- ▶ Stop when increasing fidelity further not beneficial → reduce cost
- ▶ Based on **learning curve prediction**

2. Selection with static fidelities

- ▶ Fixed schedule of fidelities to try (from low to high)
- ▶ Select which configurations to explore (further)

3. Selection with adaptive fidelities

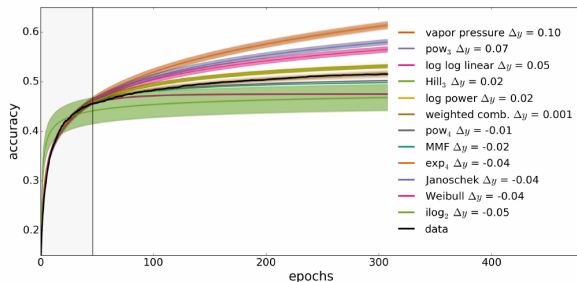
- ▶ Also choose fidelities actively

Predictive termination

- **Learning curve** = performance of an iterative ML algorithm as a function of iterations/training time
- HPO with [predictive termination](#)
 - ▶ Use any HPO method for iterative ML algorithms, but modify the validation protocol
 - ▶ Validate regularly (e.g., after every k iterations)
 - ▶ We obtain a *partial* learning curve
 - ▶ Use a learning curve model to estimate whether the partial learning curve is likely to fall behind the *best* model found so far
 - ▶ If so, terminate training and output estimated final performance
 - ▶ If not, continue training
- Learning curve model
 - ▶ Key goal is to identify “bad” configurations quickly
 - ▶ Only few observations per curve: uncertainty modeling is key
 - ▶ May exploit prior knowledge, e.g., learning curves are non-increasing and saturate

Parametric learning curve model of Domhan et al. (1)

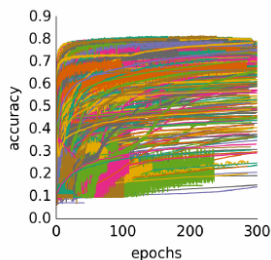
- Linear combination of 11 suitable parametric functions and additive Gaussian noise
- Prior: positive weights, non-decreasing



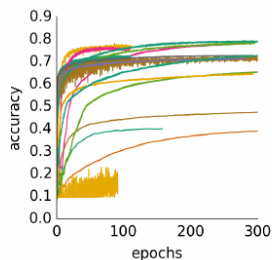
Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c \log(x))$
pow ₃	$c - ax^{-\alpha}$
log log linear	$\log(a \log(x) + b)$
Hill ₃	$\frac{y_{\max} x^{\eta}}{\kappa \eta + x^{\eta}}$
log power	$\frac{a}{1 + (\frac{x}{e^b})^c}$
pow ₄	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha - \beta}{1 + (\kappa x)^{\delta}}$
exp ₄	$c - e^{-ax^{\alpha} + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^{\delta}}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^{\delta}}$
ilog ₂	$c - \frac{a}{\log x}$

Figure 1: Left: A typical learning curve and extrapolations from its first part (the end of which is marked with a vertical line), with each of the 11 individual parametric models. The legend is sorted by the residual of the predictions at epoch 300. Right: the formulas for our 11 parameteric learning curve models $f_k(x)$.

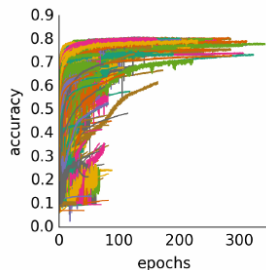
Parametric learning curve model of Domhan et al. (2)



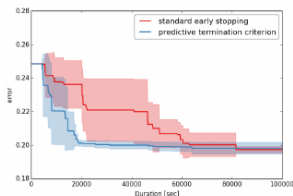
(a) Without predictive termination



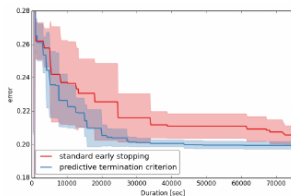
(b) Random subset of Figure 4a



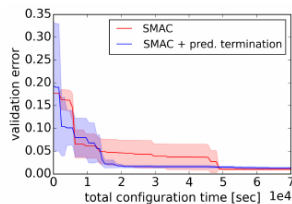
(c) With predictive termination



(a) SMAC on k-means CIFAR-10



(b) TPE on k-means CIFAR-10



(c) SMAC on MNIST

Background: Multi-armed bandits

- Multi-fidelity HPO can be modeled as particular multi-armed bandit problems
- Generally, a **multi-armed bandit** (MAB) is
 - ▶ A set of L real **reward distributions** $B = \{R_1, \dots, R_L\}$
 - ▶ Casino analogy: L slot machines, each with a single **arm** to pull
 - ▶ **Reward** obtained by sampling from R_i of some arm i
 - ▶ Key question: Which arm to pull? (**allocation**)
- MAB problem: maximize **payoff** over H pulls (**horizon**)
 - ▶ Payoff = cumulative reward
 - ▶ Reward distributions unknown, but reward observed after each pull
 - ▶ Exploration/exploitation trade-off
 - ▶ Explore: pull an arm to gather information about its reward distribution
 - ▶ Exploit: pull an arm with high (estimated) reward
- Many techniques, many variants

HPO and multi-armed bandits (1)

- HPO and MAB are closely related
 - ▶ Arm = hyperparameter configuration
 - ▶ Pull an arm = train with this hyperparameter configuration
 - ▶ Loss = observed validation loss
- **Best-arm identification**
 - ▶ Goal is not to maximize payoff, but to estimate the “best” arm
 - ▶ A [pure exploration problem](#)
 - ▶ Strategies well-suited for maximizing payoff may not be well suited for best-arm identification
- **Many-armed bandits**
 - ▶ Number of arms $>$ number of available trials
 - ▶ E.g., when continuous variable included
 - ▶ Cf. methods discussed for blackbox optimization

HPO and multi-armed bandits (2)

- Multi-fidelity HPO can also be modeled
 - ▶ First pull of arm i = train low-fidelity model
 - ▶ Subsequent pulls of arm i = continue training to obtain a higher-fidelity model
 - ▶ Example: pull an arm = train for one additional SGD epoch
- Non-stochastic multi-armed bandit problem
 - ▶ As MAP, but when pulling arm i for the T_i -th time, observe loss l_{iT_i} (no randomness)
 - ▶ Loss of an arm changes when its being pulled (oblivious to any pulls of other arms)
 - ▶ Losses converge in that $v_i = \lim_{T_i \rightarrow \infty} l_{iT_i}$ exists
 - ▶ Goal is to identify best arm (lowest v_i)
- Think: T_i = fidelity, l_{iT_i} = low-fidelity validation loss, v_i = full-fidelity validation loss
- Difficult problem; e.g., cannot reject an arm or verify best arm
 - ▶ In general, convergence to v_i may be arbitrarily slow

Successive halving

- A simple technique is successive halving (SH)
- Input: set of n arms, overall **budget** B
- Output: estimate of best arm
- Budget distributed evenly over $R = \lceil \log_2(n + 1) \rceil$ rounds ($\approx B/R$ pulls per round)
- Start with all arms active, then in each round
 - ▶ Pull each active arm equally often (\approx doubles each round)
 - ▶ Remove the half of the active arms with the largest loss
 - ▶ After R rounds, only one arm survives \rightarrow estimate of best arm
- Prespecifying the budget can be avoided by a “doubling trick”
 - ▶ Start with $B \leftarrow n$, then repeatedly double B
 - ▶ Reuse existing arms \rightarrow no repeated computation
 - ▶ When B^* is the optimal budget, effective budget with doubling trick is $\leq 2B^*$

Successive halving (example)

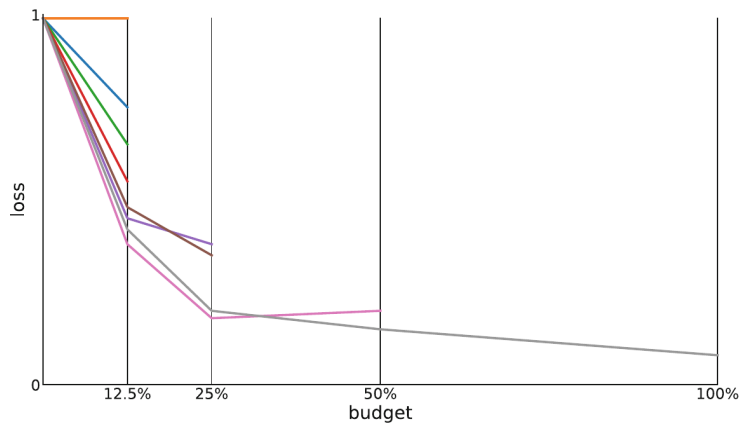


Fig. 1.3 Illustration of successive halving for eight algorithms/configurations. After evaluating all algorithms on $\frac{1}{8}$ of the total budget, half of them are dropped and the budget given to the remaining algorithms is doubled

Successive halving (discussion)

- Very simple, performs well
- Analysis and experimental study by [Jamieson and Talwalkar \(2016\)](#)
 - ▶ Under certain conditions, provably better than random search
 - ▶ Generally not much worse than random search
 - ▶ Better anytime performance than random search
 - ▶ Empirically good HPO performance w.r.t. other bandit methods
 - ▶ One reason: few runs of validation protocol necessary ($2n + 1$, which is often $\ll B$)
- Instead of halving, may use other factors
 - ▶ HPO hyperparameter η (e.g., $\eta = 2$ for SH)
 - ▶ In each round, keep $1/\eta$ of best arms, multiply budget by η
- In HPO, arms are sampled from the HP configuration space. But how many arms should we use? (" n vs. B/n problem")
 - ▶ Many with small budget each
 - may terminate good configurations prematurely
 - ▶ Few with large budget each
 - may waste resources on poor configurations
 - ▶ Trade-off may not be easy to make by user

n vs B/n problem (example)

HPO for LeNet on MNIST

- Successive halving with $\eta = 3$
- s low \rightarrow few configurations with large budget each
- s high \rightarrow many configurations with small budget each
- Optimal choice neither too low nor too high (here: $s = 3$)

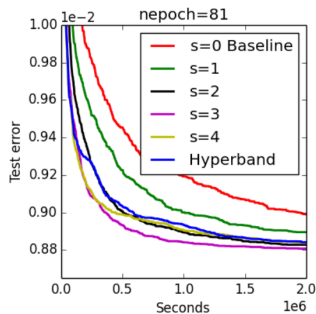


Figure 3: Performance of individual brackets s and HYPERBAND.

Hyperband

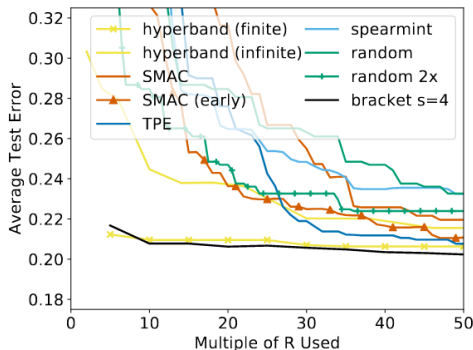
- Hyperband addresses the n vs B/n problem
- Key idea: run successive halving repeatedly from scratch with fixed budget but different number of configurations
- Input: maximum per-configuration budget R , factor η (def. 3)
 - ▶ Perform $\lfloor \log_{\eta} R + 1 \rfloor$ runs of successive halving, called **brackets**
 - ▶ Initial budget (r) in first round starts at 1 and is multiplied by η from bracket to bracket
 - ▶ Run as many configurations (n) with budget R as possible to not exceed a fixed per-bracket budget ($B' = R \lfloor \log_{\eta} R + 1 \rfloor$)

	$s = 4$		$s = 3$		$s = 2$		$s = 1$		$s = 0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Table 1: The values of n_i and r_i for the brackets of HYPERBAND corresponding to various values of s , when $R = 81$ and $\eta = 3$.

Discussion

- Introduced, analyzed, and empirically studied by Li et al. (2018)
 - ▶ Modeled as non-stochastic infinitely-armed bandit problem
 - ▶ Can be extended to avoid specifying R (similar to doubling trick)
 - ▶ Hyperband most useful when domain information on n -vs- B/n trade-off is not available
 - ▶ In practice, Hyperband outperformed by successive halving with “right” number of configurations

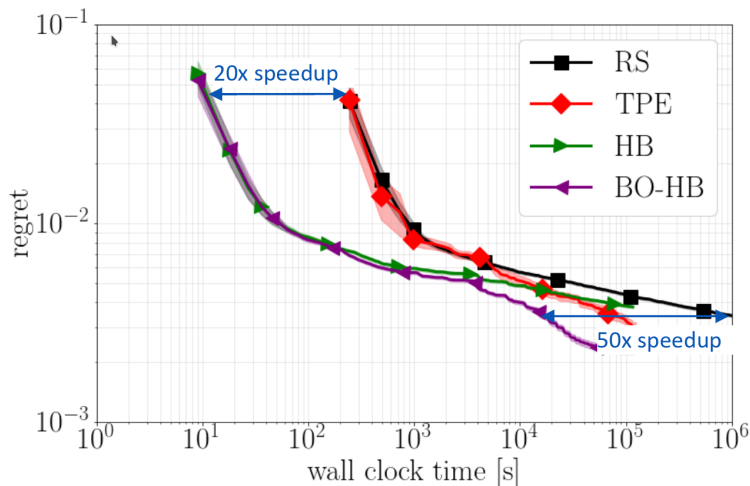


(a) CIFAR-10

Beyond Hyperband

- Hyperband is based on random search
 - ▶ Observations used to determine how many resources to spent on each configuration, but not used to select suitable configurations
 - ▶ Methods such as [BOHB](#) combine Bayesian optimization and Hyperband
 - ▶ Hyperband for quick improvements in the beginning
 - ▶ Bayesian optimization for good performance in the long run
- Hyperband uses a prespecified set of fidelities
 - ▶ Implicitly via parameter η and budget
 - ▶ Usually only a small number (e.g., ≤ 5) and only one type (e.g., iterations) of fidelities considered
 - ▶ One approach beyond: multi-fidelity BO methods such as [BOCA](#)

Example: BOHB



Best of both worlds: strong **anytime** and **final performance**

Auto-Net on dataset adult

Machine Learning

09 – Hyperparameter Optimization

Part 4: HPO in Practice

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2023-1

Which method to use?

- Choice of HPO method difficult and application-dependent
- No standard, well-accepted HPO benchmark yet
 - ▶ Evaluation protocols and metrics differ in the literature
 - ▶ In progress: [HPOBench](#), successor of [HPOlib](#)
 - ▶ Related: [COCO](#) for blackbox optimizers
 - ▶ Different authors use different evaluation methods
- Some criteria for choosing an HPO method
 - ▶ Structure of search space (small? large? real-valued? conditional?)
 - ▶ Number of possible evaluations (10s? 100s? 1000s?)
 - ▶ Degree of possible parallelization (1 worker? few? hundreds?)
 - ▶ Expertise and domain knowledge to set HPO hyperparameters
 - ▶ Multiple fidelities possible?
 - ▶ Extensibility and/or anytime properties needed?
- Recommendations of [Feurer and Hutter \(2019\)](#)
 - ▶ Multiple fidelities → BOHB
 - ▶ Real-valued, few trials → BO with Gaussian processes
 - ▶ Real-valued, many trials → CMA-ES
 - ▶ Large/categorical/conditional spaces → BO with SMAC or TPE

How to set HPO hyperparameters?

- HPO methods have hyperparameters themselves
 - ▶ Note: these are not the hyperparameters subject to HPO!
 - ▶ Even present for basic methods such as grid search (which grid?) or random search (which sampling distribution?)
 - ▶ Choice many have significant impact
- Manual specification
 - ▶ Implementations typically aim to provide sensible default choices
 - ▶ (Hopefully) easier to set by experts than hyperparameters of learning problem
- Estimate value of HPO hyperparameters from data
 - ▶ Using a point estimate (e.g., marginal likelihood of data in BO)
 - ▶ Problem: generally little data available (few trials)
 - ▶ Problem: Not always possible (e.g., grid points)
- Fully Bayesian treatment of HPO hyperparameters
 - ▶ Integrate out HPO hyperparameters
 - ▶ E.g., [possible for Gaussian processes](#) (length scales, covariance amplitude, observation noise, constant mean)

Can HPO lead to overfitting?

- Hyperparameters typically tuned on validation loss
 - ▶ No overfitting to training data
 - ▶ Validation data is limited, however
 - ▶ Overfitting to validation data is a concern
 - ▶ Observed empirically
- Strategies against overfitting
 - ▶ Vary training and validation splits across function evaluations
 - ▶ With BO, do not choose model with best observed performance, but with best estimated mean performance
 - ▶ Use a separate holdout set to detect HPO overfitting or reassess found configurations
 - ▶ Prefer **stable optima** (flat around optimum) over sharp optima
 - ▶ Use ensembles/Bayesian model selection (cf. 09-1, slide 5)

Machine Learning

A - Probability Refresher

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Probability theory is nothing but common sense reduced to calculation.

— *Pierre Laplace, 1812*

Basic terms

- **Sample space** Ω = set of outcomes
- **Event** $A \subseteq \Omega$ = some outcomes
 - ▶ Special cases: \emptyset (empty event), Ω (trivial event)
 - ▶ Can be complicated \rightarrow consider only events from some **event space** Σ
- Can combine events A and B
 - ▶ $A \cup B$, $A \cap B$, $A \setminus B$, $\bar{A} = \Omega \setminus A$, ...
 - ▶ **Disjoint** iff $A \cap B = \emptyset$
- **Probability space** = triple (Ω, Σ, p)
 - ▶ Informally, $p : \Sigma \rightarrow [0, 1]$ assigns a probability to events
 - ▶ Usual properties (e.g., $p(\Omega) = 1$)
- We will often think in terms of random variables
 - ▶ Ω and Σ implicit

Random variables

- We often associate random variables with data attributes
 - ▶ Both observed and unobserved ones
 - ▶ E.g., $p(\text{heart disease}=\text{true} \mid \text{age} \geq 50, \text{gender}=\text{male})$
 - ▶ E.g., $p(\text{shows a zero}=\text{true} \mid \text{image pixels})$
 - ▶ E.g., $p(\text{is spam}=\text{true} \mid \text{words in e-mail})$
 - ▶ E.g., $p(\text{stock tomorrow} \geq 100 \mid \text{stock history})$
- Technically, **random variables** (RV) are functions
 - ▶ $\text{Val}(X)$ = set of possible values
 - ▶ Random variable $X : \Omega \rightarrow \text{Val}(X)$,
 - ▶ Maps each outcome $\omega \in \Omega$ to a value $X(\omega)$ in $\text{Val}(X)$
 - ▶ $X = x$ means $\{\omega \mid X(\omega) = x\}$
 - ▶ $X \leq x$ means $\{\omega \mid X(\omega) \leq x\}$
 - ▶ Sample space often implicit (e.g., cross product of the possible-value sets of the considered variables)
- RV is **discrete** if possible values finite/countably infinite
 - ▶ E.g., $\text{Val}(X) = \{0, 1\}$ or $\text{Val}(X) = \mathbb{N}$
- RV is **continuous** if possible values uncountably infinite
 - ▶ E.g., $\text{Val}(X) = \mathbb{R}$

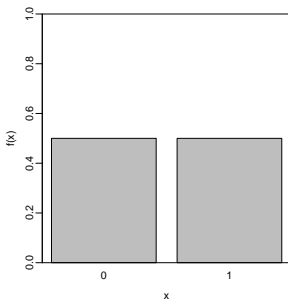
Discrete random variables (1)

- Described via **probability mass function** $f_X : \text{Val}(X) \rightarrow [0, 1]$

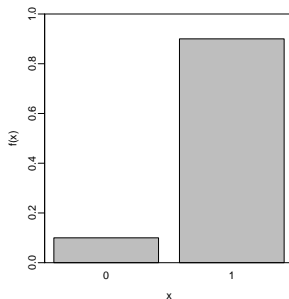
$$f_X(x) = p(X = x)$$

- Example: **Bernoulli distribution** $\text{Ber}(\theta)$
 - Models coin flip with probability θ of heads
 - $\text{Val}(X) = \{0, 1\} \rightarrow$ **binary** RV
 - $f_X(1) = \theta, f_X(0) = 1 - \theta$

Fair coin

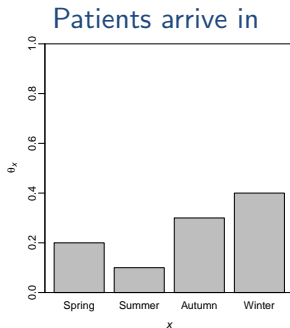


My special coin



Discrete random variables (2)

- Example: **categorical distribution** $\text{Cat}(\boldsymbol{\theta})$
 - ▶ Generalization of Bernoulli distribution to $k > 0$ categories
 - ▶ Probabilities of categories given by $\boldsymbol{\theta} = (\theta_1, \dots, \theta_k)$, summing to 1



- Others: binomial, multinomial, Poisson, empirical, ...

Union of events

- Given events A and B , the probability of “ A or B ” is given by

$$\begin{aligned} p(A \vee B) &= p(A) + p(B) - p(A \wedge B) \\ &= p(A) + p(B) \text{ if } A \text{ and } B \text{ are disjoint} \end{aligned}$$

- Generalizes to **principle of inclusion-exclusion**

$$p(\bigvee_{i=1}^n A_i) = \sum_{\emptyset \neq J \subseteq \{1, \dots, n\}} (-1)^{|J|-1} p(\bigwedge_{j \in J} A_j)$$

- Allows to derive upper and lower bounds, e.g., **union bound**

$$p(\bigvee_{i=1}^n A_i) \leq \sum_{i=1}^n p(A_i)$$

Joint probabilities and sum rule

- The joint probability of events A and B is given by

$$p(A, B) = p(A \wedge B)$$

- Sum rule** (law of total probability)

$$p(X = x) = \sum_{y \in \text{Val}(y)} p(X = x, Y = y)$$

- ▶ $p(X = x)$ is called the **marginal distribution**
- ▶ When we apply the sum rule, we say that we **marginalize out** Y

Example

$p(X, Y)$	H	T
$X=H$	0.1	0.2
$X=T$	0.3	0.4

$$\begin{aligned} p(X = H) &= \sum_{y \in \{H, T\}} p(X = H, Y = y) \\ &= p(X = H, Y = H) + p(X = H, Y = T) = 0.3 \end{aligned}$$

Conditional probability

- The **conditional probability** of A , given that B is true, is defined as

$$p(A|B) = \frac{p(A, B)}{p(B)} \quad \text{if } p(B) > 0$$

- Can be represented in a **conditional probability table** (CPT)

Joint probabilities		
$p(X, Y)$	H	T
$X=H$	0.1	0.2
$X=T$	0.3	0.4

CPT		
$p(X Y)$	H	T
$X=H$	0.25	0.33
$X=T$	0.75	0.66

Product rule

- Conditional probability: $p(A|B) = p(A, B)/p(B)$
- Implies the **product rule**

$$p(A, B) = p(A|B)p(B)$$

- Generalizes to **chain rule**

$$p(A_{1:n}) = p(A_1)p(A_2|A_1)p(A_3|A_1, A_2) \cdots p(A_n|A_{1:n-1})$$

Bayes' rule

- Using the product rule, we obtain **Bayes' rule**

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

- In terms of RVs:

$$p(X = x|Y = y) = \frac{p(X = x)p(Y = y|X = x)}{p(Y = y)}$$

- Many applications, foundation of Bayesian statistics (more later)

Example: Medical diagnosis

- A mammogram is a test for breast cancer
- Suppose you are a women in your 40s
- If you have cancer, test positive ($T = 1$) with probability 80%
- If you don't have cancer, test positive with prob. 10%
- About 0.4% of women in their 40s have breast cancer ($B = 1$)
- How likely is it that you have breast cancer if the test is positive?

$$\begin{aligned} p(B = 1|T = 1) &= \frac{p(B = 1)p(T = 1|B = 1)}{p(T = 1)} \\ &= \frac{0.004 \cdot 0.8}{0.004 \cdot 0.8 + 0.9996 \times 0.1} \\ &= 0.031 \end{aligned}$$

Independence

- We say X and Y are (unconditionally/marginally) **independent** iff

$$\forall x, y : p(X = x, Y = y) = p(X = x)p(Y = y)$$

- I.e., joint probability equals product of marginals
- Example: throw two coins X, Y
 - ▶ Assuming that the coins are fair and throws are independent, we have for example

$$p(X = \text{H}, Y = \text{T}) = p(X = \text{H})p(Y = \text{T}) = 0.25$$

- Very useful, thus special notation: $X \perp Y$
- Note: disjointness \neq independence

Conditional independence

- Unconditional independence is rare
- X and Y are **conditionally independent** given Z iff

$$\begin{aligned}\forall x, y, z : p(X = x, Y = y | Z = z) \\ = p(X = x | Z = z) p(Y = y | Z = z)\end{aligned}$$

- Denoted $X \perp Y \mid Z$
- Example
 - ▶ Event that it rains tomorrow (X)
 - ▶ Event that ground is currently wet (Y)
 - ▶ Event that it rains now (Z)
 - ▶ Then: $X \not\perp Y$ but $X \perp Y \mid Z$
- Note: $X \perp Y \mid Z$ does not imply $X \perp Y$
- Note: $X \perp Y$ does not imply $X \perp Y \mid Z$

Continuous random variables

- Described via **cumulative distribution function** (cdf)

$$F_X : \text{Val}(X) \rightarrow [0, 1]$$

$$F_X(x) = p(X \leq x)$$

- Non-decreasing
- $p(a \leq X \leq b) = F_X(b) - F_X(a)$ for $a \leq b$
- If derivative exists, we obtain the **probability density function** (pdf) $f_X : \text{Val}(X) \rightarrow \mathbb{R}^+$

$$f_X(x) = \frac{d}{dx} F_X(x)$$

- Probability of range corresponds to area below pdf:

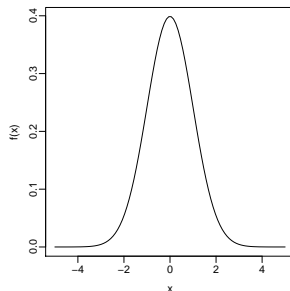
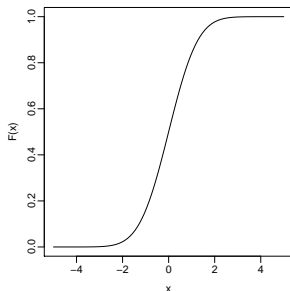
$$p(a \leq X \leq b) = \int_a^b f_X(x) \, dx$$

- Examples: uniform, Gaussian, Student's t , Laplace, Gamma, Beta, Dirichlet, ...

The Gaussian Distribution (1)

- **Normal distribution** $\mathcal{N}(\mu, \sigma^2)$
 - ▶ μ is a mean parameter
 - ▶ σ^2 is a variance parameter (sometimes: precision $\lambda = 1/\sigma^2$)
 - ▶ **Standard normal distribution**: $\mathcal{N}(0, 1)$
 - ▶ Most widely used continuous distribution in statistics and ML
- pdf given by

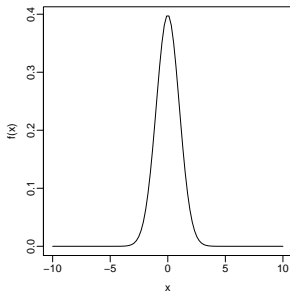
$$\mathcal{N}(x|\mu, \sigma^2) = \underbrace{\frac{1}{\sqrt{2\pi\sigma^2}}}_{\text{cdf}} \exp \left(\underbrace{-\frac{1}{2\sigma^2}(x - \mu)^2}_{\text{pdf}} \right)$$



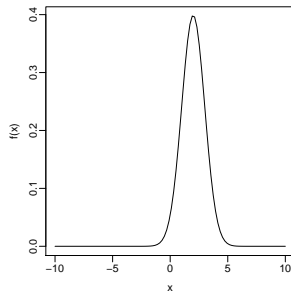
The Gaussian Distribution (2)

$$\sigma^2 = 1$$

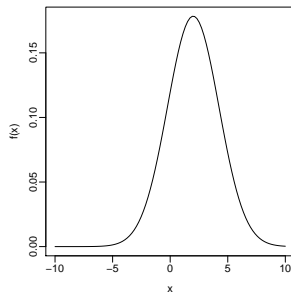
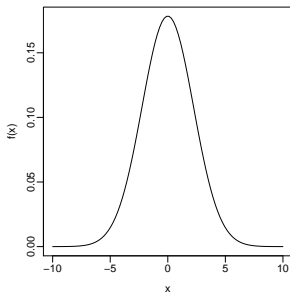
$$\mu = 0$$



$$\mu = 2$$



$$\sigma^2 = 5$$



Mean, variance, covariance

- The **mean** or **expected value** μ of a RV is given by

$$E[X] = \sum_{x \in \text{Val}(x)} x f_X(x) \quad \text{if } X \text{ is discrete}$$

$$E[X] = \int_x x f_X(x) \, dx \quad \text{if } X \text{ is continuous}$$

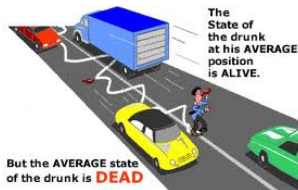
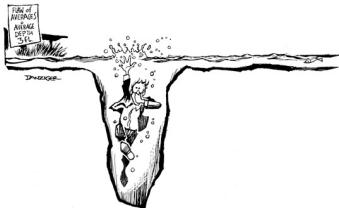
- The **variance** σ^2 is given by

$$\text{var}[X] = E[(X - \mu)^2]$$

- Useful fact: $\text{var}[X] = E[X^2] - E[X]^2$
- The **covariance** is given by

$$\text{Cov}[X, Y] = E[(X - E[X])(Y - E[Y])]$$

“Flaw of averages”



Mean correct, variance ignored

$$E[g(X)] \neq g(E[X])$$

Be careful with expected values!

Notation

We will work with many random variables.

- We write $X \sim \mathcal{N}(0, 1)$ to say that r.v. X has the specified distr.
- We write $p(X = x)$ to refer to the pmf (when X discrete) or pdf (continuous)
- We often drop the r.v. from our notation (when clear from text)
 - ▶ Write $p(x)$ instead of $p(X = x)$ (**marginal distribution**)
 - ▶ Write $p(x, y)$ instead of $p(X = x, Y = y)$ (**joint distribution**)
 - ▶ Write $p(x|y)$ instead of $p(X = x|Y = y)$ (**conditional distribution**)
- $p(x)$ can refer to a probability (x fixed) or a distribution (x variable)
- We write $X \perp Y$ if X and Y are independent
- We write $X \perp Y|Z$ if X and Y are conditionally independent given Z

Important properties

$$p(A \cup B) = p(A) + p(B) - p(A \cap B) \quad (\text{inclusion-exclusion})$$

$$p(\bar{A}) = 1 - p(A)$$

$$\text{If } B \supseteq A, \quad p(B) = p(A) + p(B \setminus A) \geq p(A)$$

$$p(X, Y) = p(Y|X)p(X) \quad (\text{product rule})$$

$$p(X) = \sum_y p(X, y) \quad (\text{sum rule, } y \text{ discrete})$$

$$p(X) = \int_y p(X, y) \, dy \quad (\text{sum rule, } y \text{ continuous})$$

$$p(X|Y) = \frac{p(Y|X)p(X)}{p(Y)} \quad (\text{Bayes theorem})$$

$$E[aX + b] = aE[X] + b \quad (\text{linearity of expectation})$$

$$E[X + Y] = E[X] + E[Y]$$

$$E_Y[E_X[X|Y]] = E[X] \quad (\text{law of total expectation})$$

Literature

- Murphy, Ch. 2, *Probability: Univariate Models*
- Goodfellow et al., Ch. 3: *Probability and Information Theory*

Machine Learning

B – Vectors & Matrices

Prof. Dr. Rainer Gemulla

Universität Mannheim

Version: 2022-1

Outline

1. Vectors
2. Matrices
3. Summary

Outline

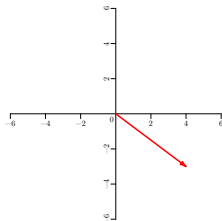
1. Vectors
2. Matrices
3. Summary

Vector

A **vector** is

- A 1D array of numbers
- A geometric entity with magnitude and direction
- A matrix with exactly one row or column
 - ▶ Called row vector and column vector, resp.
 - ▶ **Transpose** v^\top transposes a row vector into a column vector and vice versa
- A (latent) object/example
- A (latent) attribute/feature

	<i>Year</i>
<i>Stockholm</i>	9.95
<i>Minsk</i>	10.77
<i>London</i>	14.85
<i>Budapest</i>	14.91
<i>Paris</i>	15.46
<i>Bucharests</i>	16.44
<i>Barcelona</i>	19.90
<i>Rome</i>	20.44
<i>Lisbon</i>	21.36
<i>Athens</i>	22.31
<i>Valencia</i>	22.36
<i>Malta</i>	23.35



$$\begin{pmatrix} 4 \\ -3 \end{pmatrix}$$

	<i>Jan</i>	<i>Apr</i>	<i>Jul</i>	<i>Oct</i>	<i>Year</i>
<i>Stockholm</i>	-0.70	8.60	21.90	9.90	10.00

Vector norm

The **norm** of vector defines its magnitude. Let

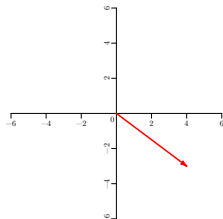
$$\mathbf{v} = (v_1 \ v_2 \ \cdots \ v_n)^\top.$$

- **Euclidean norm:** $\|\mathbf{v}\| = \sqrt{\sum_{i=1}^n v_i^2}$
 - ▶ Corresponds to intuitive notion of length in Euclidean space
- ℓ_p **norm** for $1 \leq p \leq \infty$: $\|\mathbf{v}\|_p = (\sum_{i=1}^n |v_i|^p)^{1/p}$
 - ▶ ℓ_1 norm = sum of absolute values (Manhattan distance from origin)
 - ▶ ℓ_2 norm = Euclidean norm (bird-fly distance from origin)
 - ▶ ℓ_∞ norm = maximum absolute value
 - ▶ The ℓ_p norms never increase as p increases, i.e.,

$$\|\mathbf{v}\|_{p+a} \leq \|\mathbf{v}\|_p \quad \text{for } a \geq 0$$

- Properties of vector norms

- ▶ $\|\mathbf{v}\| > 0$ when $\mathbf{v} \neq \mathbf{0}$ and $\|\mathbf{v}\| = 0$ iff $\mathbf{v} = \mathbf{0}$
- ▶ $\|a\mathbf{v}\| = |a| \|\mathbf{v}\|$ (absolute scalability)
- ▶ $\|\mathbf{v}_1 + \mathbf{v}_2\| \leq \|\mathbf{v}_1\| + \|\mathbf{v}_2\|$ (triangle inequality)



$$\mathbf{v} = \begin{pmatrix} 4 \\ -3 \end{pmatrix}$$

$$\|\mathbf{v}\|_1 = 7$$

$$\|\mathbf{v}\| = 5$$

$$\|\mathbf{v}\|_\infty = 4$$

Norms and distances

The **distance** between two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ can be quantified with norm $\|\mathbf{u} - \mathbf{v}\|$.

- | | <i>Jan</i> | <i>Apr</i> | <i>Jul</i> | <i>Oct</i> | <i>Year</i> |
|-----------------------------|------------|------------|------------|------------|-------------|
| • Stockholm, $\mathbf{s} =$ | (-0.70 | 8.60 | 21.90 | 9.90 | 10.00) |
| • Minsk, $\mathbf{m} =$ | (-2.10 | 12.20 | 23.60 | 10.20 | 10.60) |
| • Athens, $\mathbf{a} =$ | (12.90 | 20.30 | 32.60 | 23.10 | 22.30) |

ℓ_1	\mathbf{s}	\mathbf{m}	\mathbf{a}
\mathbf{s}	0.00	7.60	61.50
\mathbf{m}	7.60	0.00	56.70
\mathbf{a}	61.50	56.70	0.00

ℓ_2	\mathbf{s}	\mathbf{m}	\mathbf{a}
\mathbf{s}	0.00	4.27	27.60
\mathbf{m}	4.27	0.00	25.98
\mathbf{a}	27.60	25.98	0.00

ℓ_∞	\mathbf{s}	\mathbf{m}	\mathbf{a}
\mathbf{s}	0.00	3.60	13.60
\mathbf{m}	3.60	0.00	15.00
\mathbf{a}	13.60	15.00	0.00

Dot product (algebraic definition)

The **dot product** of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is given by

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i.$$

- Also known as **scalar product**
- An **inner product** for Euclidean space: $\langle \mathbf{u}, \mathbf{v} \rangle$
- Matrix product of a row and a column vector: $\mathbf{u}^\top \mathbf{v}$
- Properties (with $a, b \in \mathbb{R}$)
 - ▶ $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$
 - ▶ $(a\mathbf{u}) \cdot \mathbf{v} = a(\mathbf{u} \cdot \mathbf{v})$
 - ▶ $(a\mathbf{u} + b\mathbf{v}) \cdot \mathbf{w} = (a\mathbf{u}) \cdot \mathbf{w} + (b\mathbf{v}) \cdot \mathbf{w}$
- Many uses, many interpretations

With dot products, we can ...

- Compute the (squared) Euclidean norm

$$\mathbf{v} \cdot \mathbf{v} = \sum_{i=1}^n v_i^2 = \|\mathbf{v}\|^2$$

- Determine the value of a coordinate

$$v_i = \mathbf{v} \cdot \mathbf{e}_i,$$

where \mathbf{e}_i denotes the i -th **standard basis vector** (i.e., $[\mathbf{e}_i]_j = 1$ if $i = j$ else 0)

- Compute the sum of the elements of a vector

$$\mathbf{v} \cdot \mathbf{1}_n = \sum_{i=1}^n v_i,$$

where $\mathbf{1}_n$ is the all-ones vector of dimensionality n

- ...

Dot product: Weighted sum

The elements of one vector are interpreted as weights for the elements of the other vector.

Example: Anna goes shopping

Item	Bread	Butter	Pizza
Price/piece	1 €	0.50 €	3 €
Quantity bought	1	2	5

- How much does Anna pay?
- Prices can be interpreted as “weights”: $\mathbf{p} = (1 \quad 0.5 \quad 3)^\top$
- Quantities are $\mathbf{n} = (1 \quad 2 \quad 5)^\top$
- Total is $\mathbf{p} \cdot \mathbf{n} = 1 \cdot 1 + 0.5 \cdot 2 + 3 \cdot 5 = 17$
- Similarly: Can interpret quantities as weights for prices

Dot product: Expected value

One vector corresponds to probabilities, the other one to a random variable.

Example: Bob is gambling

Outcome	Jackpot	Win	Loss
Probability	0.1	0.2	0.7
Amount won	5€	1€	-2€

- How much does Bob win in expectation? (Should he play?)
- Probabilities $\mathbf{p} = (0.1 \ 0.2 \ 0.7)^\top$
 - ▶ A non-negative vector that sums to one ($\|\mathbf{p}\|_1 = 1$) is called a **probability vector**
 - ▶ Corresponds to a probability distribution over a finite set of outcomes
- Amounts won $\mathbf{x} = (5 \ 1 \ -2)^\top$
 - ▶ Corresponds to a random variable; associates a real value with each outcome
- Expected value $\mathbf{p} \cdot \mathbf{x} = 0.1 \cdot 5 + 0.2 \cdot 1 + 0.7 \cdot (-2) = -0.7$

Dot product: Sample variance

Denote by $\bar{u} = \frac{1}{n} \sum_i u_i$ the mean of \mathbf{u} . If we treat the entries of \mathbf{u} as samples from some distribution, then the **sample variance** is given by

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (u_i - \bar{u})^2 = \frac{\|\mathbf{u} - \bar{\mathbf{u}}\|^2}{n-1} = \frac{(\mathbf{u} - \bar{\mathbf{u}}) \cdot (\mathbf{u} - \bar{\mathbf{u}})}{n-1},$$

where $\bar{\mathbf{u}}$ denotes the **sample mean vector**, i.e., $[\bar{\mathbf{u}}]_i = \bar{u}$ for $1 \leq i \leq n$.

- Example

- ▶ $\mathbf{u} = (10 \ 11 \ 12)^\top$
- ▶ $\bar{u} = 11, \bar{\mathbf{u}} = (11 \ 11 \ 11)^\top$
- ▶ $\mathbf{u} - \bar{\mathbf{u}} = (-1 \ 0 \ 1)^\top$
- ▶ $s^2 = 1, \|\mathbf{u}\|^2 = 365$

- Variances are thus closely related to norms; the key difference is centering and averaging
- When we center data before analyzing it, dot products are proportional to variances ($\mathbf{u} \cdot \mathbf{u}$) or covariances ($\mathbf{u} \cdot \mathbf{v}$)

Dot product: Sets and intersections

The **indicator vector** of a subset T of a set $S = \{s_1, \dots, s_n\}$ is the vector \mathbf{x} such that $x_i = 1$ if $s_i \in T$ and $x_i = 0$ if $s_i \notin T$. If \mathbf{u} and \mathbf{v} are indicator vectors for subsets $U, V \subseteq S$, resp., then $\mathbf{u} \cdot \mathbf{v} = |U \cap V|$.

- $S = \{\text{France, Germany, Denmark, Poland}\}$
- Anna visited France, Germany, and Poland: $\mathbf{u} = (1 \ 1 \ 0 \ 1)^\top$
- Bob visited Germany, Denmark, and Poland: $\mathbf{v} = (0 \ 1 \ 1 \ 1)^\top$
- Number of countries visited by both:

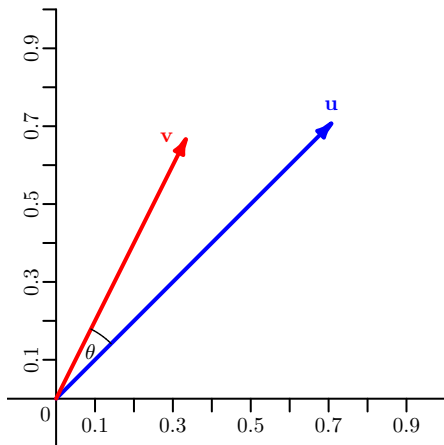
$$\mathbf{u} \cdot \mathbf{v} = 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 = 2 = |\{\text{Germany, Poland}\}|$$

Dot product (geometric definition)

An alternative geometric definition of the dot product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ is

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta,$$

where $-\pi \leq \theta \leq \pi$ denotes the angle between \mathbf{u} and \mathbf{v} .



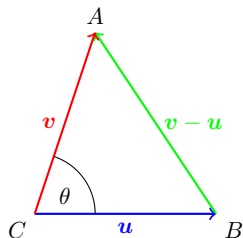
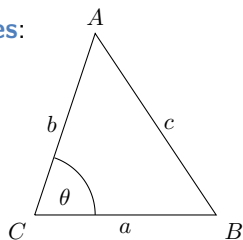
Why is this?

Let's focus on the 2D case. Recall the **law of cosines**:

$$c^2 = a^2 + b^2 - 2ab \cos \theta.$$

Now set $\mathbf{u} = B - C$ and $\mathbf{v} = A - C$ and observe that $\mathbf{v} - \mathbf{u} = A - B$.

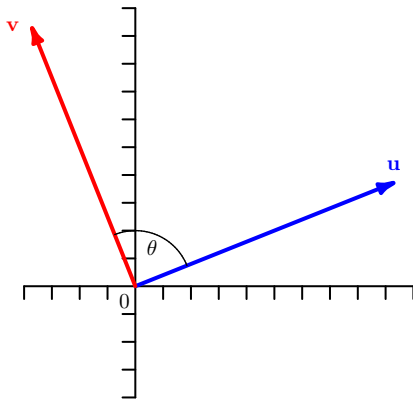
$$\begin{aligned}\cos \theta &= \frac{a^2 + b^2 - c^2}{2ab} = \frac{\|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 - \|\mathbf{v} - \mathbf{u}\|^2}{2 \|\mathbf{u}\| \|\mathbf{v}\|} \\&= \frac{\mathbf{u} \cdot \mathbf{u} + \mathbf{v} \cdot \mathbf{v} - (\mathbf{v} - \mathbf{u}) \cdot (\mathbf{v} - \mathbf{u})}{2 \|\mathbf{u}\| \|\mathbf{v}\|} \\&= \frac{\mathbf{u} \cdot \mathbf{u} + \mathbf{v} \cdot \mathbf{v} - \mathbf{v} \cdot \mathbf{v} + 2\mathbf{u} \cdot \mathbf{v} - \mathbf{u} \cdot \mathbf{u}}{2 \|\mathbf{u}\| \|\mathbf{v}\|} \\&= \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}\end{aligned}$$



Dot product: Test for orthogonality

Two nonzero vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ are **orthogonal** iff $\mathbf{u} \cdot \mathbf{v} = 0$.

- Since $0 = \mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$ and $\|\mathbf{u}\|, \|\mathbf{v}\| > 0$, we have $\cos \theta = 0$
- And this means that the angle is 90 degrees



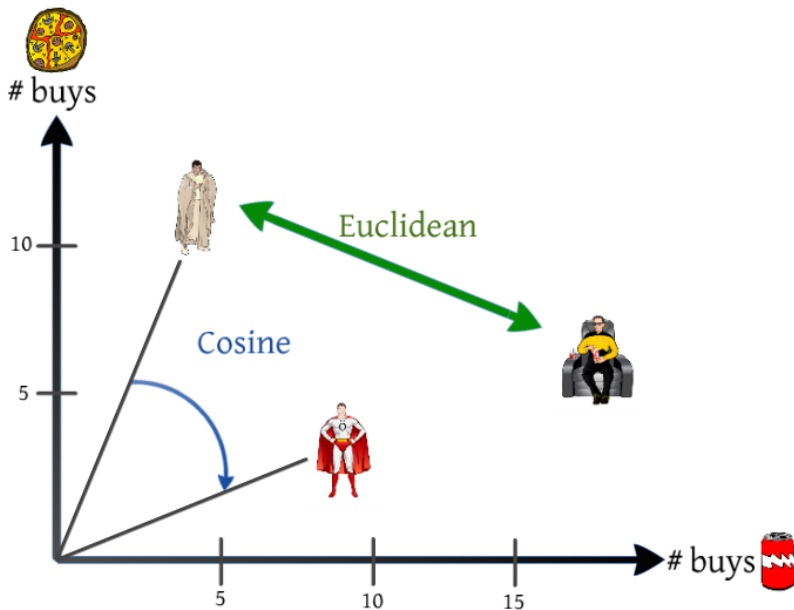
Dot product: Cosine similarity (1)

The angle between \mathbf{u} and \mathbf{v} is another way to measure the similarity between two vectors. The **cosine similarity** of \mathbf{u} and \mathbf{v} is given by

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}.$$

- $-1 \leq \cos(\mathbf{u}, \mathbf{v}) \leq 1$
- Vectors that point in roughly the same direction
→ small angle → cosine similarity ≈ 1
- Vectors that point in roughly opposite directions
→ large angle → cosine similarity ≈ -1
- Vectors that are roughly orthogonal
→ roughly right angle → cosine similarity ≈ 0
- Example: Determine the similarity between a document and a query in IR

Dot product: Cosine similarity (2)

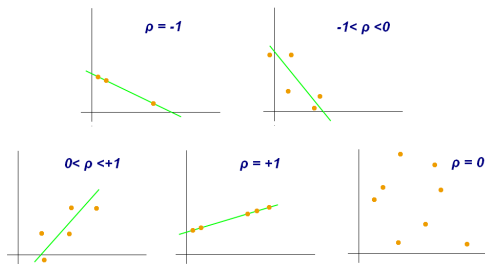


Dot product: Pearson correlation

The **sample Pearson correlation coefficient** is a measure of linear correlation. It is given by

$$r_{x,y} = \frac{(x - \bar{x}) \cdot (y - \bar{y})}{\|x - \bar{x}\| \|y - \bar{y}\|}.$$

- Numerator proportional to the sample covariance
- Denominator proportional to sample standard deviations
- Related to cosine similarity, but performs centering



Dot product: Similarity

The dot product itself can also be seen as a measure of similarity or compatibility. Recall

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta.$$

Example: Shopping transactions

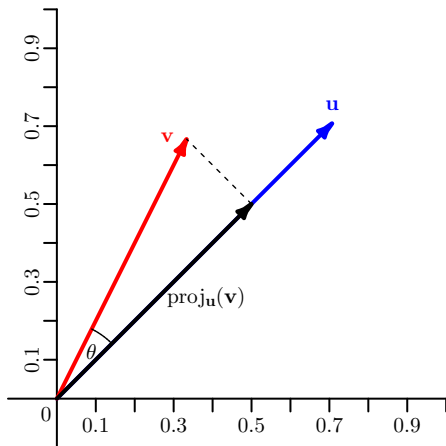
- Like in previous example, vectors \mathbf{u} and \mathbf{v} correspond to persons
- Elements corresponds to frequencies of buying each product
- We can think of the direction of a vector as “preference”
 - ▶ Which products are being bought?
 - ▶ $\cos \theta$ large when \mathbf{u} and \mathbf{v} have similar interest
- We can think of the magnitude of a vector as “strength”
 - ▶ How much is being bought?
 - ▶ $\|\mathbf{u}\| \|\mathbf{v}\|$ large when both persons buy a lot
- If $\mathbf{u} \cdot \mathbf{v}$ is large, \mathbf{u} and \mathbf{v} have similar shopping behavior and buy a lot

Dot product: Projection

The **vector projection** of v onto u is given by

$$\text{proj}_u(v) = \underbrace{\frac{u \cdot v}{\|u\|}}_{\text{scalar projection}} \frac{u}{\|u\|} = \frac{u \cdot v}{\|u\|^2} u$$

- The scalar projection describes how far v points in the direction u
- The vector projection is a vector pointing this far in the direction of u
- Note: norm of u (when $\neq 0$) does not affect result, only direction does



Outline

1. Vectors
2. Matrices
3. Summary

Notation

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a real $m \times n$ matrix. We write

- a_{ij} or \mathbf{A}_{ij} (both scalars) for the value of entry (i, j)
- \mathbf{a}_j or $\mathbf{A}_{:j}$ (both column vectors) for the j -th column of \mathbf{A}
- \mathbf{a}_i (column vector) or $\mathbf{A}_{i:}$ (row vector) for the i -th row of \mathbf{A}

Thus

$$\begin{aligned}\mathbf{A} &= \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \\ &= (\mathbf{A}_{:1} \quad \mathbf{A}_{:2} \quad \cdots \quad \mathbf{A}_{:n}) = \begin{pmatrix} \mathbf{A}_{1:} \\ \mathbf{A}_{2:} \\ \vdots \\ \mathbf{A}_{m:} \end{pmatrix}\end{aligned}$$

Full matrix ring (addition)

The set of all matrices in $\mathbb{R}^{n \times n}$ form a ring, the **full matrix ring**.

- Addition and subtraction are element-wise

$$[\mathbf{A} + \mathbf{B}]_{ij} = a_{ij} + b_{ij}$$

$$[\mathbf{A} - \mathbf{B}]_{ij} = a_{ij} - b_{ij}$$

- Addition is associative and commutative
- The additive identity is the $n \times n$ zero matrix $\mathbf{0}_{n \times n}$
- The additive inverse is $-\mathbf{A}$ with $[-\mathbf{A}]_{ij} = -a_{ij}$
- In general $[c\mathbf{A}]_{ij} = ca_{ij}$ for $c \in \mathbb{R}$ (scalar multiplication)

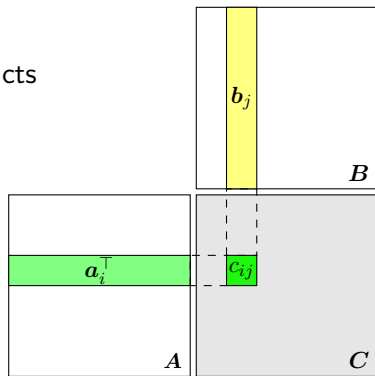
Full matrix ring (multiplication)

- For multiplication, we take dot products

$$[AB]_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j = \sum_{k=1}^n a_{ik} b_{kj}$$

- The multiplicative identity is the $n \times n$ **identity matrix** I_n

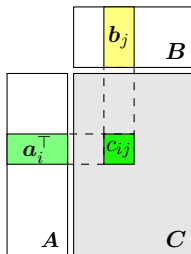
$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$



- Multiplication is associative, but not commutative ($AB \neq BA$ in general)
- Multiplication distributes over addition ($A(B + C) = AB + AC$ and $(B + C)A = BA + CA$)
- Multiplication does not always have an inverse (division)

Rectangular matrices

- We generally have rectangular matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$
- We can only add and subtract matrices of the same dimensions ($\mathbf{A}_{m \times n} + \mathbf{B}_{m \times n}$)
- We can only multiply matrices with a matching inner dimension
 - ▶ We can multiply $\mathbf{A} \in \mathbb{R}^{m \times r}$ with $\mathbf{B} \in \mathbb{R}^{r \times n}$ (inner dimension is r)
 - ▶ Gives an $m \times n$ matrix (outer dimensions)
 - ▶ $[\mathbf{AB}]_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j = \sum_{k=1}^r a_{ik} b_{kj}$



Interpretation for matrix multiplication (1)

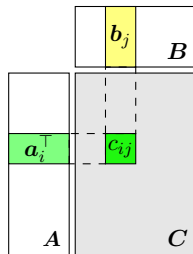
When we multiply A and B , we compute all dot products between rows of A and columns of B .

- We can apply any of the interpretations of the dot product
- E.g., weighted sum

- ▶ m supermarkets, r products, n persons
- ▶ a_{ik} = price of product k at supermarket i
- ▶ b_{kj} = quantity of product k bought by person j
- ▶ $[AB]_{ij}$ = how much the j -th person would pay when buying at the i -th supermarket

- E.g., covariance

- ▶ If all columns of $A_{m \times n}$ are centered ($\sum_k a_{ik} = 0$), then $\frac{1}{m-1} A^\top A \in \mathbb{R}^{n \times n}$ is the **sample covariance matrix**
- ▶ $[\frac{1}{m-1} A^\top A]_{ii}$ holds the sample variance of column i
- ▶ $[\frac{1}{m-1} A^\top A]_{ij}$ holds the sample covariance between columns i and j



Interpretation for matrix multiplication (2)

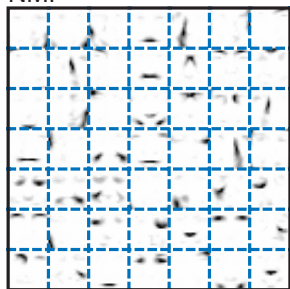
We can also interpret rows i of \mathbf{AB} as a linear combination of the rows of \mathbf{B} with the coefficients coming from \mathbf{A}

$$[\mathbf{AB}]_{i:} = a_{i1}\mathbf{B}_{1:} + a_{i2}\mathbf{B}_{2:} + \cdots + a_{ir}\mathbf{B}_{r:},$$

and, similarly, the columns of \mathbf{AB} as linear combinations of the columns of \mathbf{A}

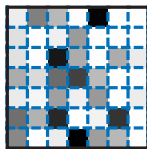
$$[\mathbf{AB}]_{:j} = b_{1j}\mathbf{A}_{:1} + b_{2j}\mathbf{A}_{:2} + \cdots + b_{rj}\mathbf{A}_{:r}.$$

NMF



\mathbf{A} (each square is a column)

\times



$\mathbf{B}_{:j}$

$=$



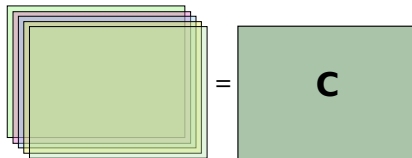
$[\mathbf{AB}]_{:j}$

Interpretation for matrix multiplication (3)

We can view matrix \mathbf{AB} as the sum of the r **component matrices** obtained by multiplying the k -th column of \mathbf{A} and the k -th row of \mathbf{B} :

$$\mathbf{AB} = \mathbf{A}_{:1}\mathbf{B}_{1:} + \mathbf{A}_{:2}\mathbf{B}_{2:} + \cdots + \mathbf{A}_{:r}\mathbf{B}_{r:}$$

- Components $\mathbf{A}_{:k}\mathbf{B}_{k:}$ are outer products ($m \times n$ matrices)
- Note: when $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, the matrix product
 - ▶ $\mathbf{u}^\top \mathbf{v}$ corresponds to a dot product (a scalar), $m = n$ required
 - ▶ $\mathbf{u}\mathbf{v}^\top$ corresponds to an **outer product** (an $m \times n$ matrix)
- In our supermarket example
 - ▶ Components correspond to products
 - ▶ Entry (i, j) of k -th component indicates how much the j -th person would pay for product k when buying at the i -th supermarket



Transposes

The **matrix transpose** \mathbf{A}^\top switches rows and columns, i.e.,

$$[\mathbf{A}^\top]_{ij} = a_{ji}.$$

The following properties hold

- $(\mathbf{A}^\top)^\top = \mathbf{A}$
- $(\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top$
- $(c\mathbf{A})^\top = c\mathbf{A}^\top$
- $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$

Summing and scaling

Let $A \in \mathbb{R}^{m \times n}$. Denote by $\mathbf{1}_n$ the all-ones vector of dimensionality n . For $s \in \mathbb{R}^n$, denote by $\text{diag}(s)$ the $n \times n$ matrix with the entries of s on the main diagonal:

$$\text{diag}(s) = \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_n \end{pmatrix}$$

- $A\mathbf{1}_n$ computes the **row sums** of A
- $\mathbf{1}_m^\top A$ computes the **column sums** of A
- $A \text{diag}(c)$ **scales each column** j of A by c_j , $c \in \mathbb{R}^n$
- $\text{diag}(r) A$ **scales each row** i of A by r_i , $r \in \mathbb{R}^m$

Matrices as linear maps

- A matrix $A \in \mathbb{R}^{m \times n}$ is a **linear map** from \mathbb{R}^n to \mathbb{R}^m
 - ▶ If $x \in \mathbb{R}^n$, then $y = Ax \in \mathbb{R}^m$ is the **image** of x
 - ▶ $y = \sum_{j=1}^n a_j x_j$, i.e., a linear combination of the columns of A
- If $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, then AB maps from \mathbb{R}^n to \mathbb{R}^m
 - ▶ Product AB corresponds to composition of linear maps A and B
- Square matrix $A \in \mathbb{R}^{n \times n}$ is **invertible** (= **nonsingular**) iff there is matrix $B \in \mathbb{R}^{n \times n}$ such that $AB = I$
 - ▶ Matrix B is the **inverse** of A , denoted A^{-1}
 - ▶ If A is invertible, then $AA^{-1} = A^{-1}A = I$
 - ▶ $AA^{-1}x = A^{-1}Ax = x$
 - ▶ Non-square matrices do not have (general) inverses but can have **right** or **left inverses**: $AR = I$ or $LA = I$
- The **transpose** of $A \in \mathbb{R}^{m \times n}$ is a linear map $A^\top : \mathbb{R}^m \rightarrow \mathbb{R}^n$
 - ▶ $(A^\top)_{ij} = A_{ji}$
 - ▶ Generally, transpose is **not** the inverse ($AA^\top \neq I$)

Matrix norms

- Matrix norms measure the magnitude of the matrix
 - ▶ Magnitude of the values in the matrix
 - ▶ Magnitude of the image
- **Operator norms** measure how large the image of a unit vector can get
 - ▶ Induced by a vector norm
 - ▶ For $p \geq 1$, $\|\mathbf{A}\|_p = \max\{\|\mathbf{A}\mathbf{x}\|_p \mid \|\mathbf{x}\|_p = 1\}$
 - ▶ $\|\mathbf{A}\|_1$ = maximum sum of absolute values of a column
 - ▶ $\|\mathbf{A}\|_\infty$ = maximum sum of absolute values of a row
 - ▶ **Spectral norm**: $\|\mathbf{A}\|_2$ = largest singular value of \mathbf{A} (more later)
- The **Frobenius norm** is the vector- ℓ_2 norm applied to the elements of a matrix (treating them as a vector)
 - ▶ $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$
 - ▶ Note: $\|\mathbf{A}\|_F \neq \|\mathbf{A}\|_2$

Matrix rank and linear independence

- A vector $\mathbf{u} \in \mathbb{R}^n$ is **linearly dependent** on set of vectors $V = \{\mathbf{v}_i\} \subset \mathbb{R}^n$ if \mathbf{u} can be expressed as a linear combination of vectors in V
 - ▶ $\mathbf{u} = \sum_i a_i \mathbf{v}_i$ for some $a_1, \dots, a_n \in \mathbb{R}$
 - ▶ Set V is linearly dependent if some $\mathbf{v}_i \in V$ is linearly dependent on $V \setminus \{\mathbf{v}_i\}$
 - ▶ If V is not linearly dependent, it is **linearly independent**
- The **column rank** of matrix \mathbf{A} is the maximum number of linearly independent columns of \mathbf{A}
- The **row rank** of \mathbf{A} is the maximum number of linearly independent rows of \mathbf{A}
- The **Schein rank** of \mathbf{A} is the least integer r such that $\mathbf{A} = \mathbf{L}\mathbf{R}$ for some $\mathbf{L} \in \mathbb{R}^{m \times r}$ and $\mathbf{R} \in \mathbb{R}^{r \times n}$
 - ▶ Equivalently, the least r such that \mathbf{A} is a sum of r vector outer products
- **All these ranks are equivalent**
 - ▶ E.g., matrix has rank 1 iff it is an outer product of two (non-zero) vectors

Matrices as systems of linear equations

- A matrix can hold the coefficients of a system of linear equations (c.f. Chinese *Nine Chapters on Arithmetic*)

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{array} \Leftrightarrow \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

- If the coefficient matrix \mathbf{A} is invertible, the system has exact solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
- If $m < n$ the system is **underdetermined** and can have an infinite number of solutions
- If $m > n$ the system is **overdetermined** and (usually) does not have an exact solution
- The **least-squares** solution is the vector \mathbf{x} that minimizes $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ (cf. linear regression)

Special types of matrices

- The **diagonals** of matrix A go from top-left to bottom-right
 - ▶ The main diagonal contains the elements a_{ii}
 - ▶ The k -th upper diagonal contains the elements $a_{i,(i+k)}$
 - ▶ The k -th lower diagonal contains the elements $a_{(i+k),i}$
 - ▶ The **anti-diagonals** go from top-right to bottom-left
- Matrix is **diagonal** if all its non-zero values are in a diagonal (typically main diagonal)
 - ▶ **Bi-diagonal** matrices have values in two diagonals, etc.
- Matrix A is **upper (right) triangular** if all of its non-zeros are in or above the main diagonal
 - ▶ **Lower (left) triangular** matrices have all non-zeros in or below main diagonal
 - ▶ Upper left and lower right triangular matrices: replace diagonal with anti-diagonal
- A square matrix P is **permutation matrix** if each row and each column of P has exactly one 1 and rest are 0s
 - ▶ If P is a permutation matrix, PA permutes the order of the rows and AP the order of the columns of A

Orthogonal matrices

- A set $V = \{\mathbf{v}_i\} \subset \mathbb{R}^n$ is **orthogonal** if all vectors in V are mutually orthogonal
 - ▶ $\mathbf{v} \cdot \mathbf{u} = 0$ for all $\mathbf{v} \neq \mathbf{u} \in V$
 - ▶ If all vectors in V also have unit norm ($\|\mathbf{v}\|_2 = 1$), V is **orthonormal**
- A square matrix \mathbf{A} is **orthogonal** if its columns are a set of orthonormal (!) vectors or equivalently
 - ▶ Its rows are orthonormal
 - ▶ $\mathbf{A}^\top \mathbf{A} = \mathbf{I}_n$
 - ▶ $\mathbf{A}^{-1} = \mathbf{A}^\top$
- An $m \times n$ matrix \mathbf{A} is
 - ▶ **column-orthogonal** if columns are a set of orthonormal vectors (only possible if $m \geq n$); then \mathbf{A}^\top is left inverse ($\mathbf{A}^\top \mathbf{A} = \mathbf{I}_n$)
 - ▶ **row-orthogonal** if rows are a set of orthonormal vectors (only possible if $m \leq n$); then \mathbf{A}^\top is right inverse ($\mathbf{A} \mathbf{A}^\top = \mathbf{I}_m$)
- If \mathbf{A} and \mathbf{B} are orthogonal, so is \mathbf{AB}
 - ▶ Similarly: column-orthogonality and row-orthogonality is preserved

Outline

1. Vectors
2. Matrices
3. Summary

Lessons learned

- Many uses, many interpretations
 - ▶ Vectors
 - ▶ Matrices
 - ▶ Dot products
 - ▶ Matrix products
- Magnitudes and distances are measured by norms
- Basic concepts of linear algebra
- Special types of matrices: diagonal, triangular, orthogonal

Suggested reading

- Murphy, Ch. 7.1–7.3.1
- A linear algebra text book such as
 - ▶ Carl Meyer
Matrix Analysis and Applied Linear Algebra
Society for Industrial and
Applied Mathematics, 2000
<http://www.matrixanalysis.com> (used to be freely available)
- [Wolfram MathWorld](#) articles
- [Wikipedia](#) articles