Introduction to Machine Learning Applications

What you guess(infer) from the following data

RollNo	Practical	DBMS	TOC	Machine Learning	Placed
	8.30	10.45	11.45	12.45	
1	A	P	P	P	Yes
2	A	A	A	A	NO
3	A	P	P	P	Yes
4	A	P	P	P	Yes
5	A	A	A	A	NO
6	A	A	A	A	NO
7	P	P	P	P	YES
8	P	P	P	P	NO
9	A	P	P	P	??

Introduction : What is Machine Learning

"Learning is any process by which a system improves performance from experience." - Herbert Simon

Definition by Tom Mitchell (1998):

Machine Learning is the study of algorithms that

- improve their performance P
- at some task T
- with experience E.

A well-defined learning task is given by <P, T, E>.

Traditional Programming

Machine Learning

When Do We Use Machine Learning?

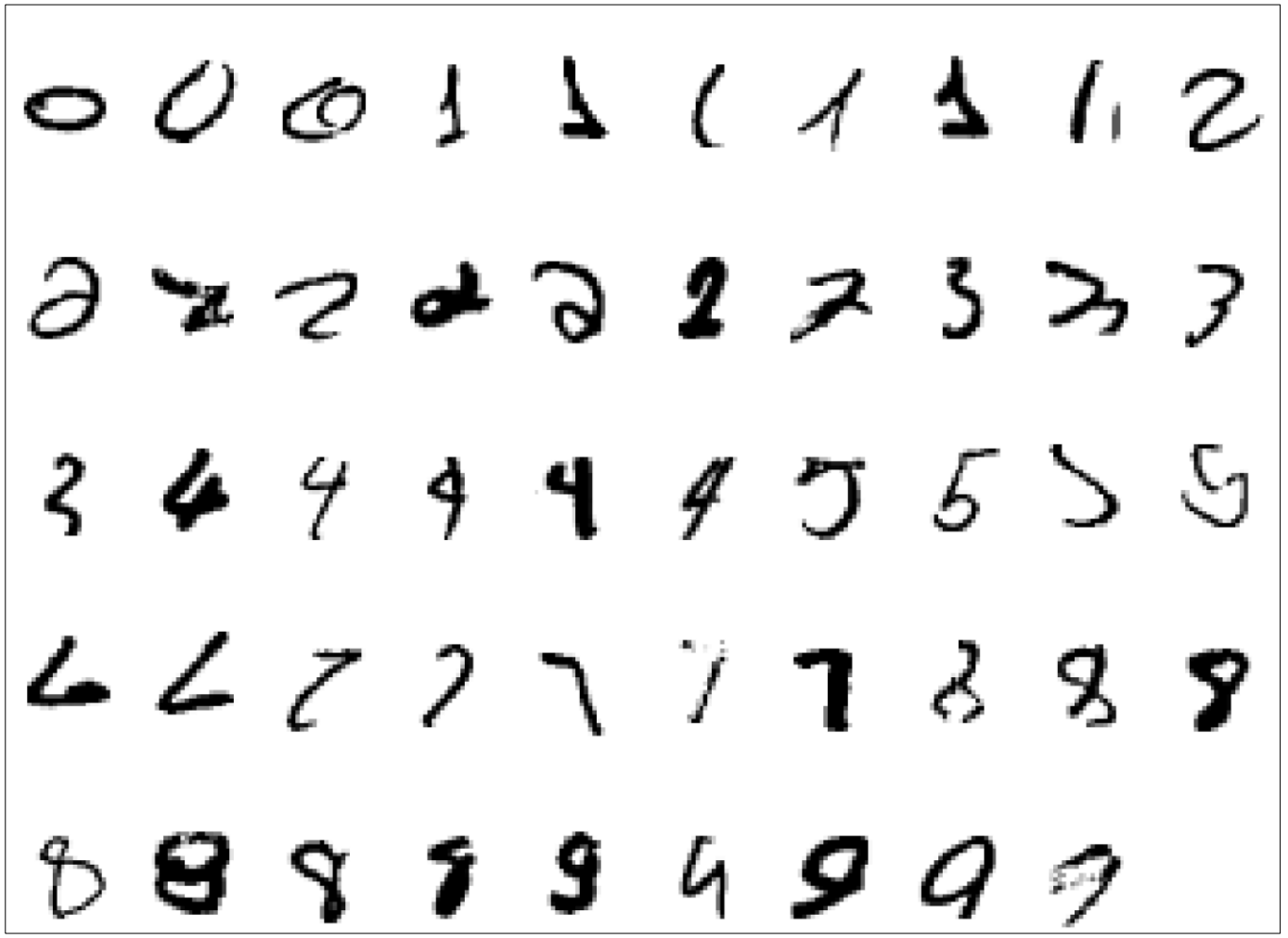
ML is used when:

- Human expertise does not exist (navigating on Mars)
- Humans can't explain their expertise (speech recognition)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)

Learning isn't always useful:

- There is no need to "learn" to calculate payroll

A classic example of a task that requires machine learning:
It is very hard to say what makes a 2



Some more examples of tasks that are best solved by using a learning algorithm

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences
- Recognizing anomalies:
  - Unusual credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant
- Prediction:
  - Future stock prices or currency exchange rates

Defining the Learning Task :Improve on task T, with respect to performance metric P, based on experience E

Q. Define the learning task for Automated checker Plyer

T: Playing checkers

P: Percentage of games won against an arbitrary opponent

E: Playing practice games against itself

Q. Define the learning task for Automated handwritten word recognition

T: Recognizing hand-written words

P: Percentage of words correctly classified

E: Database of human-labeled images of handwritten words

Q. Define a learning task for Automated Span filter

T: Categorize email messages as spam or legitimate.

P: Percentage of email messages correctly classified.

E: Database of emails, some with human-given labels

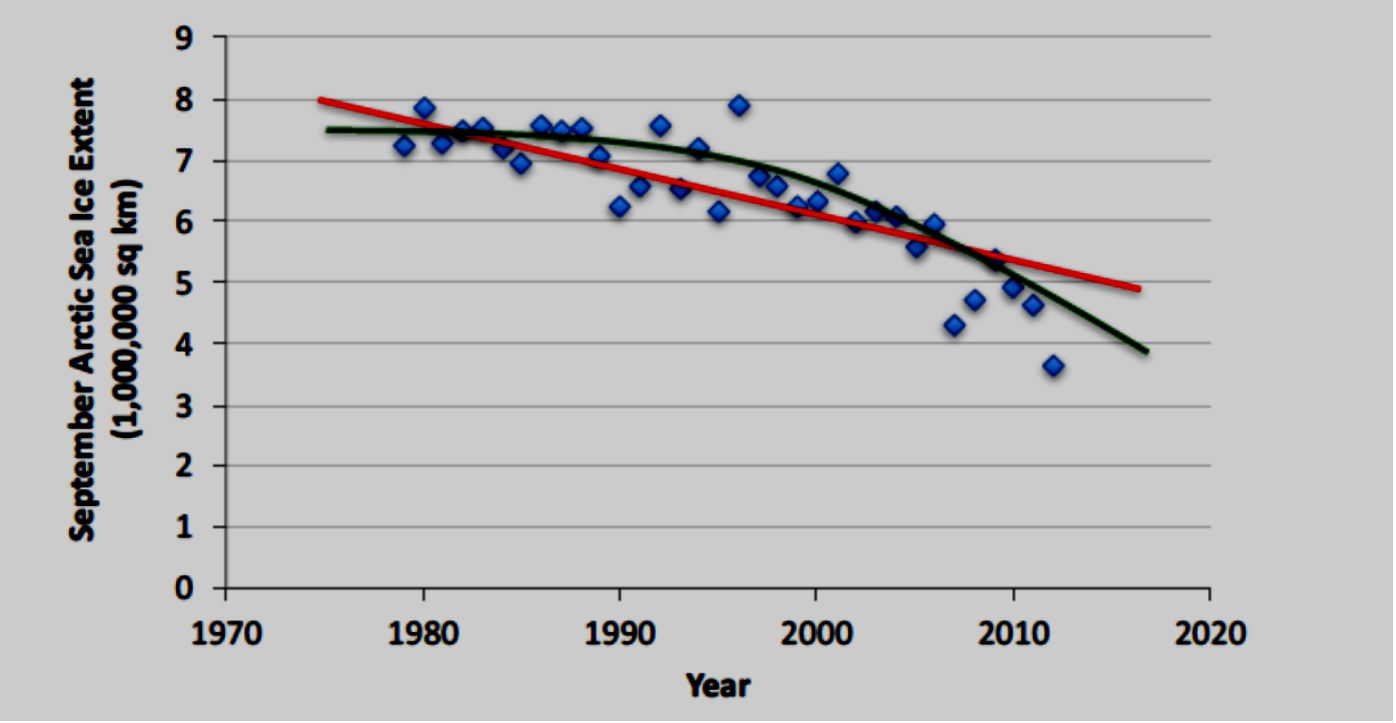
Types of Learning

- **Supervised (inductive) learning**
  - Given: training data + desired outputs (labels)
- **Unsupervised learning**
  - Given: training data (without desired outputs)
- **Semi-supervised learning**
  - Given: training data + a few desired outputs

Supervised Learning: Regression

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function f(x) to predict y given x

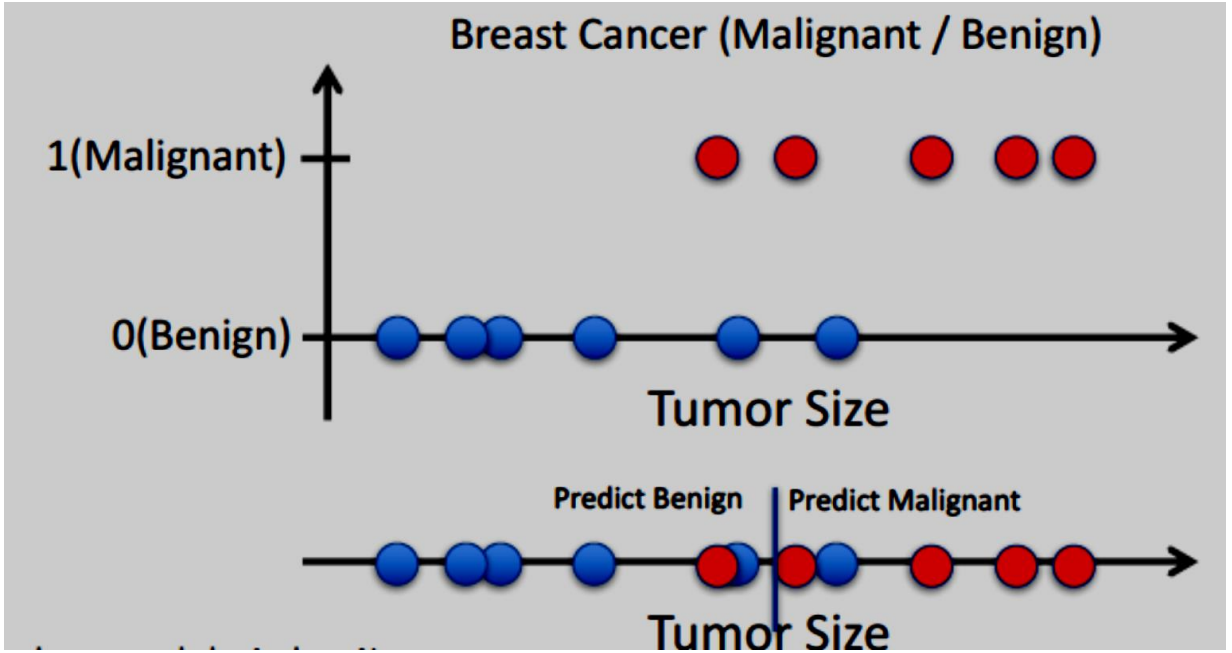
– y is real-valued == regression



Supervised Learning: Classification

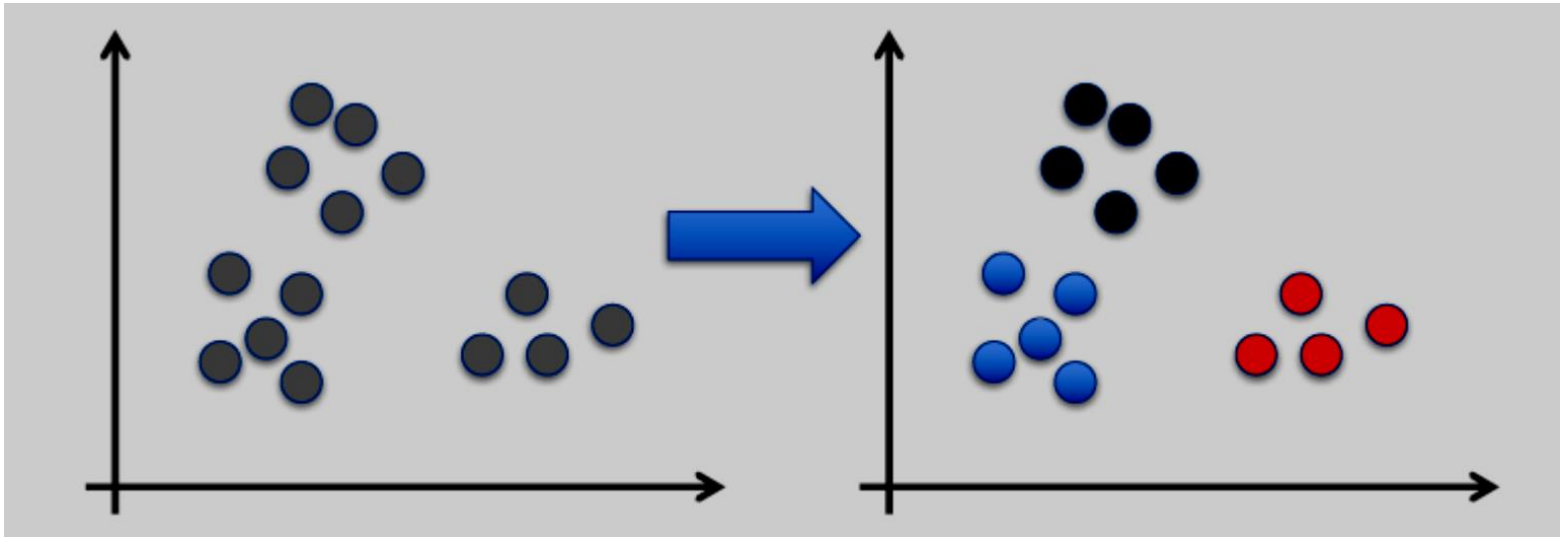
- Given (x1, y1), (x2, y2), ..., (xn, yn)
- Learn a function f(x) to predict y given x

– y is categorical == classification



Unsupervised Learning

- Given x1, x2, ..., xn (without labels)
- Output hidden structure behind the x's
  – E.g., clustering



Exercise

Solve the quiz

Solve Assignment 1 in Video Lecture 2(linear regression)

Data→ Information →Knowledge

- For Betterment of Businesses, correct decision making is important and crucial
- Right decisions are always made on past data(historical)
- Two approaches helps correct decision: Exploratory data analysis , Predictive Data Analysis
- Exploratory Data Analysis(EDA) aanalyzes the data sets to summarize their main characteristics, often with data visualization methods.
- EDA provides summary in terms of plots and charts, collect information from data

Predictive Analytics(knowledge from information)

- Predictive modelling can leverage statistics to predict future behavior
- A predictive model is consist of a number of predictors and may also contains the response variable (supervised learning).
- Predictors are the variables which are relevant to the future behavior or results.
- Once the data is collected, a statistical model can be built to learn the data, predictions are made and the model can be validated by testing the model on new data.
- Machine learning algorithms are often applied in predictive models to learn the main pattern from a training dataset in order to make predictions.

Four steps to apply data analytics:

1. Define your Objective
    - What are you trying to achieve?
    - What could the result look like?
2. Understand Your Data Source
    - What information do I need?
    - Can I get the data myself, or do I need to ask an IT resource? Or any other sources
3. Prepare Your Data
    - Does the data need to be cleansed?
    - Does the data need to be normalized?
4. Analyze Data

Data Information and Knowledge

- Data–Items that are the most elementary descriptions of things, events, activities, and transactions, May be internal or external
- Information–Organized data that has meaning and value
- Knowledge–Processed data or information that conveys understanding or learning applicable to a problem or activity

Data → Information → Knowledge

Decision Support System

- **Decision-support systems** are used to make business decisions, often based on data collected by on-line transaction-processing systems.

- Examples of business decisions:
  - What items to stock?

- Examples of data used for making decisions
  - Retail sales transaction details
  - Customer profiles (income, age, gender, etc.)

DSS Overview

- A **data warehouse** archives information gathered from multiple sources, and stores it under a unified schema, at a single site.
  - –Important for large businesses that generate data from multiple divisions, possibly at multiple sites
  - –Data may also be purchased externally
- **Data analysis** tasks are simplified by specialized tools and SQL extensions and Statistical Analysis
  - For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year
  - As above, for each product category and each customer category
- **Machine Learning** seeks to discover knowledge automatically in the form of statistical rules and patterns from large databases.

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- Pandas
- SciPy
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more …

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

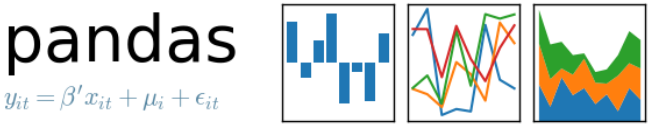
Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

Link: https://www.scipy.org/scipylib/



Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Link: https://seaborn.pydata.org/

Start Jupyter nootebook

Loading Python Libraries

In [ ]:
```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Press `Shift+Enter` to execute the *jupyter* cell

Reading data using pandas

```
In [ ]: #Read csv file
        df = pd.read_csv("C:\\Users\\Administrator\\Desktop\\Data Analytics with
        Python\\Emp.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1’)
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

Exploring data frames

In [3]:
```
#List first 5 records
df.head()
```

Out[3]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Hands-on exercises

- ✓ Try to read the first 10, 20, 50 records;
- ✓ Can you guess how to view the last few records;

Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

In [4]:
```
#Check a particular column type
df['salary'].dtype
```

Out[4]:
```
dtype('int64')
```

In [5]:
```
#Check types for all the columns
df.dtypes
```

Out[4]:

rank	object
discipline	object
phd	int64
service	int64
sex	object
salary	int64

dtype: object

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Hands-on exercises

- ✓ Read Any CSV File
- ✓ Display first 5 Rows
- ✓ Display Datatype of all columns
- ✓ How many elements are there?
- ✓ What are the column names?

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Hands-on exercises

- ✓ Give the summary for the numeric columns in the dataset
- ✓ Calculate standard deviation for all numeric columns;
- ✓ What are the mean values of the first 5 records in the dataset?

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

df[‘name’,….,…]

Method 2: Use the column name as an attribute:

df.name

Hands-on exercises: Use Emp.csv

- ✓ Calculate the basic statistics for the *salary* column;
- ✓ Find how many values in the *salary* column (use *count* method);
- ✓ Calculate the average salary;

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to dplyr() function in R

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is created we can calculate various statistics for each group:

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby('rank')[['salary']].mean()
```

rank	salary
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
- by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

In [ ]:
```
#Select column salary:
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

In [ ]:
```
#Select rows by their position:
df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

In [ ]:
```
#Select rows by their labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

In [ ]:
```
#Select rows by their labels:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

Data Frames: Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

In [ ]:
```
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation

Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R. It specifically targets statistical data visualization

To show graphs within Python notebook include inline directive:

In [ ]:
```
%matplotlib inline
```

Graphics

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Exercise

- Solve the quiz uploaded

Basic Steps of Machine Learning

- Understand domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learn models
- Interpret results
- Consolidate and deploy discovered knowledge

Data Preparation

- Data Pre-processing
  - Data Cleaning, Data Transformation, Data Reduction, Data Feature selection, Data Encoding
- Divide data into train and test samples

Data Cleaning

- Real-world data is generally incomplete and noisy, and is likely to contain irrelevant and redundant information or errors
- Data preprocessing, which is an important step in Machine Learning helps transform the raw data to an understandable format
- ML techniques are quite sensitive to the predictors, such as linear regression.
- Thus, examining and preprocessing data before entering the ML model is essential.
- We will see some important methods in data preprocessing, including data cleaning, data transformation and data reduction.

Contd..

- incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
  - e.g., occupation=“ ”
- noisy: containing errors or outliers
  - e.g., Salary=“-10”
- inconsistent: containing discrepancies in codes or names
  - e.g., Age=“42” Birthday=“03/07/1997”
  - e.g., Was rating “1,2,3”, now rating “A, B, C”
  - e.g., discrepancy between duplicate records

Dealing with missing data

- Techniques of handling missing data generally divided into two strategies: removing the missing data directly, Fill/Impute the missing data.
- If the missing data is distributed at random (not dense)and the dataset is large enough, then the removal of missing data has little effect on the performance of analysis.
- However, if one of the above conditions is not satisfied, then simply removing the missing data is inappropriate.
- The second strategy is to fill in or impute the missing data based on the rest of the data.
- Generally, there are two approaches. One method simply uses the average of the predictor to fill in the missing value.
- Alternatively, we can use a learning algorithm such as Bayes or decision tree (intelligent methods)to predict the missing value. It is worth noting that additional uncertainty is added by the imputation.

Missing Values

Missing values are marked as NaN

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

In [ ]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Pandas functions for handling missing values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Managing missing features with Imputer

Following are the ways to handle missing features

1. Removing the whole line
2. Creating sub-model to predict those features
3. Using an automatic strategy to input them according to the other known values

- With option 3 , scikit-learn offers the class Imputer, which is responsible for filling the holes using a strategy based on the mean (default choice), median, or frequency (the most frequent entry will be used for all the missing ones).

Example

```
from sklearn.preprocessing import Imputer
>>> data = np.array([[1, np.nan, 2], [2, 3,
    np.nan], [-1, 4, 2]])
>>> imp = Imputer(strategy='mean')
>>> imp.fit_transform(data)
array([[ 1. , 3.5, 2. ],
[ 2. , 3. , 2. ],
[-1. , 4. , 2. ]])
```

```
>>> imp = Imputer(strategy='median')
>>> imp.fit_transform(data)
array([[ 1. , 3.5, 2. ],
[ 2. , 3. , 2. ],
[-1. , 4. , 2. ]])
```

```
>>> imp = Imputer(strategy='most_frequent')
>>> imp.fit_transform(data)
array([[ 1., 3., 2.],
[ 2., 3., 2.],
[-1., 4., 2.]])
```

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan,
    strategy='mean')
```

Data scaling and normalization

- A generic dataset is made up of different values which can be drawn from different distributions, having different scales and, sometimes, there are also outliers
- it's always preferable to standardize datasets before processing them
- Methods:
  1. StandardScaler : Normalize data to unit variance. Influence by Outlier
  2. MinMaxScaler : MinMaxScaler rescales the data set such that all feature values are in the range [0, 1]
  3. MaxAbsScaler : MaxAbsScaler differs from the previous scaler such that the absolute values are mapped in the range [0, 1]
  4. RubustScaler : the centering and scaling statistics of this scaler are based on percentiles and are therefore not influenced by a few number of very large marginal outliers.
  5. Normalizer : L1,L2,Max

StandardScaler

```
from sklearn.preprocessing import StandardScaler
>>> ss = StandardScaler()
>>> scaled_data = ss.fit_transform(data)
```

Normalizer

$$Max\ norm: \quad \|X\|_{max} = \frac{X}{|max_i\{X\}|}$$

$$L1\ norm: \quad \|X\|_{L1} = \frac{X}{\sum_i |x_i|}$$

$$L2\ norm: \quad \|X\|_{L2} = \frac{X}{\sqrt{\sum_i |x_i|^2}}$$

Yi = [Xi - min(X)]/[max(X) - min(X)]

```
from sklearn.preprocessing import Normalizer
data = np.array([[1.0, 2.0],[2.0,3.0]])
n_max = Normalizer(norm='max')
n_max.fit_transform(data)
n_l1 = Normalizer(norm='l1')
n_l1.fit_transform(data)
```

```
array([[0.5 , 1. ], [0.66666667, 1. ]])
```

```
array([[0.33333333, 0.66666667], [0.4 , 0.6
]])
```

RobustScaler

- The centering and scaling statistics of this scaler are based on percentiles and are therefore not influenced by a few number of very large marginal outliers.

Data Transformation: Normalization

- Min-max normalization: to [new_min$_A$, new_max$_A$]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to $\frac{73,600-12,000}{98,000-12,000}(1.0-0)+0 = 0.716$

- Z-score normalization (μ: mean, σ: standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Ex. Let μ = 54,000, σ = 16,000. Then $\frac{73,600-54,000}{16,000} = 1.225$

- Normalization by decimal scaling

$$v' = \frac{v}{10^j} \quad \text{Where } j \text{ is the smallest integer such that Max}(|v'|) < 1$$

MinMaxScalar , MaxAbsScaler

- MinMaxScaler rescales the data set such that all feature values are in the range [0, 1]
- MaxAbsScaler differs from above scalar such that the absolute values are mapped in the range [0, 1]. On positive only data, this scaler behaves similarly to MinMaxScaler

```
from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1., 2.], ... [ 2., 0., 0.], ... [ 0., 1., -1.]])
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[0.5 , 0. , 1. ], [1. , 0.5 , 0.33333333], [0. , 1. , 0. ]])
```

Feature selection and filtering

- An unnormalized dataset with many features contains information proportional to the independence of all features and their variance
- entropy H(X) : Information measure for specific feature
- variance threshold is a useful approach to remove all those elements whose contribution (in terms of variability and so, information) is under a predefined level
- scikit-learn provides the class VarianceThreshold

Example

```
from sklearn.feature_selection import VarianceThreshold
>>> X[0:3, :]
array([[-3.5077778 , -3.45267063, 0.9681903 ],
[-3.82581314, 5.77984656, 1.78926338],
[-2.62090281, -4.90597966, 0.27943565]])
>>> vt = VarianceThreshold(threshold=1.5)
>>> X_t = vt.fit_transform(X)
>>> X_t[0:3, :]
array([[-0.53478521, -2.69189452],
[-5.33054034, -1.91730367],
[-1.17004376, 6.32836981]])
```

The third feature has been completely removed because its variance is under the selected threshold

Data reduction and Feature Extraction

Correlation

- df.corr()
- print(df.corr(method="pearson"))
  sns.heatmap(df.corr(method="pearson"))
- sns.heatmap(df.corr())

Principal Component Analysis

Introduction

- An important machine learning method for dimensionality reduction is called Principal Component Analysis.
- It is a method that uses simple matrix operations from linear algebra and statistics to calculate a projection of the original data into the same number or fewer dimensions.
- It can be thought of as a projection method where data with m-columns (features) is projected into a subspace with m or fewer columns, whilst retaining the essence of the original data.
- The PCA method can be described and implemented using the tools of linear algebra.

PCA is linear dimensionality reduction technique. With PCA, we take as input our original data and try to find a combination of the input features which can best summarize the original data distribution so that to reduce its original dimensions.
PCA is able to do this by maximizing variances and minimizing the reconstruction error by looking at pair wised distances. In PCA, our original data is projected into a set of orthogonal axes and each of the axes gets ranked in order of importance.

Steps

- PCA is an operation applied to a dataset, represented by an n x m matrix A
- The first step is to calculate the mean values of each column(dimention/features)
- We need to center the values in each column by subtracting the mean column value.
- Calculate the covariance matrix of the centered matrix C.

  Correlation is a normalized measure of the amount and direction (positive or negative) that two columns change together. Covariance is a generalized and unnormalized version of correlation across multiple columns. A covariance matrix is a calculation of covariance of a given matrix with covariance scores for every column with every other column, including itself.
- Calculate the Eigen decomposition of the covariance matrix V. This results in a list of eigenvalues.
- Find list of eigenvectors : *The eigenvectors represent the directions or components for the reduced subspace of B, whereas the eigenvalues represent the magnitudes for the directions.*
- Next original Data can be projected into the (new)subspace via matrix multiplication of it with Eigen vectors
- This is called the covariance method for calculating the PCA, although there are alternative ways to calculate it.

About Eigen vectors and Values

- If all eigenvalues have a similar value, then we know that the existing representation may already be reasonably compressed or dense and that the projection may offer little.
- If there are eigenvalues close to zero, they represent components or axes of B that may be discarded.
- A total of m or less components must be selected to comprise the chosen subspace.
- Ideally, we would select k eigenvectors, called principal components, that have the k largest eigenvalues.
- The eigenvectors can be sorted by the eigenvalues in descending order to provide a ranking of the components or axes of the new subspace for A.

Covariance Matrix

cor(x,y)=	cov(x,x)	cov(x,y)
	cov(y,x)	cov(y,y)

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

A	B	C	D	E	F	G	H
	x	y	x-xavg	y-yavg	D*E	D*D	E*E
1	2.5	2.4	0.69	0.49	0.3381	0.4761	0.2401
2	0.5	0.7	-1.31	-1.21	1.5851	1.7161	1.4641
3	2.2	2.9	0.39	0.99	0.3861	0.1521	0.9801
4	1.9	2.2	0.09	0.29	0.0261	0.0081	0.0841
5	3.1	3	1.29	1.09	1.4061	1.6641	1.1881
6	2.3	2.7	0.49	0.79	0.3871	0.2401	0.6241
7	2	1.6	0.19	-0.31	-0.0589	0.0361	0.0961
8	1	1.1	-0.81	-0.81	0.6561	0.6561	0.6561
9	1.5	1.6	-0.31	-0.31	0.0961	0.0961	0.0961
10	1.1	0.9	-0.71	-1.01	0.7171	0.5041	1.0201
Average OR Sum	1.81	1.91	0	3.55271E-15	5.539	5.549	6.449
	Averages		Summations				
				N-1=9			

Eigen Values and Eigen Vectors

$$[Covariance\ \ matrix] \cdot [Eigenvector] = [eigenvalue] \cdot [Eigenvector]$$

Eigenvalues and Eigenvectors

- Eigenvalues and eigenvectors of matrices are needed for principal component analysis, principal component regression, and assessment of the impact of collinearity

$$
\begin{aligned}
Az_i &= \lambda_i z_i, \\
\text{or} \quad Az_i - \lambda_i z_i &= 0, \\
\text{or} \quad (A - \lambda_i I)z_i &= 0, \quad i = 1, \ldots, n. \qquad (2.13)
\end{aligned}
$$

The λ_i are the n eigenvalues (characteristic roots or latent roots) of the matrix A and the z_i are the corresponding (column) eigenvectors (characteristic vectors or latent vectors).

There are nonzero solutions to equation 2.13 only if the matrix $(A - \lambda_i I)$ is less than full rank—that is, only if the determinant of $(A - \lambda_i I)$ is zero. The λ_i are obtained by solving the general determinantal equation

$$
|A - \lambda I| = 0. \qquad (2.14)
$$

Step by Step solution for PCA

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Step 1: Get some data

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Step 2: Subtract the mean

Xmean=1.81
Ymean=1.91

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

Step 3: Calculate the covariance matrix

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

Contd..

Step 4: Calculate the eigenvectors and eigenvalues of the covariance Matrix

Eigen Values :
|A-eI |=0 , i.e det(A-eI)=0

	X	Y
X	0.616555555555556-E	0.615444444
Y	0.615444444	0.716555555555556 -E

For illustration, consider the matrix

$$A = \begin{bmatrix} 10 & 3 \\ 3 & 8 \end{bmatrix}.$$

The eigenvalues of A are found by solving the determinantal equation (equation 2.14),

$$|(A - \lambda I)| = \left| \begin{bmatrix} 10-\lambda & 3 \\ 3 & 8-\lambda \end{bmatrix} \right| = 0$$

$$(10-\lambda)(8-\lambda) - 9 = \lambda^2 - 18\lambda + 71 = 0.$$

The solutions to this quadratic (in λ) equation are

$$\lambda_1 = 12.16228 \quad \text{and} \quad \lambda_2 = 5.83772$$

Answer: eigen values

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

Contd..

Step 5: Choosing components and forming a feature vector

$$FeatureVector = (eig_1\ eig_2\ eig_3\ ....\ eig_n)$$

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

Leave less significant component and only have a single column:

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

Contd..

Step 5: Deriving the new data set

$$FinalData = RowFeatureVector \times RowDataAdjust,$$

Where RowFeatureVector is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and is the mean-adjusted data *transposed, ie. the data* items are in each column, with each row holding a separate dimension

Transformed Data=

x	y
-.827970186	-.175115307
1.77758033	.142857227
-.992197494	.384374989
-.274210416	.130417207
-1.67580142	-.209498461
-.912949103	.175282444
.0991094375	-.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-.162675287

Data transformed with 2 eigenvectors

Transformed Data (Single eigenvector)

x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056

Coding

```
1. from numpy import array
2. from numpy import mean
3. from numpy import cov
4. from numpy.linalg import eig
5. # define a matrix
6. A = array([[1, 2], [3, 4], [5, 6]])
7. print(A)
8. # calculate the mean of each column
9. M = mean(A.T, axis=1)
10. print(M)
11. # center columns by subtracting column means
12. C = A - M
13. print(C)
```

Contd….

```
14. # calculate covariance matrix of centered matrix
15. V = cov(C.T)
16. print(V)
17. # eigendecomposition of covariance matrix
18. values, vectors = eig(V)
19. print(vectors)
20. print(values)
21. # project data
22. P = vectors.T.dot(C.T)
23. print(P.T)
```

Questions

- Find a method to calculate eigen vectors from eigen values

Data Splitting

- Models may produce overly optimistic performance estimates during learning processes
- A good approach to inspect the estimate is to test the model on samples that were not used in training process.
- For this purpose, one can split the data set into “training” data set and “test” or “validation” data set
- Using the “training” data set to create the mode
- While using the “test” or “validation” data set to qualify performance.
- A simple way to split the data is to take a simple random sample.

Scikit- learn Dataset

- scikit-learn provides some built-in datasets that can be used for testing purposes
- available in the package sklearn.datasets
- For example, considering the Boston house pricing dataset (used for regression), we have:

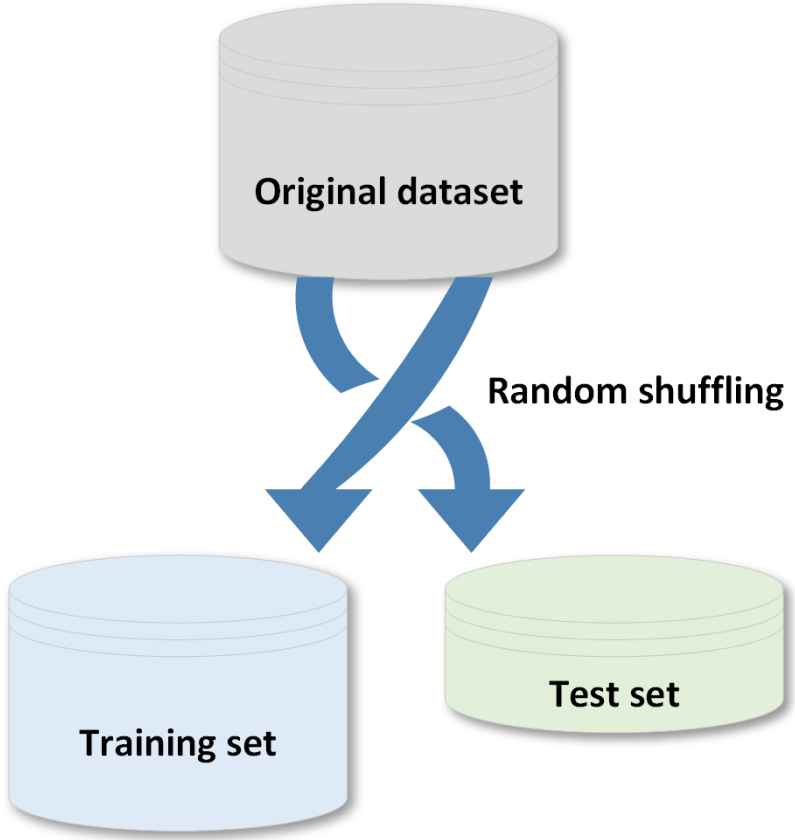
```
from sklearn.datasets import load_boston
>>> boston = load_boston()
>>> X = boston.data
>>> Y = boston.target
>>> X.shape
(506, 13)      " 506 samples with 13 features
>>> Y.shape
(506,)
```

Reference : //s c i k i t - l e a r n . o r g /s t a b l e /d a t a s e t s /

Creating training and test sets

- When a dataset is large enough, it's a good practice to split it into training and test sets; the former to be used for training the model and the latter to test its performances.

```
from sklearn.model_selection import train_test_split
>>> X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.25, random_state=1)
```



Managing categorical data

- In many classification problems, the target dataset is made up of categorical labels which cannot immediately be processed by any algorithm.
- An encoding is needed and scikit-learn offers at least two valid options.

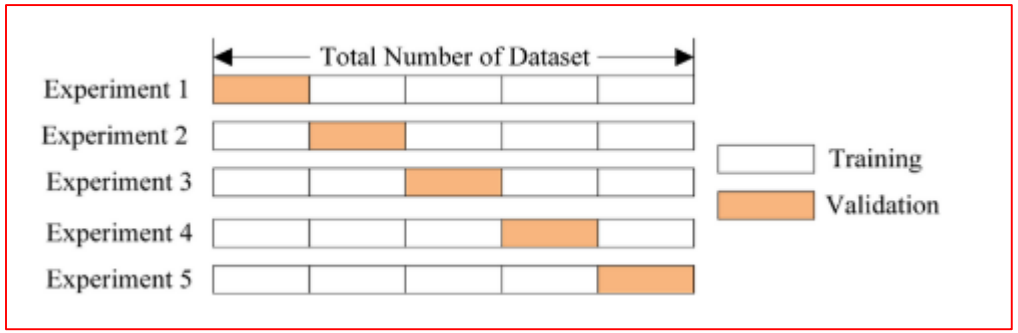
```
import numpy as np
Y = np.random.choice(('Male','Female'), size=(10))
Y
array(['Female', 'Male', 'Female', 'Male', 'Female',
'Female', 'Male', 'Male', 'Male', 'Female'],
dtype='<U6')
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
yt = le.fit_transform(Y)
print(yt)
```

Question

- What is LabelBinarizer?
- How it different from LabelEncoder() ?
- What is DictVectorizer() ?

K-Fold Cross-Validation



- In k-fold cross-validation, the samples are randomly partitioned into k roughly equal sized subsamples.
- Among the k sample sets, the first subset is retained as the validation data for testing the model and the held-out samples are used as training data to fit the model.
- The first subset is then return to the training data and the process repeats k times while each of the k subsets used exactly once as the validation data.
- The k resampled estimates of performance are aggregated and summarized to produce a single estimation

Feature Engineering

- features for representing *categorical data*, features for representing *text*, and features for representing *images*

- *Check the python script*

Classification

Classification maps data into predefined groups or classes

It is useful in

- Supervised learning
- Pattern recognition
- Prediction

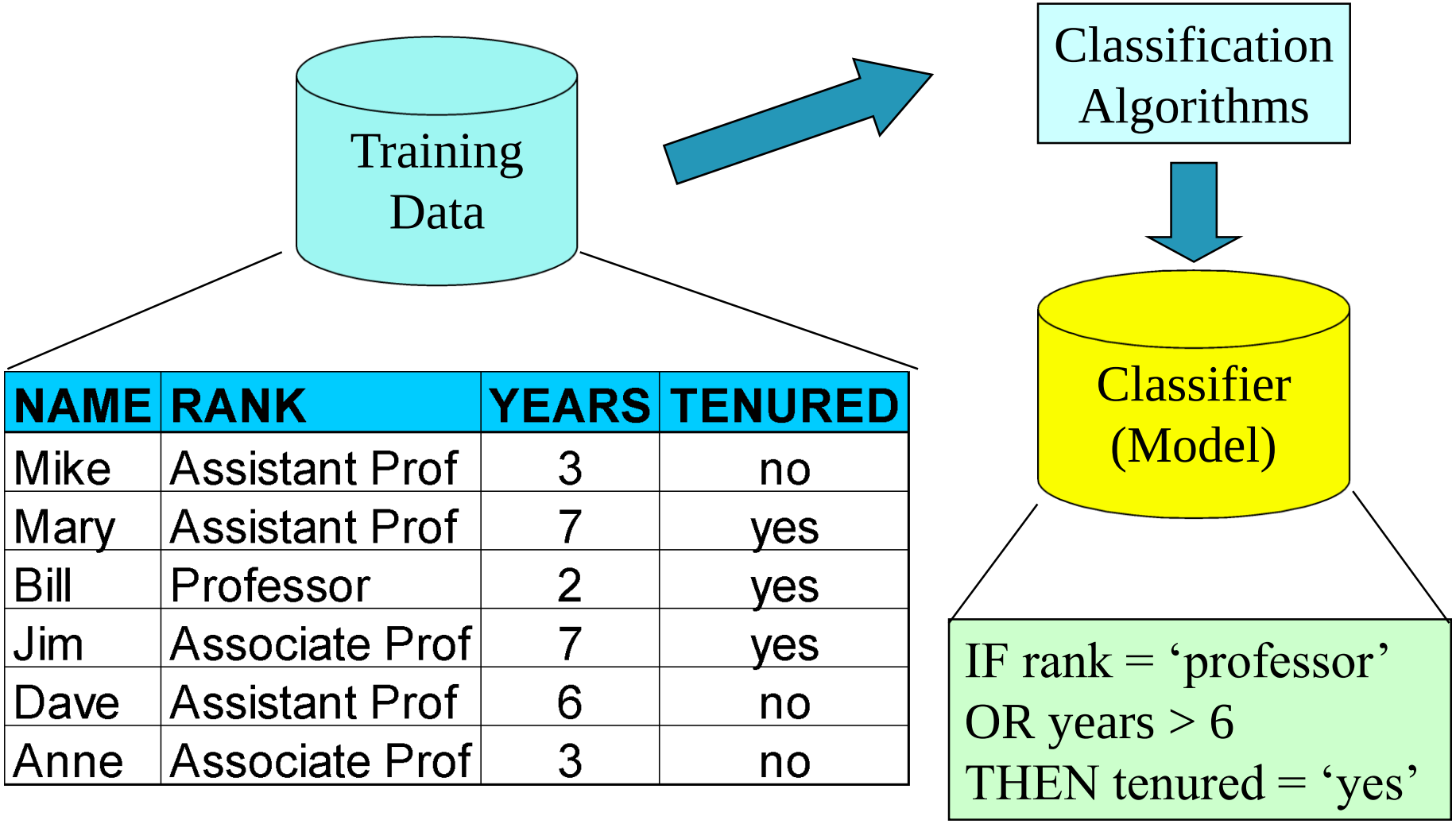
Classification Vs Prediction

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit approval
  - Target marketing
  - Medical diagnosis
  - Fraud detection

Classification Process

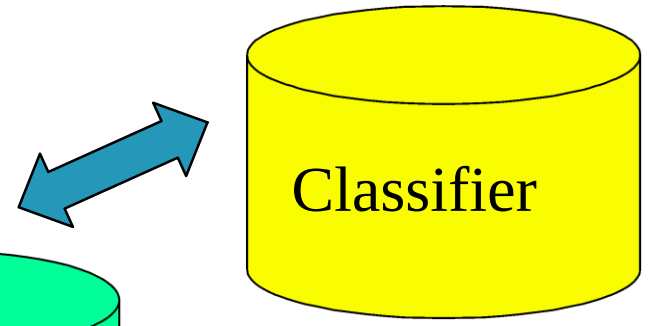
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

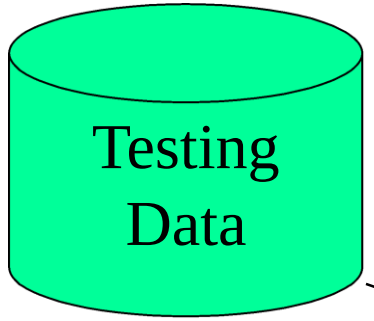
Step 1: Model Construction

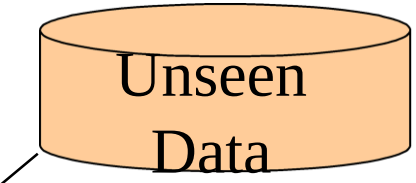


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Step 2: Model Usage







NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

(Jeff, Professor, 4)

Tenured?

Bayesian Classification

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Supervised learning : Classification

- Naive Bayes models are a group of extremely fast and simple classification algorithms
- often suitable for very high-dimensional datasets.
- Because they are so fast and have so few tunable parameters, they end up being very useful as a quick-and dirty baseline for a classification problem.
- This section will focus on an intuitive explanation of how naive Bayes classifiers work, followed by a couple examples of them in action on some datasets.

Contents

- Introduction Bayes' Theorm
- Naive Bayes Classifier
- Gaussian Naive Bayes
- Multinomial Naive Bayes

Bayesian Classification

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers

Example

age	income	student	credit rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

>40

Probability

- *Probability:* How likely something is to happen

- Probability of an event happening =

$$\frac{\text{Number of ways it can happen}}{\text{Total number of outcomes}}$$

Bayesian Theorem Basics

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$$
- Classification is to determine P(H|**X**), the probability that the hypothesis holds given the observed data sample **X**
- P(H) (*prior probability*), the initial probability
  - E.g., **X** will buy computer, regardless of age, income, …
- P(**X**): probability that sample data is observed
- P(**X**|H) (*posteriori probability*), the probability of observing the sample **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayesian Theorm

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes theorem

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

  posteriori = likelihood x prior/evidence

- ***Predicts X belongs to C_2 iff the probability $P(C_i|X)$ is the highest among all the $P(C_k|X)$ for all the k classes***
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Naiive Bayesian

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Training Data

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age =31..40,
Income = low,
Student = yes
Credit_rating = excellent)

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Test for X = (age <= 30 , income = medium, student = yes, credit_rating = fair)

- $P(C_i)$: **P(buys_computer = "yes") = 9/14 = 0.643** **P(buys_computer = "no") = 5/14= 0.357**
- Compute $P(X|C_i)$ for each class

P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222

P(age = "<=30" | buys_computer = "no") = 3/5 = 0.6

P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- **$P(X|C_i)$:** P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

  P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

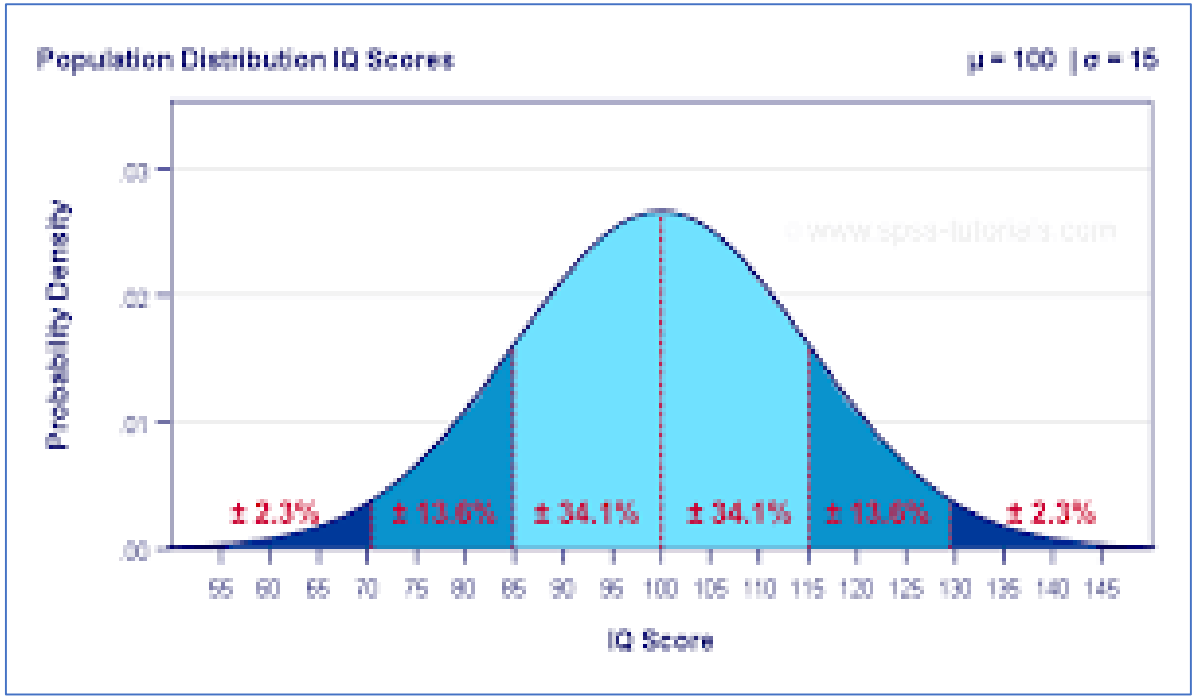
0.028>0.007 ..
Therefore, X belongs to class ("buys_computer = yes")

GaussianNB

- GaussianNB implements the Gaussian Naive Bayes algorithm for classification.
- The likelihood of the features is assumed to be Gaussian type

Normal distribution, also known as the **Gaussian distribution**, is a probability **distribution** that is symmetric about the mean, showing that **data** near the mean are more frequent in occurrence than **data** far from the mean. In graph form, **normal distribution** will appear as a bell curve

For **example**, heights, blood pressure, measurement error, and IQ scores follow the **normal distribution**. It is also known as the **Gaussian distribution** and the bell curve.



Multinomial naive Bayes

- Multinomial naive Bayes the features are assumed to be generated from a simple multinomial distribution.
- The multinomial distribution describes the probability of observing **counts among a number of categories.**
- So Multinomial naive Bayes is most appropriate for features that represent counts or count rates.
- One place where multinomial naive Bayes is often used is in text classification, where the features are related to word counts or frequencies within the documents to be classified.

Example

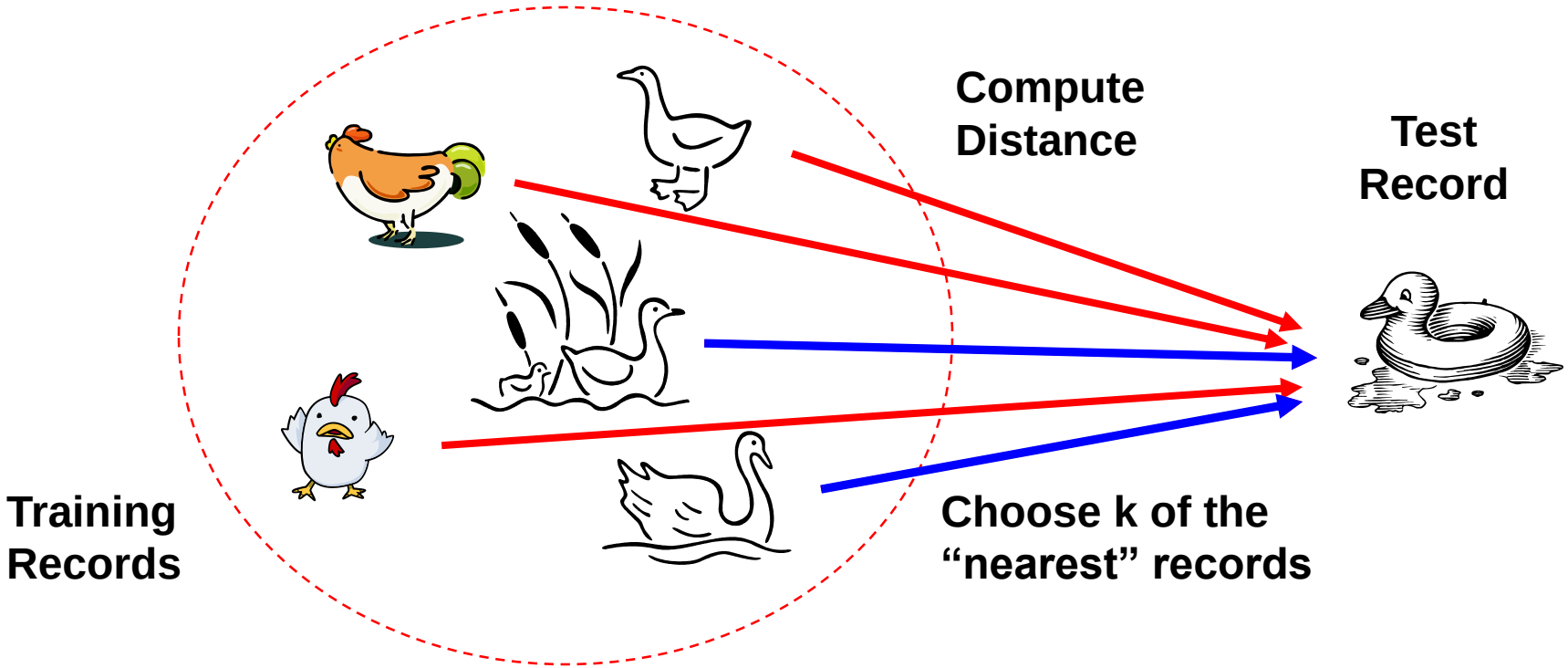
- Dataset Description: we will use the sparse word count features from the 20 Newsgroups corpus

from sklearn.datasets import fetch_20newsgroups

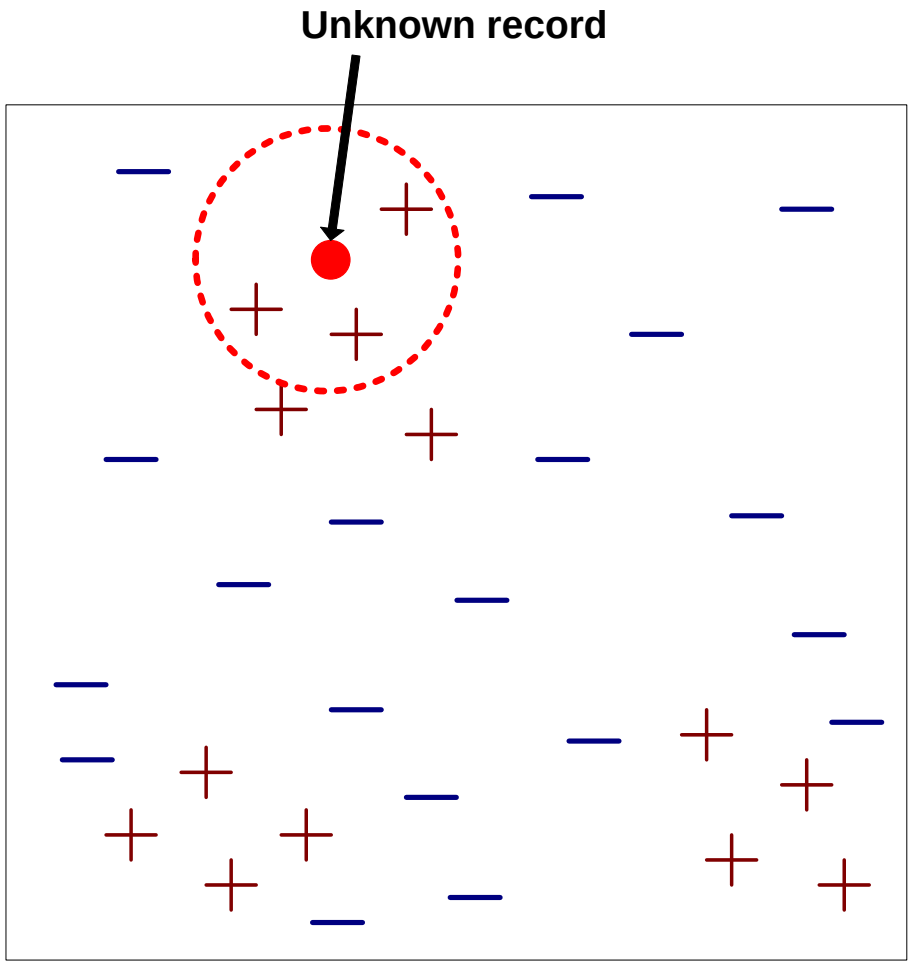
K Nearest neighbour

Nearest Neighbour Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



Nearest-Neighbor Classifiers



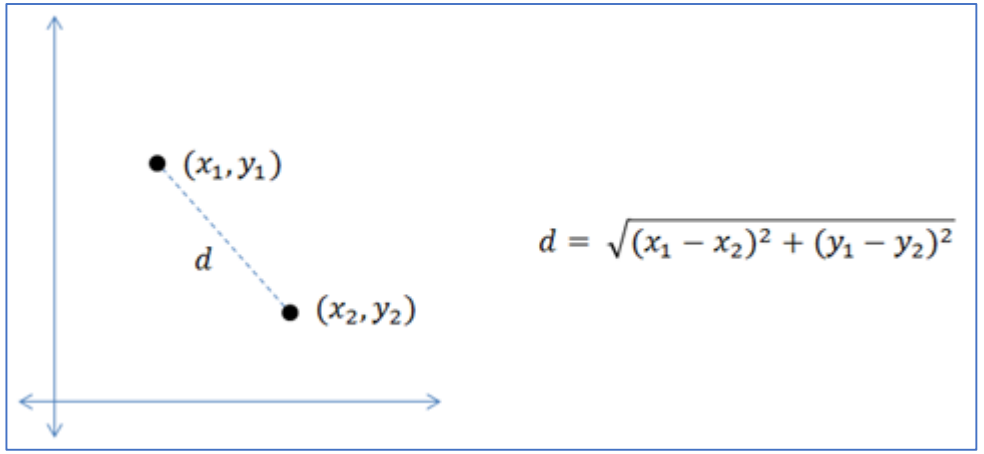
- Requires three things
  - The set of labeled records
  - Distance metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Nearest Neighbor Classification

- Compute Similarity between two points:
  - Example: Euclidean distance : minimum the distance , maximum the similarity

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_i (\boldsymbol{x_i} - \boldsymbol{y_i})^2}$$



- Determine the class from nearest neighbor list
  - Take the majority vote of class labels among the k-nearest neighbors

Exercise: Consider the following data, find the class label for (6,6) with K=3

x	y	Class
2	4	N
4	6	N
4	4	P
4	2	N
6	4	N
6	2	P

1. Determine parameter K = number of nearest neighbors.

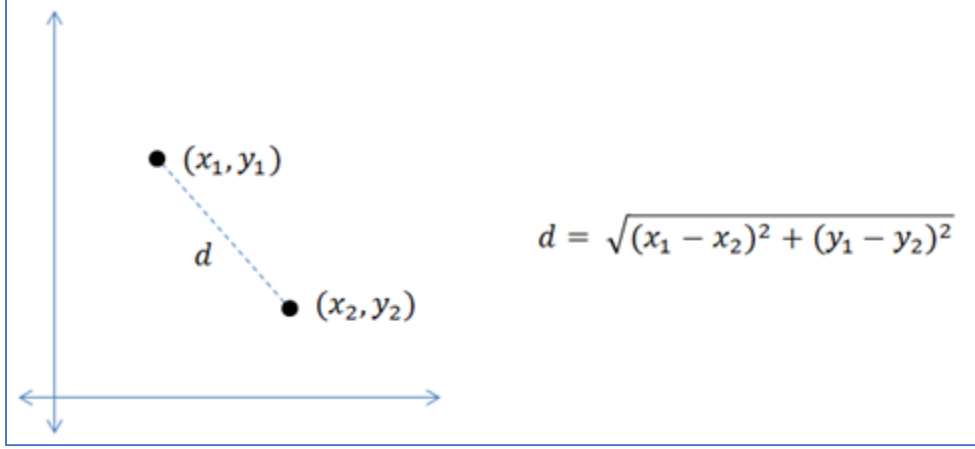
2. Calculate the distance between the query-instance and all the training samples.

3. Sort the distance and determine nearest neighbors based on the K-th minimum distance.

4. Gather the category of the nearest neighbors

5. Use a simple majority of the category of nearest neighbors as the prediction value of the query.

Example: Find the class label for (6,6) with K=3

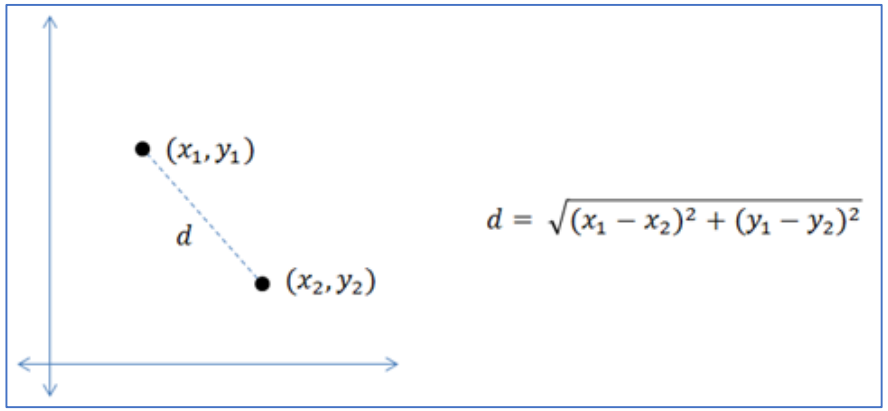


Features		Target	D	E	F	G	H	I			
X	Y	Class	X-Xi	Y-Yi	D*D	E*E	F+G	sqrt(H)	k=3	Class Lable of Neighbor	Mode
2	4	N	-4	-2	16	4	20	4.472136			
4	6	N	-2	0	4	0	4	2	Consider	N	N
4	4	P	-2	-2	4	4	8	2.828427	Consider	P	
4	2	N	-2	-4	4	16	20	4.472136			
6	4	N	0	-2	0	4	4	2	Consider	N	
6	2	P	0	-4	0	16	16	4			

Example KNN

- Find weight of [5.50,26]

- K=3
- K=4
- K=5



Height	Age	Weight	
5	45	77	
5.11	26	47	
5.6	30	55	
5.9	34	59	
4.8	40	72	
5.8	36	60	
5.3	19	40	
5.8	28	60	
5.5	23	45	
5.6	32	58	