

Data Mining

Classification

- Part 3 -



Outline

1. What is Classification?
2. K-Nearest-Neighbors
3. Decision Trees
4. Model Evaluation
5. Rule Learning
6. Naïve Bayes
7. Support Vector Machines
8. Artificial Neural Networks
9. Hyperparameter Selection

6. Naïve Bayes

- Probabilistic classification technique based on Bayes theorem
 - successful, old-school method for various tasks: NLP, recommendation, ...
- Goal: Estimate the most probable class label for a given record
- Probabilistic formulation of the classification task:
 - consider each attribute and class label as random variables
 - given a record with attributes (A_1, A_2, \dots, A_n) , the goal is to find the class C that maximizes the conditional probability

$$P(C | A_1, A_2, \dots, A_n)$$

- Example: Should we play golf?
 - $P(\text{Play=yes} | \text{Outlook=rainy}, \text{Temperature=cool})$
 - $P(\text{Play=no} | \text{Outlook=rainy}, \text{Temperature=cool})$
- Question: How to estimate these probabilities given training data?

Bayes Theorem

- Thomas Bayes (1701-1761)
 - British mathematician and priest
 - tried to formally prove the existence of God
- Bayes Theorem

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)}$$

- useful in situations where $P(C|A)$ is unknown while $P(A|C)$, $P(A)$ and $P(C)$ are known or easy to estimate



Bayes Theorem: Evidence Formulation

- **Prior probability** of event H :
 - probability of event before evidence is seen
 - we play golf in 70% of all cases $\rightarrow P(H) = 0.7$
- **Posterior probability** of event H :
 - probability of event after evidence is seen
 - evidence: It is windy and raining $\rightarrow P(H | E) = 0.2$
- Probability of event H given evidence E :

$$P(H | E) = \frac{P(E | H)P(H)}{P(E)}$$



Applying Bayes Theorem to the Classification Task

Evidence = record

Class-conditional probability of evidence

Class

Prior probability of class

Prior probability of evidence

$$P(C | A) = \frac{P(A | C)P(C)}{P(A)}$$

1. Compute the probability $P(C | A)$ for all values of C using Bayes theorem.
 - $P(A)$ is the same for all classes. Thus, we just need to estimate $P(C)$ and $P(A|C)$
2. Choose value of C that maximizes $P(C | A)$.

Example:

$$P(\text{Play}=\text{yes} | \text{Outlook}=\text{rainy}, \text{Temp}=\text{cool}) = \frac{P(\text{Outlook}=\text{rainy}, \text{Temp}=\text{cool} | \text{Play}=\text{yes})P(\text{Play}=\text{yes})}{P(\text{Outlook}=\text{rainy}, \text{Temp}=\text{cool})}$$

$$P(\text{Play}=\text{no} | \text{Outlook}=\text{rainy}, \text{Temp}=\text{cool}) = \frac{P(\text{Outlook}=\text{rainy}, \text{Temp}=\text{cool} | \text{Play}=\text{no})P(\text{Play}=\text{no})}{P(\text{Outlook}=\text{rainy}, \text{Temp}=\text{cool})}$$

Estimating the Prior Probability $P(C)$

- The prior probability $P(C_j)$ for each class is estimated by
 1. counting the records in the training set that are labeled with class C_j
 2. dividing the count by the overall number of records
- Example:
 - $P(\text{Play=no}) = 5/14$
 - $P(\text{Play=yes}) = 9/14$

Training Data

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Estimating the Class-Conditional Probability $P(A | C)$

- Naïve Bayes assumes that all attributes are *statistically independent*
 - knowing the value of one attribute says nothing about the value of another
 - this independence assumption is almost never correct!
 - but ... this scheme works well in practice
- The independence assumption allows the joint probability $P(A | C)$ to be reformulated as the product of the individual probabilities $P(A_i | C_j)$:

$$P(A_1, A_2, \dots, A_n | C_j) = \prod P(A_n | C_j) = P(A_1 | C_j) \times P(A_2 | C_j) \times \dots \times P(A_n | C_j)$$

$$P(\text{Outlook}=\text{rainy}, \text{Temperature}=\text{cool} | \text{Play}=\text{yes}) = P(\text{Outlook}=\text{rainy} | \text{Play}=\text{yes}) \times P(\text{Temperature}=\text{cool} | \text{Play}=\text{yes})$$

- Result: The probabilities $P(A_i | C_j)$ for all A_i and C_j can be estimated directly from the training data

Estimating the Probabilities $P(A_i | C_j)$

Outlook			Temperature			Humidity			Windy			Play	
	Yes	No		Yes	No		Yes	No		Yes	No	Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

The probabilities $P(A_i | C_j)$ are estimated by

1. counting how often an attribute value appears together with class C_j
2. dividing the count by the overall number of records belonging to class C_j

Example:

2 times “Yes” together with “Outlook=sunny”
out of altogether 9 “Yes” examples

→ $p(\text{Outlook=sunny}|\text{Yes}) = \mathbf{2/9}$

Classifying a New Day

Unseen record

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Class-conditional
probability of the
evidence

$$P(\text{yes} \mid E) = P(\text{Outlook} = \text{Sunny} \mid \text{yes})$$

$$\times P(\text{Temperature} = \text{Cool} \mid \text{yes})$$

$$\times P(\text{Humidity} = \text{High} \mid \text{yes})$$

$$\times P(\text{Windy} = \text{True} \mid \text{yes})$$

$$\times \frac{P(\text{yes})}{P(E)}$$

Prior probability of class “yes”

Prior probability of evidence

$$= \frac{\frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14}}{P(E)}$$

Probability of
class “yes” given
the evidence

Classifying a New Day: Weigh the Evidence!

Outlook			Temperature			Humidity			Windy			Play	
	Yes	No		Yes	No		Yes	No		Yes	No	Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

– A new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Prior probability
Evidence

Choose Maximum

Likelihood of the two classes

For "yes" = $2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$

For "no" = $3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206$

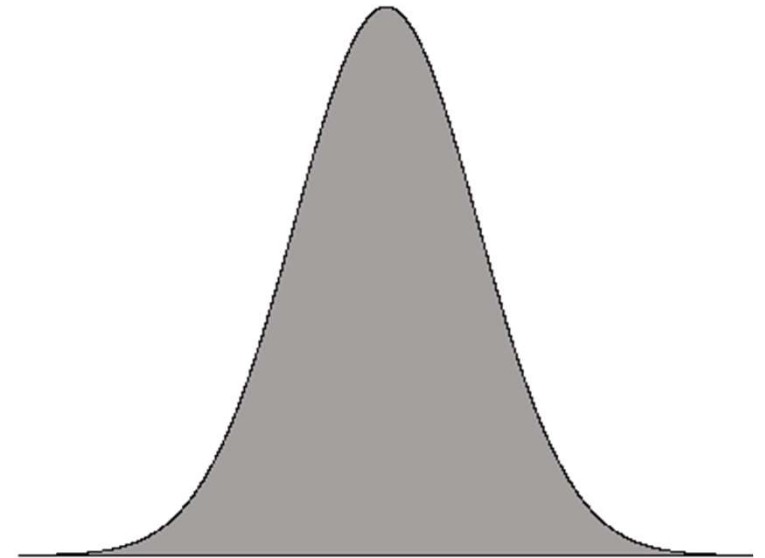
Conversion into a probability by normalization:

$P(\text{"yes"}) = 0.0053 / (0.0053 + 0.0206) = 0.205$

$P(\text{"no"}) = 0.0206 / (0.0053 + 0.0206) = 0.795$

Handling Numerical Attributes

- Option 1:
Discretize numerical attributes before learning classifier.
 - Temp= 37°C → “Hot”
 - Temp= 21°C → “Mild”
- Option 2:
Make assumption that numerical attributes have a **normal distribution** given the class.
 - use training data to estimate parameters of the distribution (e.g., mean and standard deviation)
 - once the probability distribution is known, it can be used to estimate the conditional probability $P(A_i|C_j)$



Handling Numerical Attributes

- The probability density function for the normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

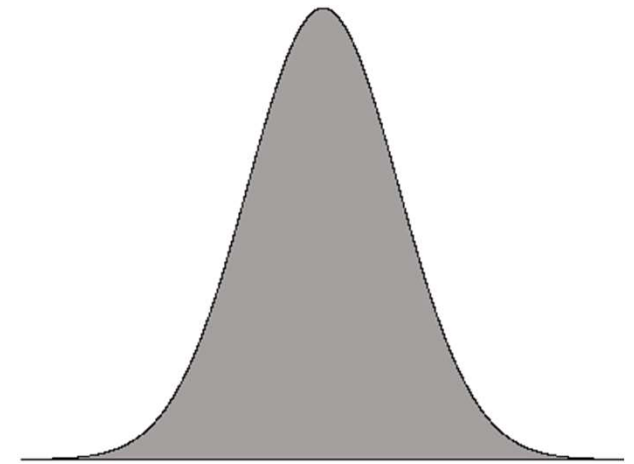
- It is defined by two parameters:

- *Sample mean* μ

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

- *Standard deviation* σ

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2}$$



- Both parameters can be estimated from the training data

Statistics for the Weather Data

Outlook			Temperature		Humidity		Windy			Play	
	Yes	No	Yes	No	Yes	No		Yes	No	Yes	No
Sunny	2	3	64, 68,	65, 71,	65, 70,	70, 85,	False	6	2	9	5
Overcast	4	0	69, 70,	72, 80,	70, 75,	90, 91,	True	3	3		
Rainy	3	2	72, ...	85, ...	80, ...	95, ...					
Sunny	2/9	3/5	$\mu = 73$	$\mu = 75$	$\mu = 79$	$\mu = 86$	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	$\sigma = 6.2$	$\sigma = 7.9$	$\sigma = 10.2$	$\sigma = 9.7$	True	3/9	3/5		
Rainy	3/9	2/5									

Example calculation:

$$f(temp = 66 | yes) = \frac{1}{\sqrt{2\pi} 6.2} e^{-\frac{(66-73)^2}{2*6.2^2}} = 0.0340$$

Classifying a New Day

Unseen record

Outlook	Temp.	Humidity	Windy	Play
Sunny	66	90	true	?

Likelihood of "yes" = $2/9 \times 0.0340 \times 0.0221 \times 3/9 \times 9/14 = 0.000036$

Likelihood of "no" = $3/5 \times 0.0291 \times 0.0380 \times 3/5 \times 5/14 = 0.000136$

$P(\text{"yes"}) = 0.000036 / (0.000036 + 0.000136) = 20.9\%$

$P(\text{"no"}) = 0.000136 / (0.000036 + 0.000136) = 79.1\%$

But note: Some numeric attributes are not normally distributed, and you may thus need to choose a different probability density function or use discretization

Handling Missing Values

- Missing values may occur in training and in unseen classification records
- **Training:** Record is not included into frequency count for attribute value-class combination
- **Classification:** Attribute will be omitted from calculation

- Example:

Unseen record

Outlook	Temp.	Humidity	Windy	Play
?	Cool	High	True	?



Likelihood of "yes" = $3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0238$

Likelihood of "no" = $1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0343$

$P(\text{"yes"}) = 0.0238 / (0.0238 + 0.0343) = 41\%$

$P(\text{"no"}) = 0.0343 / (0.0238 + 0.0343) = 59\%$

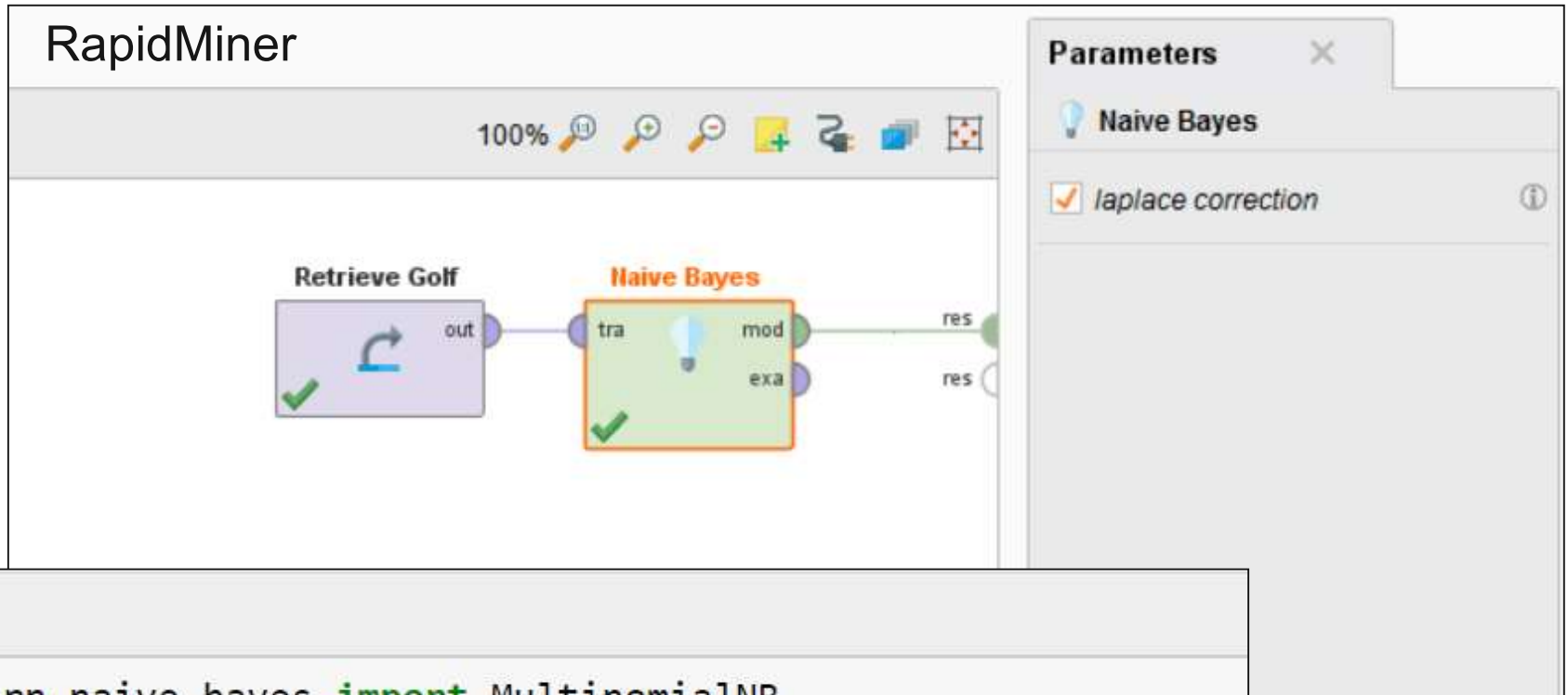
The Zero-Frequency Problem

- What if an attribute value doesn't occur with every class value?
(e.g. no "Outlook = overcast" for class "no")
 - class-conditional probability will be zero! $P[Out. = overc. | no] = \frac{0}{5} = 0$
- Problem: Posterior probability will also be zero!
No matter how likely the other values are! $P[no | E] = 0$
- Remedy: Add 1 to the count for every attribute value-class combination (*Laplace Estimator*)
- Result: Probabilities will never be zero!
also: stabilizes probability estimates

$$\text{Original : } P(A_i | C) = \frac{N_{ic}}{N_c}$$

$$\text{Laplace : } P(A_i | C) = \frac{N_{ic} + 1}{N_c + |V_i|} \quad |V_i| \text{ number of values}$$

Naïve Bayes in RapidMiner and Python



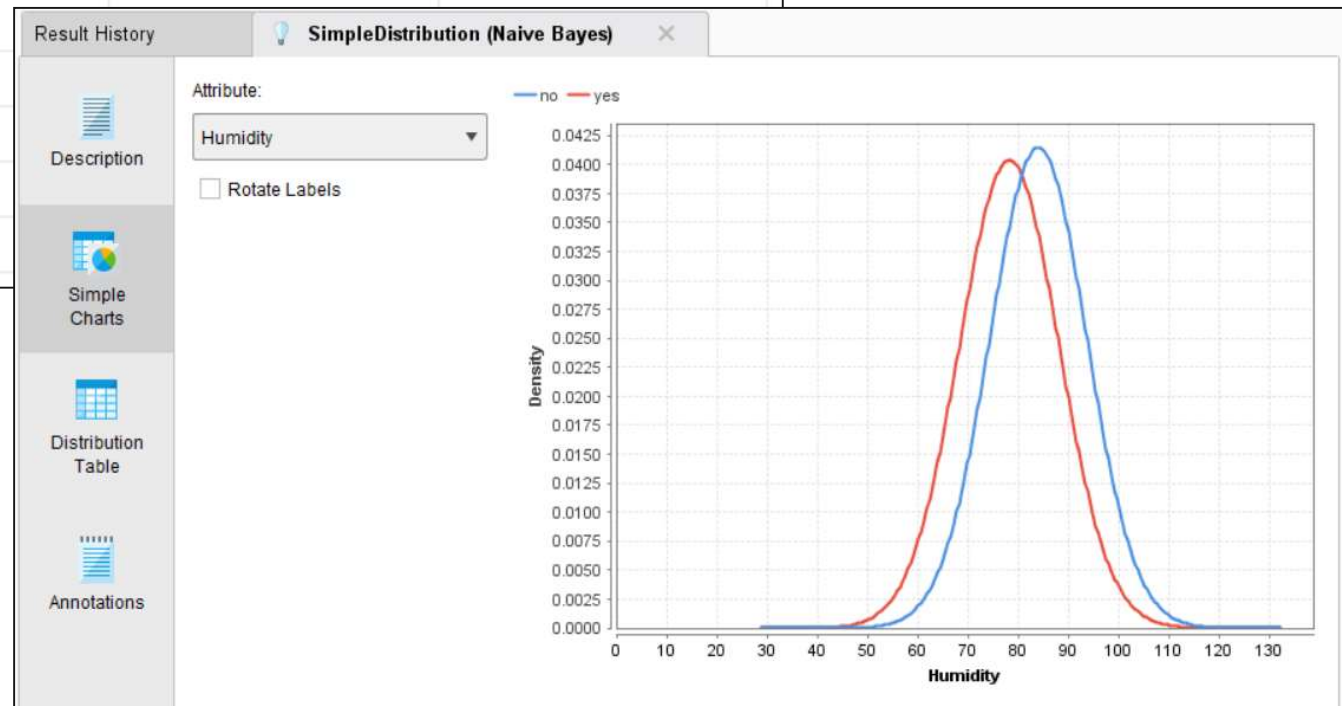
Python

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB

# Train classifier
estimator = MultinomialNB(alpha=1.0)
estimator.fit(preprocessed_training_data, training_labels)
```

Naïve Bayes in RapidMiner: Probability Distribution Table

Result History				
SimpleDistribution (Naive Bayes)				
Description				
Attribute	Parameter	no	yes	
Outlook	value=rain	0.392	0.331	
Outlook	value=overcast	0.014	0.438	
Outlook	value=sunny	0.581	0.223	
Outlook	value=unknown	0.014	0.008	
Temperature	mean	74.600	73	
Temperature	standard deviation	7.893	6.164	
Humidity	mean			
Humidity	standard deviation			
Wind	value=true			
Wind	value=false			
Wind	value=unknown			



Characteristics of Naïve Bayes

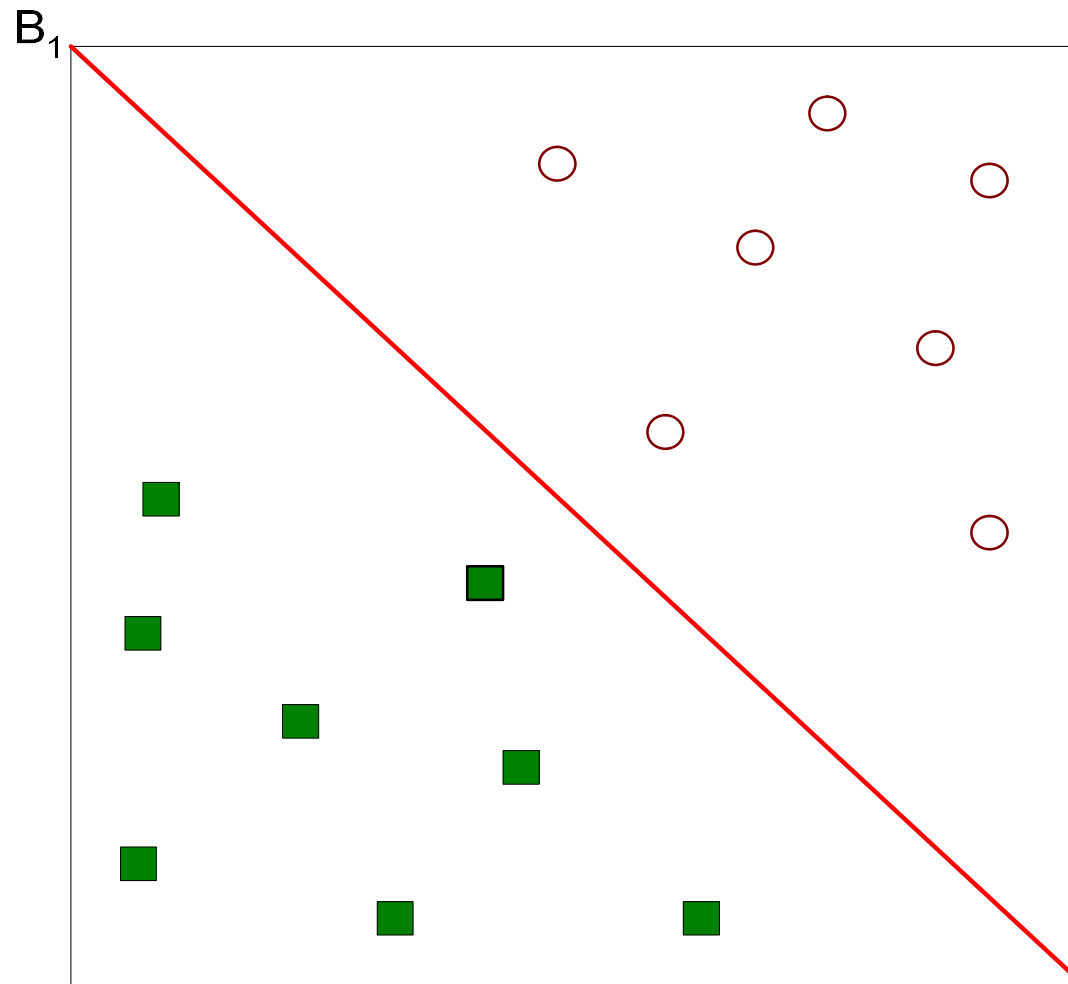
- Naïve Bayes **works surprisingly well** for many classification tasks
 - even if independence assumption is clearly violated
 - why? Because classification doesn't require accurate probability estimates as long as maximum probability is assigned to correct class
- Robust to **isolated noise points** as they will be averaged out
- Robust to **irrelevant attributes** as $P(A_i | C)$ distributed uniformly for A_i
- Adding too many **redundant attributes** can cause problems
 - solution: Select attribute subset as Naïve Bayes often works better with just a fraction of all attributes
- Technical advantages
 - learning Naïve Bayes classifiers is **computationally cheap** as probabilities can be estimated doing one pass over the training data
 - storing the probabilities does **not require a lot on memory**

7. Support Vector Machines

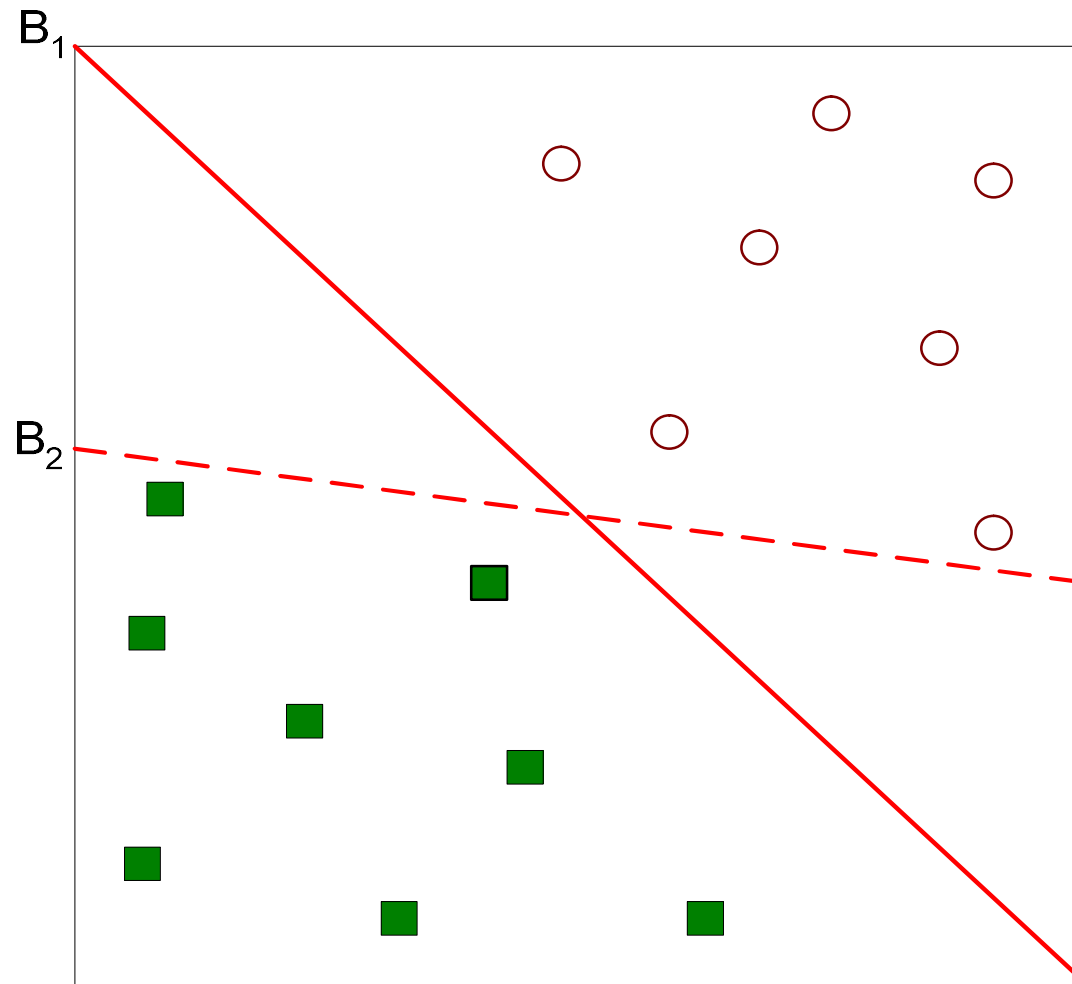
- Support vector machines (SVMs) are algorithms for learning linear classifiers for
 - **two class problems** (a positive and a negative class)
 - from examples described by **continuous attributes**
- SVMs achieve **very good results** especially for high dimensional data
- SVMs were invented by V. Vapnik and his co-workers in 1970s in Russia and became known to the West in 1992

Support Vector Machines

- SVMs find a linear **hyperplane** (decision boundary) that will separate the data



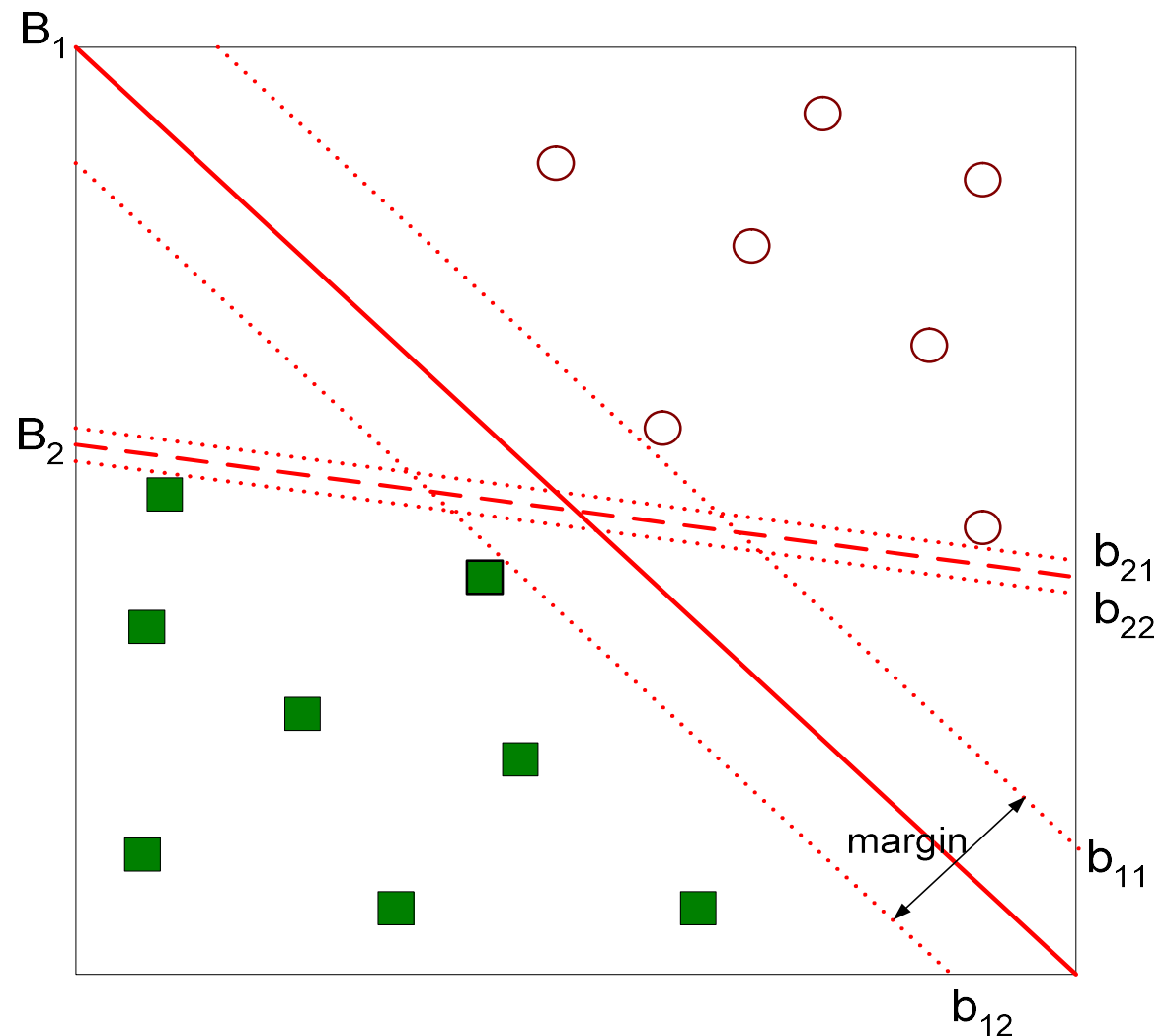
Support Vector Machines



- Which one is better? B_1 or B_2 ?
- How do you define “better”?

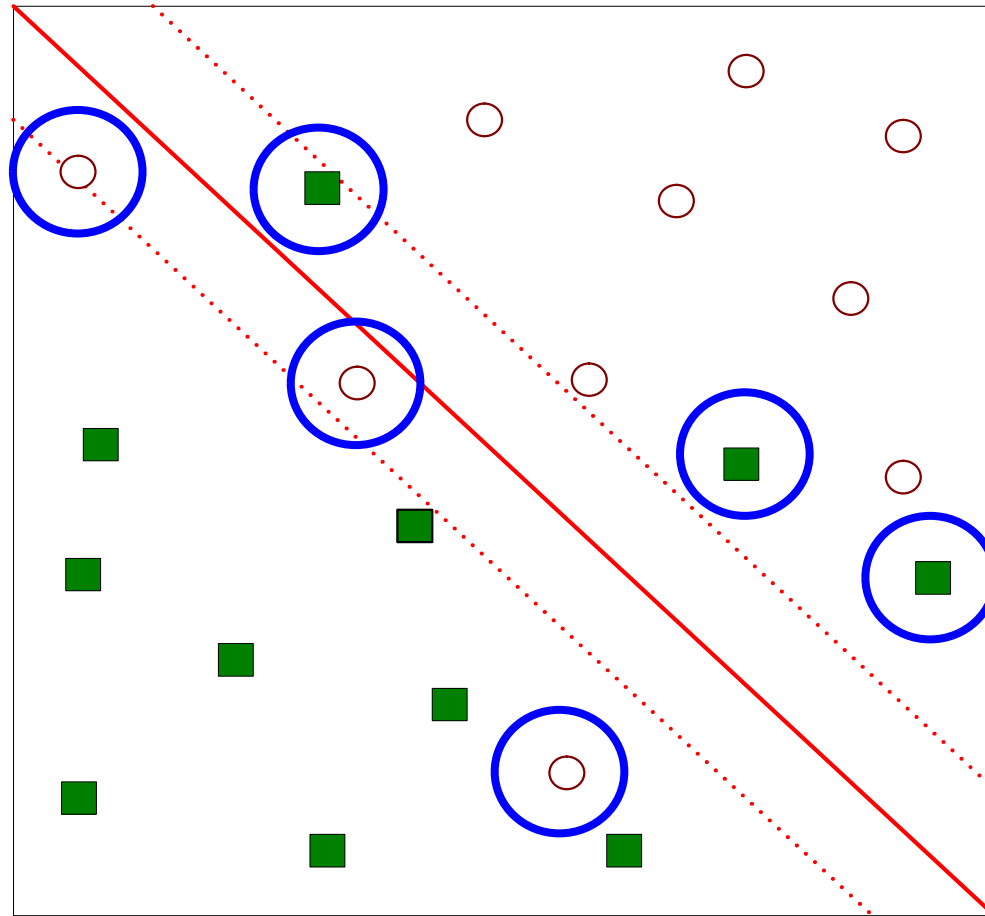
Which Hyperplane is better?

- In order to **avoid overfitting** and to generalize for unseen data, SVMs find the hyperplane that **maximizes** the margin to the closest points (support vectors)
- Visual solution:
 - B1 is better than B2
- Mathematical solution:
 - constrained optimization that can be solved using quadratic programming
 - See Tan/Steinbach/Kumar: Chapter 6.9



Dealing with Not Linearly Separable Data

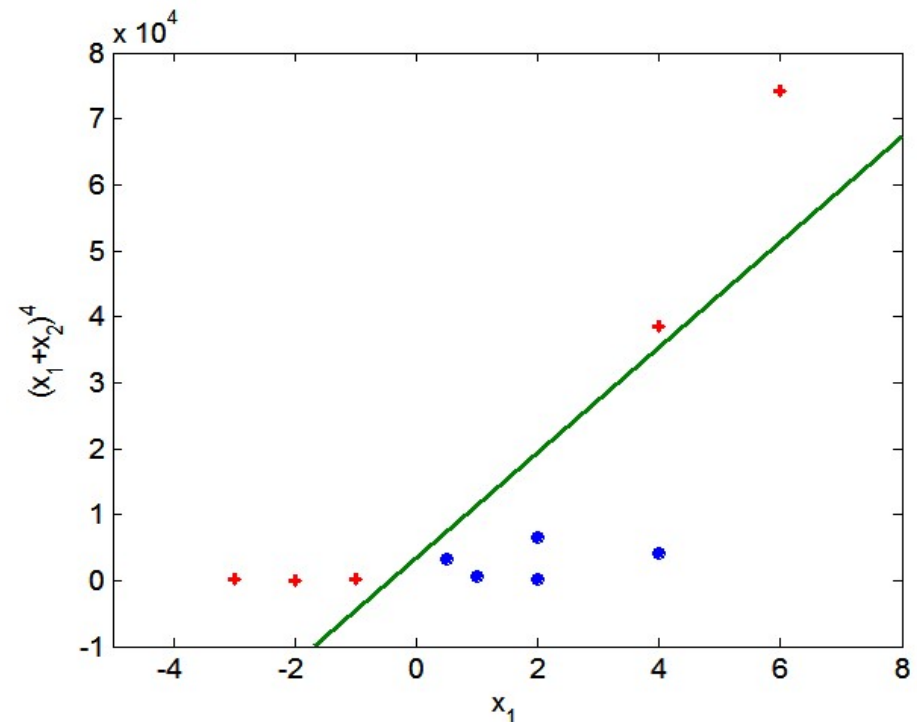
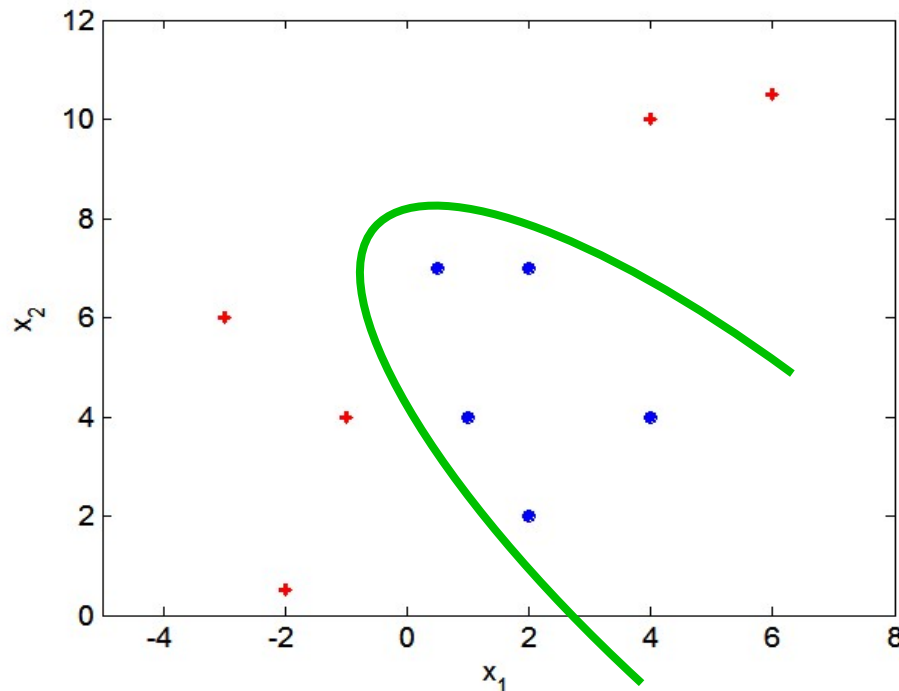
- What if the problem is not linearly separable due to noise points?



- Solution: Introduce **slack variables** in margin computation which result in a penalty for each data point that violates decision boundary

Dealing with Non-Linear Decision Boundaries

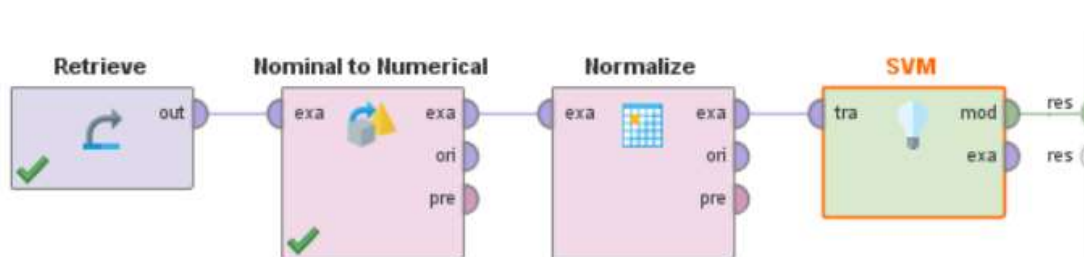
- Problem: What if decision boundary is not linear?
- Solution: Transform data into higher dimensional space where there is a linear separation
 - details see Tan/Steinbach/Kumar: Chapter 6.9
 - different types of **kernel functions** are used for this transformation



Characteristics of Support Vector Machines

- SVMs were often the **most successful** classification technique for high dimensional data **before DNNs** appeared
- Application areas of SVMs include
 - Text classification
 - Computer vision, e.g face identification
 - Handwritten digit recognition
 - SPAM detection
 - Bioinformatics
- Hyperparameter selection often has a high impact on the performance of SVNs
 - see next slide

SVMs in RapidMiner and Python



Parameters X RapidMiner

💡 SVM (Support Vector Machine (LibSVM))

svm type	C-SVC	ⓘ
kernel type	rbf	ⓘ
gamma	0.0	ⓘ
C	0.0	ⓘ
cache size	80	ⓘ
epsilon	0.001	ⓘ

Python

```
from sklearn.svm import SVC

# Train classifier
estimator = SVC(C=1.0, kernel='rbf')
estimator.fit(scaled_training_data, training_labels)
```

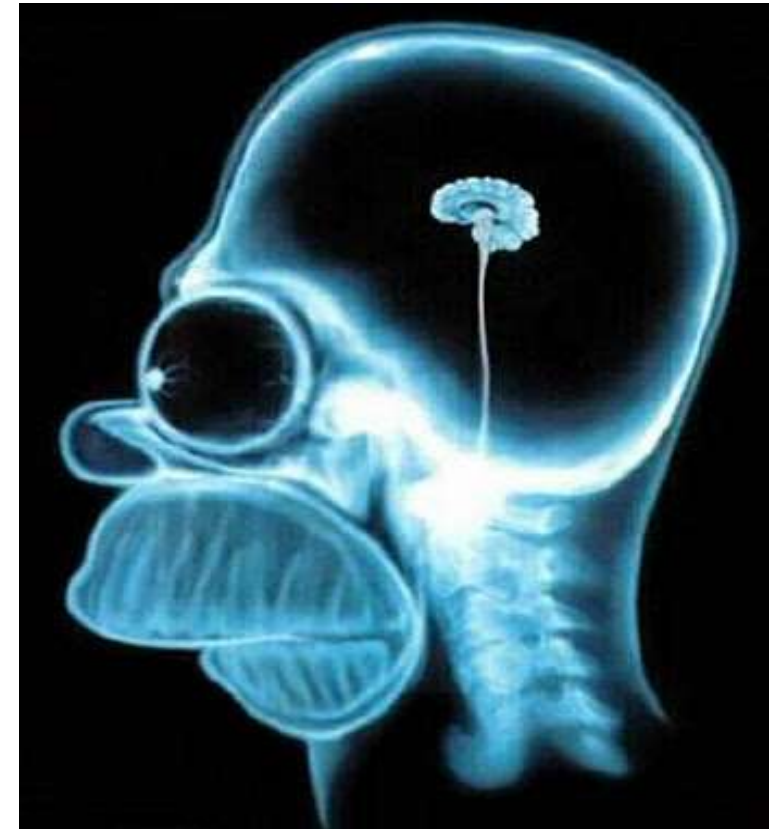
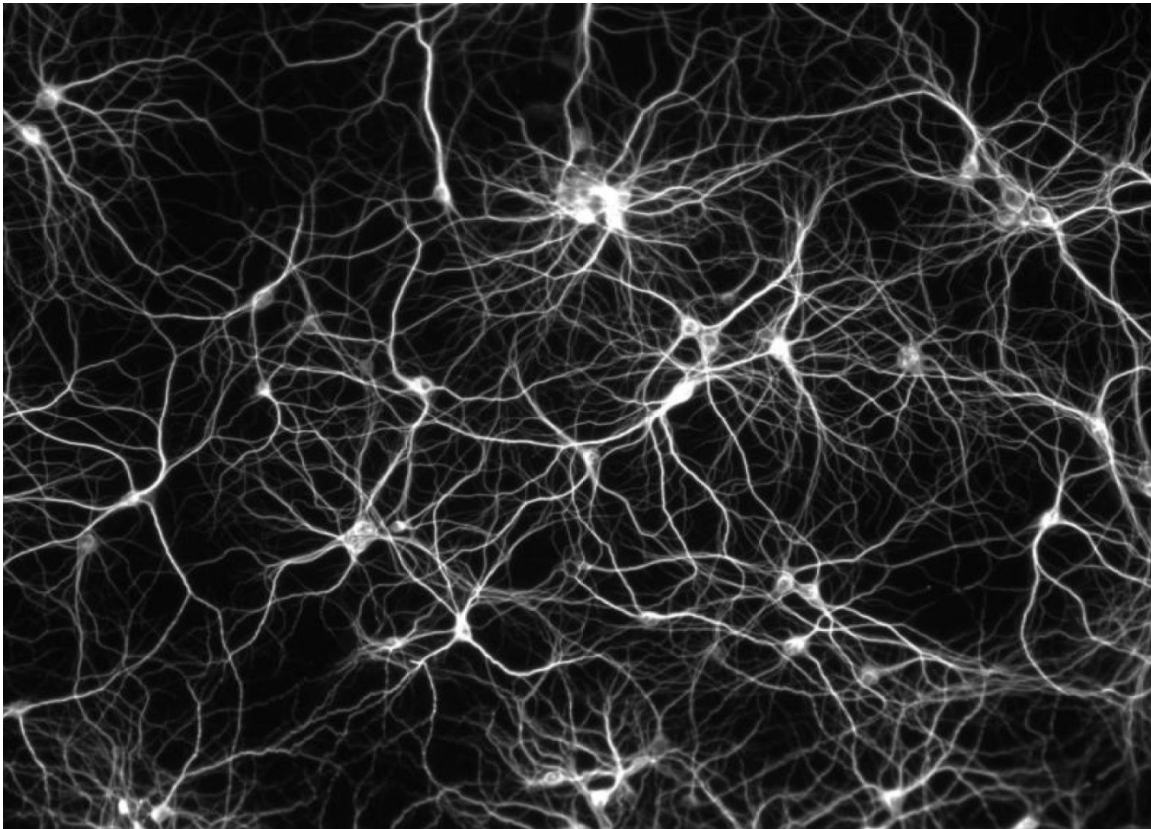
Tuning a SVM

1. Transform all attributes to numeric scale
2. Normalize all value ranges to [0,1]
3. Use the RBF kernel function
4. Use nested cross-validation to find the best values for the parameters
 1. C = weight of slack variables (Range: 0.03 to 30000)
 2. gamma = kernel parameter (Range: 0.00003 to 8)

More details on tuning: Hsu, et al: A Practical Guide to Support Vector Classification.

8. Artificial Neural Networks (ANN)

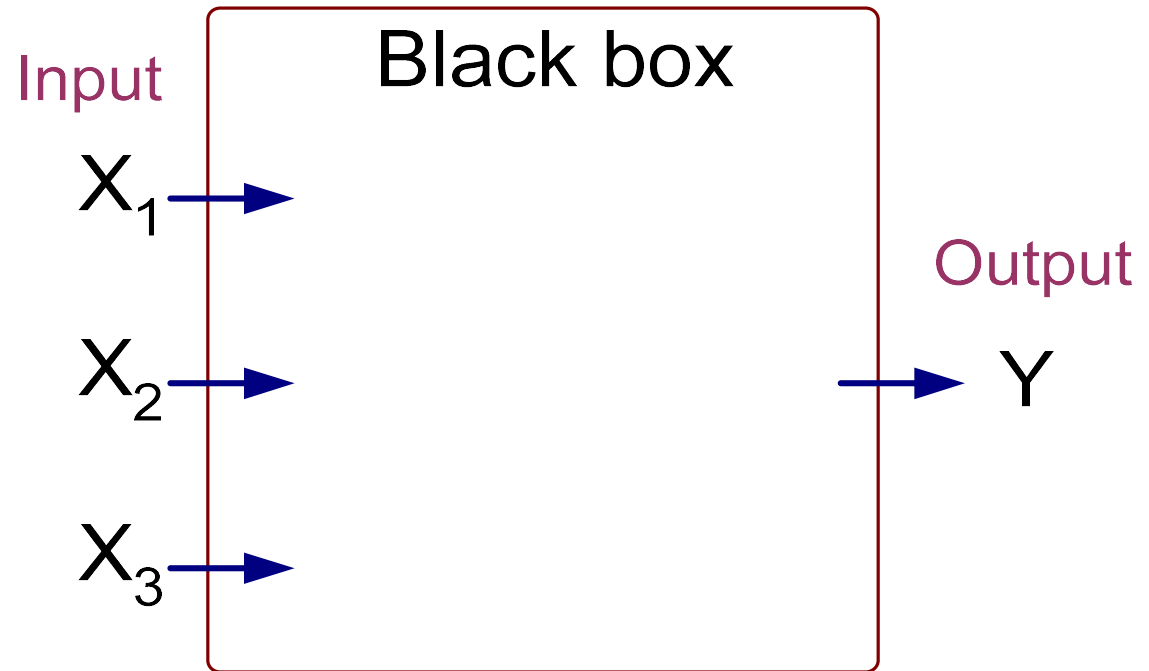
- Inspiration
 - one of the most powerful super computers in the world



Artificial Neural Networks (ANN)

Training Data

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



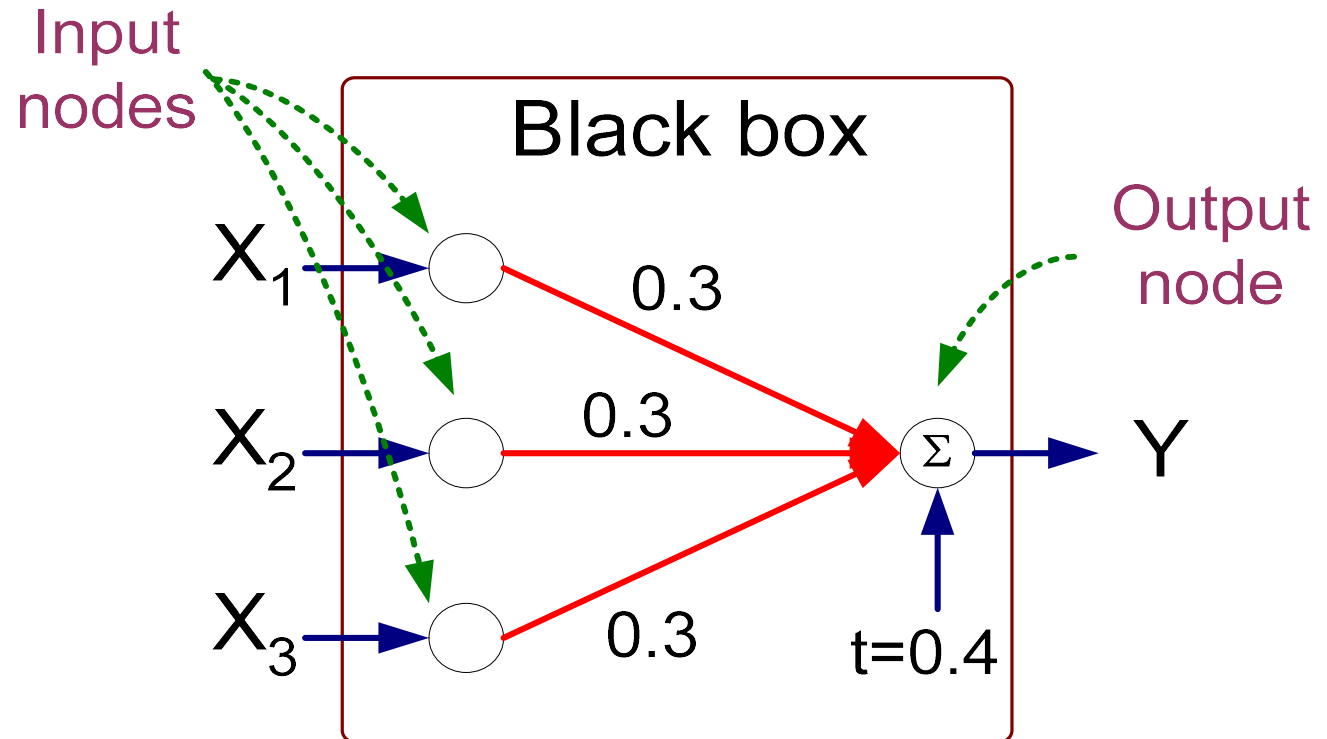
Function fitting the training data:

Output Y is 1 if at least two of the three inputs are equal to 1

Artificial Neural Networks (ANN)

Training Data

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

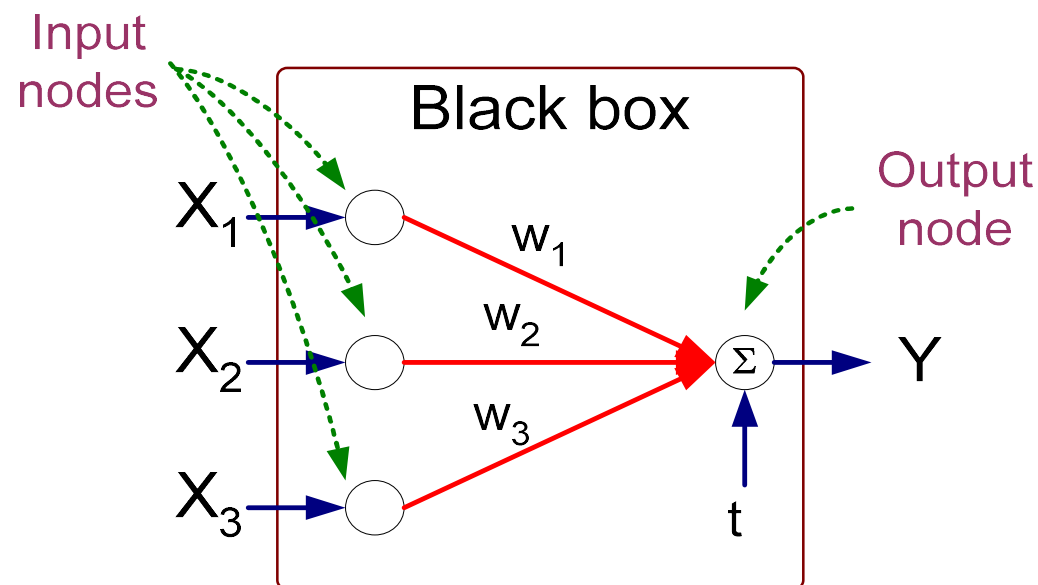


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

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

Artificial Neural Networks (ANN)

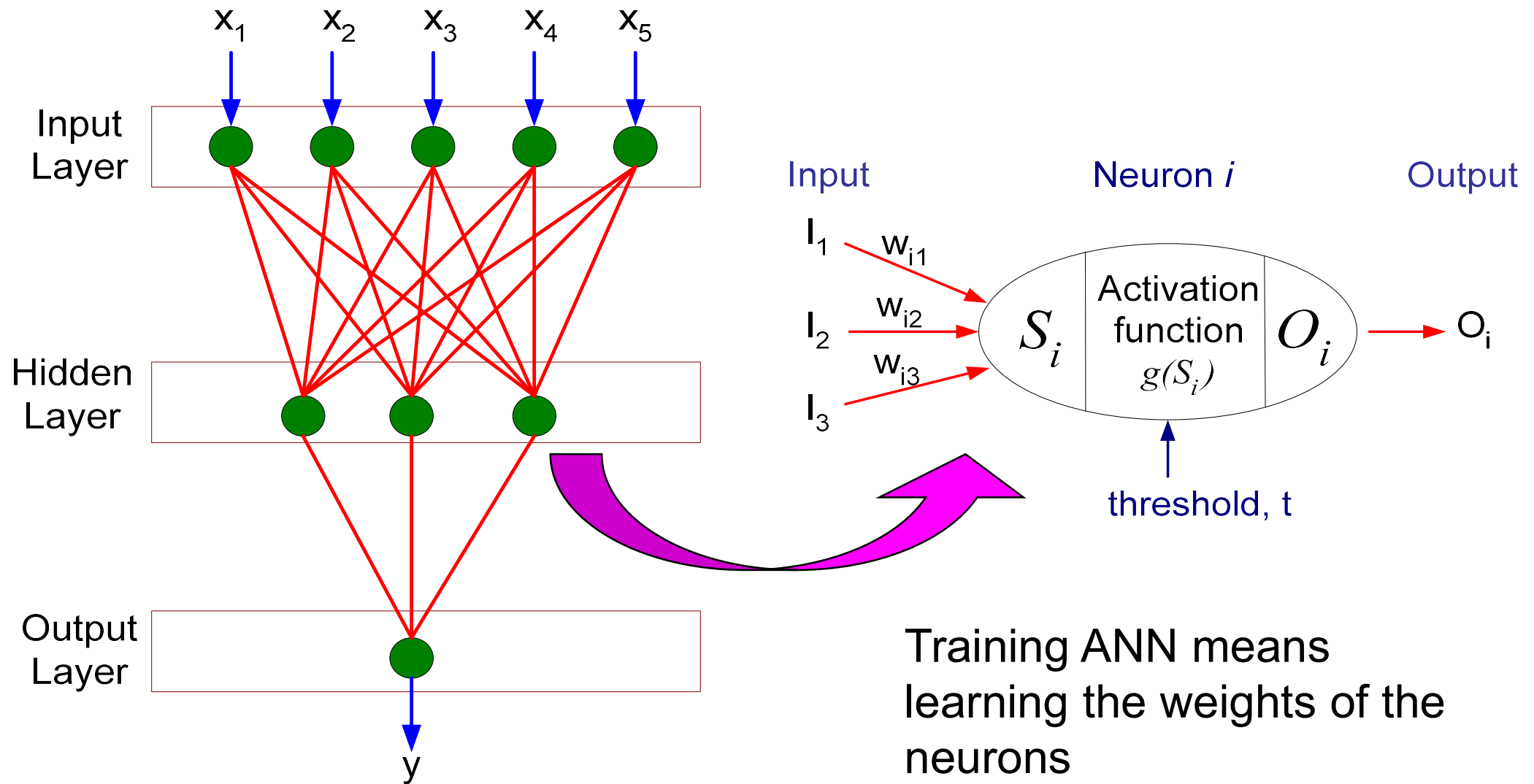
- Model is an assembly of inter-connected nodes (called neurons) and weighted links
- Output node sums up each of its input values according to the weights of its links
- Classification decision:
Compare output node against some threshold t



Perceptron Model

$$Y = I(\sum_i w_i X_i - t > 0)$$

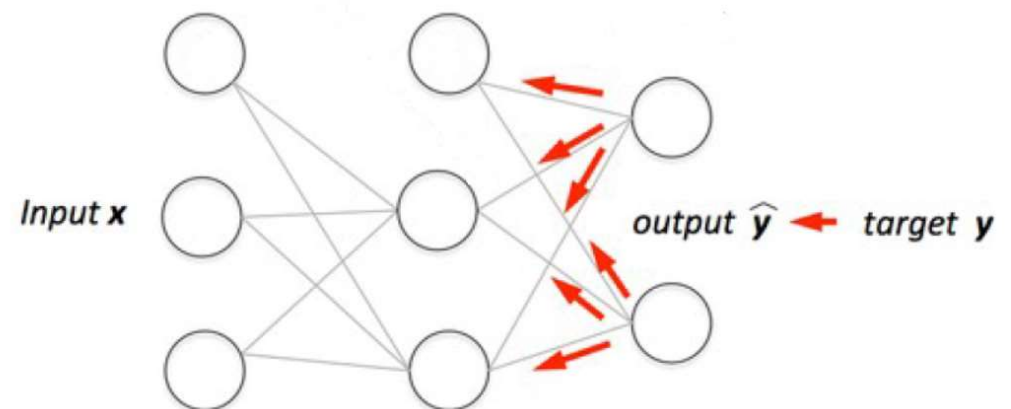
Multi-Layer Artificial Neural Networks



Algorithm for Training ANNs

1. Initialize the weights (w_0, w_1, \dots, w_k), e.g., random or pre-trained
2. Adjust the weights in such a way that the output of ANN is as consistent as possible with class labels of the training examples

- Objective function:
$$E = \sum_i [Y_i - f(w_i, X_i)]^2$$
- Find the weights w_i 's that minimize the sum of squared error E
- using the **back propagation algorithm**
(see Tan/Steinbach: Chapter 6.7,
Gemulla: Machine Learning)
- Adjustment factor: learning rate



Characteristics of Shallow Neural Networks

- ANNs can be used for classification as well as numerical regression tasks (more on this next week)
- Multi-layer neural networks are **universal approximators**
 - meaning that they can approximate any target function
- Very important but difficult to choose the right network topology
 - Expressive hypothesis space often leads to **overfitting**
 - Possible approaches to deal with overfitting:
 - use more training data (a lot more might be necessary)
 - step-by-step simplify the topology (regularization) and test against validation set
- Can handle redundant attributes, difficult to handle missing values
- Training is time consuming, model application is relatively fast

Overview: Types of Deep Learning Models

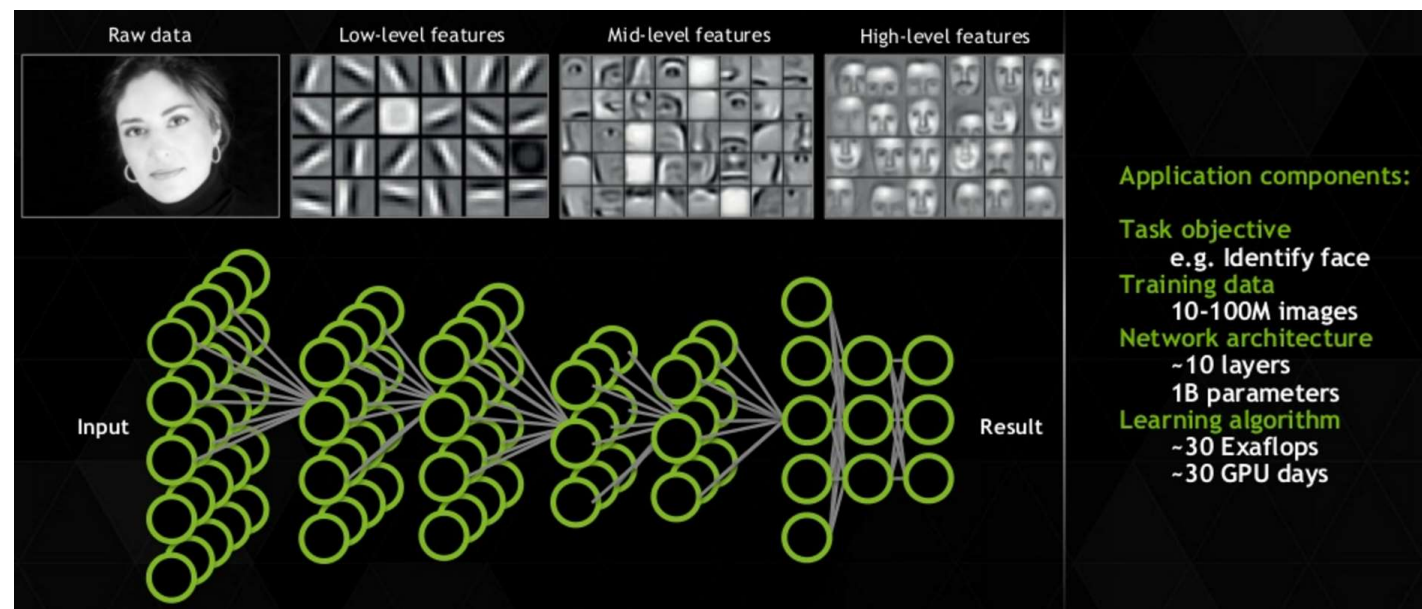
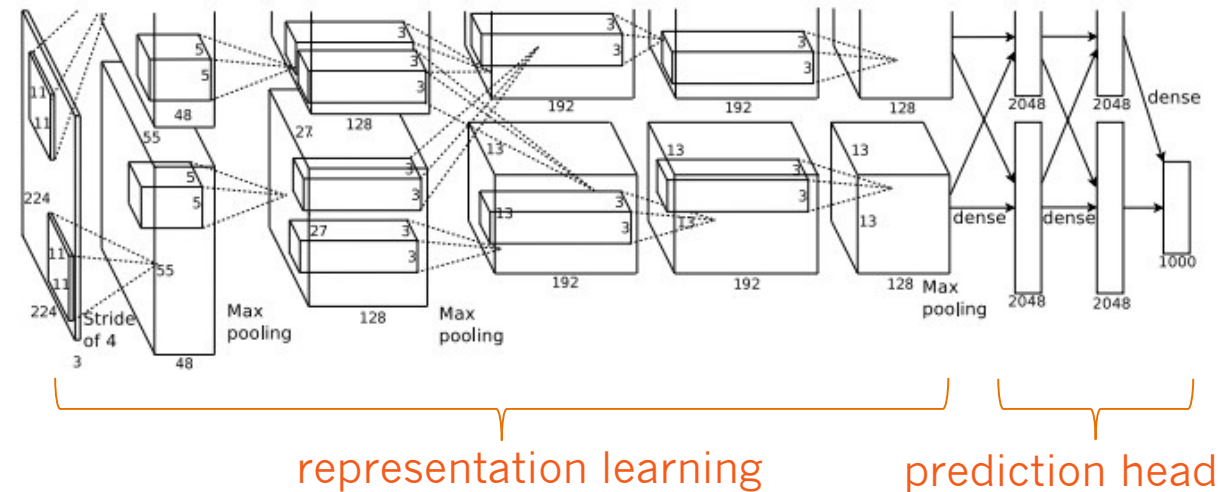
1. Convolutional Neural Networks
2. Graph Neural Networks
3. Pretrained Language Models: BERT
4. Generative Models: GPT3, DALL·E
5. Instruct Models: GPT4, Claude, Gemini

Lectures discussing deep learning in more detail

1. Rainer Gemulla: Deep Learning
2. Simone Ponzetto: Advanced Methods in Text Analytics
3. Margret Keuper: Higher Level Computer Vision

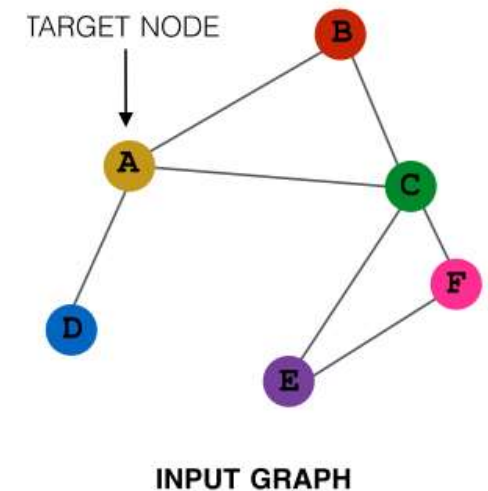
Convolutional Neural Networks (CNNs)

- invented in computer vision
- combine
 1. representation learning (convolutions and pooling)
 2. prediction head (densely connected layers)
- reduce number of input features via convolutions and pooling
- high capacity of models requires
 - lots of training data
 - lots of GPU time

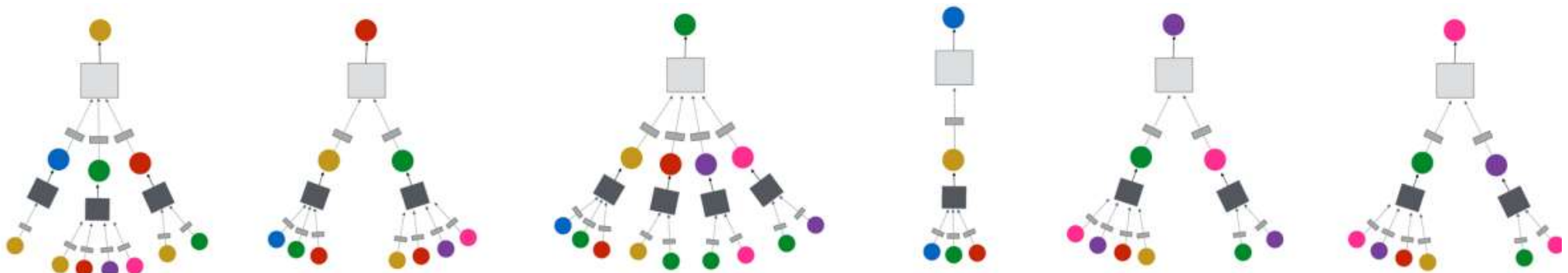


Graph Neural Networks (GNN)

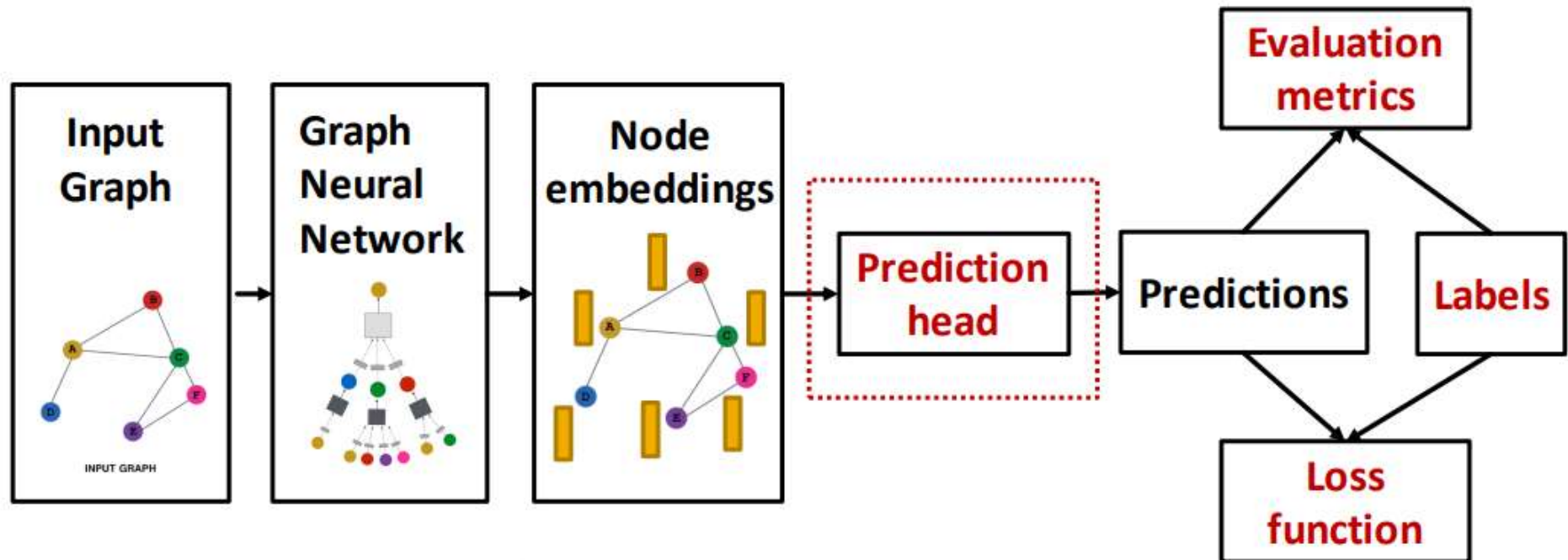
- used for predicting edges and node attributes in graphs
 - link prediction in social networks, topic detection in citation graphs
- intuition: nodes aggregate information from their neighbors using neural networks
- Each node defines its own **Computation Graph**
 - each edge in this graph is a transformation/aggregation function
 - node embeddings are learned using message passing



RESULTING COMPUTATION GRAPHS



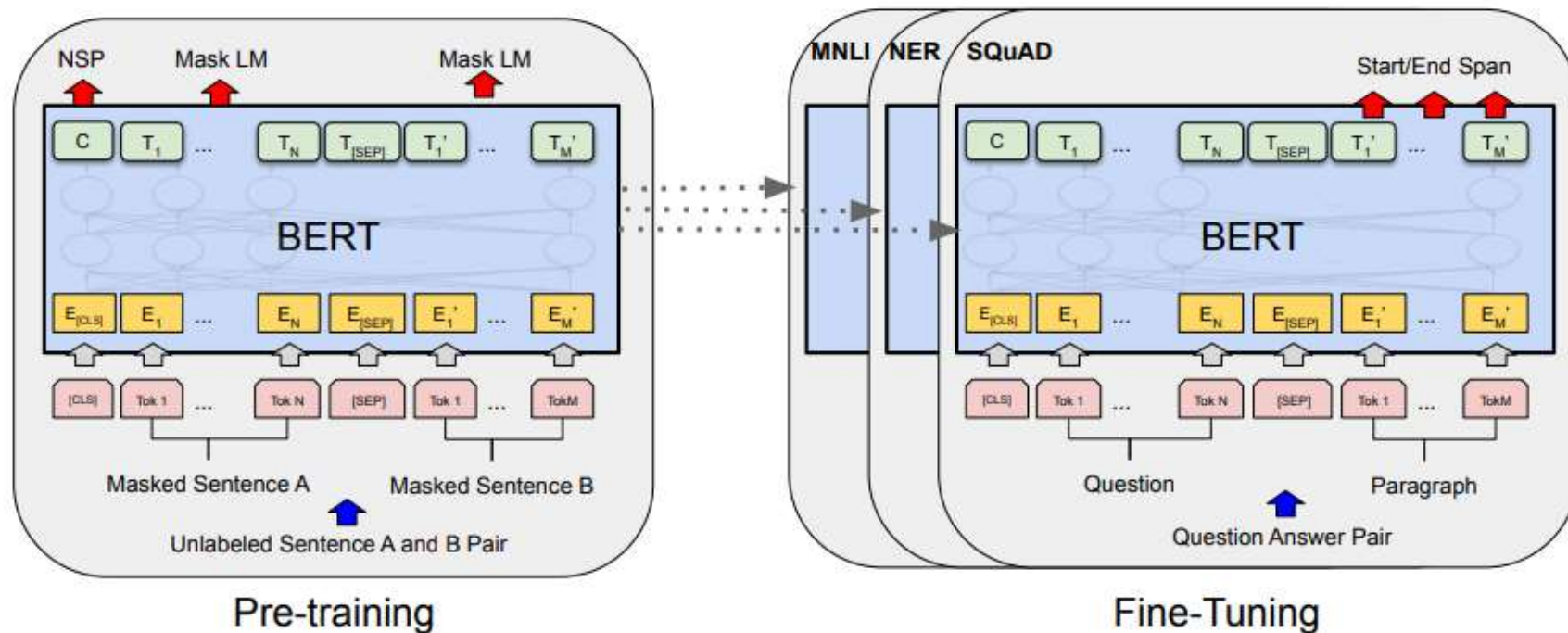
Supervised Training of a GNN



- Examples of prediction heads
 - Node classification: Linear layer on top of node embedding
 - Link prediction: Dot product of the two node embeddings
- Further details on GNNs
 - Stanford Lecture: Machine Learning with Graphs
 - <http://web.stanford.edu/class/cs224w/>

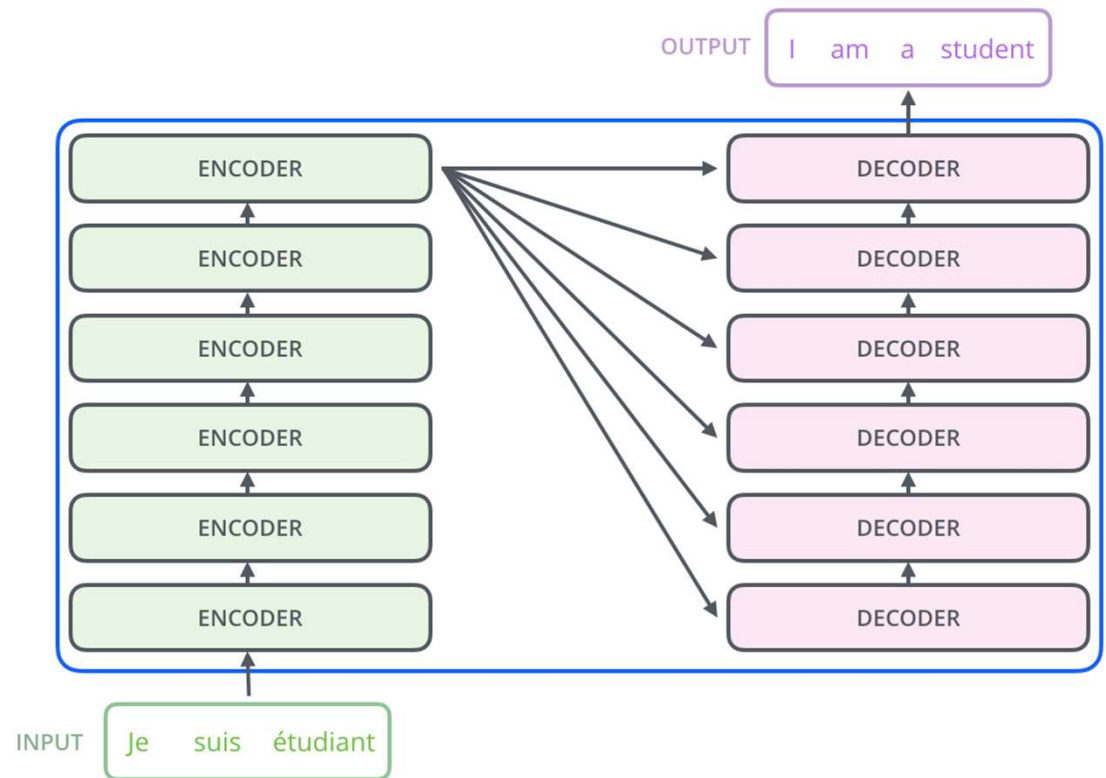
Pre-Trained Language Models

- introduce **pre-training**, **fine-tuning** paradigm
 - pre-trained on large text corpora
 - model size: BERT-base 110 million parameters
- outperform previous models on most NLP tasks



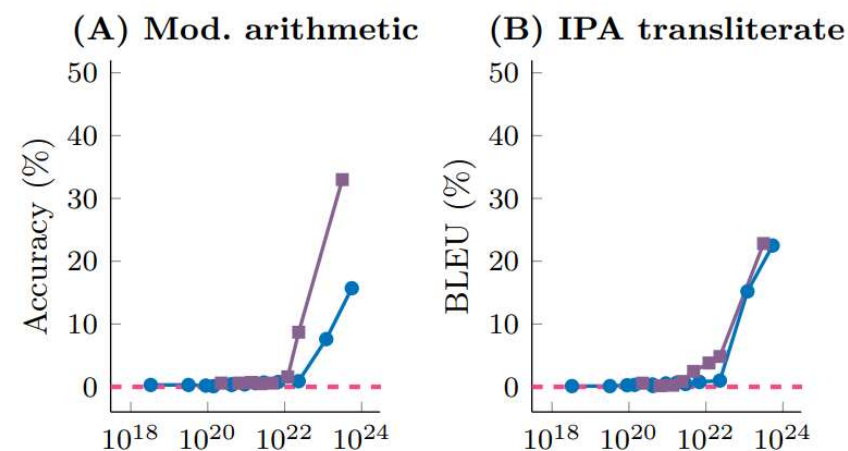
Generative Models

- use transformer architecture to generate text or images based on embeddings of input sequence
- Models for Text
 - GPT3
- Models for Images
 - DALL·E, Stable Diffusion
- pretrained on large text and image corpora
 - Web crawls
 - ImageNet, LAION-5B
- model sizes: 7 to 175 billion parameters
 - accessible mostly via APIs



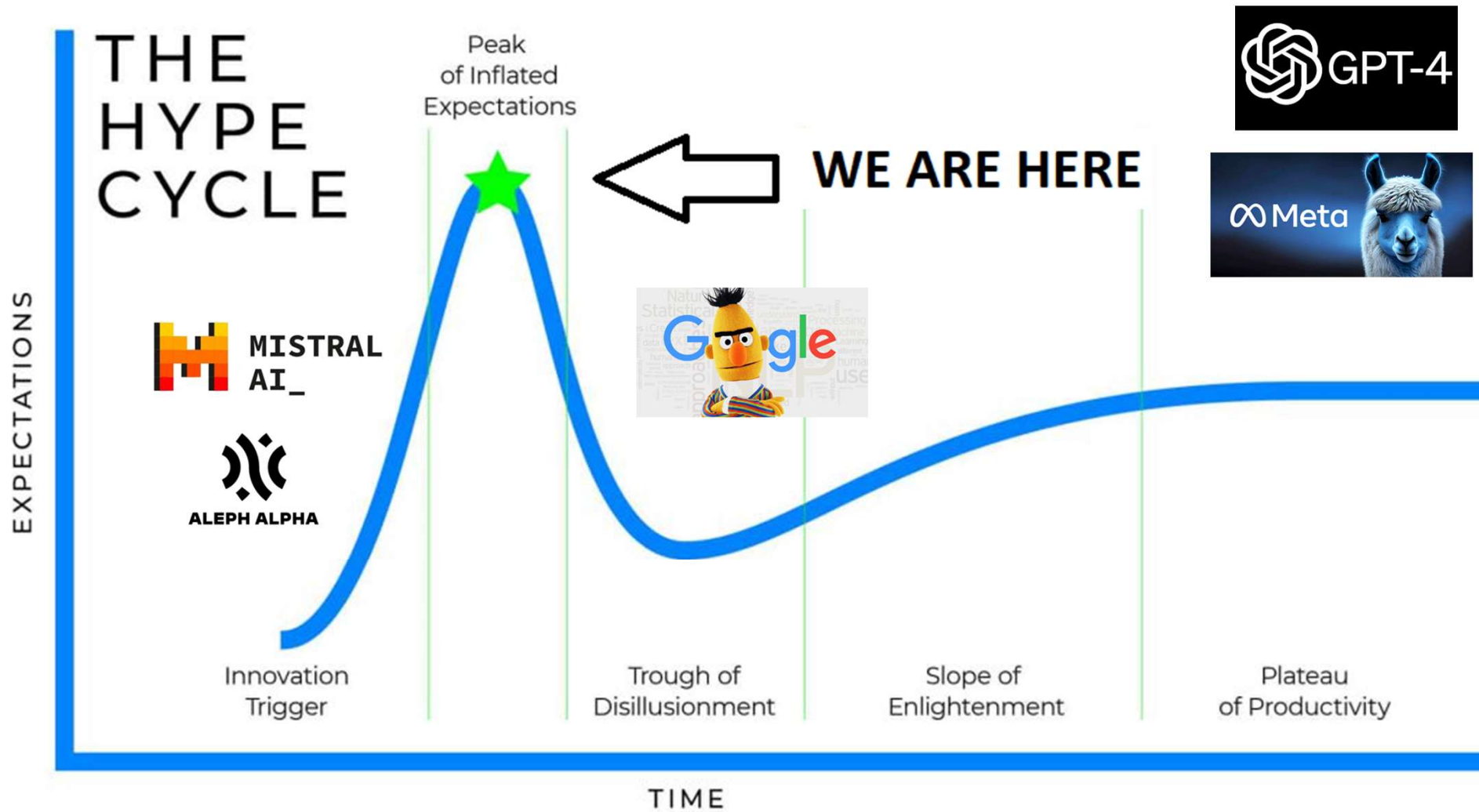
Instruct Language Models

- after being pre-trained on large text corpora, instruct models are **fine-tuned with instruction/output pairs**
- show good few-shot performance on wide range of task
 - HELM and BIG-bench collects 200+ tasks
- models show emergent abilities
 - can perform tasks they were not directly trained for
- prompt design and in-context learning determine performance of frozen models



Wei: Emergent Abilities of Large Language Models. TMLR 2022.

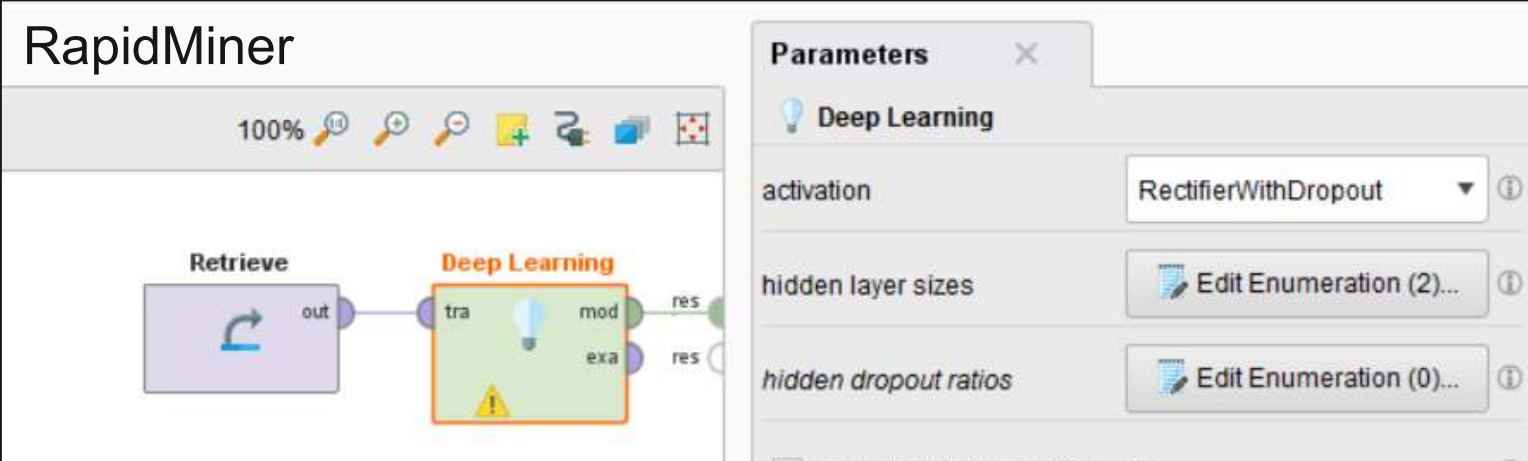
Large Language Models



Neural Networks and LLMs in Python and RapidMiner

- both platforms provide their own neural network implementations
- deep learning library, model and data hub: 🤗 **Hugging Face**
- library for programming with LLMs: 🦜🔗 **LangChain**

RapidMiner



Python

```
from sklearn.neural_network import MLPClassifier

# Train classifier
neuralnet = MLPClassifier(hidden_layer_sizes=(100, ), learning_rate_init=.1)
neuralnet.fit(training_data, training_labels)
```

9. Hyperparameter Selection

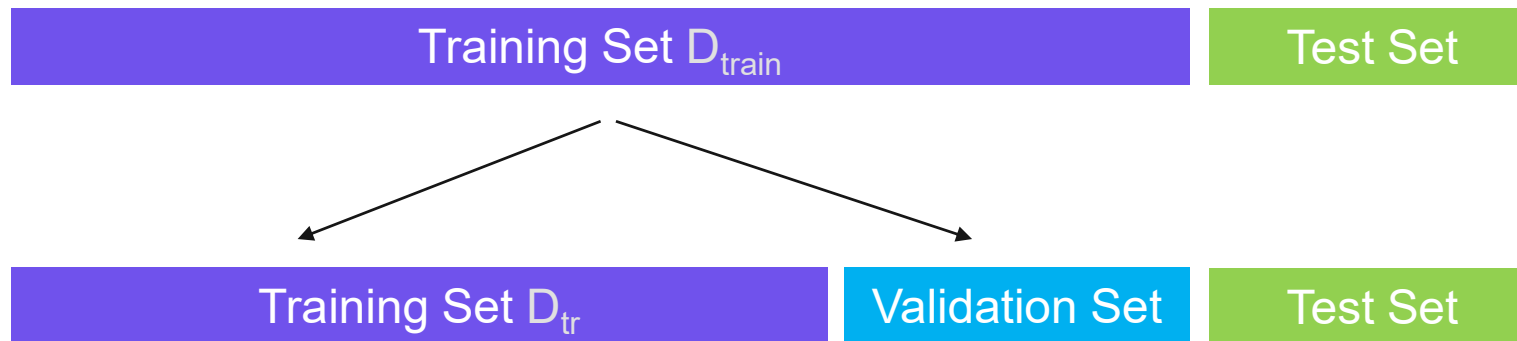
- A **hyperparameter** is a parameter which influences the learning process and whose value is set before the learning begins
 - pruning thresholds for trees and rules
 - gamma and C for SVMs
 - learning rate, hidden layers for ANNs
- By contrast, **parameters** are learned from the training data
 - weights in an ANN, probabilities in Naïve Bayes, splits in a tree
- Many methods work poorly with the default hyperparameters 😞
- How to determine good hyperparameters?
 - manually test some hyperparameter settings
 - have your machine automatically test many different settings (hyperparameter optimization)

Hyperparameter Optimization

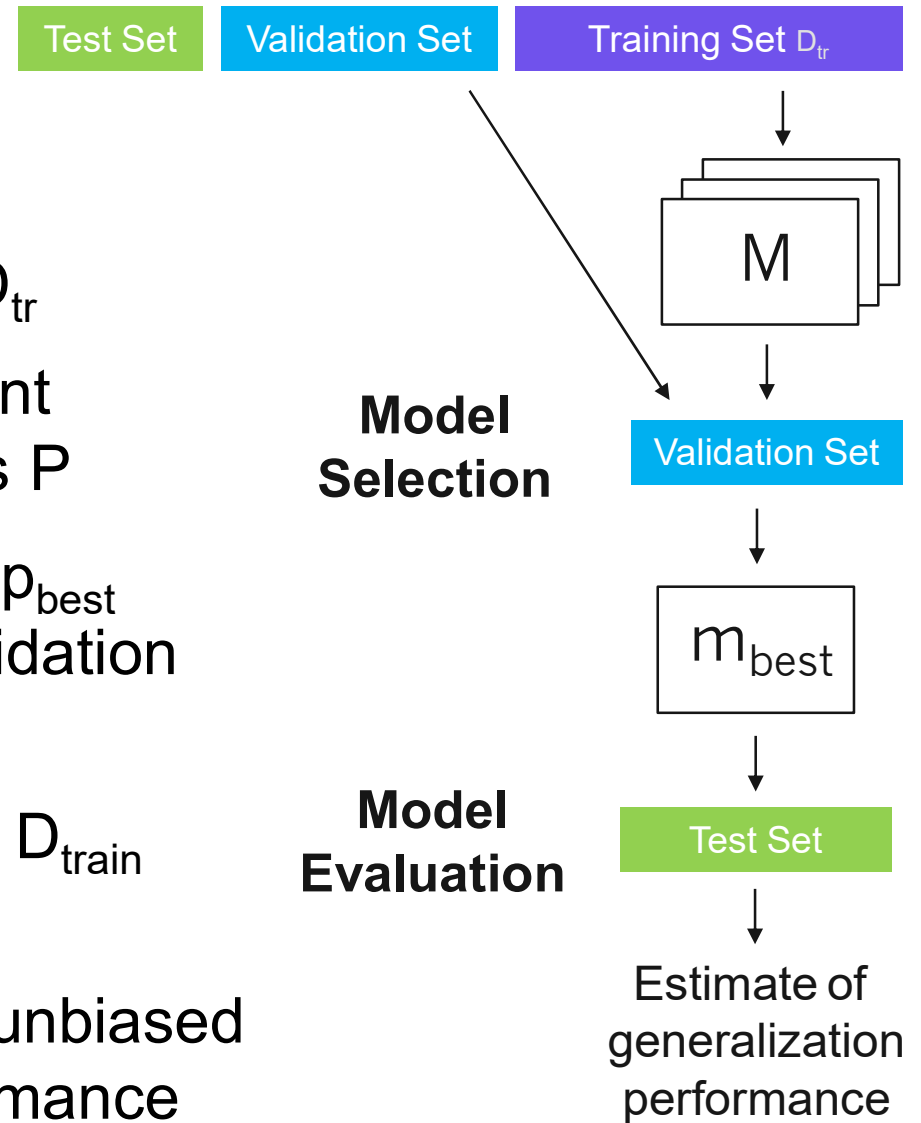
- Goal: Find the combination of hyperparameter values that results in the model with the lowest generalization error
- How to determine the parameter value combinations to be tested?
 - **Grid Search**: Test all combinations in user-defined ranges
 - **Random Search**: Test combinations of random parameter values
 - **Evolutionary Search**: Keep specific parameter values that worked well
- Often hundreds of combinations are tested
 - reason for cloud computing
- **Model Selection**: From all learned models M , select the model m_{best} that is expected to generalize best to unseen records

Model Selection Using a Validation Set

- We need to keep data used for model selection strictly separate from data used for model evaluation, otherwise:
 - selected model m_{best} will overfit to patterns in test set
 - estimate of generalization error will be too optimistic
- Thus, we split the training set D_{train} into
 1. **Training set** D_{tr} for training models with different hyperparameter settings
 2. **Validation set** D_{val} evaluating the generalization error of these models



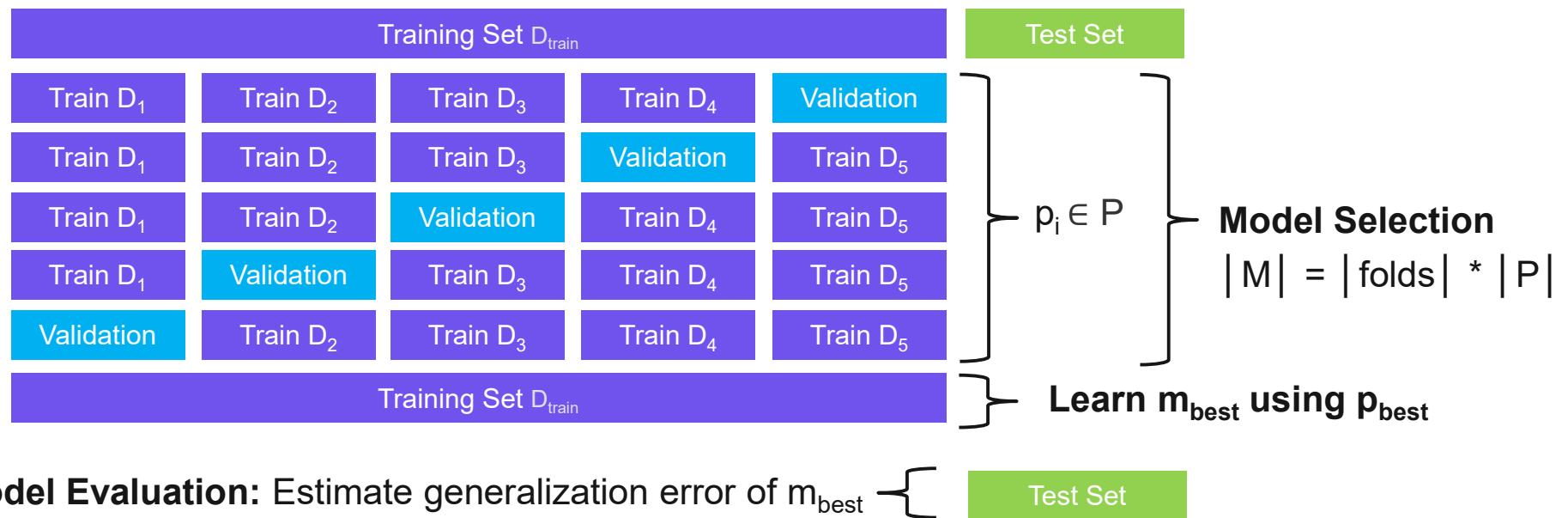
Process of Model Selection Using a Validation Set



1. Split training set D_{train} into validation set D_{val} and training set D_{tr}
2. Learn models M on D_{tr} using different hyperparameter value combinations P
3. Select best hyperparameter values p_{best} by testing each model m_i on the validation set D_{val}
4. Learn final model m_{best} on complete D_{train} using the parameter values p_{best}
5. Evaluate m_{best} on test set in get an unbiased estimate of its generalization performance

Model Selection using Cross-Validation

- But wait, we want to
 1. make sure that all examples are used for validation once (remember: red tree)
 2. use as much labeled data as possible for training (remember: learning curve)
- Both goals are met by using cross-validation for model selection



- 5 folds, 100 hyperparameter sets \rightarrow 501 models learned

Model Evaluation using Nested Cross-Validation

- Nest two cross-validation loops into each other in order to:
 1. find the best hyperparameter setting (**model selection**)
 2. get a reliable estimate of the generalization error (**model evaluation**)

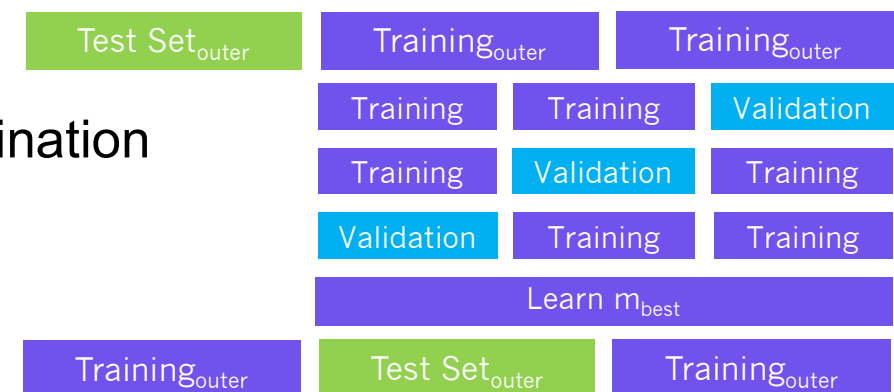
– Outer Cross-Validation

- estimates generalization error of m_{best}
- training set is passed on to inner cross-validation in each iteration



– Inner Cross-Validation

- searches for best hyperparameter combination
- splits outer training set into inner training and validation set
- learns model m_{best} using all outer training data



- $5 \text{ folds}_{\text{Outer}} * ((5 \text{ folds}_{\text{Inner}} * 100 \text{ para.sets}) + 1) \rightarrow 2505 \text{ models learned}$

Nested Cross-Validation in Python

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

# Specify hyperparameter combinations for search
parameter_grid = {"C": [1, 10, 100, 1000], "gamma": [.001, .01, .1, 1]}

# Create SVM
estimator_svm = SVC(kernel='rbf')

# Create the grid search for model selection
estimator_gs = GridSearchCV(estimator_svm, parameter_grid, scoring='accuracy', cv=5)

# Run nested cross-validation for model evaluation
accuracy_cv = cross_val_score(estimator_gs, dataset, labels, cv=5, scoring='accuracy')
```

scikit-learn Documentation: Tuning the hyper-parameters of an estimator

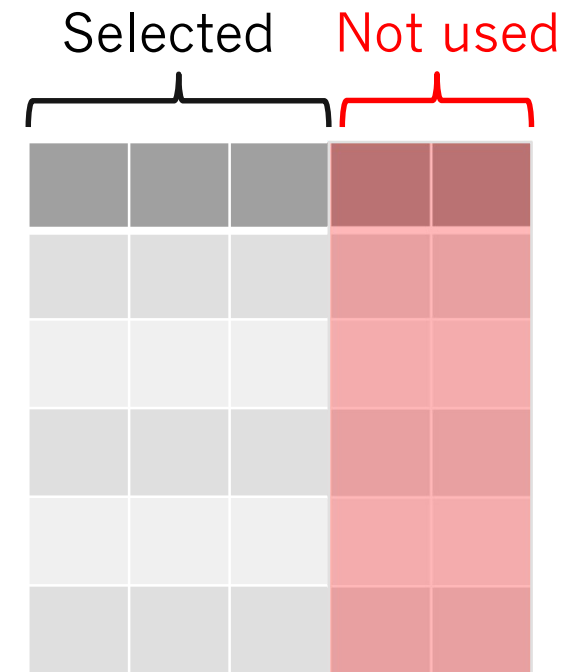
https://scikit-learn.org/stable/modules/grid_search.html

scikit-learn Documentation: Nested versus non-nested cross-validation

https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

Feature Selection

- Some classification methods automatically select the relevant feature subset as part of the learning process
 - e.g. Decision Trees, Random Forests, ANNs, SVMs
- The performance of other methods depends on the subset of the features provided
 - e.g. KNN, Naïve Bayes
- Automated feature selection approaches
 - **Backward selection:** start using all features, remove features, test again
 - **Forward selection:** Find best single feature, add further features, test again
- Use nested cross-validation to estimate the generalization error



Summary: Hyperparameter and Feature Selection

- Hyperparameter selection
 - Default: **Always run hyperparameter optimization!**
 - Otherwise, you cannot say that a method does not work for a task
- Feature selection
 - Default: **Check if classification method requires feature selection**
 - If yes, run automated feature selection
- Model selection
 - Default: **Use nested cross-validation**
 - If computation takes too long: use better hardware, reduce number of folds, reduce parameter search space, sample data to reduce size
 - If exact replicability of results is required: Use single train, validation, test split
- If your dataset is imbalanced
 - don't forget to **balance your training set**, not your test set!

Literature for this Slideset

Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar: **Introduction to Data Mining**. 2nd Edition. Pearson.

Chapter 6.4: Naïve Bayes

Chapter 6.9: Support Vector Machines

Chapter 6.7 and 6.8: Artificial Neural Networks

Chapter 3.5: Model Selection

Chapter 3.7: Presence of Hyperparameters

