

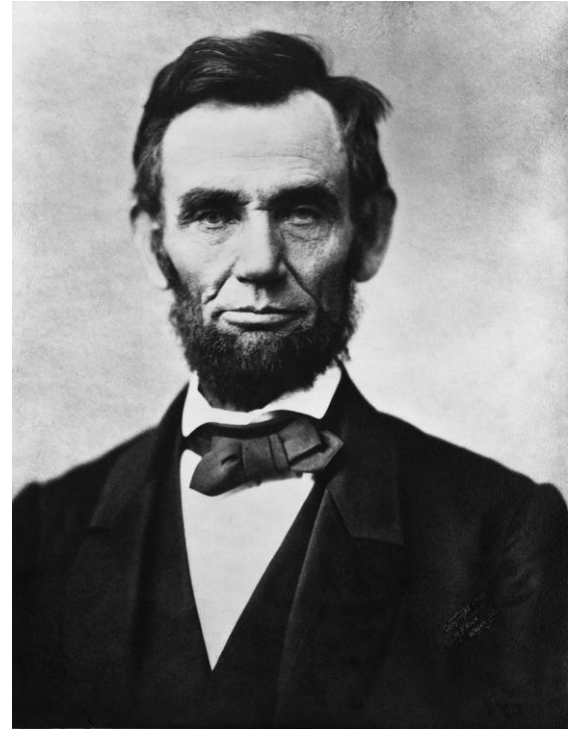
Preprocessing

IE500 Data Mining



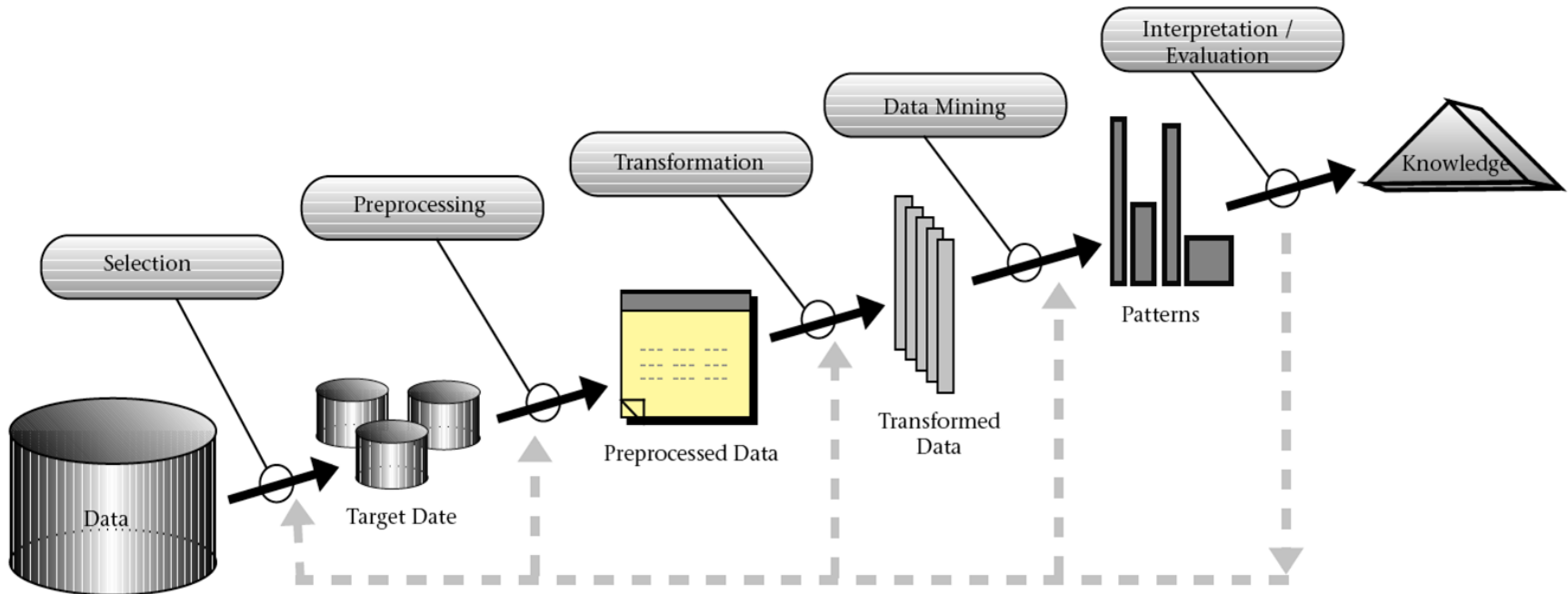
Introduction

- “Give me six hours to chop down a tree and I will spend the first four sharpening the axe.”



Abraham Lincoln, 1809-1865

Recap: The Data Mining Process



Data Preprocessing

- Your data may have some problems
 - i.e., it may be problematic for the subsequent mining steps
- Fix those problems before going on
- Which problems can you think of?



Data Preprocessing

- Problems that you may have with your data
 - Errors
 - Missing values
 - Unbalanced distribution
 - Different Scales
 - False predictors
 - Unsupported data types
 - Categorical data and Dates
 - Textual values
 - High dimensionality
 - Feature Subset Selection
 - PCA
 - Sampling

Errors in Data

- Sources
 - Malfunctioning sensors
 - Errors in manual data processing (e.g., twisted digits)
 - Storage/transmission errors
 - Encoding problems, misinterpreted file formats
 - Bugs in processing code
 - ...

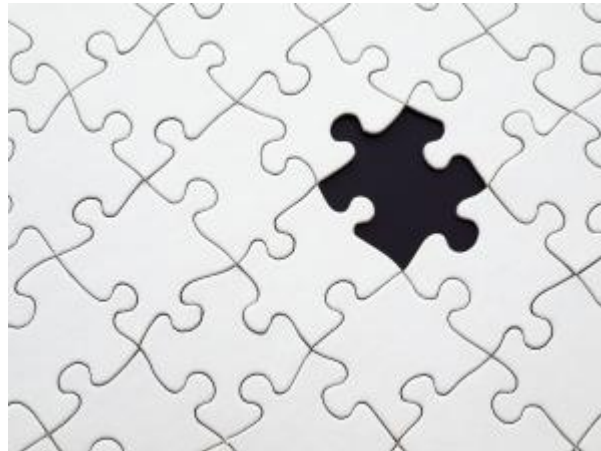


Errors in Data

- Simple remedy
 - Remove data points outside a given interval
 - This requires some domain knowledge
- Typical Examples
 - Remove temperature values outside -30 and +50 °C
 - Remove negative durations
 - Remove purchases above 1M Euro
- Advanced remedies
 - Automatically find suspicious data points (Anomaly Detection)

Missing Values

- Possible reasons
 - Failure of a sensor
 - Data loss
 - Information was not collected
 - Customers did not provide their age, sex, marital status, ...
 - ...



Missing Values

- Treatments
 - Ignore records with missing values in training data
 - Replace missing value with...
 - Default or special value (e.g., 0, “missing”)
 - Average/median value for numerics
 - Most frequent value for nominals
 - `SimpleImputer(missing_values=np.nan, strategy='mean')`
 - Try to predict missing values:
 - Handle missing values as learning problem
 - Target: attribute which has missing values
 - Training data: instances where the attribute is present
 - Test data: instances where the attribute is missing
 - `KNNImputer(n_neighbors=2, weights="uniform")`

<https://scikit-learn.org/1.5/modules/generated/sklearn.impute.SimpleImputer.html>

<https://scikit-learn.org/1.5/modules/generated/sklearn.impute.KNNImputer.html>

Missing Values

- Note: values may be missing for various reasons
 - ...and, more importantly: **at random** vs. **not at random**
- Examples for not random
 - Non-mandatory questions in questionnaires
 - e.g., “how often do you drink alcohol?”
 - Values that are only collected under certain conditions
 - e.g., final grade of your university degree (if any)
 - Values only valid for certain data sub-populations
 - e.g., “are you currently pregnant”?
 - Sensors failing under certain conditions
 - e.g., at high temperatures
- In those cases, averaging and imputation causes information loss
 - In other words: “missing” can be information!

Handling Missing Values: Caveats

- Imagine a medical trial checking for side effects of a particular drug with 250 participants
- In the trial, there are 50 people who know their blood sugar value
 - Out of those, 40/50 = 80% have an increased blood sugar value

Overall, the side effects are moderate (58/250=23,2%), but people with an increased blood sugar value have a 75% (30/40) risk of side effects

		side effects	
		Yes (n=58)	No (n=192)
increased blood sugar	Yes (n=40)	30	10
	No (n=10)	8	2
	-- (n=200)	20	180

Handling Missing Values: Caveats (ctd.)

- Assume you handle the missing value by filling in the majority value (“yes”)

Overall, the side effects are moderate ($58/250=23,2\%$), and even slightly lower ($50/240\approx 20,8\%$) for people with an increased blood sugar value

		side effects	
		Yes (n=58)	No (n=192)
increased blood sugar	Yes (n=240)	50	190
	No (n=10)	8	2

Unbalanced Distribution

- Example:
 - learn a model that recognizes HIV
 - given a set of symptoms
- Data set:
 - records of patients who were tested for HIV
- Class distribution:
 - 99.9% negative
 - 0.01% positive
- Learn a decision tree
 - It will be hard to find any splitting that significantly improves the quality



Decision tree learned:

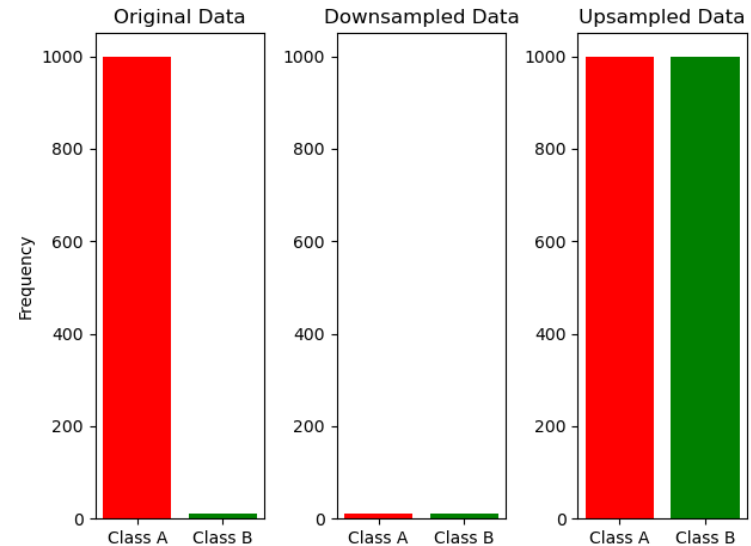
false

Resampling Unbalanced Data

- Two conflicting goals
 - Use as *much* training data as possible
 - Use as *diverse* training data as possible
- Strategies
 - Downsampling larger class
 - Conflicts with goal 1
 - Upsampling smaller class
 - Conflicts with goal 2

Resampling Unbalanced Data

- Consider an extreme example
 - 1,000 examples of class A
 - 10 examples of class B
- Downsampling
 - does not use 990 examples
- Upsampling
 - creates 100 copies of each example of B
 - likely for the classifier to simply *memorize* the 10 B cases

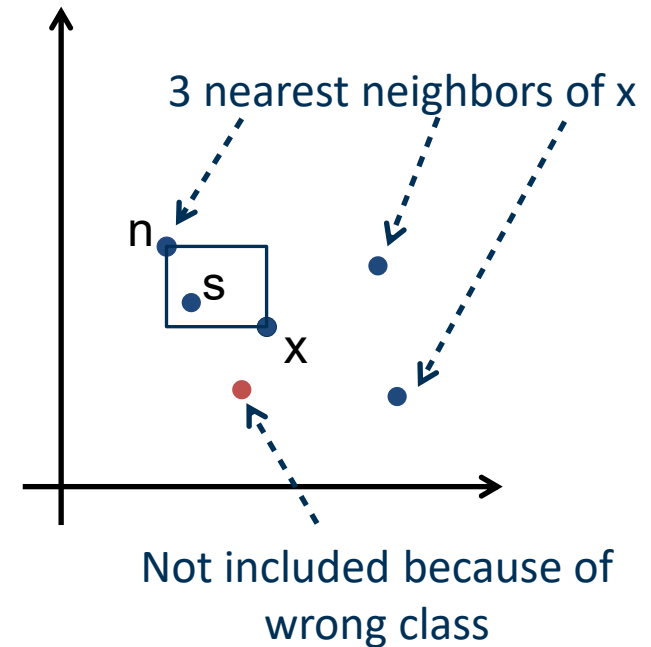


- Python:
 - <https://imbalanced-learn.org/>



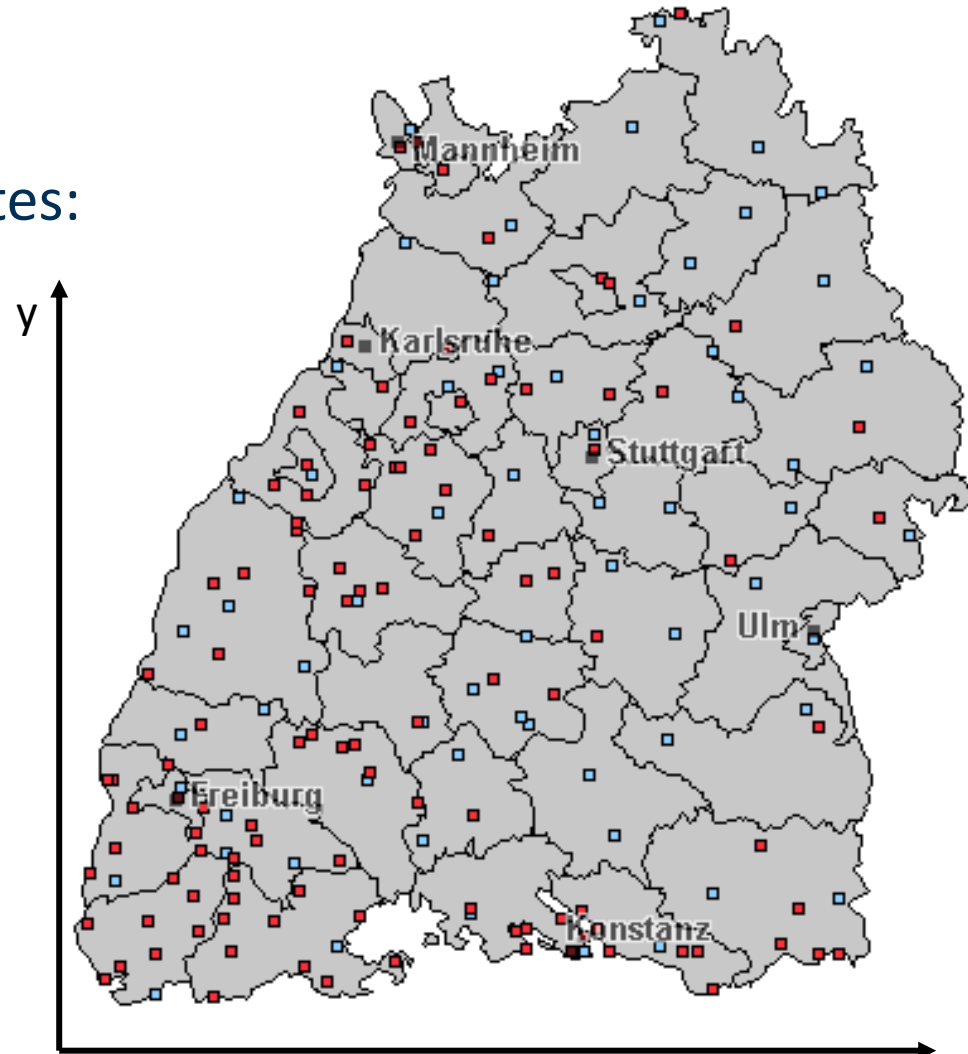
Resampling

- SMOTE (Synthetic Minority Over Sampling Technique)
 - Creates synthetic examples of minority class
- Given an example x
 - Choose one neighbor n among the k nearest neighbors (w/in same class) of x
 - Create synthetic example s
 - For each attribute a
 - $s.a \leftarrow x.a + \text{rand}(0,1) * (n.a - x.a)$



Multiple Attributes

- In previous example, we already had two attributes: x and y coordinates
- But need one distance between the examples/
weather stations
- Used linear distance (Euclidean Distance)



Multiple Attributes

- **Euclidean Distance (L_2 - norm)**

- $dist = \sqrt{\sum_{k=1}^n (p_k - q_k)^2}$

Where n is the number of dimensions (attributes) and p_k and q_k are the k -th attributes of data points p and q

- **More general (L_p - norm)**

- $dist = \sqrt[p]{\sum_{k=1}^n |p_k - q_k|^p}$
 - $= (\sum_{k=1}^n |p_k - q_k|^p)^{\frac{1}{p}}$

Example:

$L_2 = \sqrt{4^2 + 3^2} = 5$

$L_1 = 4 + 3 = 7$

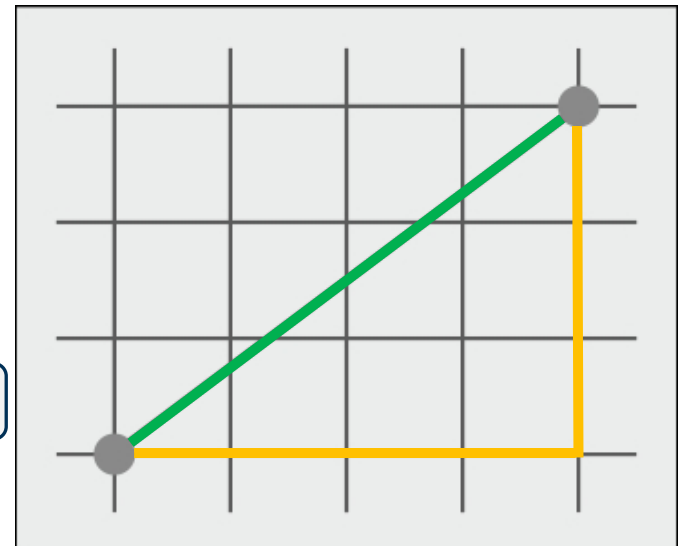
- **Manhattan distance (L_1 - norm)**

- $dist = \sum_{k=1}^n |p_k - q_k|$

- Minimum distance to go from one crossing to another

- In a squared city (like Manhattan)

Or Mannheim;



Example

	Hours of sleep	Calories burned	Well Rested?
Training Data	4	3,000	No
	8	2,500	Yes
	5	4,000	No
	8	2,000	Yes
Test Data	9	3,500	?

- Calculate the prediction for 2-NN
 - Use Mannheim/Manhattan distance

Example

	Hours of sleep	Calories burned	Well Rested?	Distances to last
Training Data	4	3,000	No	505
	8	2,500	Yes	1,001
	5	4,000	No	504
	8	2,000	Yes	1,501
Test Data	9	3,500	?	No

- Calculate the prediction for 2-NN
 - Use Mannheim/Manhattan distance
 - Distances to last example (test data):
 - $D_1 = \sum_{k=1}^n |p_k - q_k| = |4 - 9| + |3,000 - 3,500| = 5 + 500 = 505$
 - $D_2 = \sum_{k=1}^n |p_k - q_k| = |8 - 9| + |2,500 - 3,500| = 1 + 1,000 = 1,001$
 - $D_3 = \sum_{k=1}^n |p_k - q_k| = |5 - 9| + |4,000 - 3,500| = 4 + 500 = 504$
 - $D_4 = \sum_{k=1}^n |p_k - q_k| = |8 - 9| + |2,000 - 3,500| = 1 + 1,500 = 1,501$

Different scales: Normalization

	Hours of sleep	Calories burned	Well Rested?	Distances to last
Training Data	4	3,000	No	505
	8	2,500	Yes	1,001
	5	4,000	No	504
	8	2,000	Yes	1,501
Test Data	9	3,500	?	No

- Calculate the prediction for 2-NN
 - Use Mannheim/Manhattan distance
 - Distances to last example (test data):
 - $D_1 = \sum_{k=1}^n |p_k - q_k| = \underbrace{|4 - 9|}_{\text{“Hours of sleep”}} + |3,000 - 3,500| = 5 + 500 = 505$
 “Hours of sleep” does not contribute much
 - **K-NN is sensitive to scaling**

Different scales: Normalization

	Hours of sleep	Calories burned	Well Rested?	Distances to last
Training Data	4	3,000	No	505
	8	2,500	Yes	1,001
	5	4,000	No	504
	8	2,000	Yes	1,501
Test Data	9	3,500	?	No

- Needs normalization

- StandardScaler $z = \frac{x - \mu}{\sigma}$

- Standardize features by removing the mean and scaling to unit variance

- MinMaxScaler $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$

- Transform features by scaling each feature to a given range e.g. [0,1]

Different scales: Normalization

	Hours of sleep	Calories burned	Well Rested?	Distances to last
Training Data	4	3,000	No	505
	8	2,500	Yes	1,001
	5	4,000	No	504
	8	2,000	Yes	1,501
Test Data	9	3,500	?	No

- Apply MinMaxScaler $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$
 - Important: Also preprocessing steps can be “trained”
 - Hours of Sleep
 - $X_{min} = 4$ and $X_{max} = \mathbf{8}$ (not 9)
 - $H_z = \frac{z-4}{8-4} = \frac{z-4}{2}$
 - Calories Burned $C_z = \frac{z-2,000}{4,000-2,000} = \frac{z-2,000}{2,000}$

Different scales: Normalization

	Hours of sleep (normalized)	Calories burned (normalized)	Well Rested?	Distances to last	Distances to last(normalized)
Training Data	4 = 0	3,000 = 0.5	No	505	1.5
	8 = 1	2,500 = 0.25	Yes	1,001	0.75
	5 = 0.25	4,000 = 1	No	504	1.25
	8 = 1	2,000 = 0	Yes	1,501	0.5
Test Data	9 = 1.25	3,500 = 0.75	?	No	Yes

- Apply MinMaxScaler $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$
 - Important: Also preprocessing steps can be “trained”
 - Hours of Sleep
 - $X_{min} = 4$ and $X_{max} = \mathbf{8}$ (not 9)
 - $H_z = \frac{z-4}{8-4} = \frac{z-4}{2}$
 - Calories Burned $C_z = \frac{z-2,000}{4,000-2,000} = \frac{z-2,000}{2,000}$

Preprocessors also Need to be Trained

- Many preprocessing methods also have an internal representation
 - E.g. Mean and variance, minimum and maximum values
 - Do NOT apply it on the whole dataset before splitting etc

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

golf = pd.read_csv('golf.csv')

scaler = MinMaxScaler()
golf_scaled = scaler.fit_transform(golf[['Temperature', 'Humidity']])

X_train, X_test, y_train, y_test = train_test_split(golf_scaled, golf['Play'], test_size=0.3, random_state=0)
```

<https://scikit-learn.org/1.5/modules/compose.html>

Preprocessors also Need to be Trained

- Many preprocessing methods also have an internal representation
 - Function **fit_transform** for training and **transform** for testing

```
golf = pd.read_csv('golf.csv')
X_train, X_test, y_train, y_test = train_test_split(golf, golf['Play'], test_size=0.3, random_state=0)

scaler = MinMaxScaler()
golf_scaled = scaler.fit_transform(X_train[['Temperature', 'Humidity']], y_train)

golf_test_scaled = scaler.transform(X_test[['Temperature', 'Humidity']])
```


No fit on test data

Preprocessors also Need to be Trained

- How to compose multiple components

- Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
estimators = [
    ('scale', MinMaxScaler()),
    ('clf', SVC())
]
pipe = Pipeline(estimators)
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

- How to apply transformations to only a few columns

- ColumnTransformer

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

column_trans = ColumnTransformer(
    [
        ('categories', OneHotEncoder(dtype='int'), ['Outlook']),
        ('scale', MinMaxScaler(), ['Temperature'])
    ],
    remainder='drop'
)

estimators = [
    ('preprocessing', column_trans),
    ('clf', SVC())
]

pipe = Pipeline(estimators)
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Name

Component

Columns

False Predictors

- ~100% accuracy are a great result
 - ...and a result that should make you suspicious!
- A tale from the road
 - Working with our Linked Open Data extension
 - Trying to predict the world university rankings
 - With data from DBpedia
- Goal:
 - Understand what makes a top university



False Predictors

- The Linked Open Data extension
 - Extracts additional attributes from public knowledge graphs
 - e.g., DBpedia
 - Unsupervised (i.e., attributes are created fully automatically)
- Model learned: $THE < 20 \rightarrow TOP = true$
 - False predictor: target variable was included in attributes
- Other examples
 - $Mark < 5 \rightarrow passed = true$
 - $Sales > 1000000 \rightarrow bestseller = true$



Recognizing False Predictors

- By analyzing models
 - Rule sets consisting of only one rule
 - Decision trees with only one node
- Process: learn model, inspect model, remove suspect, repeat
 - until the accuracy drops
 - Tale from the road example: there were other indicators as well
- By analyzing attributes
 - Compute correlation of each attribute with label
 - Correlation near 1 (or -1) marks a suspect
- Caution: there are also strong (but not false) predictors
 - it's not always possible to decide automatically!




Unsupported Data Types

- Not every learning operator supports all data types
 - Some (e.g., SVM) cannot handle categorical data
 - Some (e.g., ID3) cannot handle numeric data
 - Dates are difficult for most learners
 - Textual values need to be transformed
- Solutions
 - Convert categorical to numeric data
 - Convert numeric to nominal data (discretization, binning)
 - Extract valuable information from dates
 - Transform textual attributes to vector representations

Conversion: Categorical to Numeric

- Two common ways to encode categorical attributes:
 - For **ordinal attributes (order is important)**
 - e.g. Grade=A, A-, B+, B, B-, C+, C, C-
 - Assign each distinct value a corresponding number preserving the order
 - A=8, A-=7, B+=6, B=5, B-=4, C+=3, C=2, C-=1



ID	Grade	...
371	A-	
433	B	


ID	Grade	...
371	7	
433	5	

- Using such a coding schema allows learners to learn valuable rules, e.g.
 - $\text{grade} > 6 \rightarrow \text{excellent_student} = \text{true}$
- Python: [OrdinalEncoder](#)

Conversion: Categorical to Numeric

- Two common ways to encode categorical attributes:
 - For **nominal attributes (no order)**
 - e.g. Color=Red, Orange,..., Violet
 - One Hot Encoding: For each value v , create a binary “flag” variable C_v , which is 1 if Color= v , 0 otherwise

ID	Color	..
371	red	
433	yellow	



ID	Color_red	Color_orange	Color_yellow	...
371	1	0	0	
433	0	0	1	

- Python: [OneHotEncoder](#)
- Special case: Binary attribute e.g. student=yes, no
 - Student = yes → student_binary = 0
 - Student = no → student_binary = 1

Conversion: Categorical to Numeric

- Many values:
 - US State Code (50 values)
 - Profession Code (7,000 values, but only few frequent)
- Approaches:
 - manual, with background knowledge
 - e.g., group US states
 - Use binary attributes
 - then apply dimensionality reduction (see later today)



Conversion: Numeric to Ordinal

- Discretization
 - Values of the attribute, e.g., age of a person:
 - 0, 4, 12, 16, 16, 18, 24, 26, 30
 - **Equal-interval binning** – for bin width of e.g., 10:
 - Bin 1: 0, 4 $[-\infty, 10)$ bin
 - Bin 2: 12, 16, 16, 18 $[10, 20)$ bin
 - Bin 3: 24, 26, 30 $[20, +\infty)$ bin
 - **Equal-frequency binning** – for bin density of e.g., 3:
 - Bin 1: 0, 4, 12 $[-, 14)$ bin
 - Bin 2: 16, 16, 18 $[14, 21)$ bin
 - Bin 3: 24, 26, 30 $[21, +]$ bin

Dealing with Date Attributes

- Dates (and times) can be formatted in various ways
 - first step: normalize and parse
- Dates have lots of interesting information in them
- Example: analyzing shopping behavior
 - time of day
 - weekday vs. weekend
 - begin vs. end of month
 - month itself
 - quarter, season
- Python: use, e.g., `datetime`

Dealing with Textual Attributes

- Preprocessing
 - Text Cleanup (remove punctuation and HTML tags)
 - Tokenization (break text into single words or N-grams)
 - Stopword Removal (e.g. the, of, and, to, an, is, that, ...)
 - Stemming (find the **stem of a word**)
 - User, users, used, using → Stem: use

```
from nltk.stem.porter import PorterStemmer

# Stem tokens
stemmer = PorterStemmer()
tokens = ['Jupiter', 'is', 'the', 'largest', 'gas', 'planet']
stems = []
for item in tokens:
    stems.append(stemmer.stem(item))
```

Dealing with Textual Attributes

- Feature Generation: Bag-of-Words
 - Each word/term becomes a feature
 - Order of words/terms is ignored
- Each document is represented by a vector

Term	Dokument																				Σ
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
oil	5	12	2	1	1	7	3	3	5	9	5	4	5	4	3	4	5	3	3	1	85
price	5	6	2	2	0	8	1	2	2	10	5	1	5	2	0	3	3	3	3	0	63
opeac	0	15	0	0	0	8	1	2	2	6	5	2	2	4	0	0	0	0	0	0	47
mln	0	4	0	0	2	4	1	0	0	3	9	0	0	0	0	3	3	0	0	2	31
market	2	5	0	0	0	3	0	2	0	10	1	2	2	0	0	0	0	0	3	0	30
barrel	2	0	1	1	0	4	0	0	1	3	3	0	1	1	0	3	3	1	0	2	26
bpd	0	4	0	0	0	7	0	0	0	2	8	0	0	2	0	0	0	0	0	0	23
dllrs	2	0	1	2	2	2	1	0	0	4	2	0	0	0	0	1	1	5	0	0	23
crude	2	0	2	3	0	2	0	0	0	0	5	2	0	2	0	0	0	2	0	1	21
saudi	0	0	0	0	0	0	0	1	0	5	7	1	4	0	0	0	0	0	0	0	18
kuwait	0	0	0	0	0	10	0	1	0	3	0	1	0	2	0	0	0	0	0	0	17
offici	0	0	0	0	0	5	1	1	0	1	4	3	1	0	0	0	0	0	1	0	17
meet	0	6	0	0	0	3	0	1	0	1	0	1	0	2	0	0	0	0	0	0	14
pct	0	0	0	0	2	0	2	2	2	1	0	0	1	0	0	1	1	0	0	2	14
product	1	6	0	0	0	1	0	0	0	0	4	0	0	0	0	0	0	0	0	1	13
accord	0	0	0	0	0	0	0	0	0	5	1	0	2	0	0	0	0	0	4	0	12
futur	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	9	0	12
minist	0	0	0	0	0	3	0	0	1	3	1	2	1	1	0	0	0	0	0	0	12
govern	0	0	0	0	0	0	5	0	6	0	0	0	0	0	0	0	0	0	0	0	11
month	0	1	0	0	0	2	2	0	1	0	5	0	0	0	0	0	0	0	0	0	11
report	0	1	0	0	0	1	8	0	0	0	1	0	0	0	0	0	0	0	0	0	11
sheikh	0	0	0	0	0	3	0	0	5	2	0	0	0	1	0	0	0	0	0	0	11
industri	0	2	0	0	0	1	1	1	1	0	0	0	0	0	0	1	2	0	1	0	10
produc	0	0	0	0	0	4	1	1	0	3	0	0	0	0	0	0	0	0	0	1	10
quota	0	2	0	0	0	4	0	0	1	1	1	0	0	1	0	0	0	0	0	0	10
reserv	0	0	0	0	3	0	0	0	1	0	0	0	0	0	0	3	3	0	0	0	10
world	0	1	0	0	0	1	3	0	1	1	0	0	1	1	0	0	0	0	1	0	10
.																					
.																					
Σ	48	204	34	39	46	219	219	73	161	180	208	57	61	54	56	68	89	44	147	32	2039

Dealing with Textual Attributes

- Different techniques for vector creation:
 - **Binary Term Occurrence:** Boolean attributes describe whether or not a term appears in the document (one-hot encoding)
 - Python: `CountVectorizer(binary=true)`
 - **Term Occurrence:** Number of occurrences of a term in the document (problematic if documents have different length)
 - Python: `CountVectorizer(binary=false)`
 - **Terms Frequency:** Attributes represent the frequency in which a term appears in the document (number of occurrences / number of words in document)
 - Python: `TfidfVectorizer(use_idf=False)`
 - **TF-IDF:** see next slide
 - Python: `TfidfVectorizer`

Dealing with Textual Attributes

- The **TF-IDF weight** (term frequency–inverse document frequency) is used to evaluate how important a word is to a corpus of documents.

- TF: Term Frequency (see last slide)

$$w_{ij} = tf_{ij} * idf_i$$

- IDF: Inverse Document Frequency.

$$idf_i = \log\left(\frac{N}{df_i}\right)$$

N : total number of docs in corpus

df_i : the number of docs in which t_i appears

- Gives more weight to rare words
- Give less weight to common words (domain-specific stopwords)

In scikit-learn:

$$idf_i = \log\left(\frac{1 + N}{1 + df_i}\right) + 1$$

Similarity of Documents

- **Jaccard Coefficient**

- Similarity measure for vectors consisting of asymmetric binary attributes

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

- Used together with binary term occurrence vectors (one-hot vectors)
 - 1 represents occurrence of specific word
 - 0 represents absence of specific word
 - most values are 0 as only small subset of the vocabulary is used in a document

Similarity of Documents

- **Jaccard Coefficient**

- Similarity measure for vectors consisting of asymmetric binary attributes

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

	Saturn	is	the	gas	planet	with	rings	Jupiter	largest	Roman	god	of	sowing
d1	1	1	1	1	1	1	1	0	0	0	0	0	0
d2	0	1	1	1	1	0	0	1	1	0	0	0	0
d3	1	1	1	0	0	0	0	0	0	1	1	1	1

- Jaccard similarities between the documents

- $Jaccard(d_1, d_2) = \frac{4}{9} = 0.44$ $Jaccard(d_2, d_3) = \frac{2}{11} = 0.18$

- $Jaccard(d_1, d_3) = \frac{3}{11} = 0.27$

Similarity of Documents

- **Cosine similarity**

- Similarity measure for comparing weighted document vectors such as term-frequency or TF-IDF vectors

$$\cos(d_1, d_2) = \frac{d_1 \bullet d_2}{\|d_1\| \|d_2\|}$$

where \bullet indicates vector dot product

$$a \bullet b = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

and $\|d\|$ is the length of the vector

$$\|d\| = \sqrt{\sum_{i=1}^n d_i^2}$$

- Example

$$d_1 = 3 \ 2 \ 0 \ 5 \ 0 \ 0 \ 0 \ 2 \ 0 \ 0$$

$$d_2 = 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 2$$

$$d_1 \bullet d_2 = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5$$

$$\|d_1\| = (3*3 + 2*2 + 0*0 + 5*5 + 0*0 + 0*0 + 0*0 + 2*2 + 0*0 + 0*0)^{0.5} = (42)^{0.5} = 6.481$$

$$\|d_2\| = (1*1 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 1*1 + 0*0 + 2*2)^{0.5} = (6)^{0.5} = 2.245$$

$$\cos(d_1, d_2) = 0.3150$$

Dense and Sparse Representation

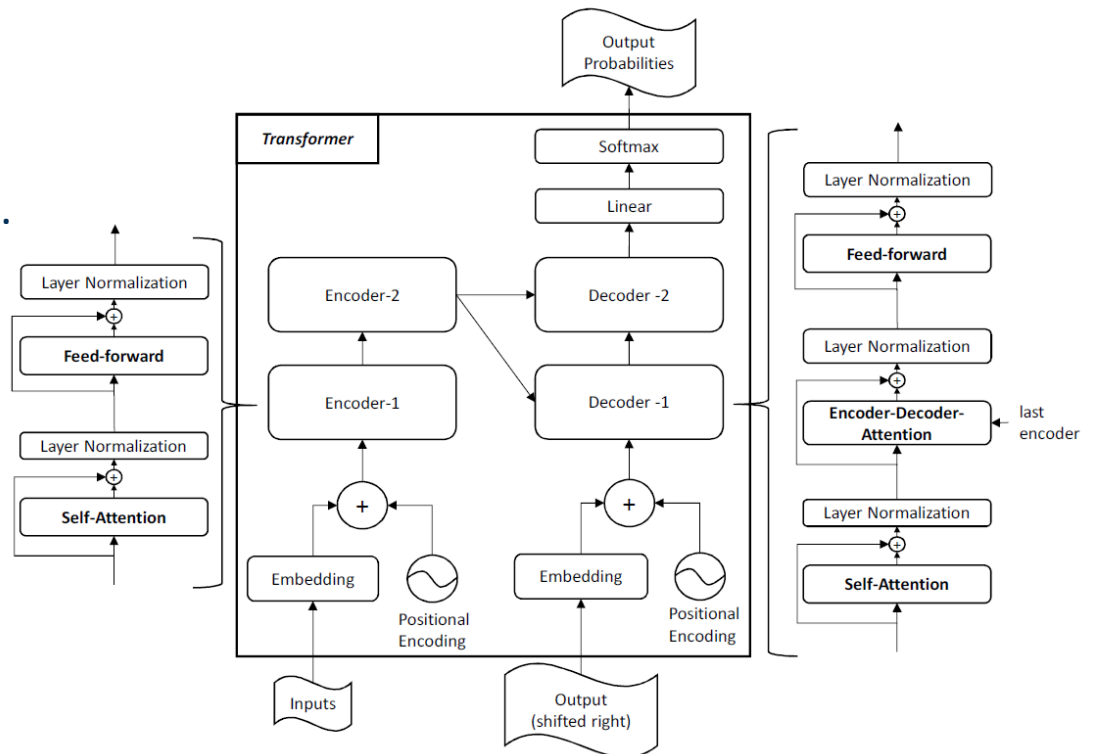
- Bag of Words is **sparse**
 - Vector dimension is tens of thousands
 - Most are zero

Term	Dokument																				Σ
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
oil	5	12	2	1	1	7	3	3	5	9	5	4	5	4	3	4	5	3	3	1	85
price	5	6	2	2	0	8	1	2	2	10	5	1	5	2	0	3	3	3	3	0	63
opec	0	15	0	0	0	8	1	2	2	6	5	2	2	4	0	0	0	0	0	0	47
mln	0	4	0	0	2	4	1	0	0	3	9	0	0	0	0	3	3	0	0	2	31
market	2	5	0	0	0	3	0	2	0	10	1	2	2	0	0	0	0	0	0	3	30
barrel	2	0	1	1	0	4	0	0	1	3	3	0	1	1	0	3	3	1	0	2	26
bpd	0	4	0	0	0	7	0	0	0	2	8	0	0	2	0	0	0	0	0	0	23
dlrs	2	0	1	2	2	2	1	0	0	4	2	0	0	0	0	1	1	5	0	0	23
crude	2	0	2	3	0	2	0	0	0	0	5	2	0	2	0	0	0	2	0	1	21
saudi	0	0	0	0	0	0	0	1	0	5	7	1	4	0	0	0	0	0	0	0	18
kuwait	0	0	0	0	0	10	0	1	0	3	0	1	0	2	0	0	0	0	0	0	17
offici	0	0	0	0	0	5	1	1	0	1	4	3	1	0	0	0	0	0	1	0	17
meet	0	6	0	0	0	3	0	1	0	1	0	1	0	2	0	0	0	0	0	0	14
pct	0	0	0	0	2	0	2	0	2	1	0	0	1	0	0	1	1	0	0	2	14
product	0	6	0	0	0	1	0	0	0	0	4	0	0	0	0	0	0	0	0	1	13
accord	0	0	0	0	0	0	0	0	0	5	1	0	2	0	0	0	0	0	4	0	12
futur	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	9	0	12
minist	0	0	0	0	0	3	0	0	1	3	1	2	1	1	0	0	0	0	0	0	12
govern	0	0	0	0	0	0	5	0	6	0	0	0	0	0	0	0	0	0	0	0	11
month	0	1	0	0	0	2	2	0	1	0	5	0	0	0	0	0	0	0	0	0	11
report	0	1	0	0	0	1	8	0	0	0	0	1	0	0	0	0	0	0	0	0	11
sheikh	0	0	0	0	0	3	0	0	5	2	0	0	0	1	0	0	0	0	0	0	11
industri	0	2	0	0	0	1	1	1	1	0	0	0	0	0	0	1	2	0	1	0	10
produc	0	0	0	0	0	4	1	1	0	3	0	0	0	0	0	0	0	0	0	1	10
quota	0	2	0	0	0	4	0	0	1	1	1	0	0	1	0	0	0	0	0	0	10
reserv	0	0	0	0	3	0	0	0	1	0	0	0	0	0	0	3	3	0	0	0	10
world	0	1	0	0	0	1	3	0	1	1	0	0	1	1	0	0	0	0	1	0	10
...																					
Σ	48	204	34	39	46	219	219	73	161	180	208	57	61	54	56	68	89	44	147	32	2039

- **Dense representation**
 - Word embeddings
 - Vector of a few hundred dimensions

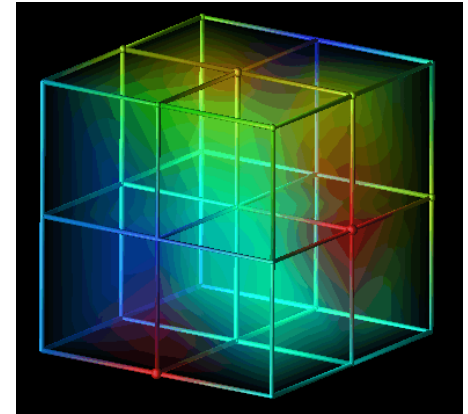
Transformer Architecture

- Encoder-only:
 - BERT, ALBERT (a lite version of **BERT**),
RoBERTa (A **R**obustly **O**ptimized **BERT** Pretraining **A**pproach)
- Decoder-only:
 - LLMs like GPT, LLaMa, ...
- Seq2Seq:
 - T-5 (Text-to-Text Transfer Transformer)
 - BART



High Dimensionality

- Datasets with large number of attributes
- Examples:
 - Text classification
 - Image classification
 - Genome classification
 - ...
- (not only a) scalability problem
 - e.g., decision tree:
search all attributes for determining one single split



Feature Subset Selection

- Preprocessing step
- Idea: only use valuable features
 - “feature”: machine learning terminology for “attribute”
- Basic heuristics: remove nominal attributes...
 - Which have more than $p\%$ identical values
 - Example: millionaire=false
 - Which have more than $p\%$ different values
 - Example: names, IDs
- Basic heuristics: remove numerical attributes
 - Which have little variation, i.e., standard deviation $< s$

Feature Subset Selection

- Basic Distinction: Filter vs. Wrapper Methods
- Filter methods
 - Use attribute weighting criterion, e.g., Chi², Information Gain, ...
 - Select attributes with highest weights
 - Fast (linear in no. of attributes), but not always optimal
- Example:
 - `X_f = SelectKBest(chi2, k=20).fit_transform(X, y)`

Feature Subset Selection

- Remove redundant attributes
 - e.g., temperature in °C and °F
 - e.g., textual features “Barack” and “Obama”
- Method:
 - compute pairwise correlations between attributes
 - remove highly correlated attributes
- Recap:
 - Naive Bayes requires independent attributes
 - Will benefit from removing correlated attributes

Feature Subset Selection

- Wrapper methods
 - Use classifier internally
 - Run with different feature sets
 - Select best feature set
- Advantages
 - Good feature set for given classifier
- Disadvantages
 - Expensive (naively: at least quadratic in number of attributes)
 - Heuristics can reduce number of classifier runs

Feature Subset Selection

- Forward selection:

```
start with empty attribute set
```

```
do {
```

```
    for each attribute {
```

```
        add attribute to attribute set
```

```
        compute performance (e.g., accuracy)
```

```
    }
```

```
    use attribute set with best performance
```

```
} while performance increases
```

- An learning algorithm is used for computing the performance
 - Cross validation is advised

Feature Subset Selection

- **Backward elimination:**

```
start with full attribute set
```

```
do {
```

```
    for each attribute in attribute set {
```

```
        remove attribute to attribute set
```

```
        compute performance (e.g., accuracy)
```

```
    }
```

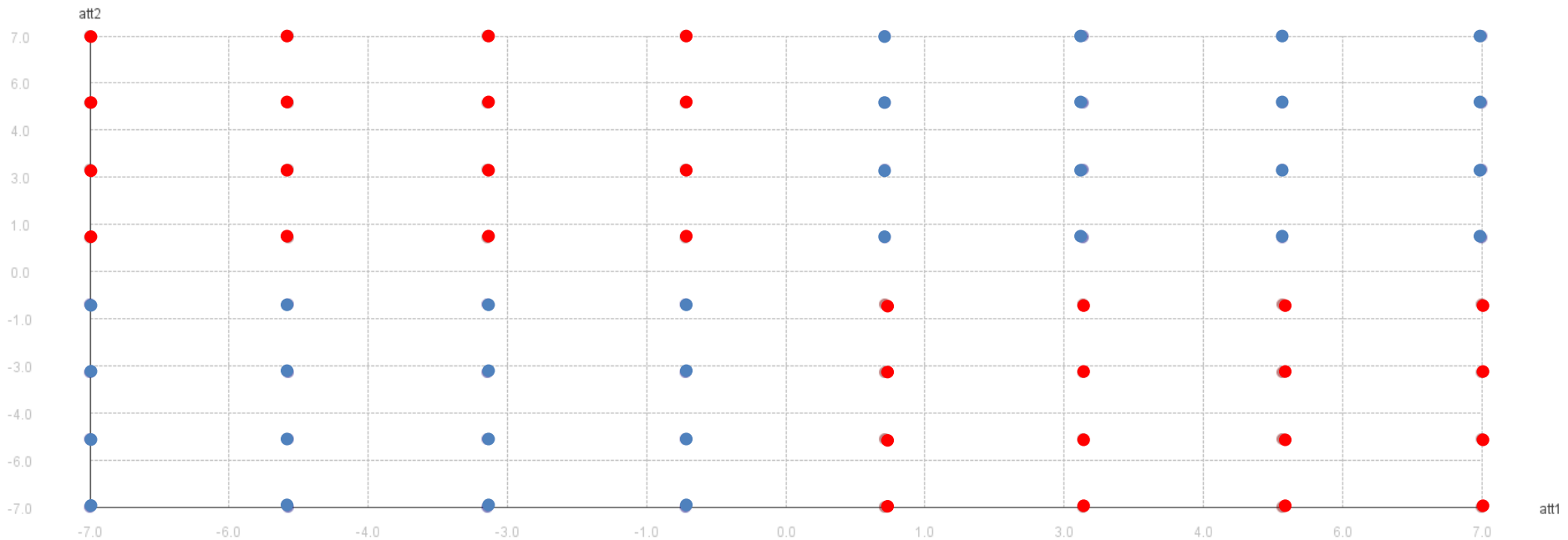
```
    use attribute set with best performance
```

```
} while performance increases
```

- **An learning algorithm is used for computing the performance**
 - Cross validation is advised

Feature Subset Selection

- The checkerboard dataset
 - Decision tree learners can perfectly learn this!
 - But what happens if we apply forward selection here?



Feature Subset Selection

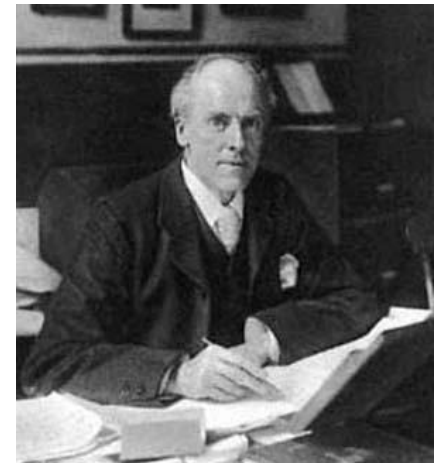
- Python:
 - Forward selection:
`SequentialFeatureSelector(direction='forward')`
 - Backward elimination:
`SequentialFeatureSelector(direction='backward')`
 - If estimator has feature importances:
 - RFECV (**R**ecursive **f**eature **e**limination with **c**ross-**v**alidation)
 - Just one selection step based on feature importances
 - SelectFromModel

Feature Subset Selection

- Further approaches
 - Brute Force search
 - Evolutionary algorithms
- Trade-off
 - Simple heuristics are fast
 - But may not be the most effective
 - Brute-force is most effective
 - But the slowest
 - Forward selection, backward elimination, and evolutionary algorithms
 - Are often a good compromise

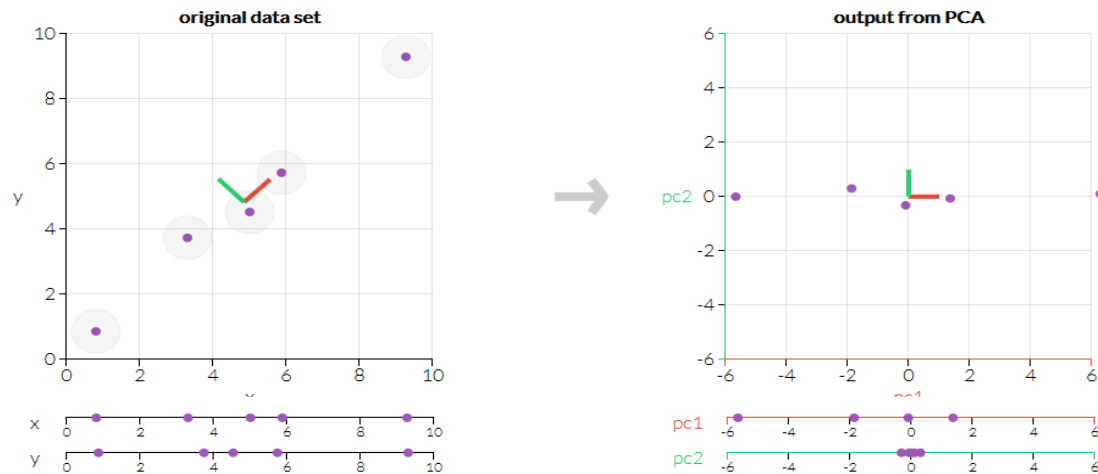
Principal Component Analysis (PCA)

- So far, we have looked at feature selection methods
 - We select a subset of attributes
 - No new attributes are created
- PCA creates a (smaller set of) new attributes
 - Artificial linear combinations of existing attributes
 - As expressive as possible
- Dates back to the pre-computer age
 - Invented by Karl Pearson (1857-1936)
 - Also known for Pearson's correlation coefficient



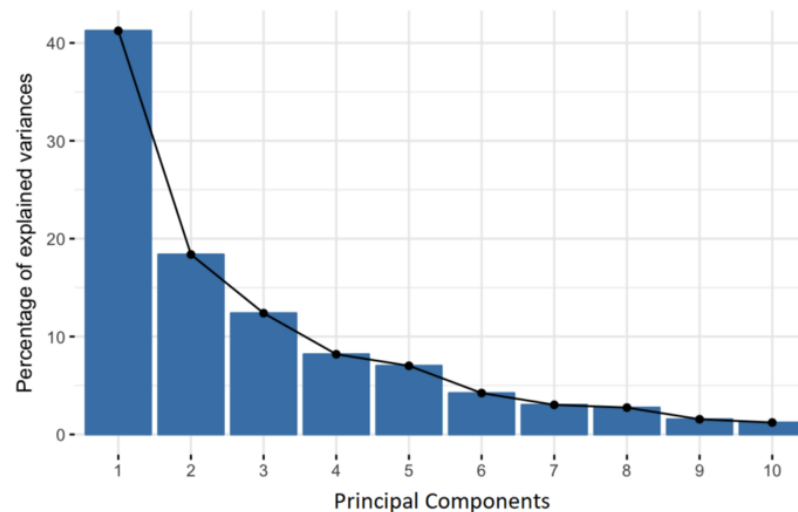
Principal Component Analysis (PCA)

- Idea: transform coordinate system so that each new coordinate (principal component) is as expressive as possible
 - Expressivity: variance of the variable
 - The 1st, 2nd, 3rd... PC should account for as much variance as possible
 - further PCs can be neglected



Principal Component Analysis (PCA)

- Principal components
 - Are *linear* combinations of the existing features
- General approach:
 - The first component should have as much variance as possible
 - The subsequent ones should also have as much variance as possible
 - And be perpendicular to the first one



Sampling

height/
examples

width/features

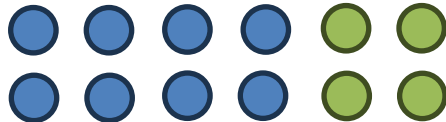
Hours of sleep	Calories burned	Well Rested?
4	3,000	No
8	2,500	Yes
...

- Feature Subset Selection reduces the **width** of the dataset
- Sampling reduces the **height** of the dataset
 - i.e., the number of instances
- Trade-off
 - Maximum usage of information
 - Fast computation
- Two approaches
 - *Stratified* sampling respects class distribution
 - *Kennard-Stone* sampling tries to select heterogenous points

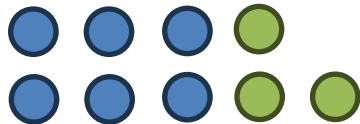
Stratified Sampling

- **Stratified sample:** Sample each class independently, so that records of the minority class are present in each sample
 - Make sure that each class is represented with approximately equal proportions in both subsets
 - Other attributes may also be considered for stratification
 - e.g., gender, age, ...

Whole data



Training (3/4)

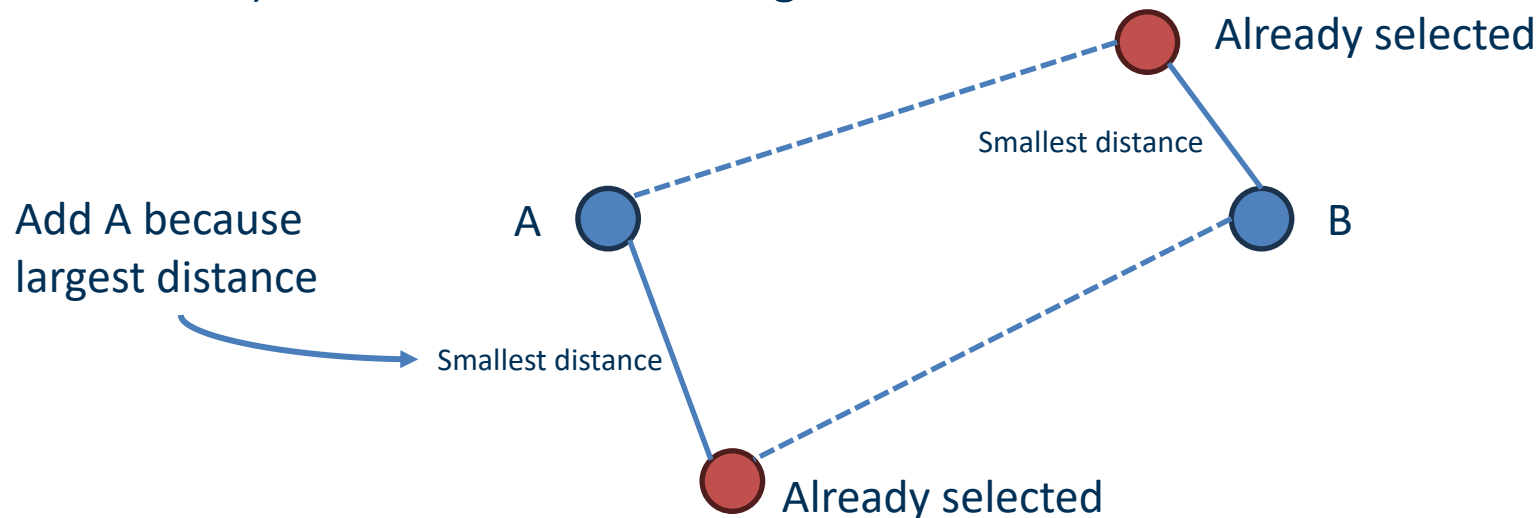


Testing (1/4)



Kennard-Stone Sampling

- 1) Compute pairwise distances of points
- 2) Add points with largest distance from one another
- 3) While target sample size not reached
 - 1) For each candidate, find smallest distance to any point in the sample
 - 2) Add candidate with largest of those smallest distances

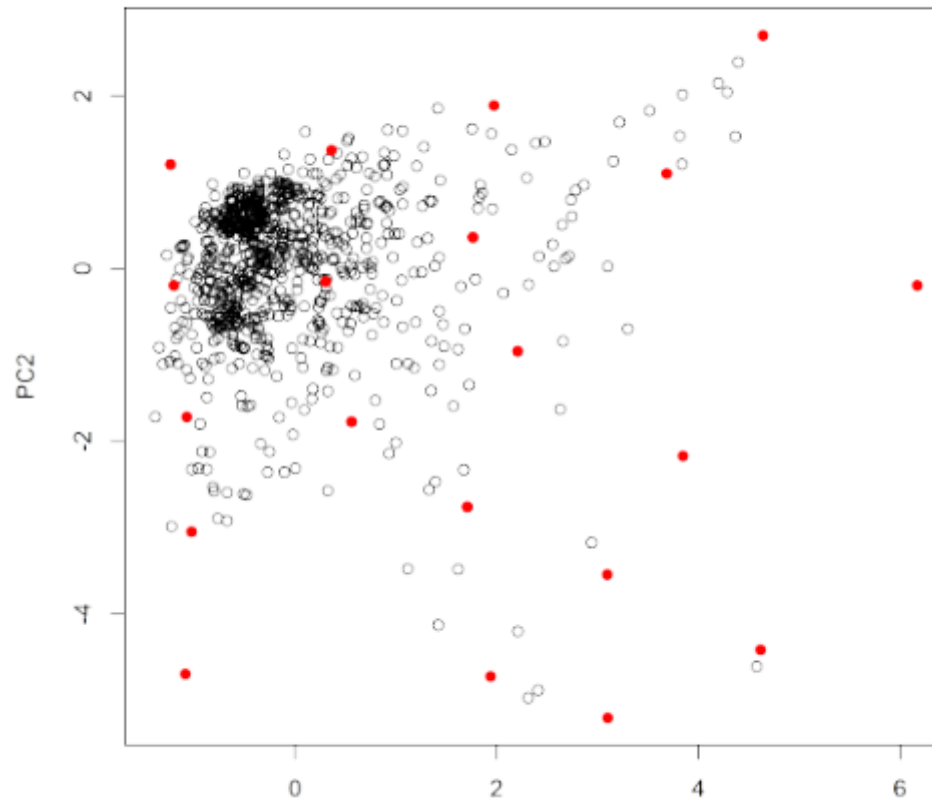


Kennard-Stone Sampling

- This guarantees that heterogeneous data points are added
 - i.e., sample gets more diverse
 - Includes more corner cases
 - But potentially also more outliers
 - Distribution may be altered
- Python: Not included in scikit-learn by default
 - Need to install separate package "kennard-stone"
 - <https://pypi.org/project/kennard-stone/>

Kennard-Stone Sampling (Example)

- Pro: a lot of rare cases covered
- Con: original distribution gets lost



Sampling Strategies and Learning Algorithms

- There are interaction effects
- Some learning algorithms rely on distributions
 - e.g., Naive Bayes
 - Usually, stratified sampling works better
- Some rely less on distributions
 - And may work better if they see more corner cases
 - e.g., Decision Trees

Titanic Dataset
Filter: 50 training examples

	Decision Tree	Naive Bayes
Stratified	.727	.752
Kennard Stone	.742	.721

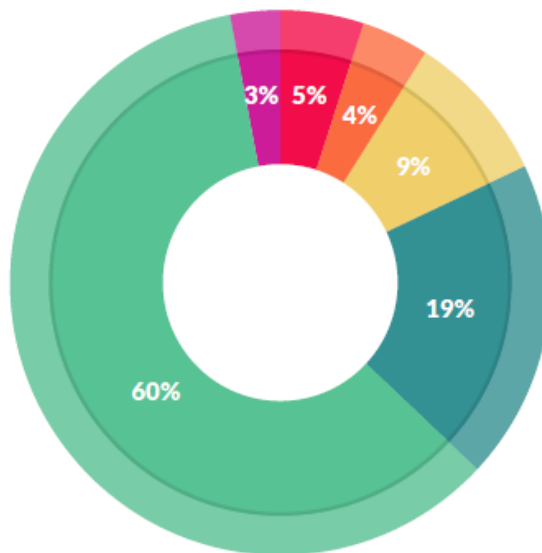
A Note on Sampling

- Often, the training data in a real-world project is already a sample
 - e.g., sales figures of last month
 - To predict the sales figures for the rest of the year
- How representative is that sample?
 - What if last month was December? Or February?
- Effect known as *selection bias*
 - Example: phone survey with 3,000 participants, carried out Monday, 9-17
 - Thought experiment: effect of selection bias for prediction, e.g., with a Naive Bayes classifier

Summary Data Preprocessing

- Raw data has many problems
 - Missing values
 - Errors
 - High dimensionality
 - ...
- Good preprocessing is essential for good data mining
 - One of the first steps in the pipeline
 - Requires lots of experimentation and fine-tuning
 - often the most time consuming step of the pipeline

Prepare Your Data



What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets; 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

Source: CrowdFlower Data Science Report 2016: <http://visit.crowdfower.com/data-science-report.html>

Questions?



Literature for this Slideset

- Python:
 - Imputation
 - <https://scikit-learn.org/1.5/modules/impute.html>
 - Preprocessing
 - <https://scikit-learn.org/1.5/modules/preprocessing.html>
 - Text feature extraction
 - https://scikit-learn.org/1.5/modules/feature_extraction.html#text-feature-extraction
 - Feature Selection
 - https://scikit-learn.org/1.5/modules/feature_selection.html