



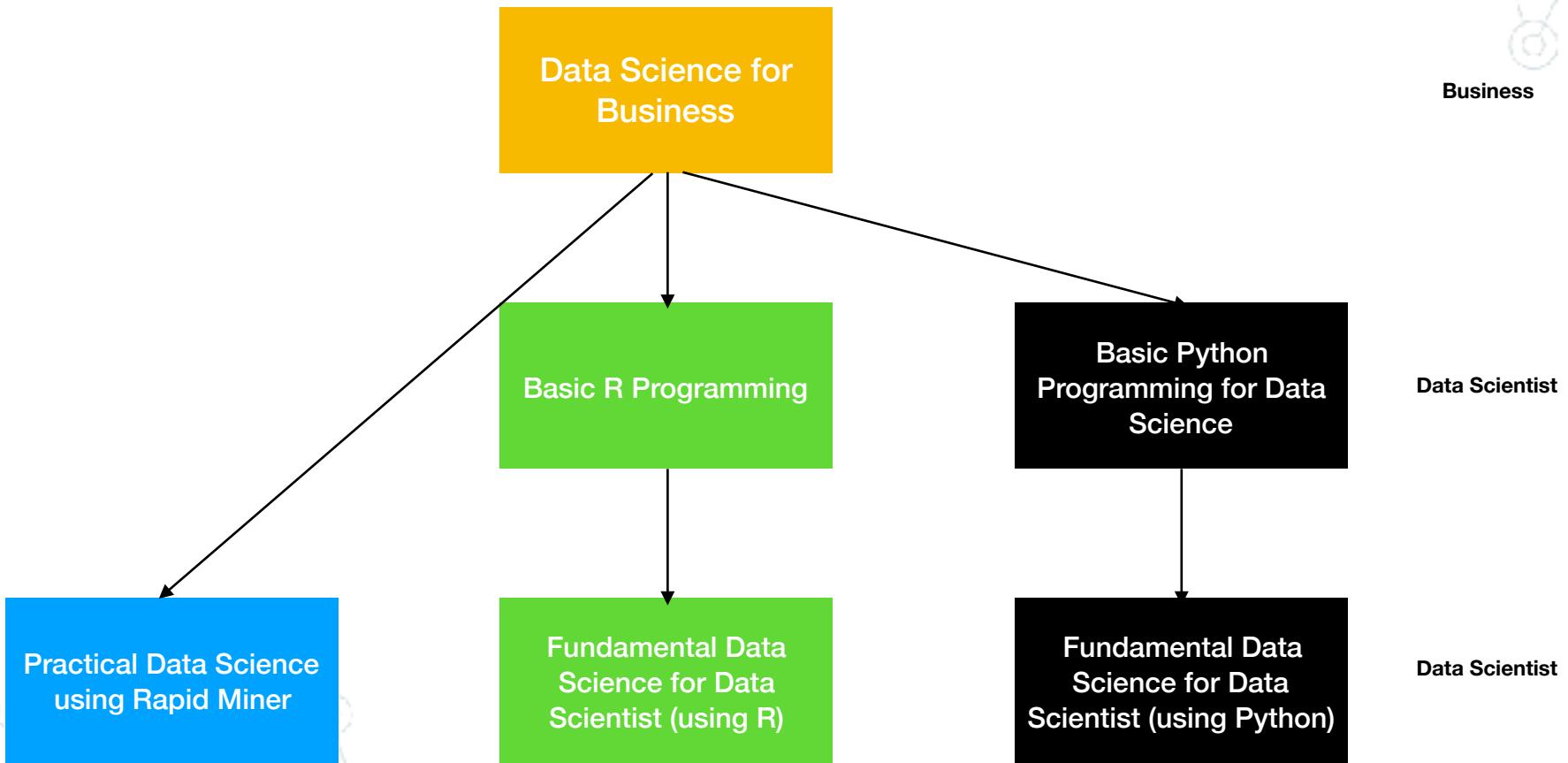
A close-up photograph of a person's hands interacting with a white computer mouse on a dark surface. The hands are shown from a top-down perspective, with fingers gripping the mouse. The background is blurred, focusing attention on the hands and the mouse.

**Fundamental Data
Science for Data
Scientist using
Python**

Veerasak
Kritsanapraphan,
Software Park Thailand



Data Science Series



About myself?

- Graduated from San Francisco State University in Master of Science in Computer Information System, 1997
- PhD. Candidate at Chulalongkorn University, research focus on Data Science, Big Data, Mobile Computing and Internet of Thing (IOT)
- Assistant Chief Information Officer at Bangkok Hospital Group
- Instructor for Software Park in Data Science, Requirement Discovery and Practical Enterprise Integration
- Instructor for Chulalongkorn University in R, Python for Data Mining

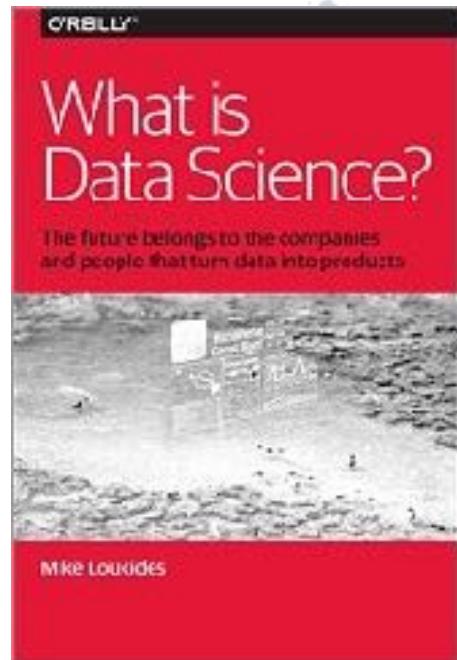
Introduce yourself?

- What is your name/position/role in your organization?
- Tell you a bit about yourself?
- What is your passion?
- What is your expectation for this class?



What is Data Science?

- Data Science aims to derive **knowledge** from **big data, efficiently and intelligently**
- Data Science encompasses the **set of activities, tools, and methods** that enable **data-driven activities** in science, business, medicine, and government



<http://www.oreilly.com/data/free/what-is-data-science.csp>

Google snaps up object recognition startup DNNr

« Search needs a shake-up



Songbirds use grammar rules »

Google has acquired DNNr, a Toronto, who
by [Josh Lowensohn](#)



Google has acquired research company

Topic: Cloud

Follow us:

Microsoft acquires legal-focused machine-learning vendor Equivio

Summary: Microsoft has purchased Equivio, maker of a machine-learning platform for the legal industry, for an undisclosed amount.



By [Mary Jo Foley](#) for All About Microsoft | January 20, 2015 -- 16:24 GMT (08:24 EST)

[Follow @maryjofoley](#)

[RT @MJoFoley](#)

[Get the ZDNet Cloud newsletter now](#)

Microsoft has purchased Equivio, an eDiscovery/compliance vendor with a specialization in text analysis, for an undisclosed amount.

Microsoft officials announced the acquisition of the Israeli company -- its first acquisition of 2015 using more of its offshore cash -- on January 20.

Update: The Wall Street Journal reported back in October last year that Microsoft planned to buy Equivio for \$200 million.

Update No. 2: A Microsoft spokesperson said the \$200 million estimate was inflated and incorrect, but declined to provide a different figure.



FOUNDED	2011
OWNER/CEO	DeepMind is a cutting-edge artificial intelligence company. We combine the best techniques from machine learning and systems neuroscience to build powerful general-purpose learning algorithms.
DESCRIPTION	Founded by Demis Hassabis, Shane Legg and Mustafa Suleyman, the company is based in London and supported by some of the most iconic technology entrepreneurs and investors of the past decade. Our first commercial...

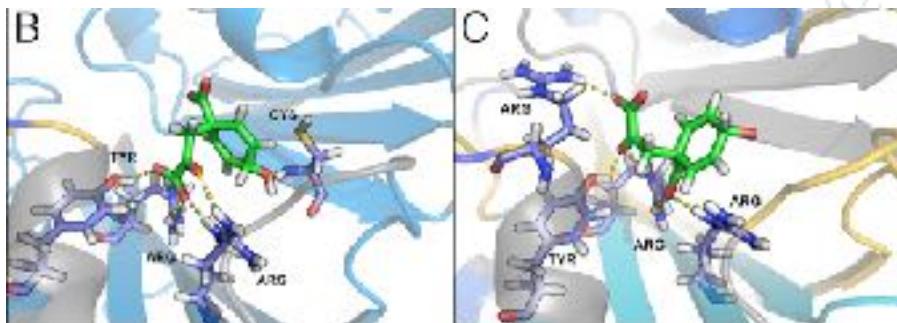
Examples of Data Science

Text Recognition



[http://commons.wikimedia.org/wiki/
File:American_book_company_1916_letter_envelope-2.JPG#filelinks](http://commons.wikimedia.org/wiki/File:American_book_company_1916_letter_envelope-2.JPG#filelinks)
[public domain]

Biology



<https://flic.kr/p/5BLW6G> [CC BY 2.0]

Spam Filtering

Examples of Data Science

Self-driving car



By Steve Jurvetson [CC BY 2.0]

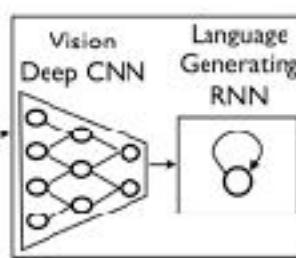
Recommendation systems



[http://commons.wikimedia.org/wiki/
File:Netflix_logo.svg](http://commons.wikimedia.org/wiki/File:Netflix_logo.svg) [public domain]

Photo search

and many, many more



A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

<http://googleresearch.blogspot.com/2014/11/a-picture-is-worth-thousand-coherent.html>

Example: Netflix Challenge

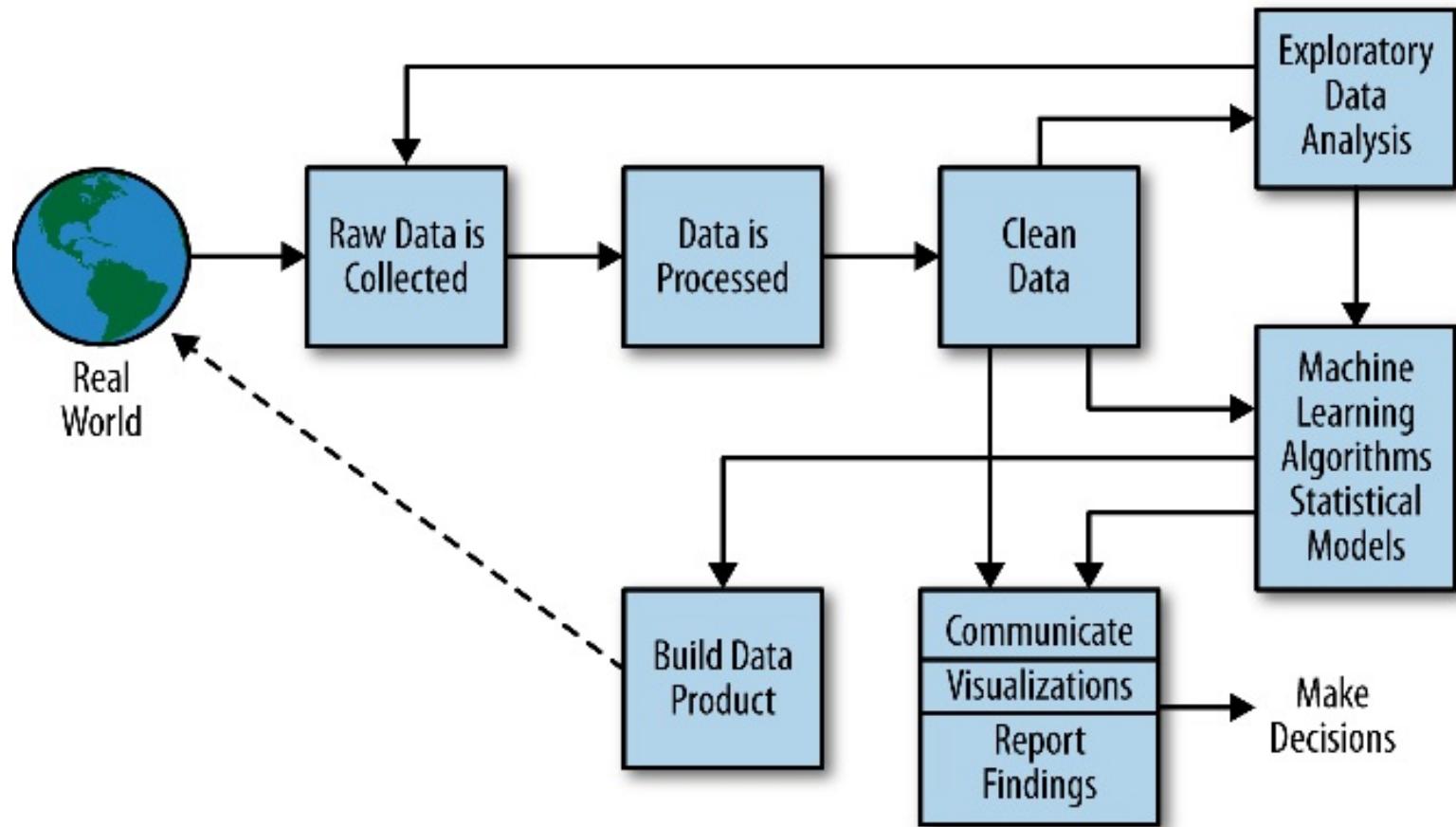
- Goal: Predict how a viewer will rate a movie
- 10% improvement = 1 million dollars



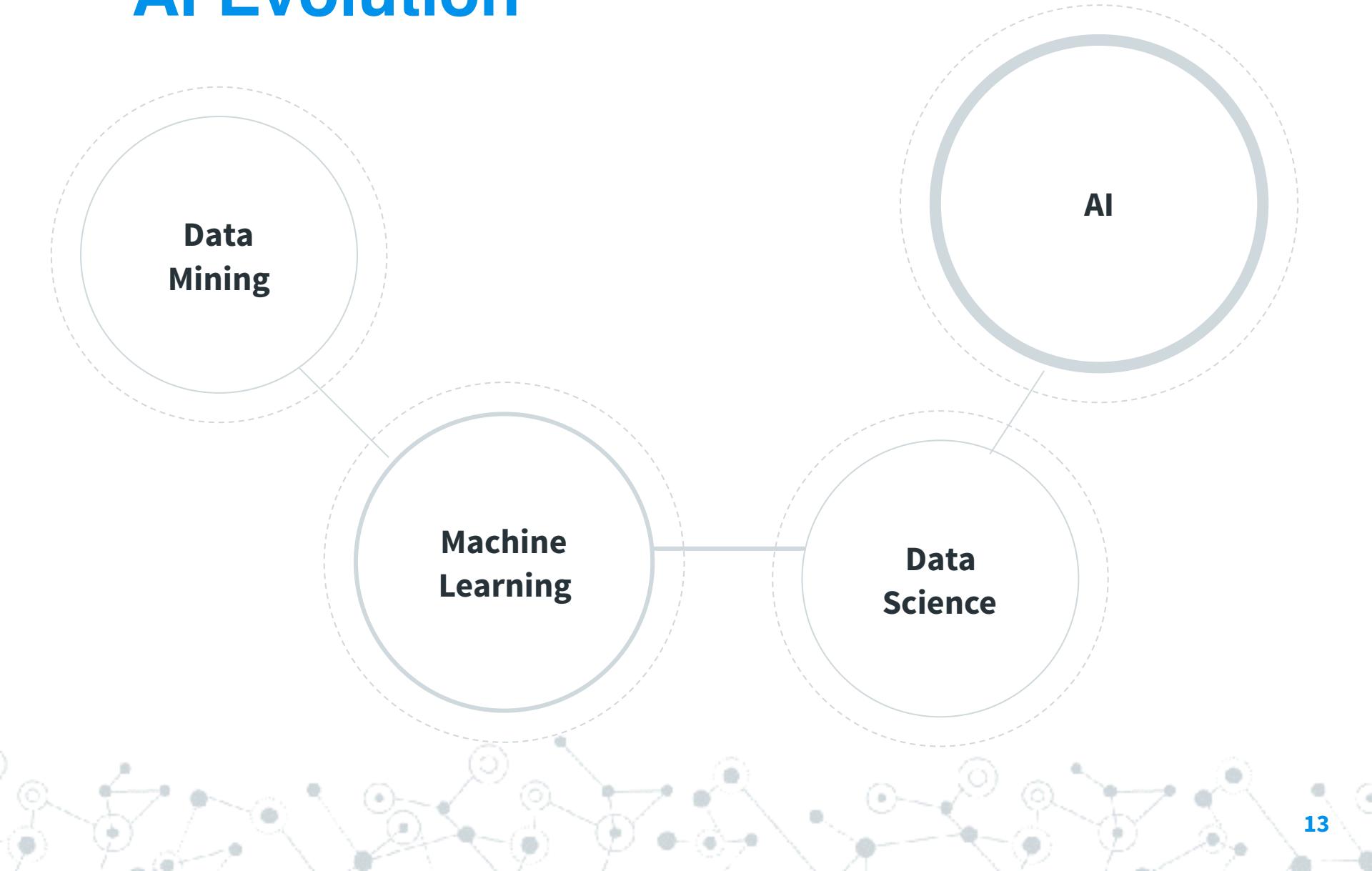
Concept

Fundamental Data Science for Data Scientist
using Python

Data Science Life Cycle

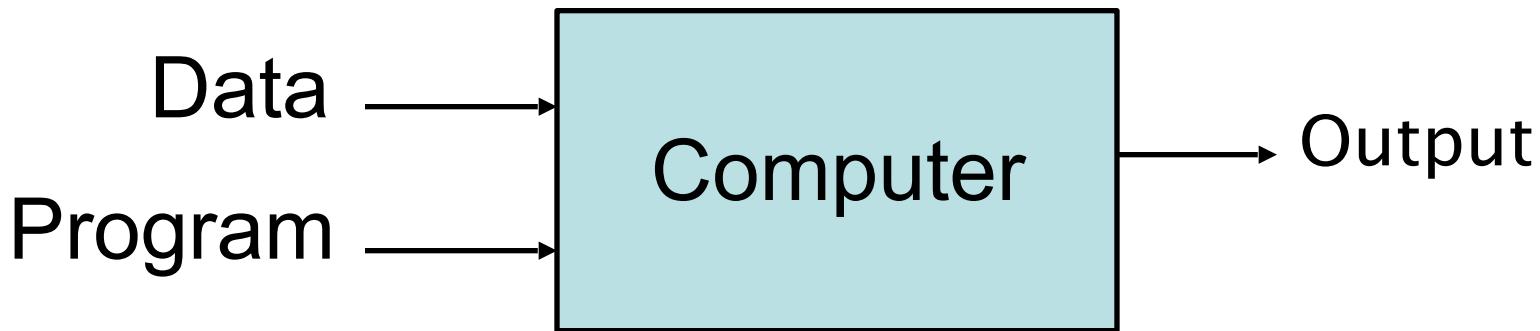


AI Evolution

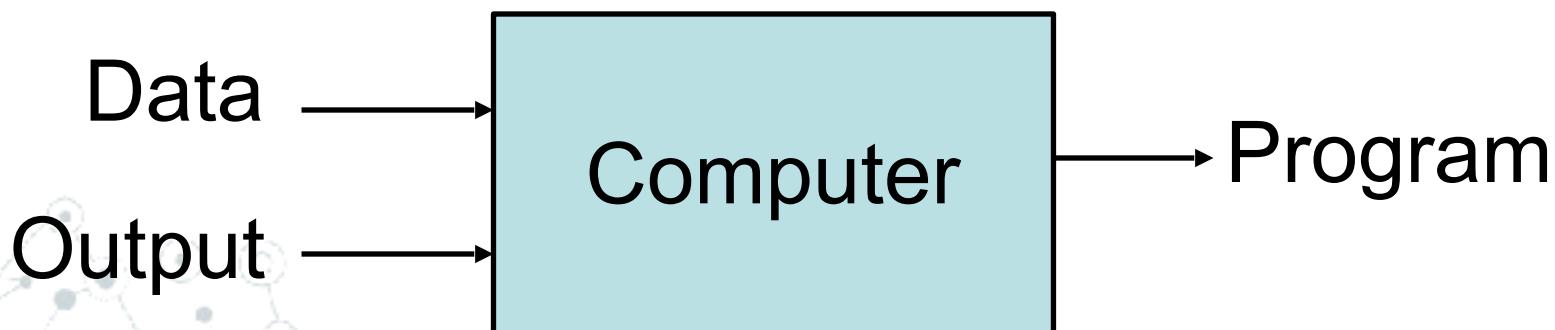


What is Machine Learning?

◎ Traditional Programming

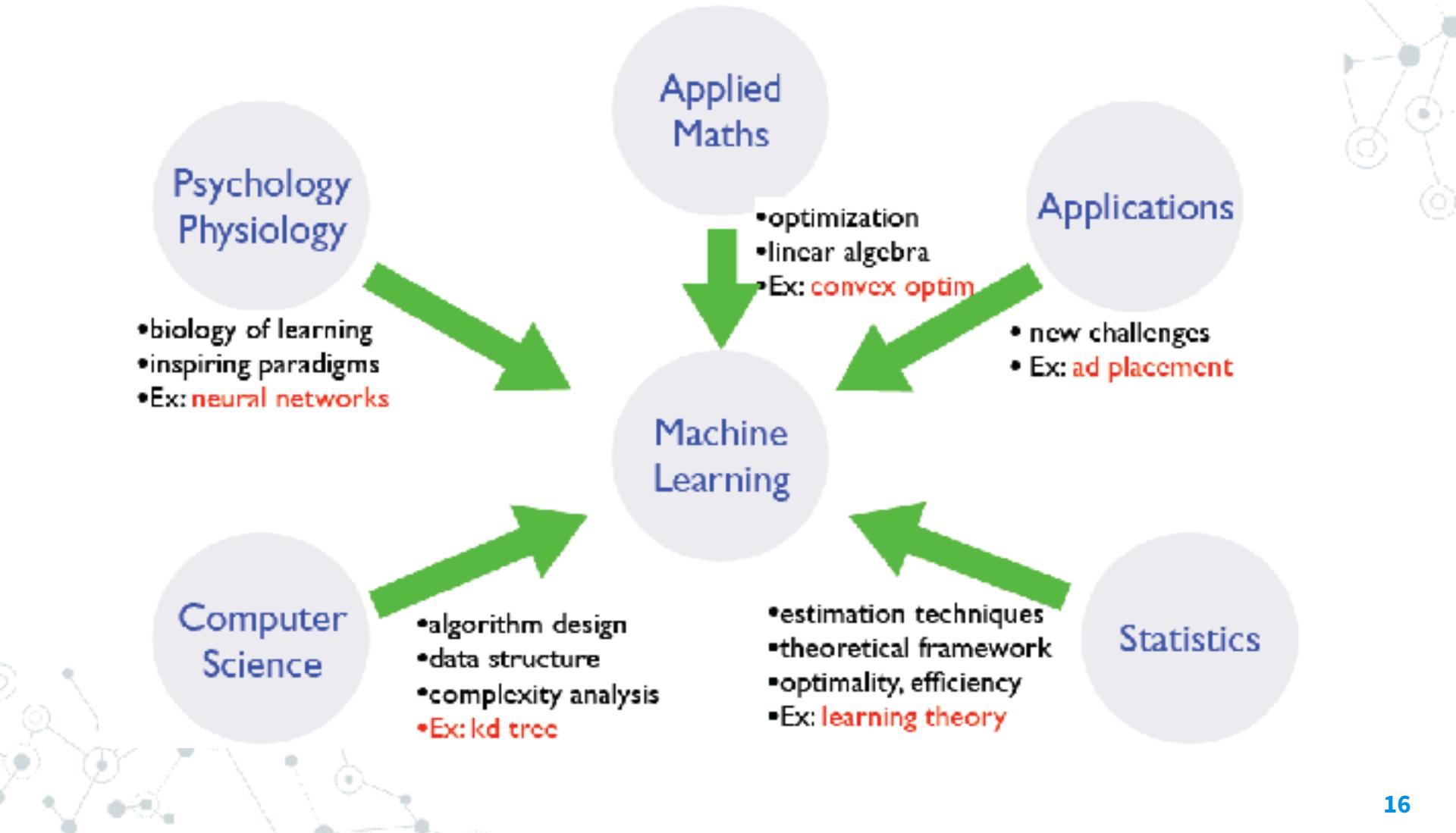


◎ Machine Learning





Where does ML fit in?



Why Study Machine Learning?

The Time is Ripe

◎ Data

- Large amounts of on-line data available.

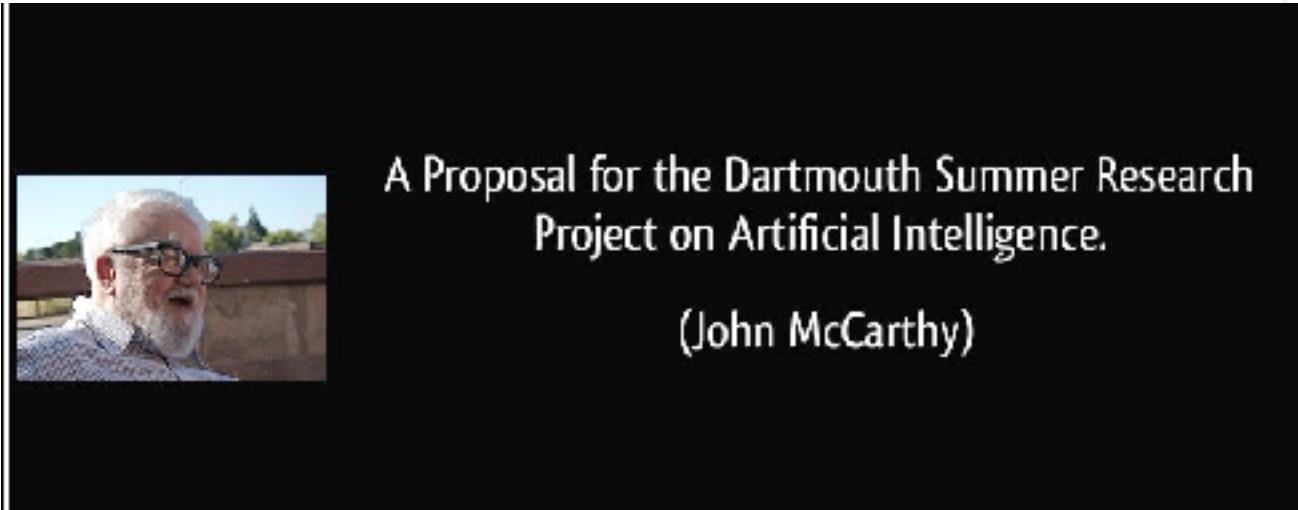
◎ Computing

- Large amounts of computational resources available.

◎ Algorithms

- Many basic effective and efficient algorithms available.
 - Finally!

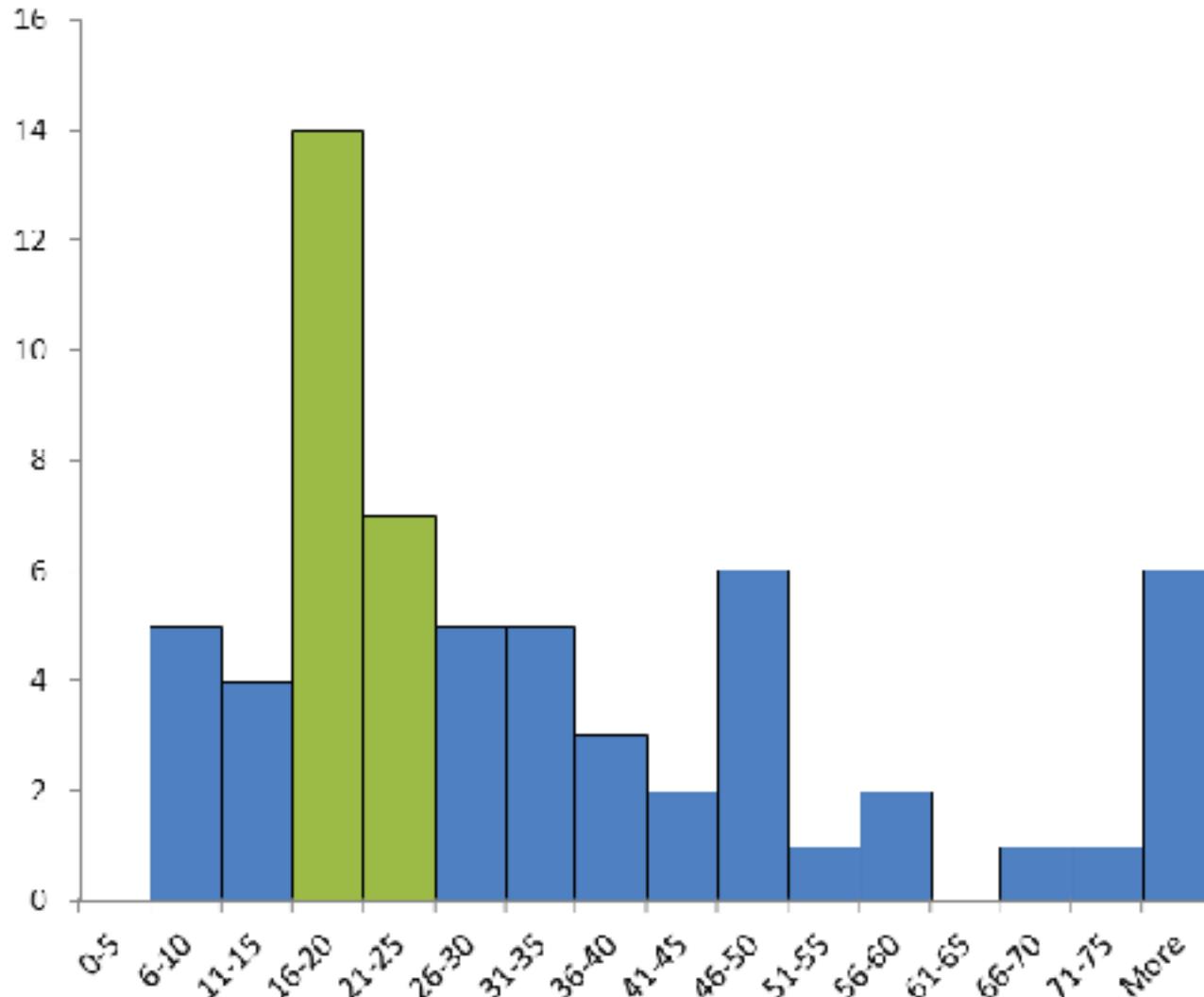
A Brief History of AI



A Brief History of AI

- ◎ “We propose that a **2 month, 10 man study** of artificial intelligence be carried out during the summer of **1956** at Dartmouth College in Hanover, New Hampshire.”
- ◎ The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.
- ◎ An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.
- ◎ We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.”

AI Predictions: Experts



We've come a long way...

◎ What is Jeopardy? Watson

- <https://youtu.be/q01i7-Qx00k?t=72>
- https://youtu.be/_429UlzN1JM?t=36



We've come a long way...

◎ Alpha Go





Hair Salon

Watch Jordan Peele use AI to make Barack Obama deliver a PSA about fake news

Machine Learning

- ◎ Tens of thousands of machine learning algorithms
 - Hundreds new every year
- ◎ Decades of ML research oversimplified:
 - All of Machine Learning:
 - Learn a mapping from input to output $f: X \rightarrow Y$
 - X : emails, Y : {spam, notspam}

Machine Learning

◎ Input: x (images, text, emails...)

◎ Output: y (spam or non-spam...)

◎ (Unknown) Target Function

- $f: X \rightarrow Y$ (the “true” mapping / reality)

◎ Data

- $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$

◎ Model / Hypothesis Class

- $g: X \rightarrow Y$
- $y = g(x) = \text{sign}(w^T x)$

Machine Learning

- ◎ Every machine learning algorithm has three components:
 - Representation / Model Class
 - Evaluation / Objective Function
 - Optimization

Representation / Model Class

- Decision trees
- Sets of rules / Logic programs
- Instances
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles
- Etc.

Evaluation / Objective Function

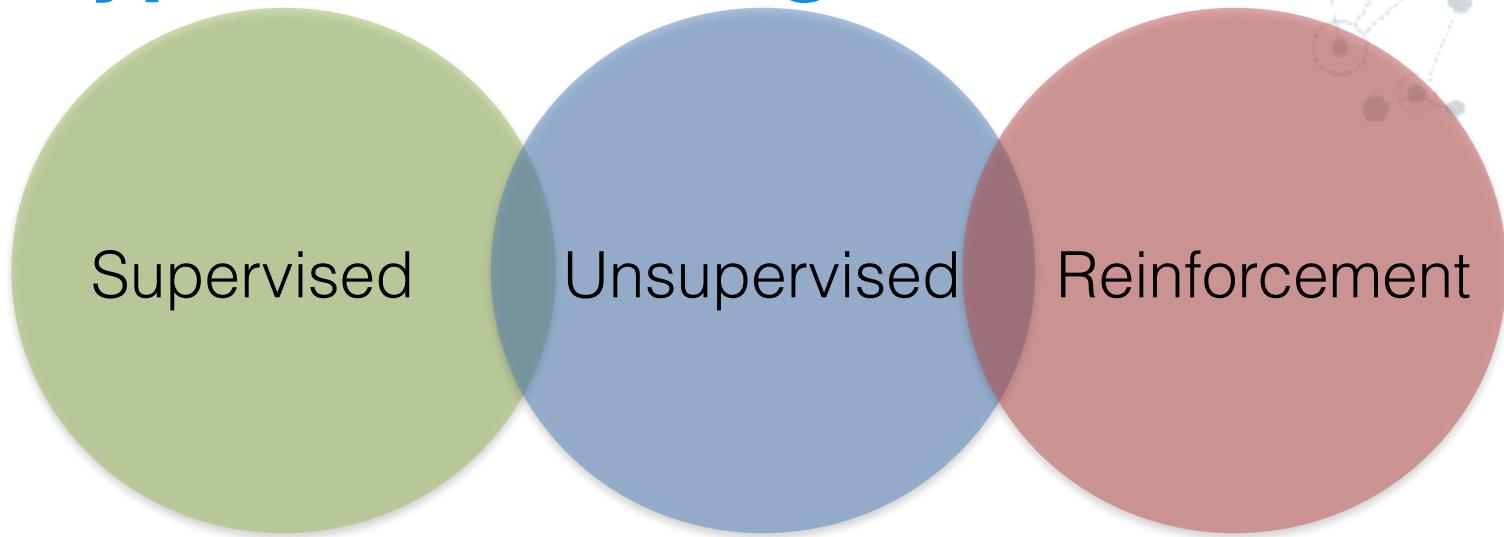
- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.

Optimization

- ◎ Discrete/Combinatorial optimization
 - greedy search
 - Graph algorithms (cuts, flows, etc)

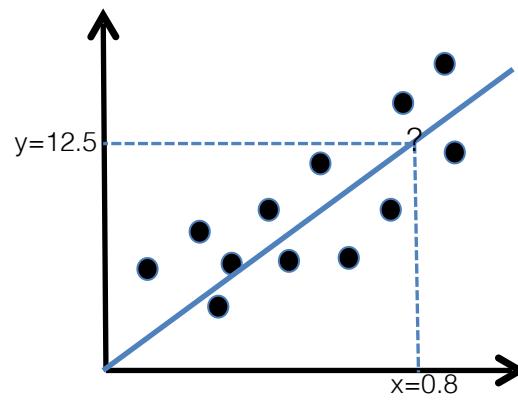
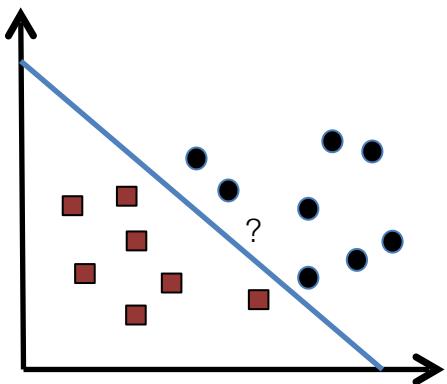
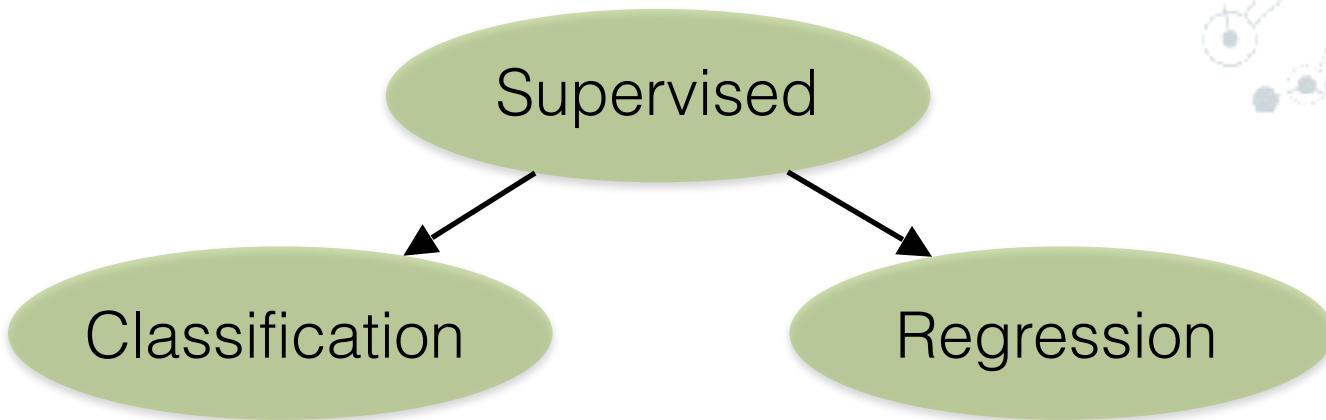
- ◎ Continuous optimization
 - Convex/Non-convex optimization
 - Linear programming

3 Types of Learning

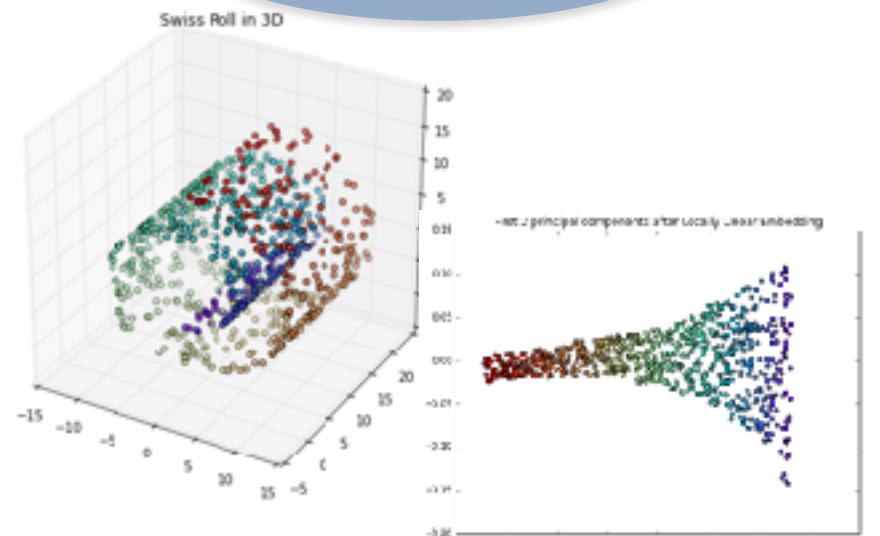
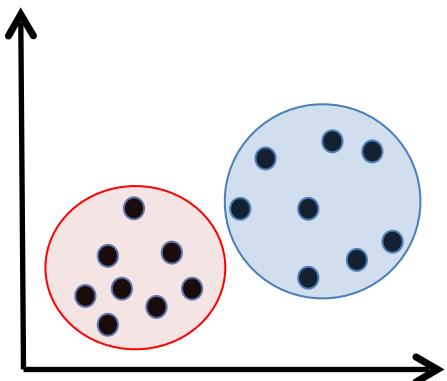
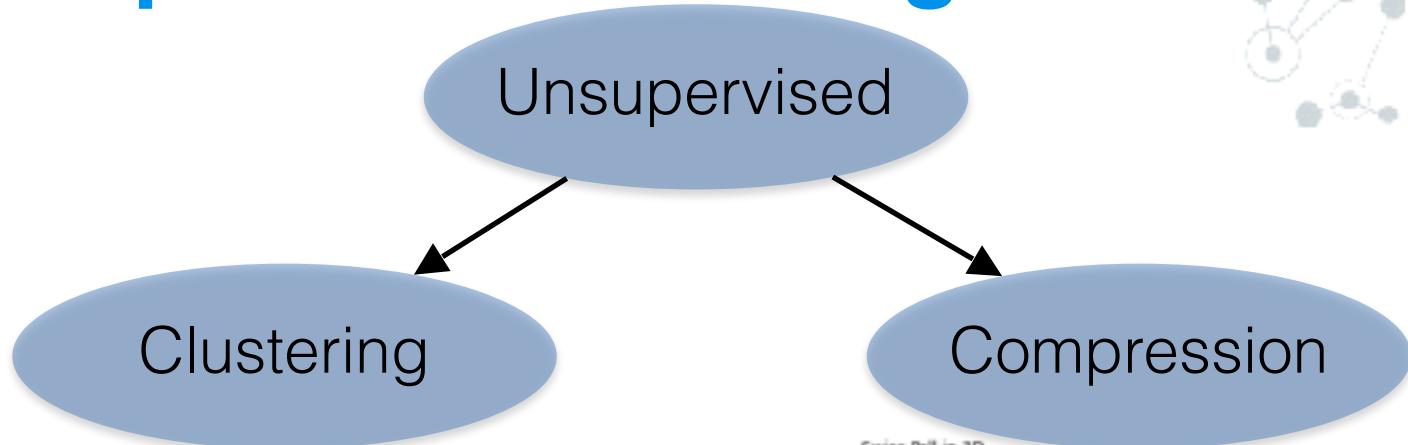


- Learning from labeled data
 - E.g., Spam classification
- Discover structure in unlabeled data
 - E.g., Document clustering
- Learning by “doing” with delayed reward
 - E.g., Chess computer

Supervised Learning



Unsupervised Learning

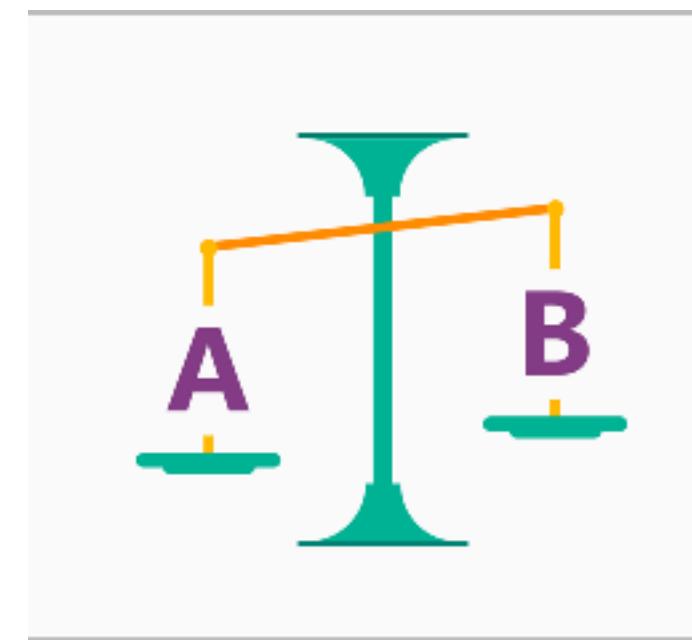


What questions can Machine Learning answer?

1. Is this **A** or **B**?
2. Is this **weird**?
3. How **much** or how **many**?
4. How is this **organized**?
5. What should I **do next**?

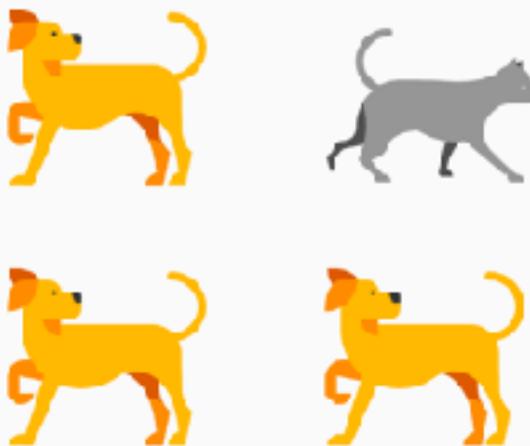
Is this A or B?

Classification algorithm



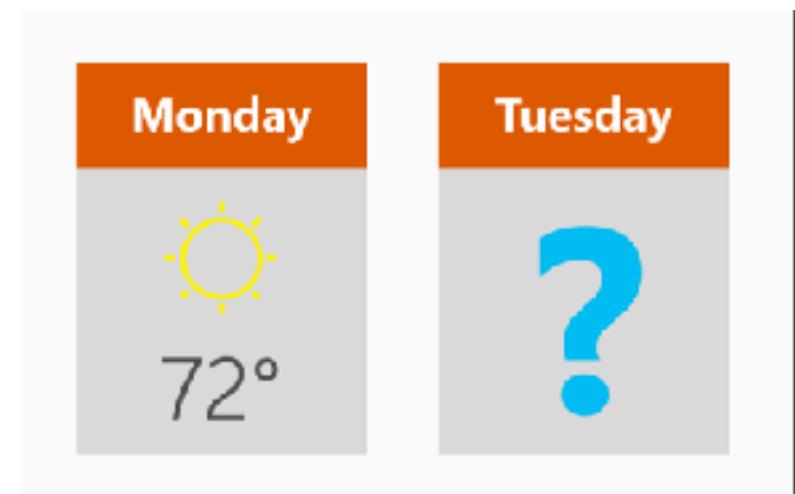
Is this weird?

Anomaly detection algorithm



How much and How many?

Regression algorithm



How is this organized?

- ◎ Clustering algorithm



What should I do next?

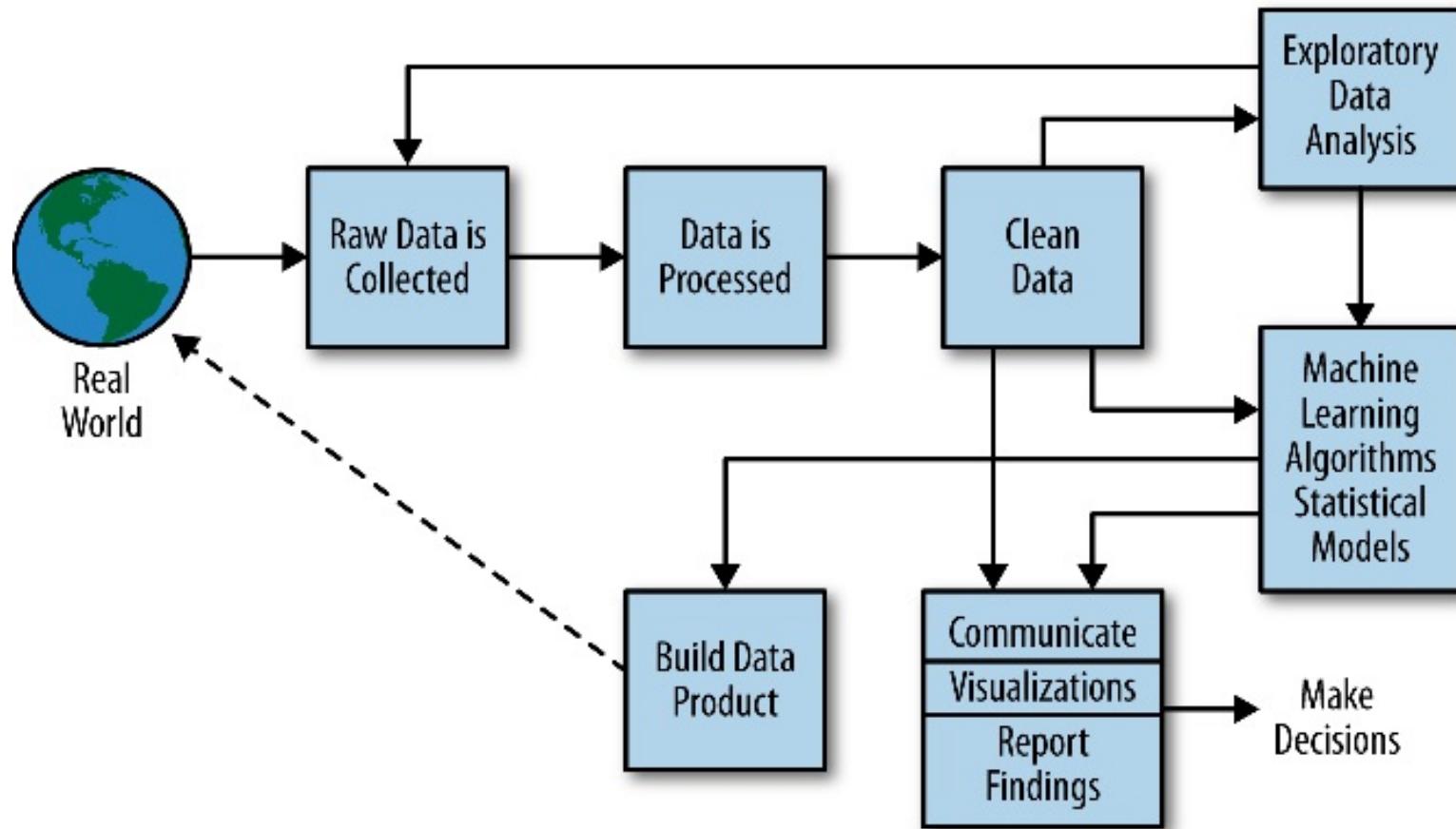
- ◎ Reinforcement Learning algorithm



Data Science Life Cycle

Fundamental Data Science for Data
Scientist using Python

Data Science Process



Data and Getting Data

Fundamental Data Science for Data
Scientist using Python

Data

- Before one can present and interpret information, there has to be a process of gathering and sorting data.
- Definition of data is "**facts or figures from which conclusions can be drawn**".
- Usually we collect many measurement on a person or object. Each measurement we call "**Variable**" and each person or object we call "**Observation**".

Types of Data

1. Categorical Data (Qualitative)

- I. Nominal
- II. Ordinal

2. Numerical Data

- 1. Interval/Ratio (Scale, Parametric, Quantitative)
 - 1. Discrete : Whole Number
 - 2. Continuous : Fraction
- 2. Ratio : True Zero
- 3. Interval : False Zero

Source Data and Jupyter Notebook Files

<https://github.com/vkrit/fundamental-data-science-python>



Getting Data

- ◎ Iris Data Set from UCI Machine Learning Repository
(<https://archive.ics.uci.edu/ml/datasets/Iris>)

- 1) Sepal Length
- 2) Sepal Width
- 3) Petal Length
- 4) Petal Width
- 5) Species



Nomenclature

Instances (samples, observations)

	sepal_length	sepal_width	petal_length	petal_width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
...
50	6.4	3.2	4.5	1.5	veriscolor
...
150	5.9	3.0	5.1	1.8	virginica

<https://archive.ics.uci.edu/ml/datasets/Iris>



Features (attributes, dimensions, variables)

Classes (targets)

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$\mathbf{y} = [y_1, y_2, y_3, \dots, y_N]$$

Iris setosa



Iris virginica



Iris versicolor



Getting Data

`scikit-learn` embeds a copy of the iris CSV file along with a helper function to load it into numpy arrays:

```
In [1]: from sklearn.datasets import load_iris  
iris = load_iris()
```

The resulting dataset is a `Bunch` object: you can see what's available using the method `keys()`:

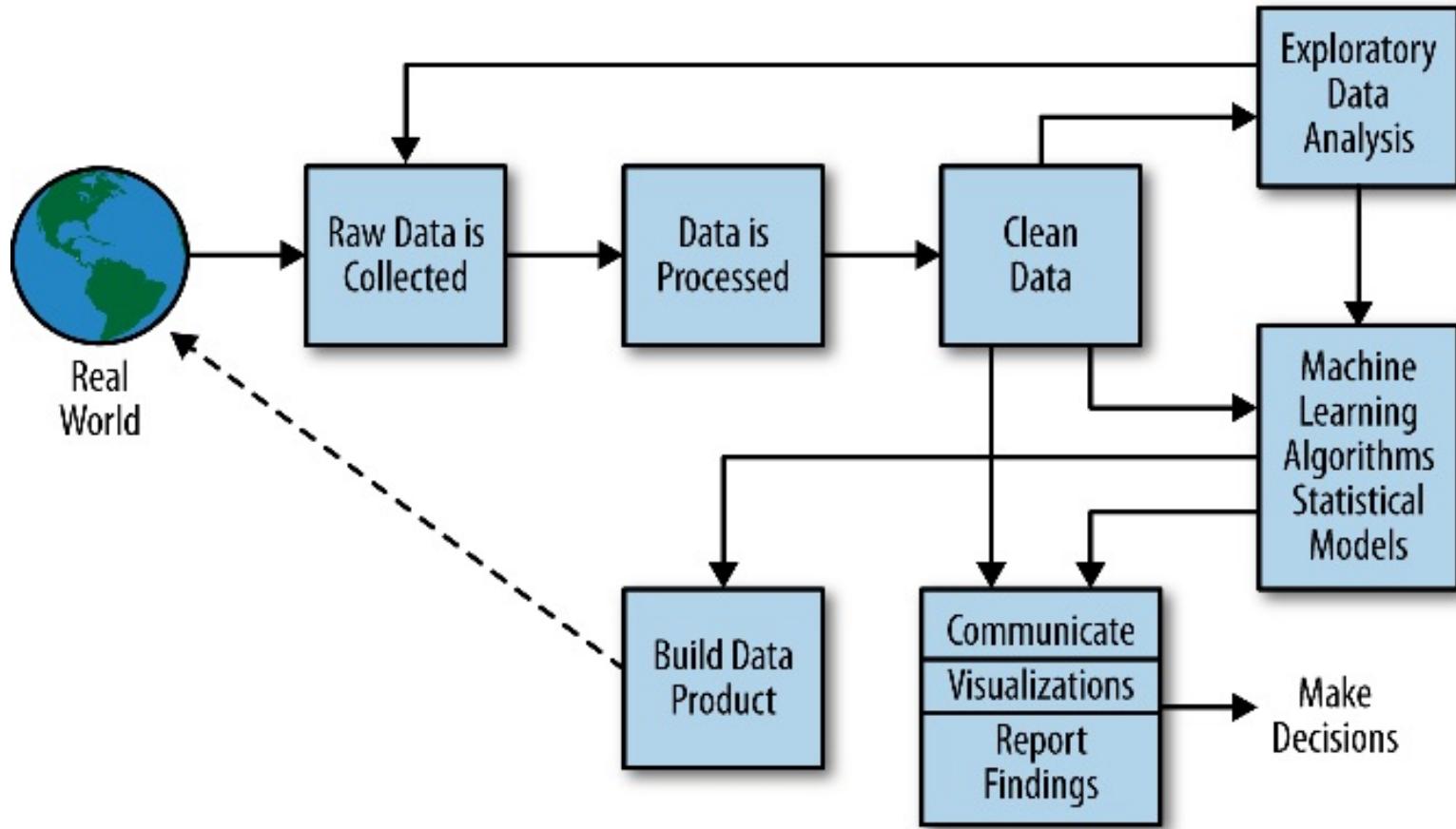
```
In [2]: iris.keys()
```

```
Out[2]: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

The features of each sample flower are stored in the `data` attribute of the dataset:

```
In [3]: n_samples, n_features = iris.data.shape  
print('Number of samples:', n_samples)  
print('Number of features:', n_features)  
# the sepal length, sepal width, petal length and petal width of the first sample (first flower)  
print(iris.data[0])
```

```
Number of samples: 150  
Number of features: 4  
[5.1 3.5 1.4 0.2]
```



Exploratory Data Analysis (EDA)

- ◎ The goal during EDA is to develop an understanding of your data.
- ◎ The easiest way to do this is to use questions as tools to guide your investigation.
- ◎ When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.
- ◎ **Two types of questions** will always be useful for making discoveries within your data.

What type of **variation** occurs within my variables?

What type of **covariation** occurs between my variables?

Variation

- ◎ The **tendency of the values of a variable to change from measurement to measurement.**
- ◎ If you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light. Categorical variables can also vary if you measure across different subjects or different times.
- ◎ **Every variable has its own pattern of variation, which can reveal interesting information.** The best way to understand that pattern is to visualize the distribution of the variable's values.

Visualizing distributions

- ◎ How you visualize the distribution of a variable will depend on whether the variable is categorical or continuous.
- ◎ A variable is **categorical** if it can only take one of a small set of values. In R, **categorical variables** are usually saved as factors or character vectors. To examine the distribution of a categorical variable, **use a bar chart**.
- ◎ A variable is **continuous** if it can take any of an infinite set of ordered values. Numbers and date-times are two examples of continuous variables. To examine the distribution of a **continuous** variable, **use a histogram**:

```
In [8]: %matplotlib inline
```

```
import matplotlib.pyplot as plt
```

```
In [9]: x_index = 3
```

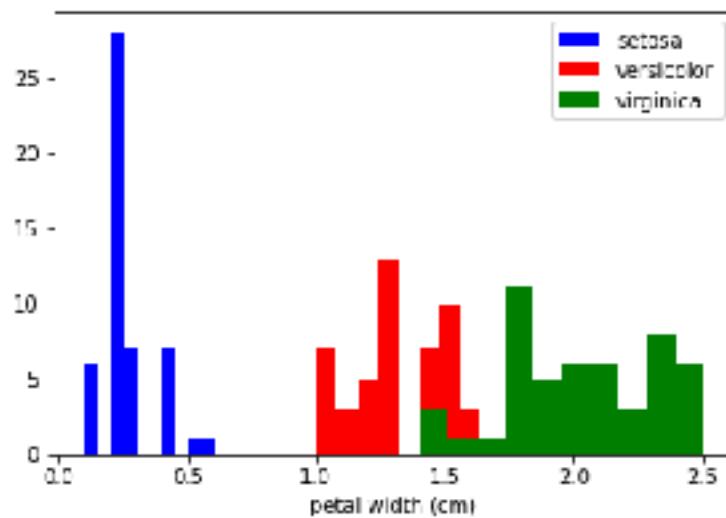
```
colors = ['blue', 'red', 'green']
```

```
for label, color in zip(range(len(iris.target_names)), colors):
    plt.hist(iris.data[iris.target==label, x_index],
             label=iris.target_names[label],
             color=color)
```

```
plt.xlabel(iris.feature_names[x_index])
```

```
plt.legend(loc='upper right')
```

```
plt.show()
```

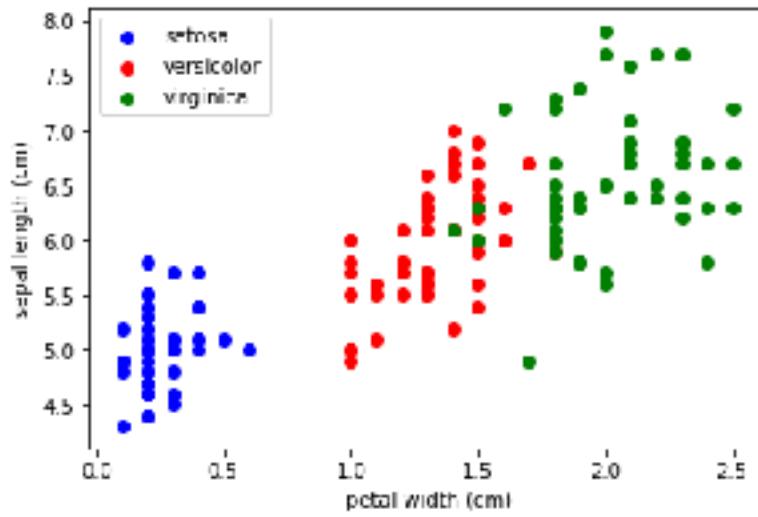


```
In [10]: x_index = 3
y_index = 0

colors = ['blue', 'red', 'green']

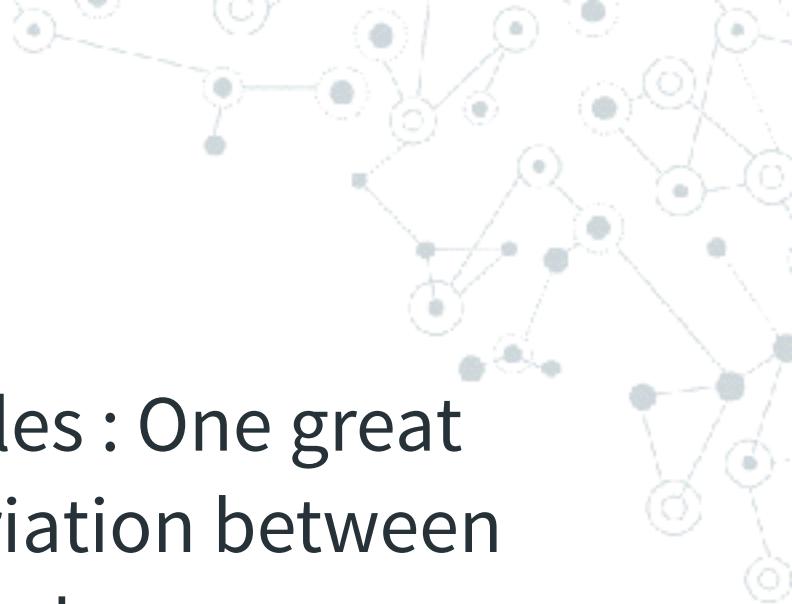
for label, color in zip(range(len(iris.target_names)), colors):
    plt.scatter(iris.data[iris.target==label], x_index,
                iris.data[iris.target==label], y_index,
                label=iris.target_names[label],
                c=color)

plt.xlabel(iris.feature_names[x_index])
plt.ylabel(iris.feature_names[y_index])
plt.legend(loc='upper left')
plt.show()
```



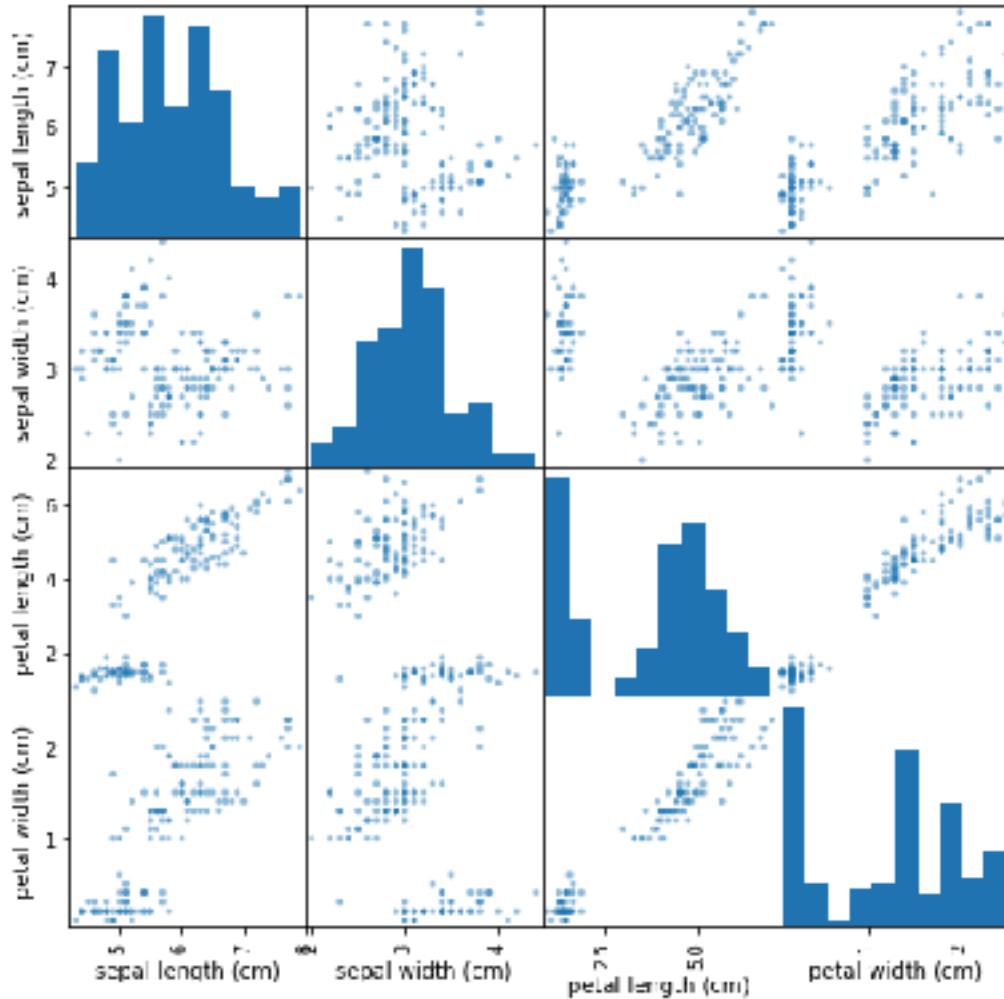
Covariation

- ◎ If variation describes the behavior within a variable, **covariation describes the behavior between variables. Covariation is the tendency for the values of two or more variables to vary together in a related way.** The best way to spot covariation is to visualize the relationship between two or more variables. How you do that should again depend on the type of variables involved.
- ◎ 1) **A categorical and continuous variable** : We want to explore the distribution of a continuous variable broken down by a categorical variable. Boxplot can help us

- 
- 
- ◎ 2) Two continuous variables : One great way to visualize the covariation between two continuous variables: draw a **scatterplot**.

In [12]:

```
import pandas as pd  
  
iris_dt = pd.DataFrame(iris.data, columns=iris.feature_names)  
pd.plotting.scatter_matrix(iris_dt, figsize=(8, 8));
```



Loading Digits Data

```
In [15]: from sklearn.datasets import load_digits  
digits = load_digits()
```

```
In [16]: digits.keys()
```

```
Out[16]: dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

```
In [17]: n_samples, n_features = digits.data.shape  
print((n_samples, n_features))  
  
(1797, 64)
```

```
In [18]: print(digits.data[0])  
print(digits.target)  
  
[ 0.  0.  5. 13.  9.  1.  0.  8.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.  
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  8.  0.  5.  8.  0.  
  0.  9.  8.  8.  0.  4. 11.  6.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.  
  0.  0.  0.  8.  6. 13. 10.  8.  0.  0.]  
[0 1 2 ... 8 9 8]
```

The target here is just the digit represented by the data. The data is an array of length 64... but what does this data mean?

There's a clue in the fact that we have two versions of the data array: `data` and `images`. Let's take a look at them:

```
In [19]: print(digits.data.shape)
print(digits.images.shape)

(1797, 64)
(1797, 8, 8)
```

We can see that they're related by a simple reshaping:

```
In [20]: import numpy as np
print(np.all(digits.images.reshape((1797, 64)) == digits.data))

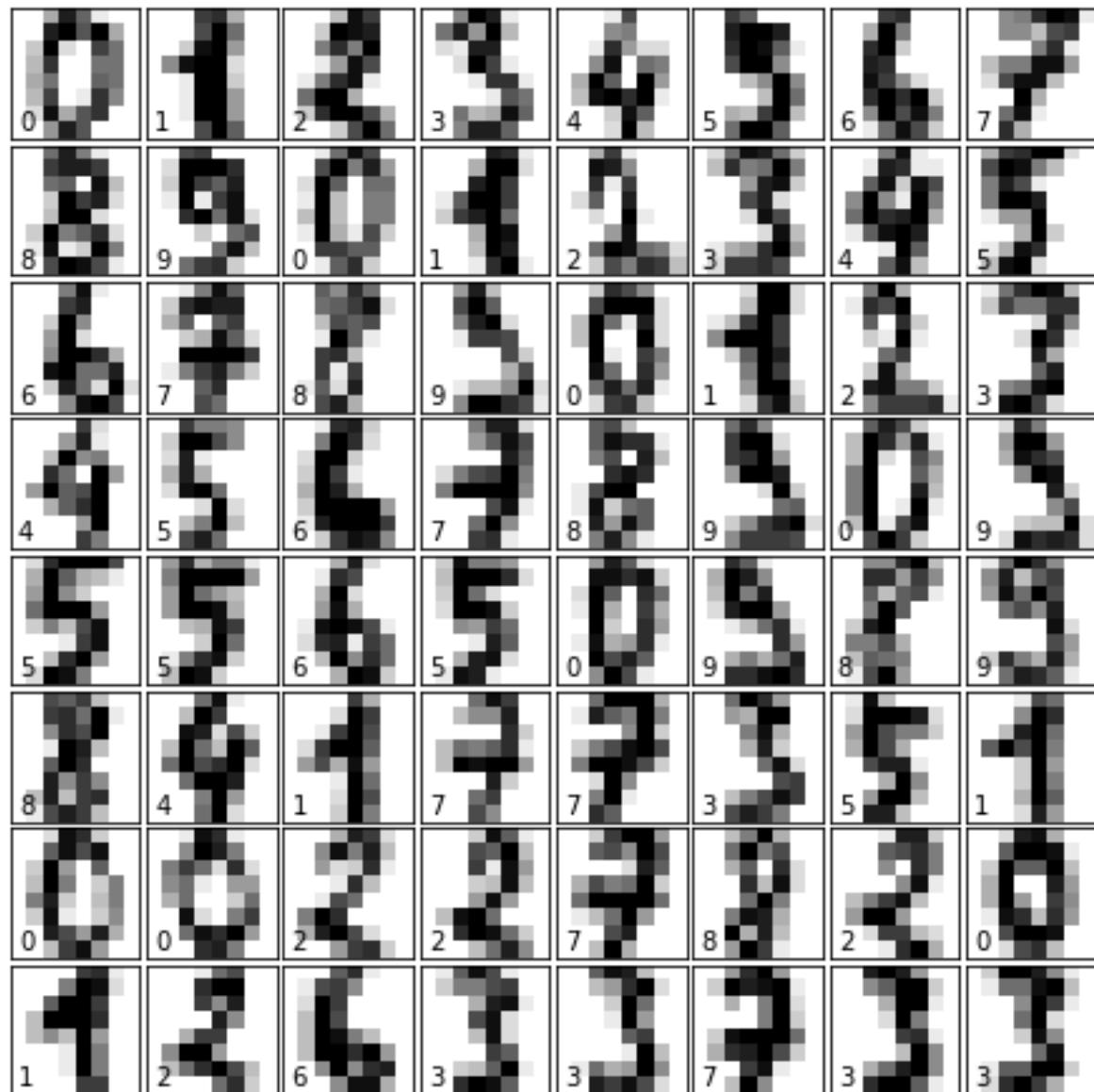
True
```

Let's visualize the data. It's little bit more involved than the simple scatter-plot we used above, but we can do it rather quickly.

```
In [21]: # set up the figure
fig = plt.figure(figsize=(6, 6)) # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

# plot the digits: each image is 8x8 pixels
for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks[])
    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')

    # label the image with the target value
    ax.text(0, 7, str(digits.target[i]))
```



Workshop 1 - Getting Data

- Choose data from dataset below.
 - Sales Win or Loss Dataset
 - HR Employee Attrition Dataset
 - Telco Customer Churn Dataset
 - Titanic Survival Dataset
 - Face Data (fetch_olivetti_faces from sklearn.datasets)
- Import data using `read_csv`
- Practice data visualization
- Do step on page 54-60

Data Preparation

Fundamental Data Science for Data
Scientist using Python

Preparing Training Data

- ◎ At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.
 - Data sampling
 - Data validation and handle missing data
 - Normalize numeric value into a uniform range
 - Data Standardization
 - Binarize Data
 - Reduce dimensionality

Data Sampling

training set					
1.1	2.2	3.4	5.6	1.0	
6.7	0.5	0.4	2.6	1.6	
2.4	9.3	7.3	6.4	2.8	
1.5	0.0	4.3	8.3	3.4	
0.5	3.5	8.1	3.6	4.6	
5.1	9.7	3.5	7.9	5.1	
3.7	7.8	2.6	3.2	6.3	
test set					
					1.6
					2.7
					4.4
					0.5
					0.2
					5.6
					6.7

```
In [4]: from sklearn.model_selection import train_test_split  
  
train_X, test_X, train_y, test_y = train_test_split(X, y,  
                                                train_size=0.7,  
                                                random_state=123)  
print("Labels for training and testing data")  
print(train_y)  
print(test_y)  
  
Labels for training and testing data  
[2 2 1 0 0 2 0 0 1 1 1 1 2 1 2 0 2 1 0 0 2 1 2 2 0 1 1 2 0 2 1 1 0 2 2 0 0  
 1 1 2 0 0 1 0 1 2 0 2 0 0 1 0 0 1 2 1 1 1 0 0 1 2 0 0 1 1 1 2 1 1 1 2 0 0  
 1 2 2 2 2 0 1 0 1 1 0 1 2 1 2 2 0 1 0 2 2 1 1 2 2 1 0 1 1 2 2]  
[1 2 2 1 0 2 1 0 0 1 2 0 1 2 2 2 0 0 1 0 0 2 0 2 0 0 0 2 2 0 2 2 0 0 1 1 2  
 0 0 1 1 0 2 2 2]
```

Stratified Split

```
In [5]: print('All:', np.bincount(y) / float(len(y)) * 100.0)
print('Training:', np.bincount(train_y) / float(len(train_y)) * 100.0)
print('Test:', np.bincount(test_y) / float(len(test_y)) * 100.0)
```

```
All: [33.33333333 33.33333333 33.33333333]
Training: [30.47619048 38.0952381 31.42857143]
Test: [40.          22.22222222 37.77777778]
```

```
In [6]: train_X, test_X, train_y, test_y = train_test_split(X, y,
                                                       train_size=0.5,
                                                       random_state=123,
                                                       stratify=y)

print('All:', np.bincount(y) / float(len(y)) * 100.0)
print('Training:', np.bincount(train_y) / float(len(train_y)) * 100.0)
print('Test:', np.bincount(test_y) / float(len(test_y)) * 100.0)
```

```
All: [33.33333333 33.33333333 33.33333333]
Training: [33.33333333 33.33333333 33.33333333]
Test: [33.33333333 33.33333333 33.33333333]
```

Handling Missing Data

- ◎ Remove entire records using Pandas dropna()
- ◎ Impute data using Means, Median, Zeros etc. using Pandas fillna()

Missing Values

Missing values are marked as NaN

```
In [ ]: # Read a dataset with missing values  
flights = pd.read_csv("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	1007.0	29.0	2251.0	NaN	UA	N31412	1229	EWR	SAN	NaN	2425	10.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3E+AA	751	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1095	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1269	EWR	RSW	NaN	1069	21.0	45.0
856	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	130	JFK	LAX	NaN	2475	NaN	NaN

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

Normalize numeric value

Normalization refers to rescaling real valued numeric attributes into the range 0 and 1.

```
In [14]: from sklearn import preprocessing  
  
# normalize the data attributes  
normalized_X = preprocessing.normalize(X)  
  
normalized_X
```

```
Out[14]: array([[0.80377277, 0.55160877, 0.22064351, 0.0315205 ],  
[0.82813287, 0.50702013, 0.23560939, 0.03380134],  
[0.80533308, 0.54831188, 0.2227517 , 0.03426949],  
[0.80003025, 0.53915082, 0.26067943, 0.03478392],  
[0.790965 , 0.5694948 , 0.2214702 , 0.0316386 ],  
[0.78417499, 0.5663486 , 0.2458699 , 0.05808704],  
[0.78010936, 0.57660257, 0.23742459, 0.0508767 ],  
[0.80218492, 0.54548574, 0.24065548, 0.0320874 ],  
[0.80642366, 0.5315065 , 0.25558935, 0.03665562],  
[0.81803119, 0.51752994, 0.25041771, 0.01669451],  
[0.80373519, 0.55070744, 0.22325977, 0.02976797],  
[0.786691 , 0.55745196, 0.26233033, 0.03279129],  
[0.82307218, 0.51442011, 0.24006272, 0.01714734],  
[0.8025126 , 0.55989251, 0.20529392, 0.01866308],  
[0.81120865, 0.55945424, 0.16783627, 0.02797271],  
[0.77361111, 0.59732787, 0.2036345 , 0.05430253],  
[0.79428944, 0.57365349, 0.19121783, 0.05883625],  
[0.80327412, 0.55126656, 0.22050662, 0.04725142],
```

Standardize Numeric Data

```
In [26]: from sklearn import preprocessing
```

```
# standardize the data attributes  
standardized_X = preprocessing.scale(X)  
  
standardized_X
```

```
Out[26]: array([[-9.00681170e-01,  1.03205722e+00, -1.34127240e+00,  
                 -1.31297573e+00],  
                [-1.14301591e+00, -1.24957661e-01, -1.34127240e+00,  
                 -1.31297573e+00],  
                [-1.38535265e+00,  3.37848329e-01, -1.39813811e+00,  
                 -1.31297573e+00],  
                [-1.50652052e+00,  1.06445364e-01, -1.28440570e+00,  
                 -1.31297573e+00],  
                [-1.02184904e+00,  1.26346019e+00, -1.34127240e+00,  
                 -1.31297573e+00],  
                [-5.37177559e-01,  1.95766969e+00, -1.17067529e+00,  
                 -1.05003079e+00],  
                [-1.50652052e+00,  8.00654259e-01, -1.34127240e+00,  
                 -1.18150376e+00],  
                [-1.02184904e+00,  8.00654259e-01, -1.28440570e+00,  
                 -1.31297573e+00],  
                [-1.74885526e+00, -3.56360566e-01, -1.34127240e+00,  
                 -1.31297573e+00],  
                [-1.14001591e+00, -1.03205722e+00, -1.34127240e+00])
```

Label Encoding

```
In [4]: from sklearn import preprocessing  
  
le = preprocessing.LabelEncoder()  
le.fit(["paris", "paris", "tokyo", "amsterdam"])  
list(le.classes_)
```

```
Out[4]: ['amsterdam', 'paris', 'tokyo']
```

```
In [6]: le.transform(["tokyo", "tokyo", "paris"])
```

```
Out[6]: array([2, 2, 1])
```

```
In [7]: list(le.inverse_transform([2, 2, 1]))
```

```
/Users/kris/anaconda3/anaconda/lib/python3.6/site-packages/sk  
nWarning: The truth value of an empty array is ambiguous. Ret  
in an error. Use 'array.size > 0' to check that an array is n  
if diff:
```

```
Out[7]: ['tokyo', 'tokyo', 'paris']
```

Binarize

```
In [8]: lb = preprocessing.LabelBinarizer()  
lb.fit(["paris", "paris", "tokyo", "amsterdam"])
```

```
Out[8]: LabelBinarizer(neg_label=0, pos_label=1, sparse_output=False)
```

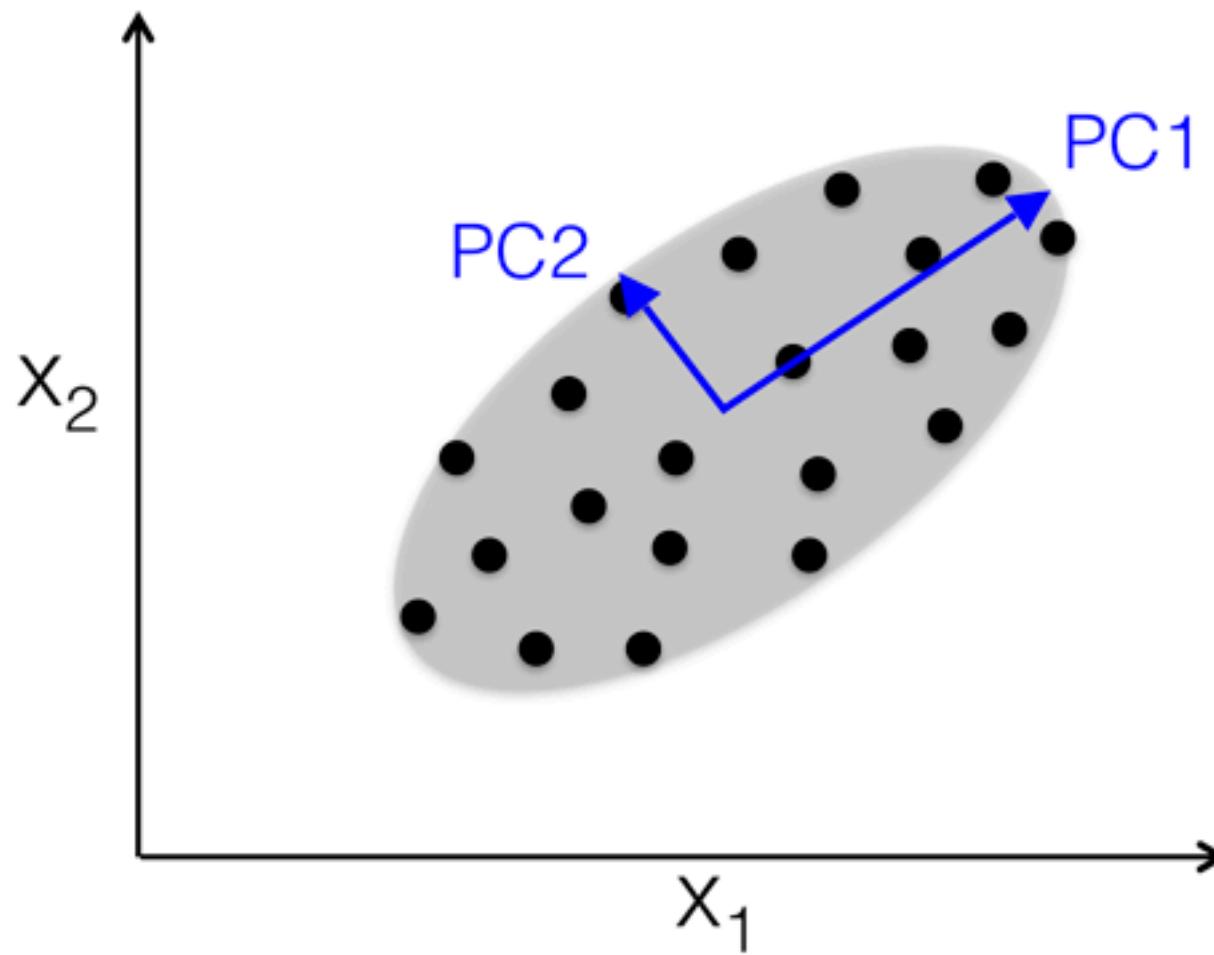
```
In [9]: lb.classes_
```

```
Out[9]: array(['amsterdam', 'paris', 'tokyo'], dtype='<U9')
```

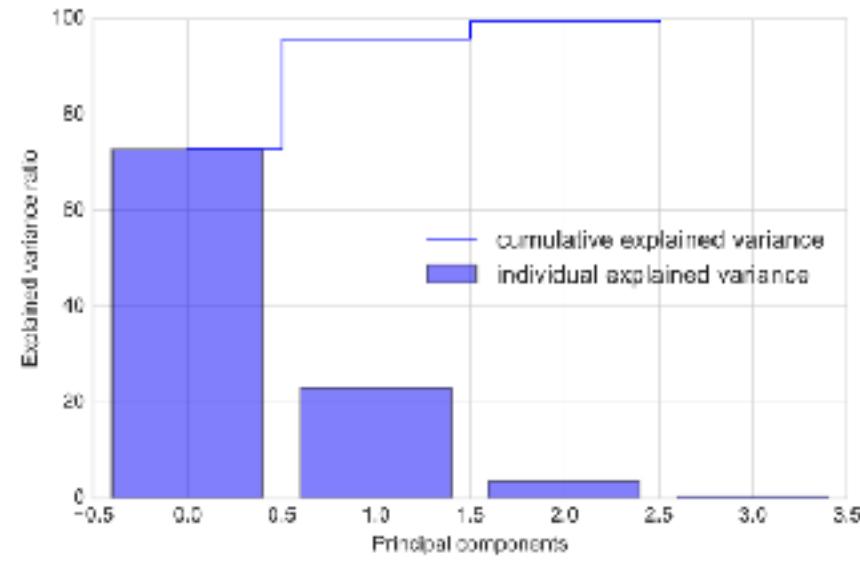
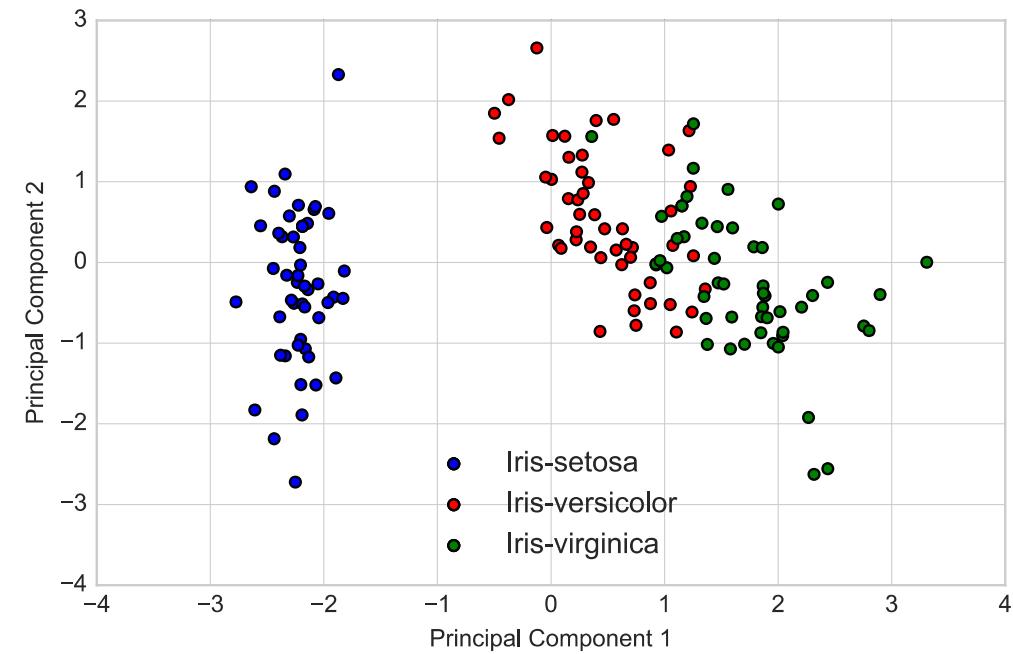
```
In [10]:  
lb.transform(['paris', 'tokyo'])
```

```
Out[10]: array([[0, 1, 0],  
                 [0, 0, 1]])
```

PCA



PCA for Dimensionality Reduction



PCA

```
In [13]: %matplotlib inline
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

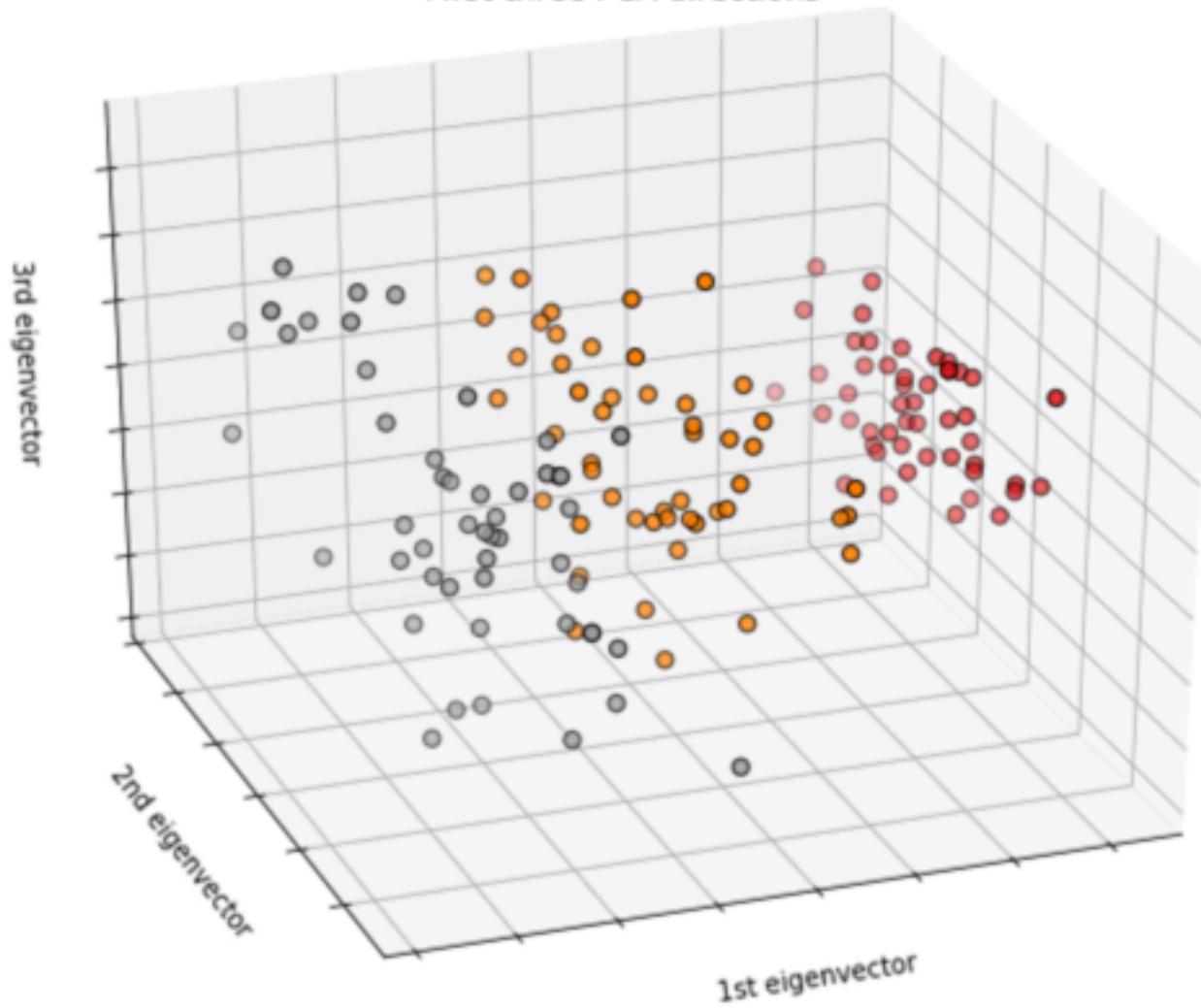
# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# To get a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

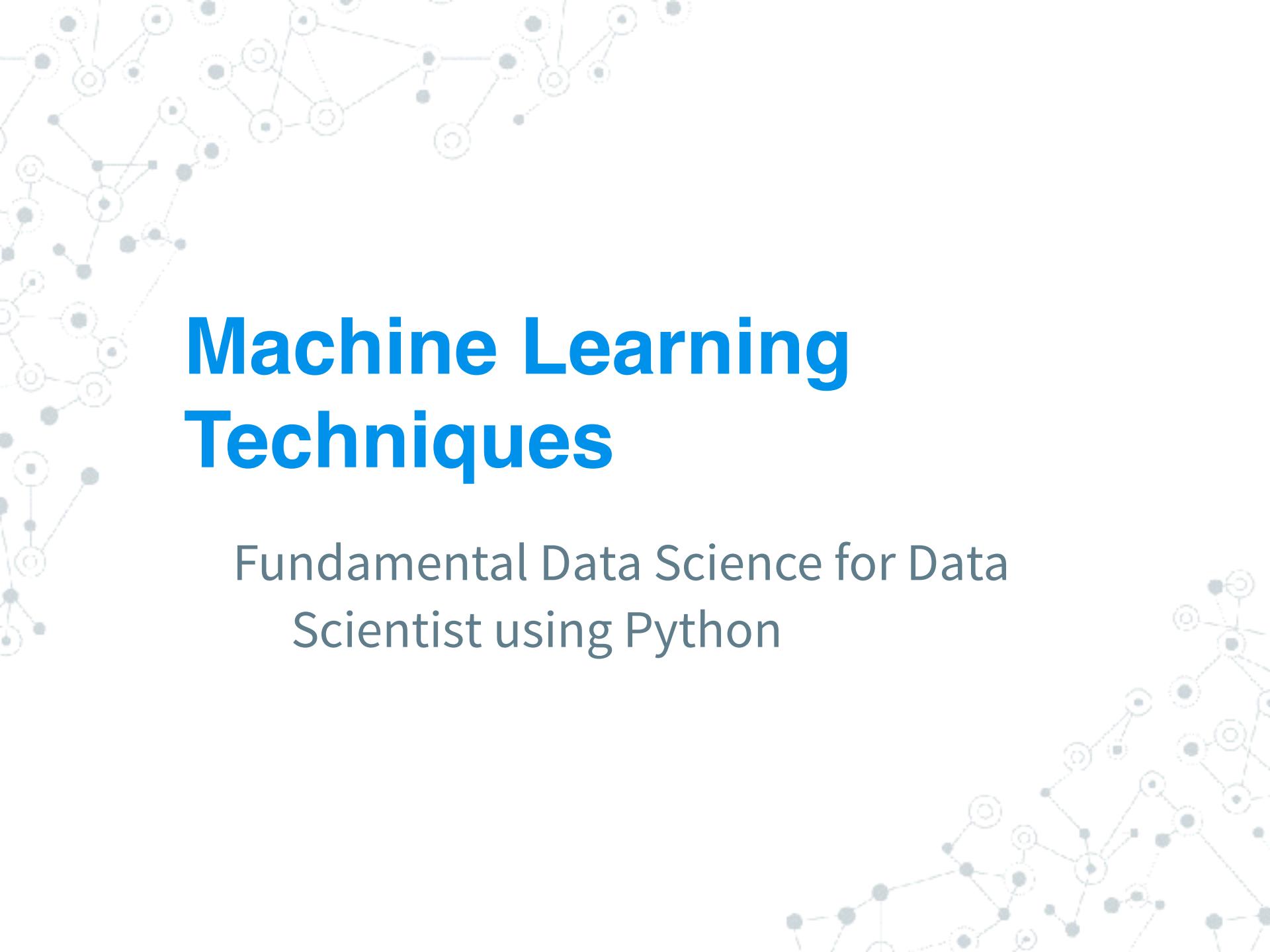
plt.show()
```

First three PCA directions



Workshop 2 - Data Preparation

- Use data from Workshop 1
- Practice data preparation using step from Page 61-75



Machine Learning Techniques

Fundamental Data Science for Data
Scientist using Python

Topic

- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

An example application

- ◎ An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- ◎ **A decision is needed:** whether to put a new patient in an intensive-care unit.
- ◎ Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- ◎ **Problem:** to predict **high-risk patients** and discriminate them from **low-risk patients**.

Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
 - age
 - Marital status
 - annual salary
 - outstanding debts
 - credit rating
 - etc.
- **Problem:** to decide whether an application should be approved, or to classify applications into two categories, approved and **not approved**.

The data and the goal

- **Data:** A set of data records (also called examples, instances or cases) described by
 - ***k* attributes:** A_1, A_2, \dots, A_k .
 - **a class:** Each example is labelled with a pre-defined class.
- **Goal:** To learn a **classification model** from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

An example: the learning task

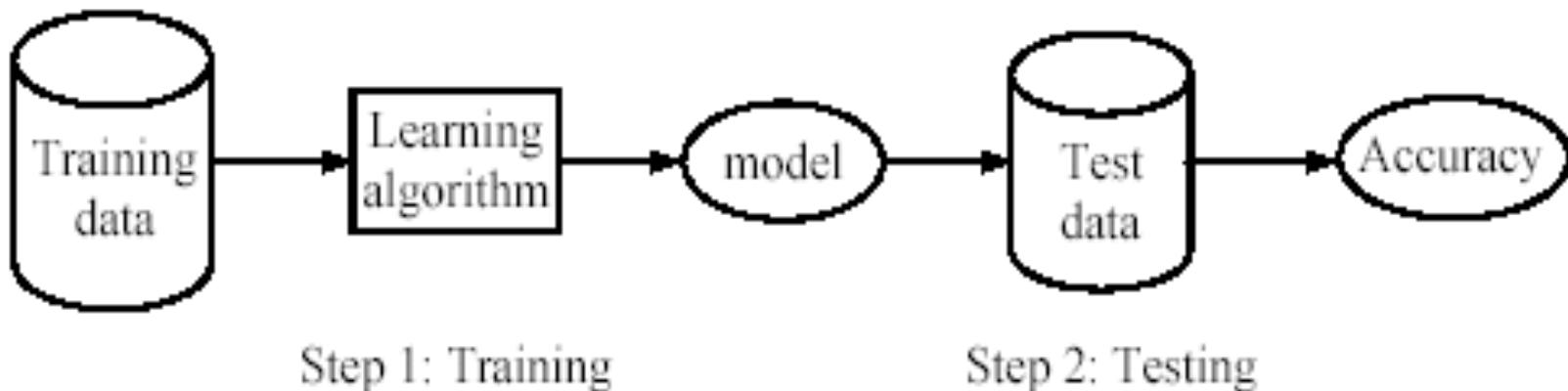
- Learn a classification model from the data
- Use the model to classify future loan applications into
 - Yes (approved) and
 - No (not approved)
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

Supervised learning process: two steps

- **Learning (training)**: Learn a model using the **training data**
- **Testing**: Test the model using **unseen test data** to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



An example

- **Data:** Loan application data
 - **Task:** Predict whether a loan should be approved or not.
 - **Performance measure:** accuracy.
- ◎ **No learning:** classify all future applications (test data) to the majority class (i.e., Yes):
- Accuracy = $9/15 = 60\%$.**
- We can do better than 60% with learning.

Fundamental assumption of learning

◎ Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Topic

- Basic concepts
- **Decision tree induction**
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Decision tree learning is one of the most widely used techniques for classification.
 - Its classification accuracy is competitive with other methods, and
 - it is very efficient.
- The classification model is a tree, called **decision tree**.
- C4.5 by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

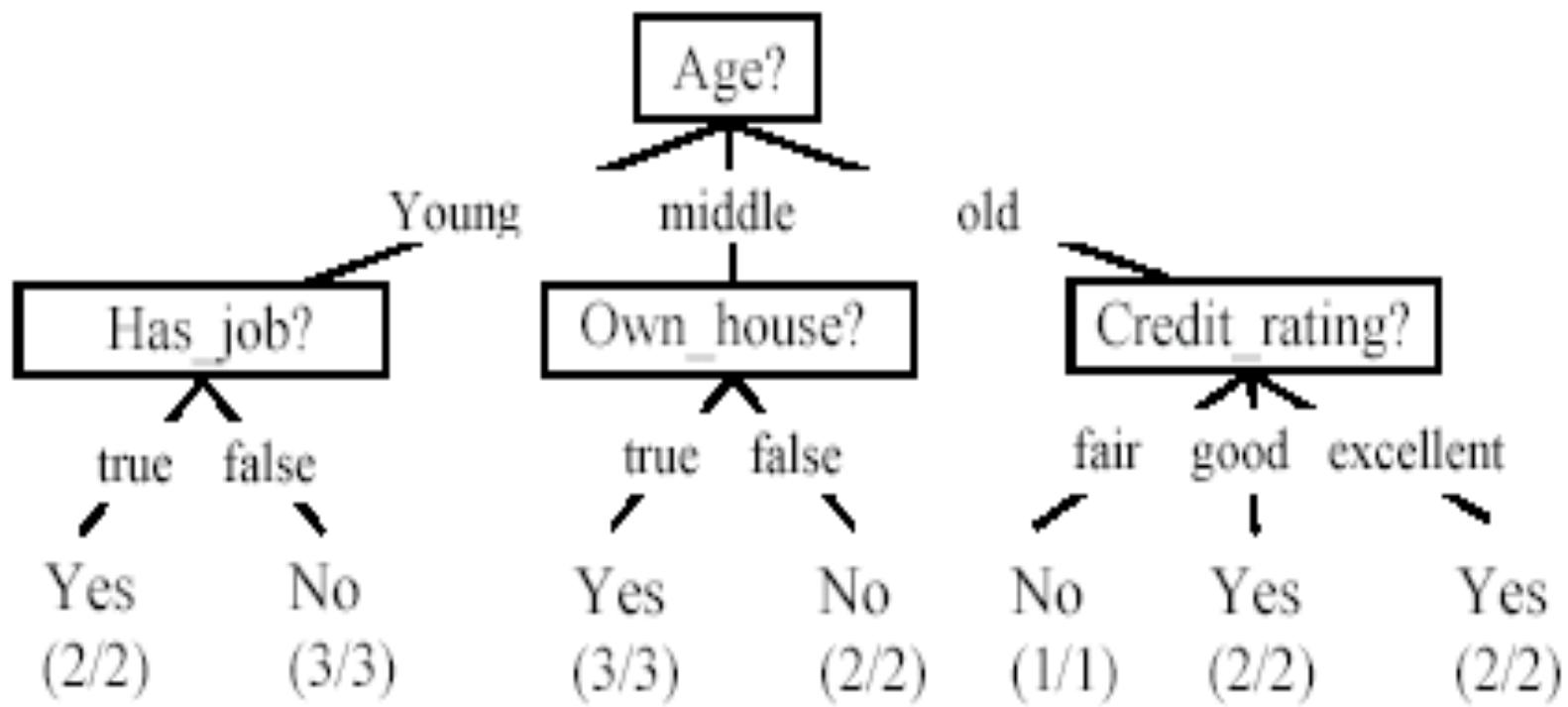
The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

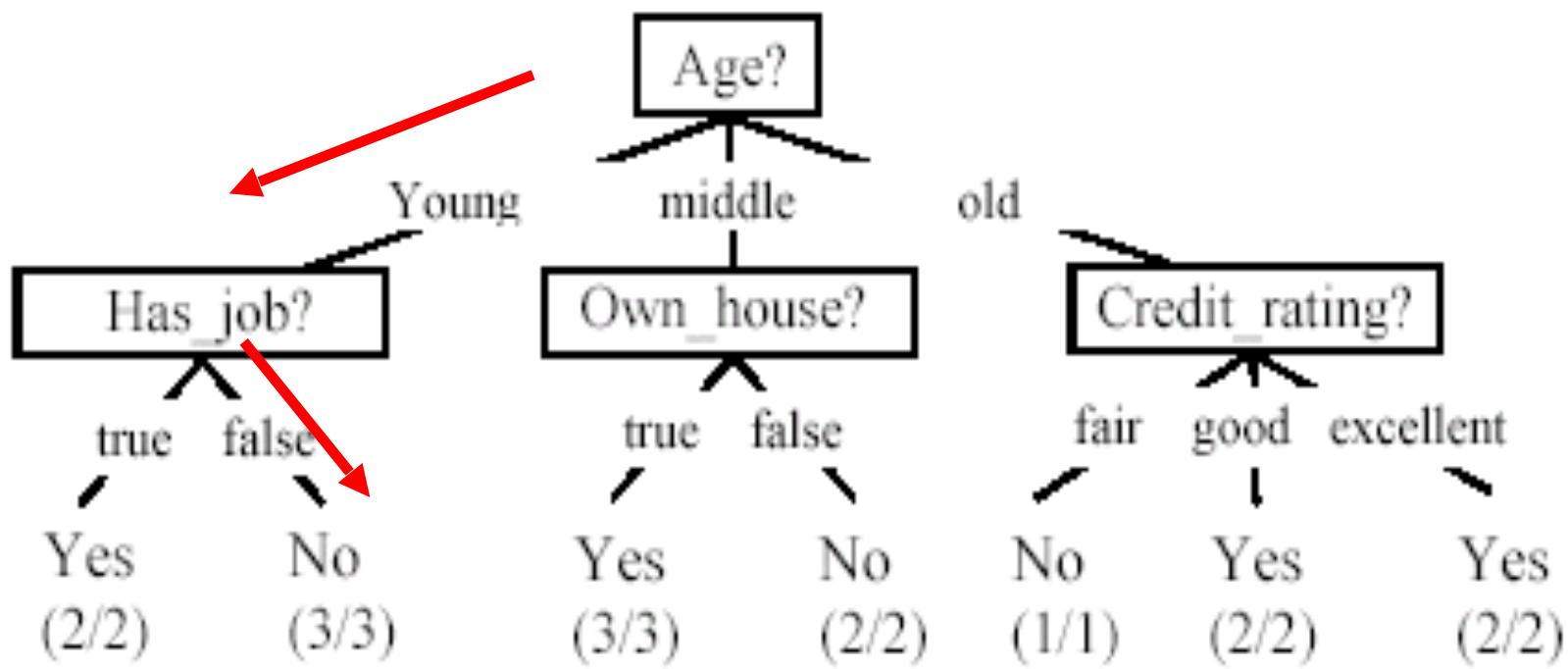
A decision tree from the loan data

- Decision nodes and leaf nodes (classes)



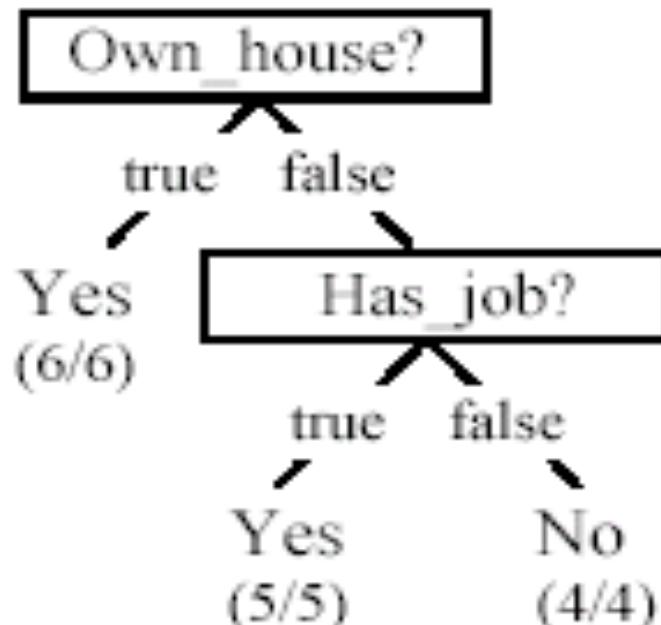
Use the decision tree

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	No



From a decision tree to a set of rules

- A decision tree can be converted to a set of rules
- Each path from the root to a leaf is a rule.



Own_house = true → Class = Yes [sup=6/15, conf=6/6]

Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]

Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithm for decision tree learning

- Basic algorithm (a greedy **divide-and-conquer** algorithm)
 - Assume attributes are categorical now (continuous attributes can be handled too)
 - Tree is constructed in a **top-down recursive manner**
 - At start, all the training examples are at the root
 - Examples are partitioned recursively based on selected attributes
 - Attributes are selected on the basis of an impurity function (e.g., **information gain**)
- Conditions for stopping partitioning
 - All examples for a given node belong to the same class
 - There are no remaining attributes for further partitioning – majority class is the leaf
 - There are no examples left

Decision tree learning algorithm

```
. Algorithm decisionTree( $D, A, T$ )
1   if  $D$  contains only training examples of the same class  $c_j \in C$  then
2       make  $T$  a leaf node labeled with class  $c_j$ ;
3   elseif  $A = \emptyset$  then
4       make  $T$  a leaf node labeled with  $c_j$ , which is the most frequent class in  $D$ 
5   else //  $D$  contains examples belonging to a mixture of classes. We select a single
6       // attribute to partition  $D$  into subsets so that each subset is purer
7        $p_0 = \text{impurityEval-1}(D)$ ;
8       for each attribute  $A_i \in \{A_1, A_2, \dots, A_k\}$  do
9            $p_i = \text{impurityEval-2}(A_i, D)$ 
10      end
11      Select  $A_g \in \{A_1, A_2, \dots, A_k\}$  that gives the biggest impurity reduction,
12          computed using  $p_0 - p_g$ 
13      if  $p_0 - p_g < \text{threshold}$  then //  $A_g$  does not significantly reduce impurity  $p_0$ 
14          make  $T$  a leaf node labeled with  $c_j$ , the most frequent class in  $D$ .
15      else //  $A_g$  is able to reduce impurity  $p_0$ 
16          Make  $T$  a decision node on  $A_g$ ;
17          Let the possible values of  $A_g$  be  $v_1, v_2, \dots, v_m$ . Partition  $D$  into  $m$ 
18          disjoint subsets  $D_1, D_2, \dots, D_m$  based on the  $m$  values of  $A_g$ .
19          for each  $D_j$  in  $\{D_1, D_2, \dots, D_m\}$  do
20              if  $D_j \neq \emptyset$  then
21                  create a branch (edge) node  $T_j$  for  $v_j$  as a child node of  $T$ ;
22                  decisionTree( $D_j, A-\{A_g\}, T_j$ ) //  $A_g$  is removed
23              end
24          end
25      end
26  end
```

Choose an attribute to partition data

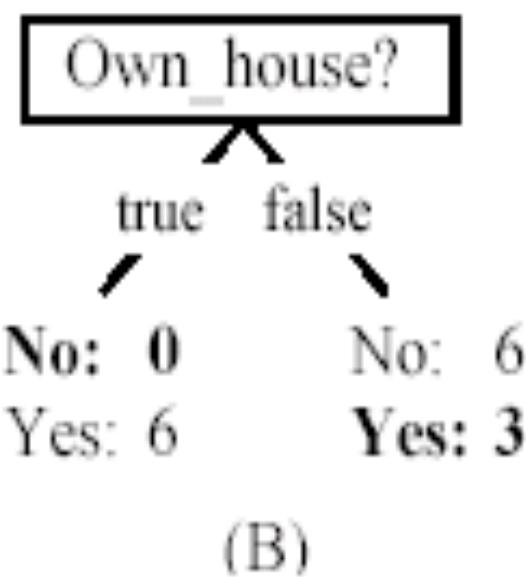
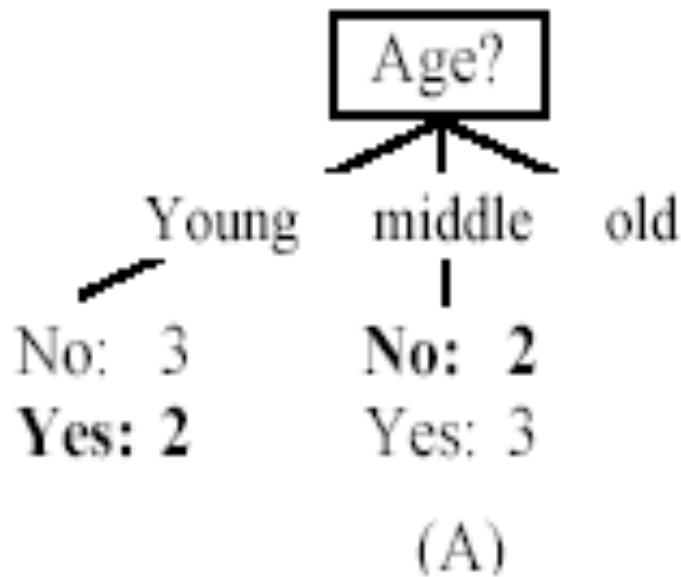
- The **key** to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.
A subset of data is **pure** if all instances belong to the same class.
- The **heuristic** in C4.5 is to choose the attribute with the maximum **Information Gain** or **Gain Ratio** based on information theory.

The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Two possible roots, which is better?

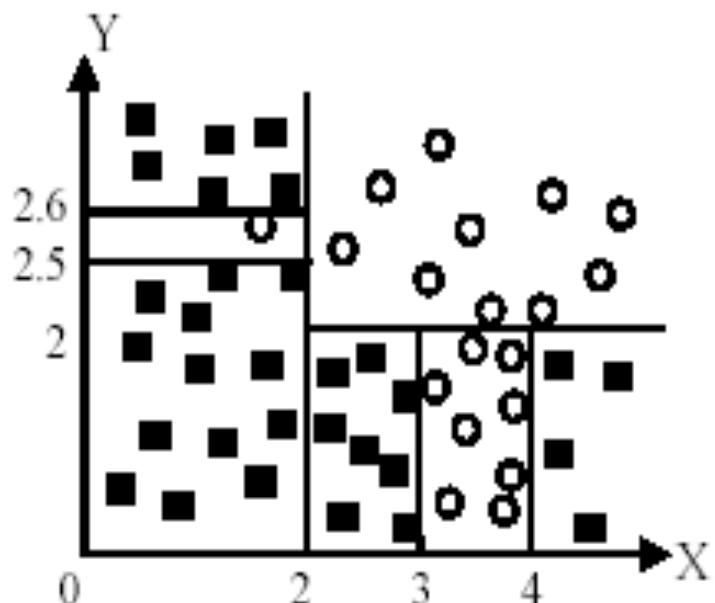


- Fig. (B) seems to be better.

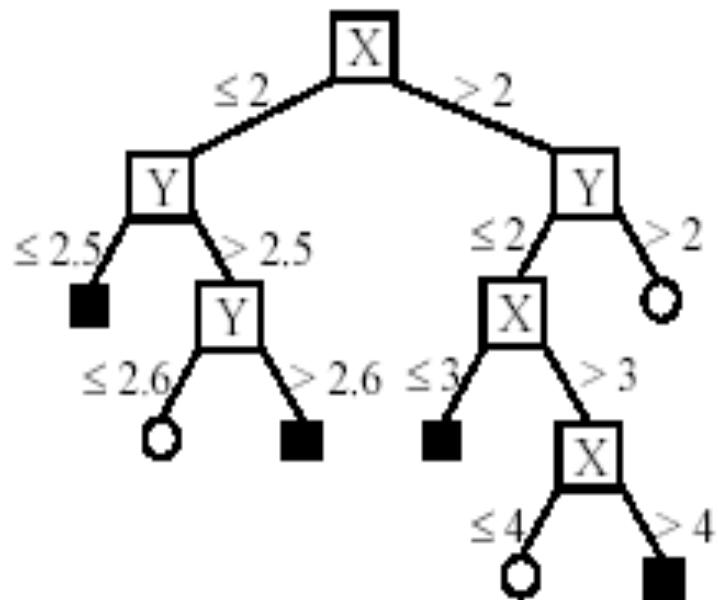
Handling continuous attributes

- Handle continuous attribute by splitting into two intervals (can be more) at each node.
- How to find the best threshold to divide?
 - Use information gain or gain ratio again
 - Sort all the values of an continuous attribute in increasing order $\{v_1, v_2, \dots, v_r\}$,
 - One possible threshold between two adjacent values v_i and v_{i+1} . Try all possible thresholds and find the one that maximizes the gain (or gain ratio).

An example in a continuous space



(A) A partition of the data space



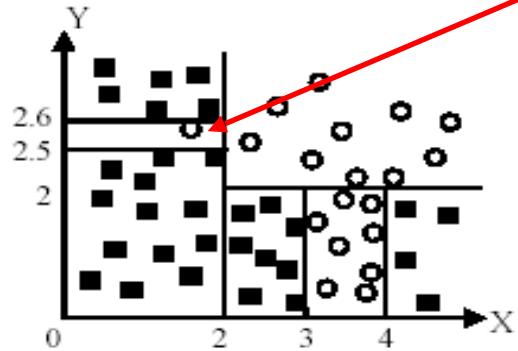
(B). The decision tree

Avoid overfitting in classification

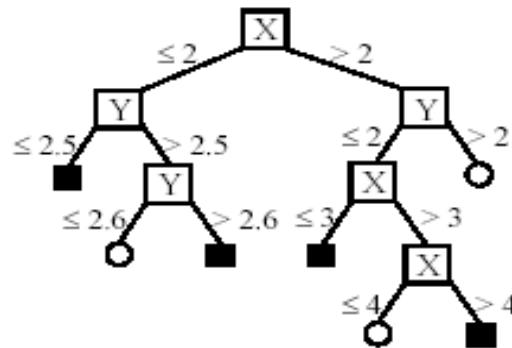
- **Overfitting:** A tree may overfit the training data
 - Good accuracy on training data but poor on test data
 - Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers
- Two approaches to avoid overfitting
 - **Pre-pruning:** Halt tree construction early
 - Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
 - **Post-pruning:** Remove branches or sub-trees from a “fully grown” tree.
 - This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.
 - A validation set may be used for pruning as well.

An example

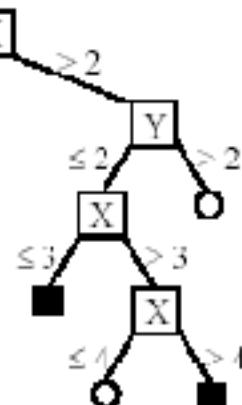
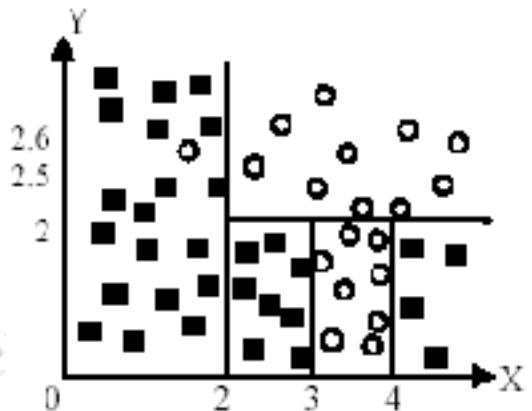
Likely to overfit the data



(A) A partition of the data space



(B). The decision tree



Other issues in decision tree learning

- From tree to rules, and rule pruning
- Handling of miss values
- Handing skewed distributions
- Handling attributes and classes with different costs.
- Attribute construction
- Etc.

Evaluating classification methods

- **Predictive accuracy**

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

- **Efficiency**

- time to construct the model
 - time to use the model

- **Robustness:** handling noise and missing values

- **Scalability:** efficiency in disk-resident databases

- **Interpretability:**

- understandable and insight provided by the model

Compactness of the model: size of the tree, or the number of rules.

Decision Tree in Python

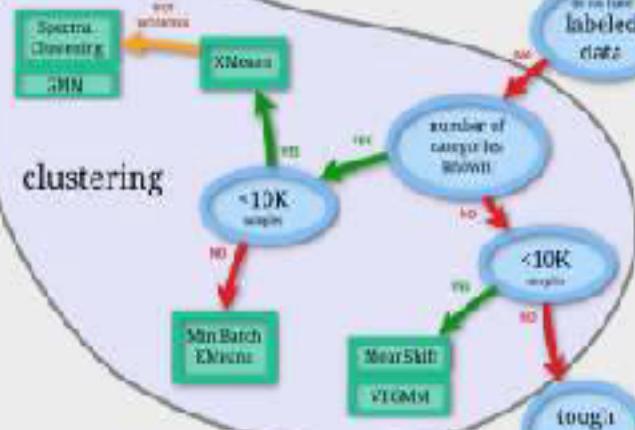
Fundamental Data Science for Data
Scientist using Python

scikit-learn algorithm cheat-sheet

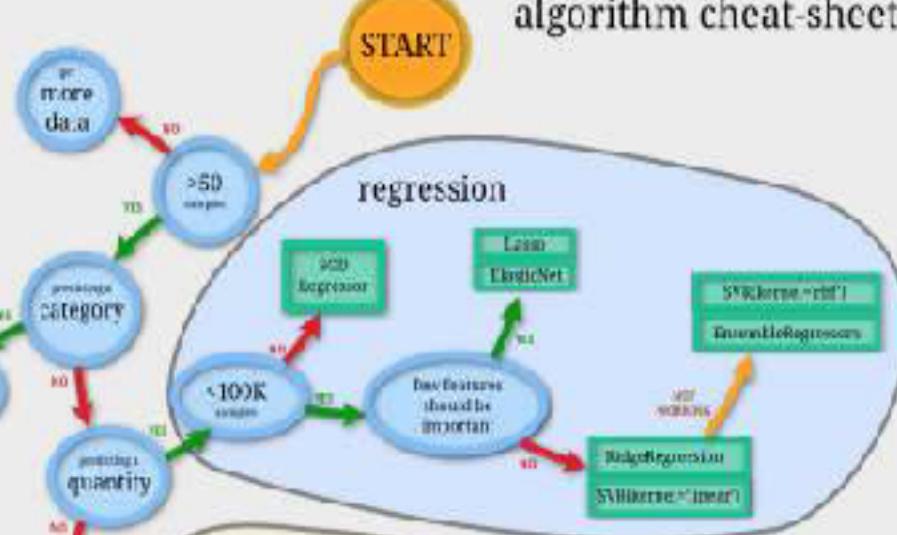
classification



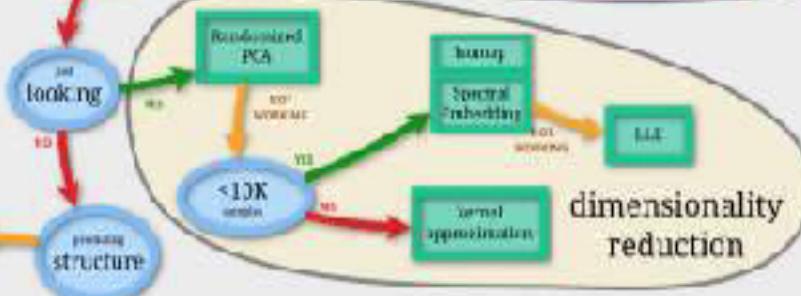
clustering



regression



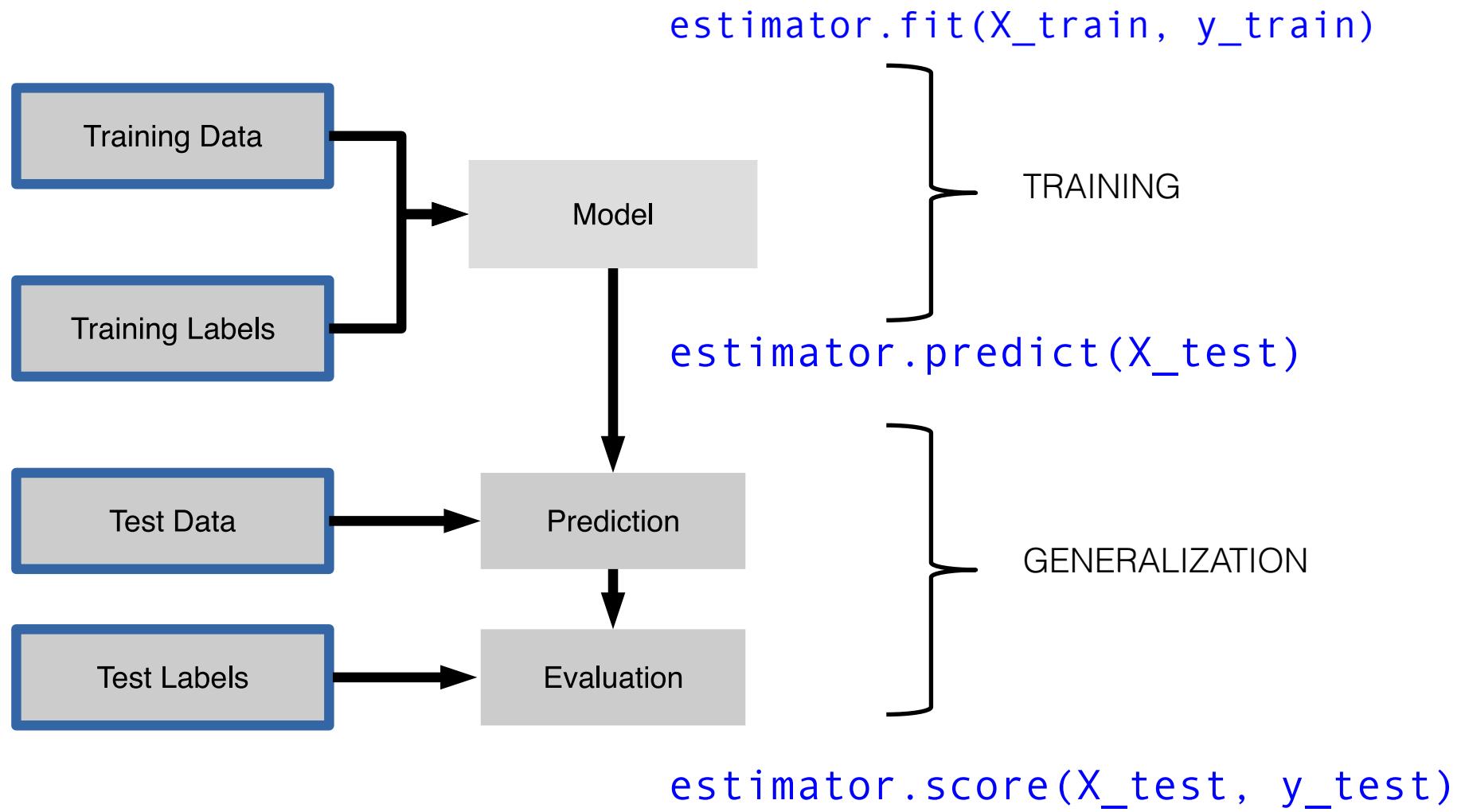
dimensionality reduction



Back

scikit-learn

Supervised Workflow



Scikit-learn API

estimator.fit(X_train, [y_train])	
estimator.predict(X_test)	estimator.transform(X_test)
Classification	Preprocessing
Regression	Dimensionality Reduction
Clustering	Feature Extraction
	Feature selection

Iris Data Preparation

```
In [34]: from time import time
import logging
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import tree
from figures import plot_2d_separator

import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns

iris = pd.read_csv("iris.csv")

iris
```

```
In [35]: from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(iris.iloc[:, 0:4],  
                                                iris['species'],  
                                                train_size=0.7,  
                                                random_state=123)  
  
print("Labels for training and testing data")  
print(y_train)
```

Labels for training and testing data

```
114    Iris-virginica  
136    Iris-virginica  
53     Iris-versicolor  
19     Iris-setosa  
38     Iris-setosa  
110    Iris-virginica  
23     Iris-setosa  
9      Iris-setosa  
..    ...
```

Decision Tree

```
In [50]: from sklearn import tree  
from sklearn.metrics import classification_report  
from sklearn.metrics import confusion_matrix
```

```
In [51]: clf = tree.DecisionTreeClassifier(max_depth=10)  
clf.fit(X_train, y_train)
```

```
Out[51]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,  
max_features=None, max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, presort=False, random_state=None,  
splitter='best')
```

Decision Tree - Create Model



In [37]:

```
clf = tree.DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
                                  max_features=None, max_leaf_nodes=None, min_samples_leaf=5,
                                  min_samples_split=2, min_weight_fraction_leaf=0.0,
                                  presort=False, random_state=100, splitter='best')

clf.fit(X_train, y_train)
```

Out[37]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=5, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=100,
splitter='best')

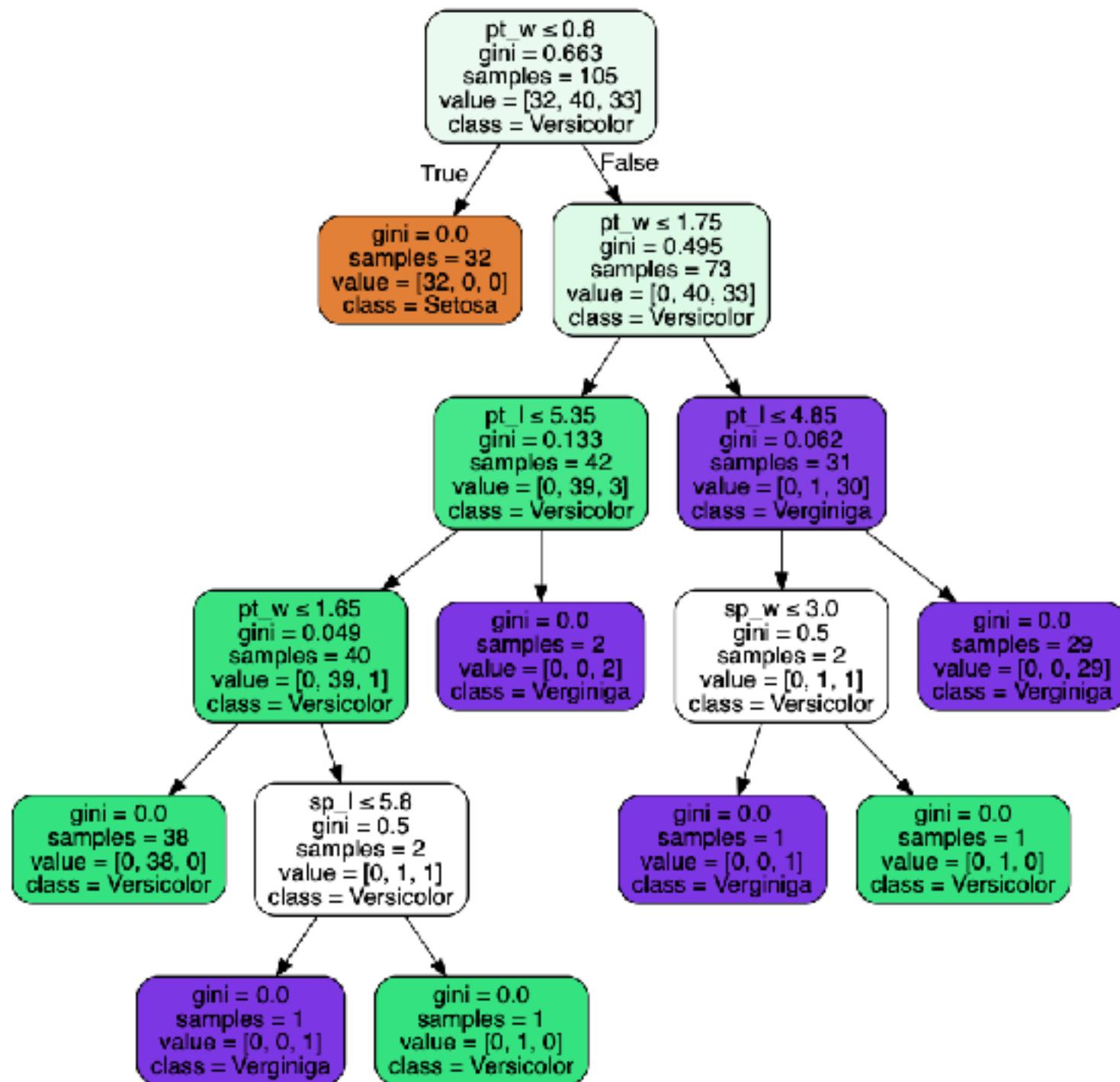


Decision Tree

```
In [48]: from IPython.display import Image
from sklearn.externals.six import StringIO
import pydotplus

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data, feature_names=['sp_l','sp_w','pt_l','pt_w'],
                     class_names=['Setosa','Versicolor','Verginica'],filled=True, rounded=True,
                     special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[48]:



Decision Tree - Prediction

```
In [49]: print("Predicting iris on the test set")
t0 = time()
y_pred = clf.predict(X_test)
print("done in %0.3fs" % (time() - t0))
```

```
print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))
print(confusion_matrix(y_test, y_pred))
```

```
Predicting iris on the test set
done in 0.001s
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.83	1.00	0.91	10
Iris-virginica	1.00	0.88	0.94	17
avg / total	0.96	0.96	0.96	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  2 15]]
```

Workshop 3 - Decision Tree

1. Use data from Workshop 2
2. Perform Decision Tree Algorithm to predict outcomes

Evaluation methods

- **Holdout set:** The available data set D is divided into two disjoint subsets,
 - the *training set* D_{train} (for learning a model)
 - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
 - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the **holdout set**. (the examples in the original data set D are all labeled with classes.)
- This method is mainly used when the data set D is large.

Evaluation methods (cont...)

- **n-fold cross-validation:** The available data is partitioned into n equal-size disjoint subsets.
- Use each subset as the test set and combine the rest $n-1$ subsets as the training set to learn a classifier.
- The procedure is run n times, which give n accuracies.
- The final estimated accuracy of learning is the average of the n accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

All Data

Training data

Test data

However, often (labeled) data is precious, and this approach lets us only use ~ 3/4 of our data for training. On the other hand, we will only ever try to apply our model 1/4 of our data for testing. A common way to use more of the data to build a model, but also get a more robust estimate of the generalization performance, is cross-validation. In cross-validation, the data is split repeatedly into a training and non-overlapping test-sets, with a separate model built for every pair. The test-set scores are then aggregated for a more robust estimate.

The most common way to do cross-validation is k-fold cross-validation, in which the data is first split into k (often 5 or 10) equal-sized folds, and then for each iteration, one of the k folds is used as test data, and the rest as training data:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Evaluation methods (cont...)

- **Leave-one-out cross-validation:** This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has m examples, this is m -fold cross-validation

Evaluation methods (cont...)

- **Validation set:** the available data is divided into three subsets,
 - a training set,
 - a validation set and
 - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications.**
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, **we are interested only in the minority class.**
 - High accuracy does not mean any intrusion is detected.
 - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Precision and recall measures (cont...)

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP} . \quad r = \frac{TP}{TP + FN} .$$

- Precision p is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.
- Recall r is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- ▶ This confusion matrix gives
 - precision $p = 100\%$ and recall $r = 1\%$
- ▶ because we only classified one positive example correctly and no negative examples wrongly.
- ▶ Note: precision and recall only measure classification on the positive class.

F_1 -value (also called F_1 -score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p+r}$$

F_1 -score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The F_1 -value is the harmonic mean of the two.
- For F_1 -value to be large, both p and r must be large.

Receive operating characteristics curve

- It is commonly called the **ROC curve**.
- It is a plot of the **true positive rate (TPR)** against the **false positive rate (FPR)**.
- **True positive rate:**

$$TPR = \frac{TP}{TP + FN}$$

- **False positive rate:**

$$FPR = \frac{FP}{TN + FP}$$

Sensitivity and Specificity

- In statistics, there are two other evaluation measures:
 - **Sensitivity**: Same as TPR
 - **Specificity**: Also called **True Negative Rate** (TNR)
- Then we have

$$TNR = \frac{TN}{TN + FP}$$

$$FPR = 1 - specificity$$

Example ROC curves

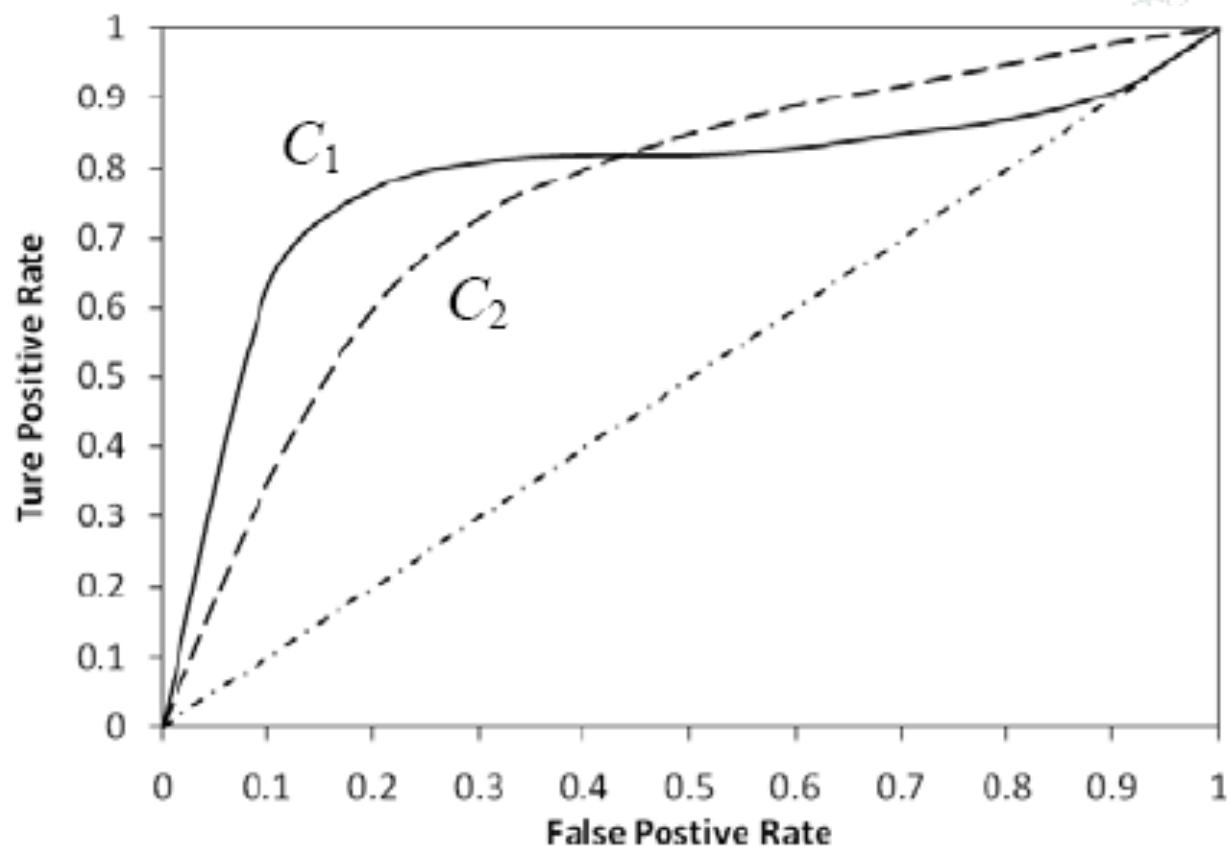


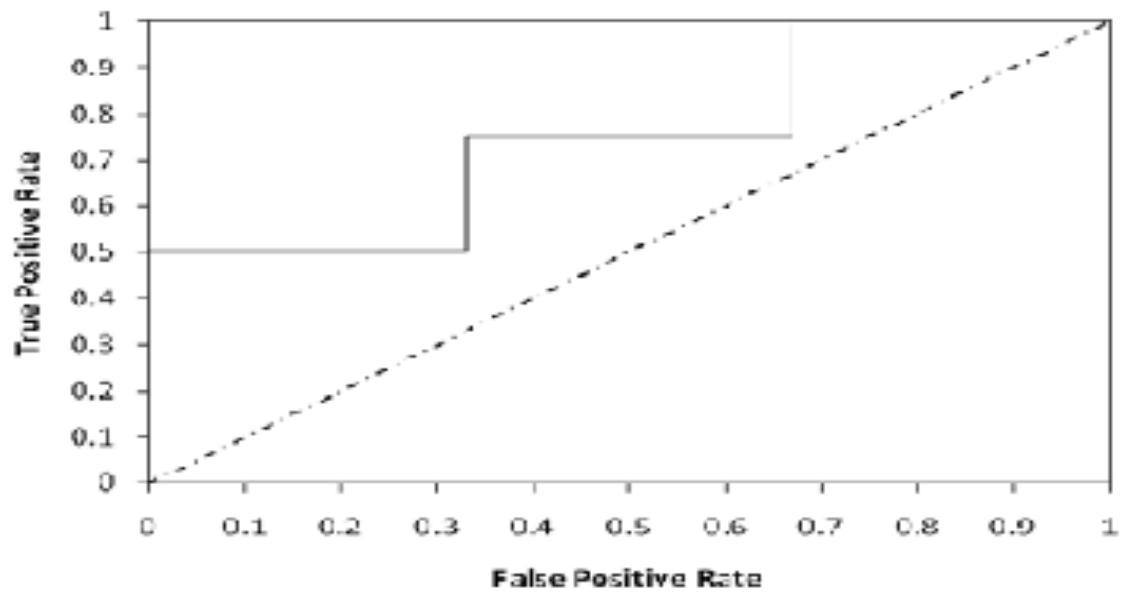
Fig. 3.8. ROC curves for two classifiers (C_1 and C_2) on the same data

Area under the curve (AUC)

- Which classifier is better, C_1 or C_2 ?
 - It depends on which region you talk about.
- Can we have one measure?
 - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j , it is said that C_i is better than C_j .
 - If a classifier is perfect, its AUC value is 1
 - If a classifier makes all random guesses, its AUC value is 0.5.

Drawing an ROC curve

Rank	1	2	3	4	5	6	7	8	9	10
Actual class	+	-			+			+		
TP	0	1	2	2	2	3	3	3	4	4
FP	0	0	0	1	2	2	3	4	4	5
TN	6	6	6	5	4	4	3	2	2	1
FN	1	3	2	2	2	1	1	1	0	0
TPR	0	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83



Another evaluation method: Scoring and ranking

- **Scoring** is related to classification.
- We are interested in a single class (**positive class**), e.g., buyers class in a marketing database.
- Instead of assigning each test instance a definite class, scoring assigns a probability estimate (PE) to indicate the likelihood that the example belongs to the positive class.

Ranking and lift analysis

- After each example is given a PE score, we can rank all examples according to their PEs.
- We then divide the data into n (say 10) bins. A lift curve can be drawn according how many positive examples are in each bin. This is called **lift analysis**.
- Classification systems can be used for scoring. Need to produce a probability estimate.
 - E.g., in decision trees, we can use the confidence value at each leaf node as the score.

An example

- We want to send promotion materials to potential customers to sell a watch.
- Each package cost \$0.50 to send (material and postage).
- If a watch is sold, we make \$5 profit.
- Suppose we have a large amount of past data for building a predictive/classification model. We also have a large list of potential customers.
- How many packages should we send and who should we send to?

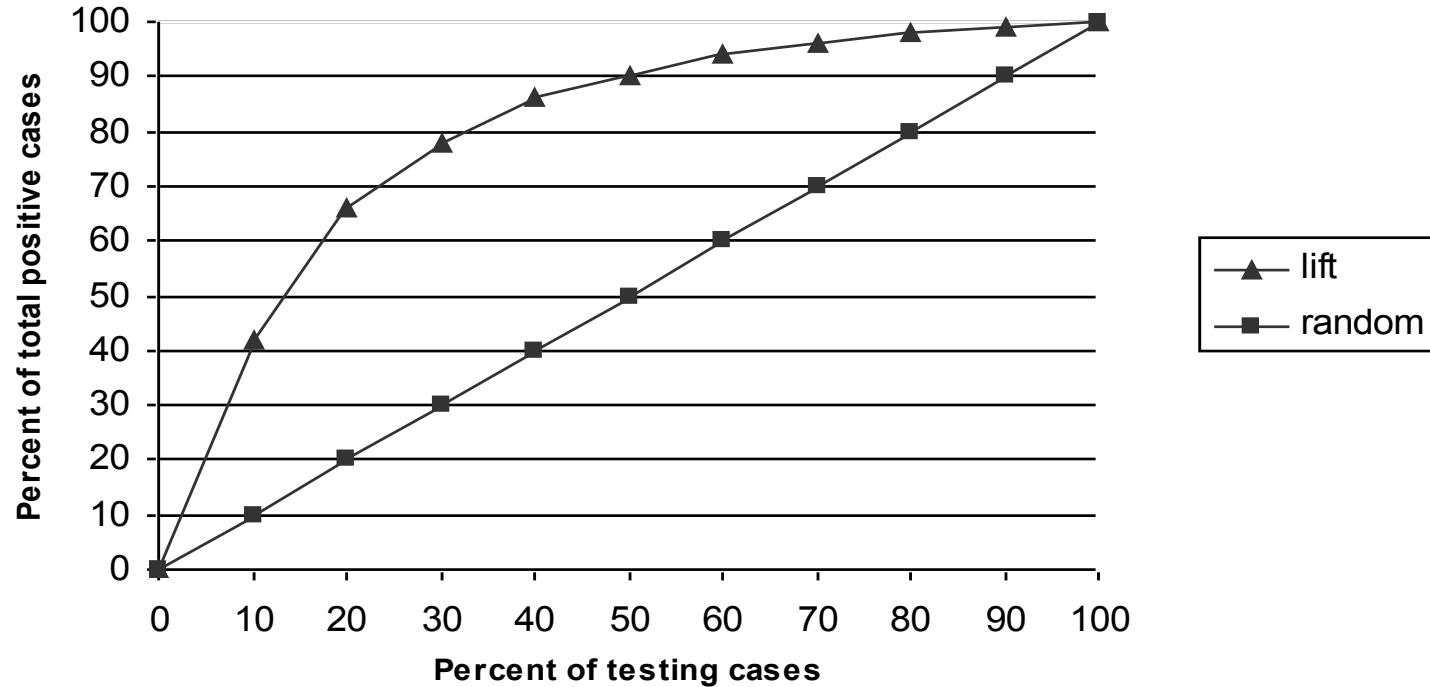
An example

- Assume that the test set has 10000 instances. Out of this, 500 are positive cases.
- After the classifier is built, we score each test instance. We then rank the test set, and divide the ranked test set into 10 bins.
 - Each bin has 1000 test instances.
 - Bin 1 has 210 actual positive instances
 - Bin 2 has 120 actual positive instances
 - Bin 3 has 60 actual positive instances
 - ...
 - Bin 10 has 5 actual positive instances

Lift curve

Bin 1 2 3 4 5 6 7 8 9 10

210	120	60	40	22	18	12	7	6	5
42%	24%	12%	8%	4.40%	3.60%	2.40%	1.40%	1.20%	1%
42%	66%	78%	86%	90.40%	94%	96.40%	97.80%	99%	100%



Topic

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- **Naïve Bayesian classification**
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Bayesian classification

- **Probabilistic view:** Supervised learning can naturally be studied from a probabilistic point of view.
- Let A_1 through A_k be attributes with discrete values.
The class is C .
- Given a test example d with observed attribute values a_1 through a_k .
- Classification is basically to compute the following posteriori probability. The prediction is the class c_j such that

$$\Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|})$$

is maximal

Apply Bayes' Rule

$$\begin{aligned} & \Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|}) \\ = & \frac{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_j) \Pr(C = c_j)}{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|})} \\ = & \frac{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_j) \Pr(C = c_j)}{\sum_{r=1}^{|C|} \Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_r) \Pr(C = c_r)} \end{aligned}$$

- $\Pr(C=c_j)$ is the class prior probability: easy to estimate from the training data.

Computing probabilities

- The denominator $P(A_1=a_1, \dots, A_k=a_k)$ is irrelevant for decision making since it is the same for every class.
- We only need $P(A_1=a_1, \dots, A_k=a_k | C=c_i)$, which can be written as
 - ◎ $\Pr(A_1=a_1 | A_2=a_2, \dots, A_k=a_k, C=c_j) * \Pr(A_2=a_2, \dots, A_k=a_k | C=c_j)$
- Recursively, the second factor above can be written in the same way, and so on.
Now an assumption is needed.

Final naïve Bayesian classifier

$$\Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|}) \\ = \frac{\Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)}{\sum_{r=1}^{|C|} \Pr(C = c_r) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_r)}$$

- We are done!
- How do we estimate $\Pr(A_i = a_i \mid C = c_j)$? Easy!.

Classify a test instance

- If we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class.
- Thus, given a test example, we compute the following to decide the most probable class for the test instance

$$c = \arg \max_{c_j} \Pr(c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

An example

- Compute all probabilities required for classification

A	B	C
m	b	t
m	s	t
g	q	t
h	s	t
g	q	t
g	q	f
g	s	f
h	b	f
h	q	f
m	b	f

$$\Pr(C = t) = 1/2,$$

$$\Pr(C = f) = 1/2$$

$$\Pr(A=m \mid C=t) = 2/5$$

$$\Pr(A=g \mid C=t) = 2/5$$

$$\Pr(A=h \mid C=t) = 1/5$$

$$\Pr(A=m \mid C=f) = 1/5$$

$$\Pr(A=g \mid C=f) = 2/5$$

$$\Pr(A=h \mid C=f) = 2/5$$

$$\Pr(B=b \mid C=t) = 1/5$$

$$\Pr(B=s \mid C=t) = 2/5$$

$$\Pr(B=q \mid C=t) = 2/5$$

$$\Pr(B=b \mid C=f) = 2/5$$

$$\Pr(B=s \mid C=f) = 1/5$$

$$\Pr(B=q \mid C=f) = 2/5$$

Now we have a test example:

$$A = m \quad B = q \quad C = ?$$

An Example (cont ...)

- For $C = t$, we have

$$\Pr(C = t) \prod_{j=1}^2 \Pr(A_j = a_j | C = t) = \frac{1}{2} \times \frac{2}{5} \times \frac{2}{5} = \frac{2}{25}$$

- For class $C = f$, we have

$$\Pr(C = f) \prod_{j=1}^2 \Pr(A_j = a_j | C = f) = \frac{1}{2} \times \frac{1}{5} \times \frac{2}{5} = \frac{1}{25}$$

- $C = t$ is more probable. t is the final class.

Additional issues

- **Numeric attributes:** Naïve Bayesian learning assumes that all attributes are categorical. Numeric attributes need to be discretized.
- **Zero counts:** An particular attribute value never occurs together with a class in the training set. We need smoothing.

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda n_i}$$

- **Missing values:** Ignored

On naïve Bayesian classifier

- **Advantages:**

- Easy to implement
- Very efficient
- Good results obtained in many applications

- **Disadvantages**

- Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)

Naïve Bayesian classifier

Fundamental Data Science for Data
Scientist using Python

Naive Bayes

```
In [14]: from sklearn.naive_bayes import GaussianNB  
model = GaussianNB()  
model.fit(X_train, y_train);
```

```
In [17]: print("Predicting iris on the test set using Naive Bayes")  
t0 = time()  
y_pred = model.predict(X_test)  
print("done in %0.3fs" % (time() - t0))  
  
print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))  
print(confusion_matrix(y_test, y_pred))
```

Naive Bayes

Predicting iris on the test set using Naive Bayes
done in 0.001s

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.83	1.00	0.91	10
Iris-virginica	1.00	0.88	0.94	17
avg / total	0.96	0.96	0.96	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  2 15]]
```

Workshop 4 - Naive Bayes Classifier

1. Use data from Workshop 2
2. Perform Naive Bayes Classifier Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- **K-nearest neighbor**
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

k-Nearest Neighbor Classification (kNN)

- Unlike all the previous learning methods, kNN does not build model from the training data.
- To classify a test instance d , define k -neighborhood P as k nearest neighbors of d
- Count number n of training instances in P that belong to class c_j
- Estimate $\Pr(c_j|d)$ as n/k
- No training is needed. Classification time is linear in training set size for each test case.

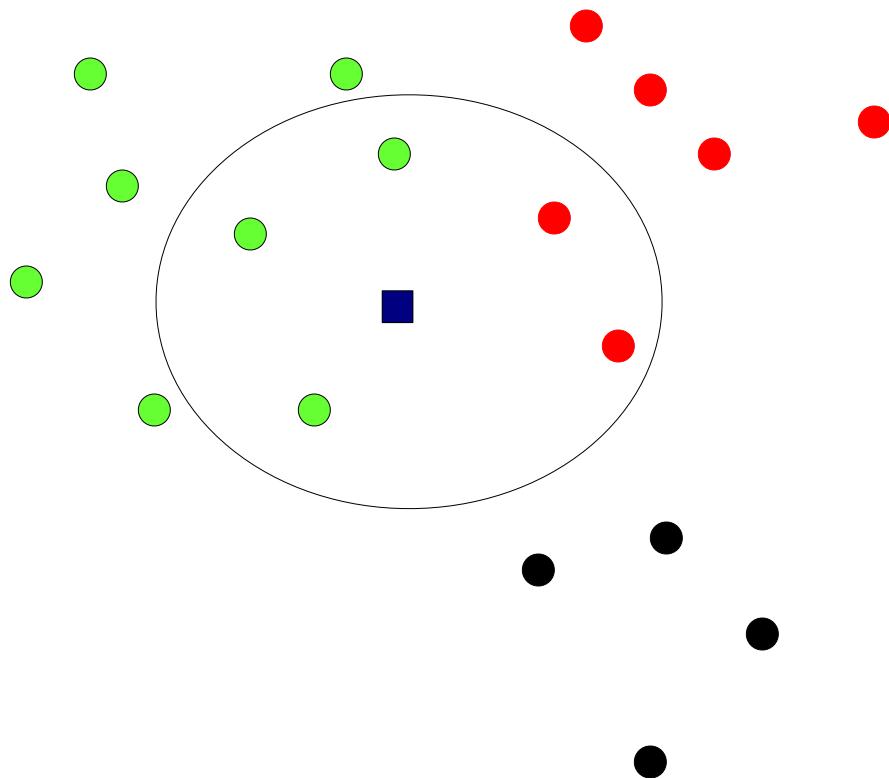
kNNAlgorithm

Algorithm kNN(D, d, k)

- 1 Compute the distance between d and every example in D ;
- 2 Choose the k examples in D that are nearest to d , denote the set by P ($\subseteq D$);
- 3 Assign d the class that is the most frequent class in P (or the majority class);

- k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.
- **Distance function** is crucial, but depends on applications.

Example: k=5



- Government
- Science
- Arts

A new point



Discussions

- kNN can deal with complex and arbitrary decision boundaries.
- Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong and in many cases as accurate as those elaborated methods.
- kNN is slow at the classification time
- kNN does not produce an understandable model

k-NN in Python

Fundamental Data Science for Data
Scientist using Python

K-NN



```
In [7]: from sklearn.neighbors import KNeighborsClassifier
```

This time we set a parameter of the `KNeighborsClassifier` to tell it we only want to look at one nearest neighbor:

```
In [8]: knn = KNeighborsClassifier(n_neighbors=3)
```

We fit the model with our training data

```
In [9]: knn.fit(X_train, y_train)
```

```
Out[9]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                           weights='uniform')
```



K-NN



```
In [12]: knn.score(X_test, y_test)
```

```
Out[12]: 0.9555555555555556
```

```
In [13]: print("Predicting iris on the test set using K-NN")
t0 = time()
y_pred = knn.predict(X_test)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))
print(confusion_matrix(y_test, y_pred))
```

```
Predicting iris on the test set using K-NN
done in 0.002s
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.99	0.90	0.90	10
Iris-virginica	0.94	0.94	0.94	17
avg / total	0.96	0.96	0.96	45

```
[[18  0  0]
 [ 0  9  1]
 [ 0  1 16]]
```

Workshop 5 - k-NN

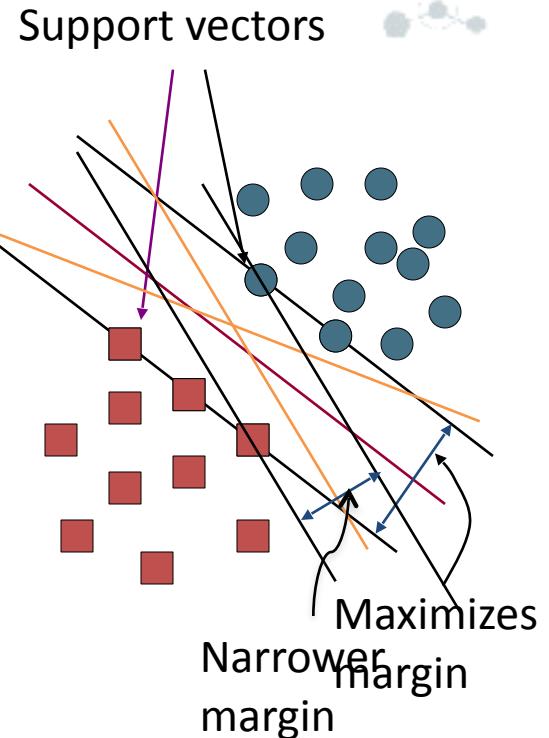
1. Use data from Workshop 2
2. Perform k-NN Algorithm to predict outcomes
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- **Support Vector Machine**
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Support Vector Machine (SVM)

- SVMs maximize the *margin* around the separating hyperplane.
 - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem
- Seen by many as the most successful current text classification method*



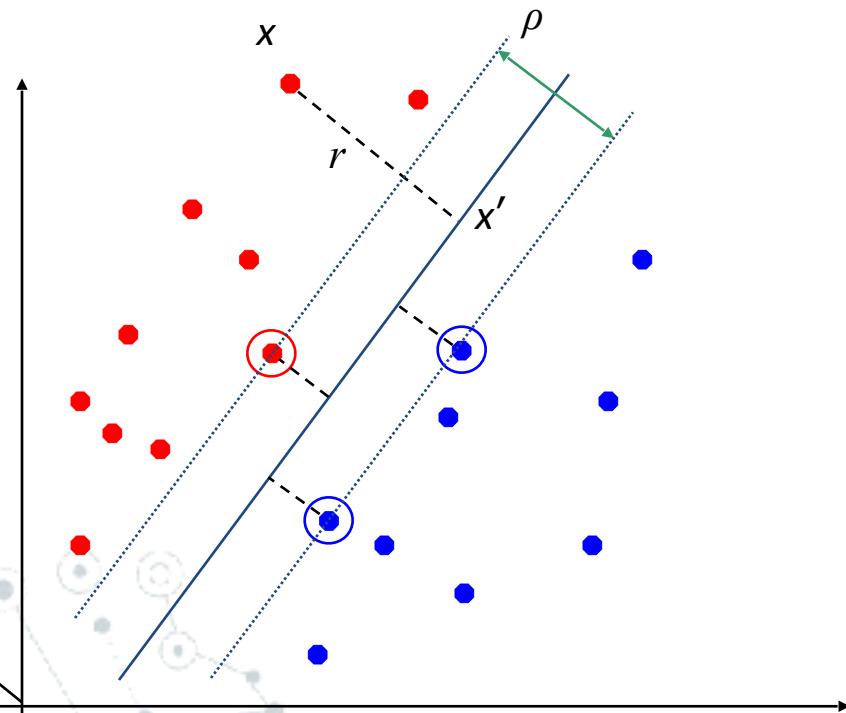
Maximum Margin: Formalization

- ◎ \mathbf{w} : decision hyperplane normal vector
- ◎ \mathbf{x}_i : data point i
- ◎ y_i : class of data point i (+1 or -1) NB: Not 1/0
- ◎ Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T \mathbf{x}_i + b)$
- ◎ Functional margin of \mathbf{x}_i is: $y_i (\mathbf{w}^T \mathbf{x}_i + b)$
 - But note that we can increase this margin simply by scaling \mathbf{w}, b
- ◎ Functional margin of dataset is twice the minimum functional margin for any point
 - The factor of 2 comes from measuring the whole width of the margin

Geometric Margin

- Distance from example to the separator is
- Examples closest to the hyperplane are **support vectors**.
- Margin** ρ of the separator is the width of separation between support vectors of classes.

$$r = \frac{y(\mathbf{w}^T \mathbf{x} + b)}{\|\mathbf{w}\|}$$



Derivation of finding r :
Dotted line $\mathbf{x}' - \mathbf{x}$ is perpendicular to decision boundary so parallel to \mathbf{w} .
Unit vector is $\mathbf{w}/\|\mathbf{w}\|$, so line is $\mathbf{rw}/\|\mathbf{w}\|$.
 $\mathbf{x}' = \mathbf{x} - yr\mathbf{w}/\|\mathbf{w}\|$.
 \mathbf{x}' satisfies $\mathbf{w}^T \mathbf{x}' + b = 0$.
So $\mathbf{w}^T(\mathbf{x} - yr\mathbf{w}/\|\mathbf{w}\|) + b = 0$
Recall that $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T \mathbf{w}}$.
So $\mathbf{w}^T \mathbf{x} - yr\|\mathbf{w}\| + b = 0$
So, solving for r gives:
 $r = y(\mathbf{w}^T \mathbf{x} + b)/\|\mathbf{w}\|$

Linear SVM Mathematically

The linearly separable case

- ◎ Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x}_i, y_i)\}$

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1 \quad \text{if } y_i = 1$$

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1 \quad \text{if } y_i = -1$$

- ◎ For support vectors, the inequality becomes an equality
- ◎ Then, since each example's distance from the hyperplane is

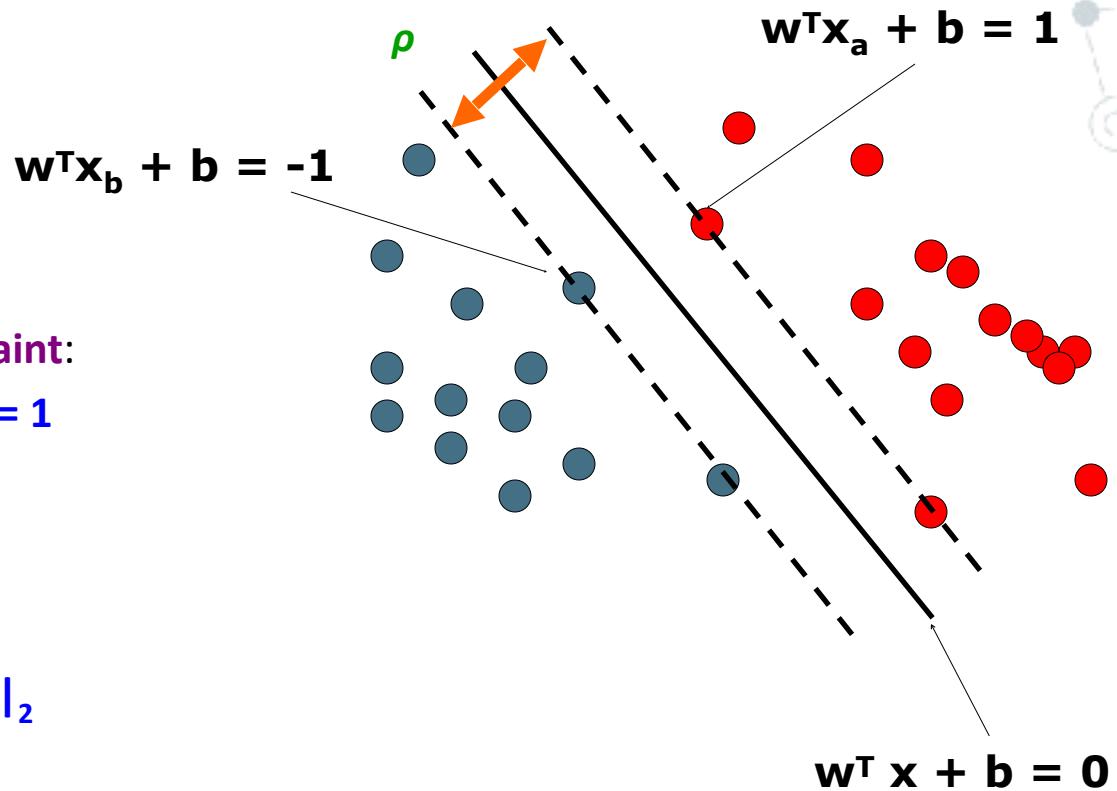
$$r = y \frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|}$$

- ◎ The margin is:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Linear Support Vector Machine (SVM)

- Hyperplane $w^T x + b = 0$
- Extra scale constraint:
 $\min_{i=1,\dots,n} |w^T x_i + b| = 1$
- This implies:
- $w^T(x_a - x_b) = 2$
- $\rho = \|x_a - x_b\|_2 = 2/\|w\|_2$



Support Vector Machine in Python

Fundamental Data Science for Data
Scientist using Python

Support Vector Machine

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$. d is specified by keyword `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$. γ is specified by keyword `gamma`, must be greater than 0.
- sigmoid ($\tanh(\gamma \langle x, x' \rangle + r)$), where r is specified by `coef0`.

```
In [11]: from sklearn.svm import SVC
```

```
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

```
Out[11]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
      max_iter=-1, probability=False, random_state=None, shrinking=True,
      tol=0.001, verbose=False)
```

Support Vector Machine

```
In [14]: print("Predicting iris on the test set using K-NN")
t0 = time()
y_pred = svclassifier.predict(X_test)
print("done in %0.3fs" % (time() - t0))
```

```
print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))
print(confusion_matrix(y_test, y_pred))
```

Predicting iris on the test set using K-NN
done in 0.001s

	precision	recall	f1 score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.83	1.00	0.91	10
Iris-virginica	1.00	0.88	0.94	17
avg / total	0.96	0.96	0.96	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  2 15]]
```

Workshop 6 - SVM

1. Use data from Workshop 2
2. Perform SVM Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

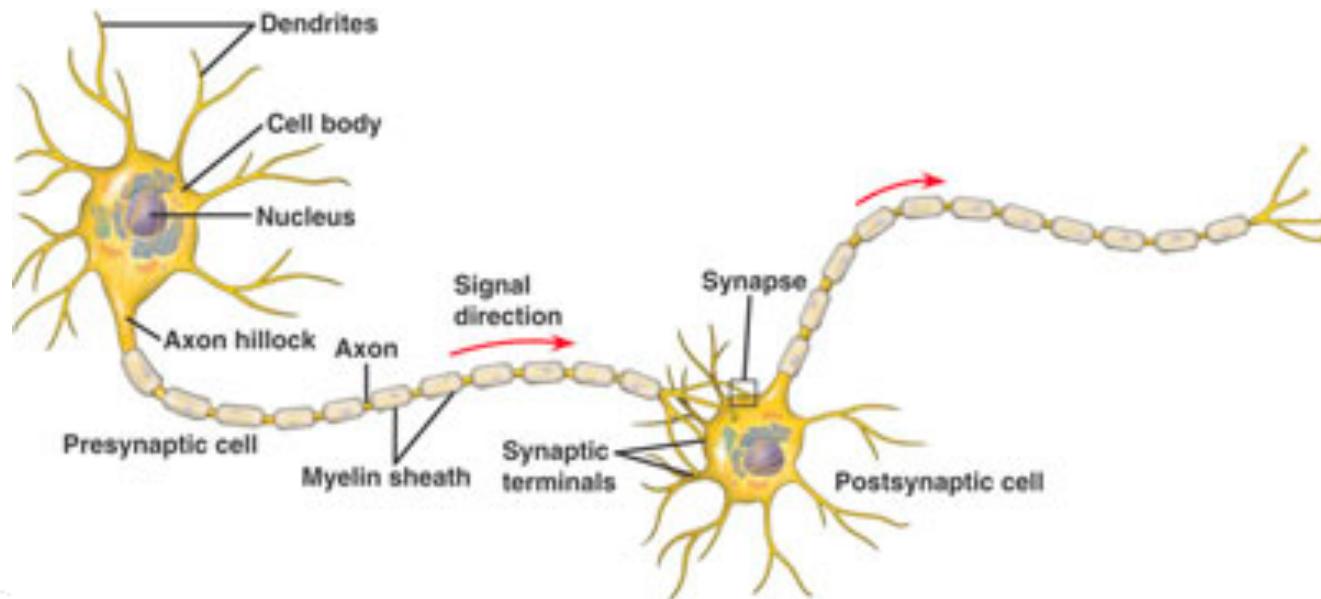
- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- **Neural Net**
- Ensemble methods: Bagging and Boosting
- Summary

Neural Network

- Neural networks learning methods provide a robust approach to approximating real-valued, discrete valued and vector valued target functions.
- Often used in problems such as handwriting, voice, or image recognition
- **Feed-Forward Neural Networks, Convolutional Neural Networks, Deep Learning**

Biological Motivation

- Inspired by biological learning systems that are built from very complex webs of interconnected neurons

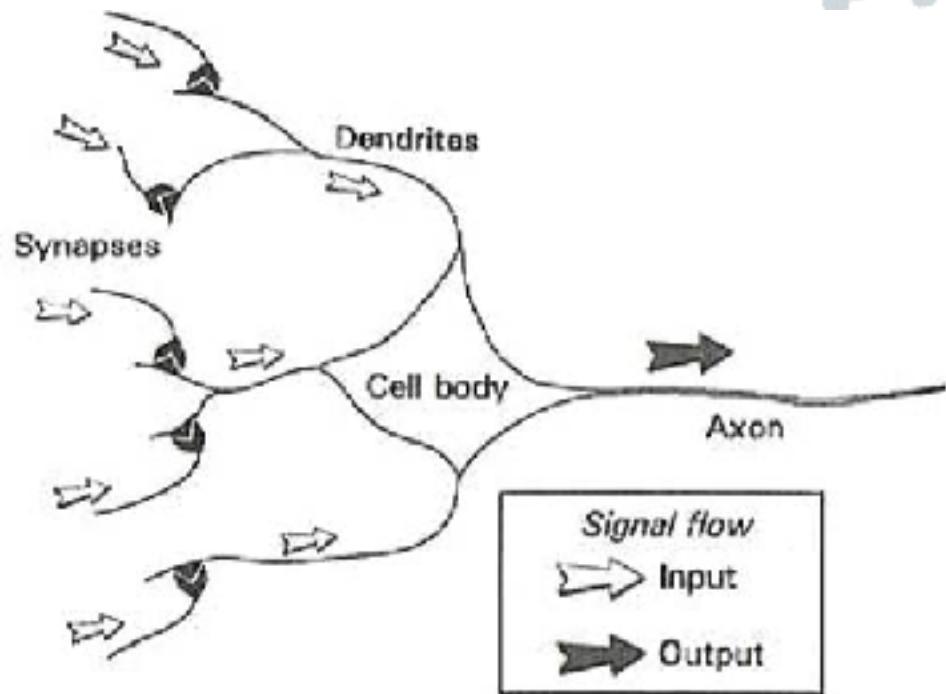


Facts About Neuro-biology

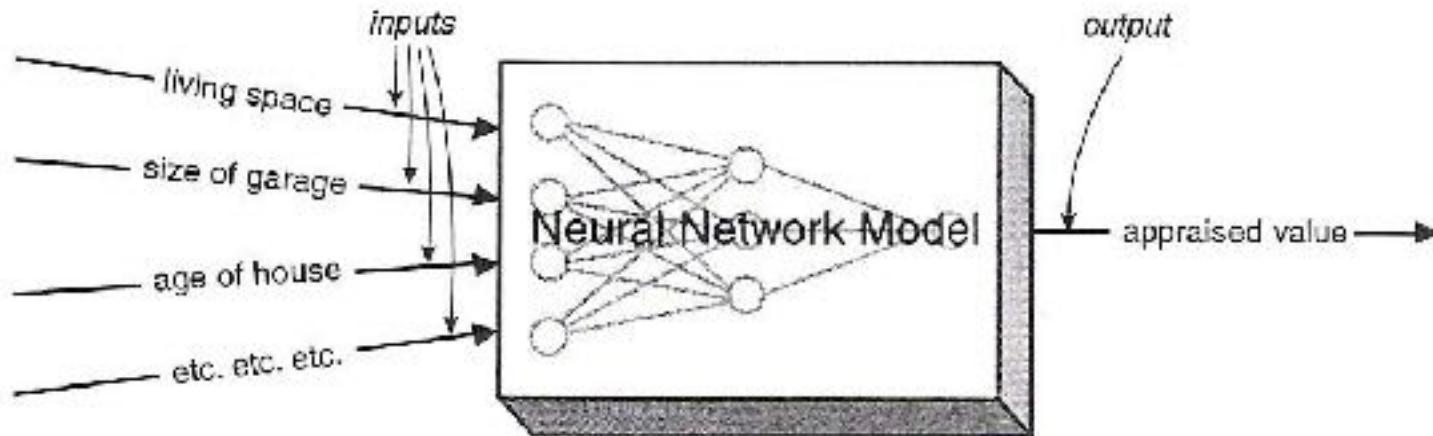
- Human brain is estimated to contain a densely interconnected network of approximately 10^{11} neurons.
- Each connected, on average, to 10^4 other neurons.
- Neuron activity is typically excited through connections to other neurons.
- The fastest neuron switching times are in the order of 10^{-3} seconds.
- It requires 10^{-1} seconds to visibly recognize your mother.

Analogy of Biological Learning Systems

- Artificial neural networks are built out of a interconnected set of simple units, where each unit takes a number of real-valued inputs (can be the outputs of other units) and produces a single real-valued output (which may become the input of other units).



Example

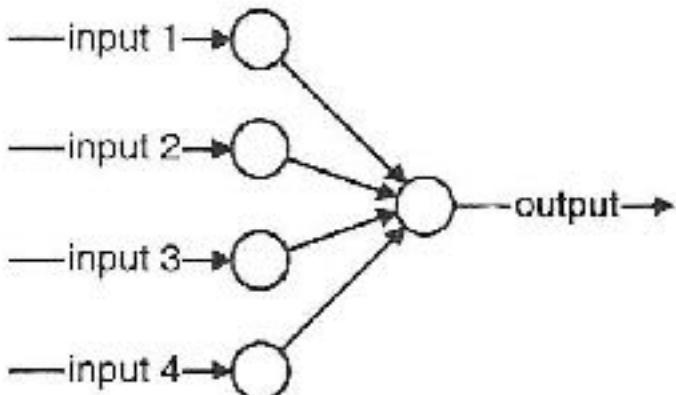


A neural network is like a black box that know how to process input to create an output. The calculation is quite complex and difficult to understand.

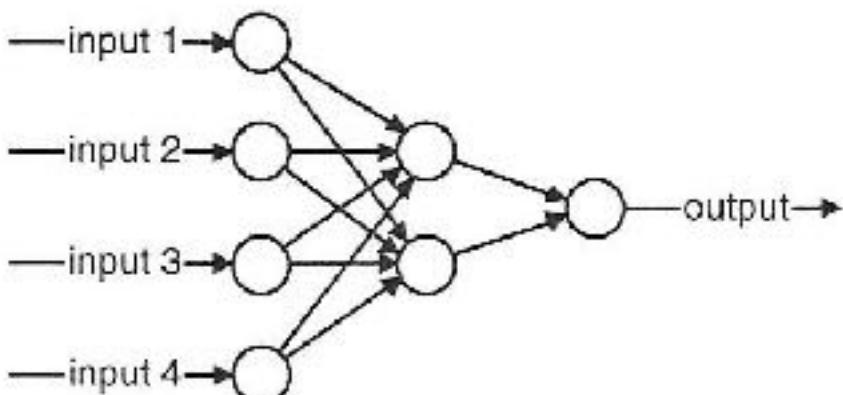
Neural Networks Process

1. Identify the input and output features
2. Transform the input and output so that they are in a small range (-1 to 1)
3. Set up a network with an appropriate topology
4. Train the network on a representative set of training examples
5. Use the validation set to choose the set of weights that minimizes the error
6. Evaluate the network using the test set to see how well it performs
7. Apply the model generated by the network to predict outcomes for unknown input

What is a Neural Network? (1)

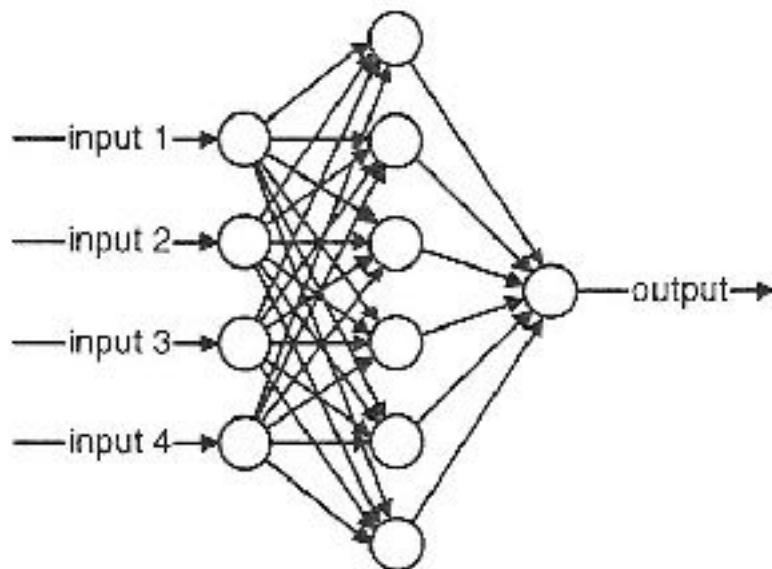


This simple neural network takes four inputs and produces an output. This result of training this network is equivalent to the statistical technique called logistic regression.

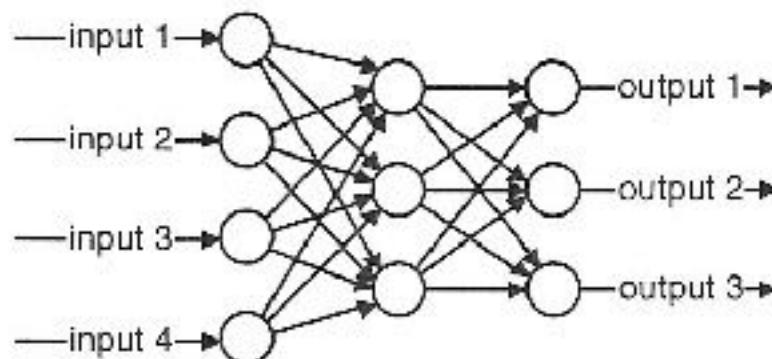


This network has a middle layer called the *hidden layer*, which makes the network more powerful by enabling it to recognize more patterns.

What is a Neural Network? (2)



Increasing the size of the hidden layer makes the network more powerful but introduces the risk of overfitting. Usually, only one hidden layer is needed.



A neural network can produce multiple output values.

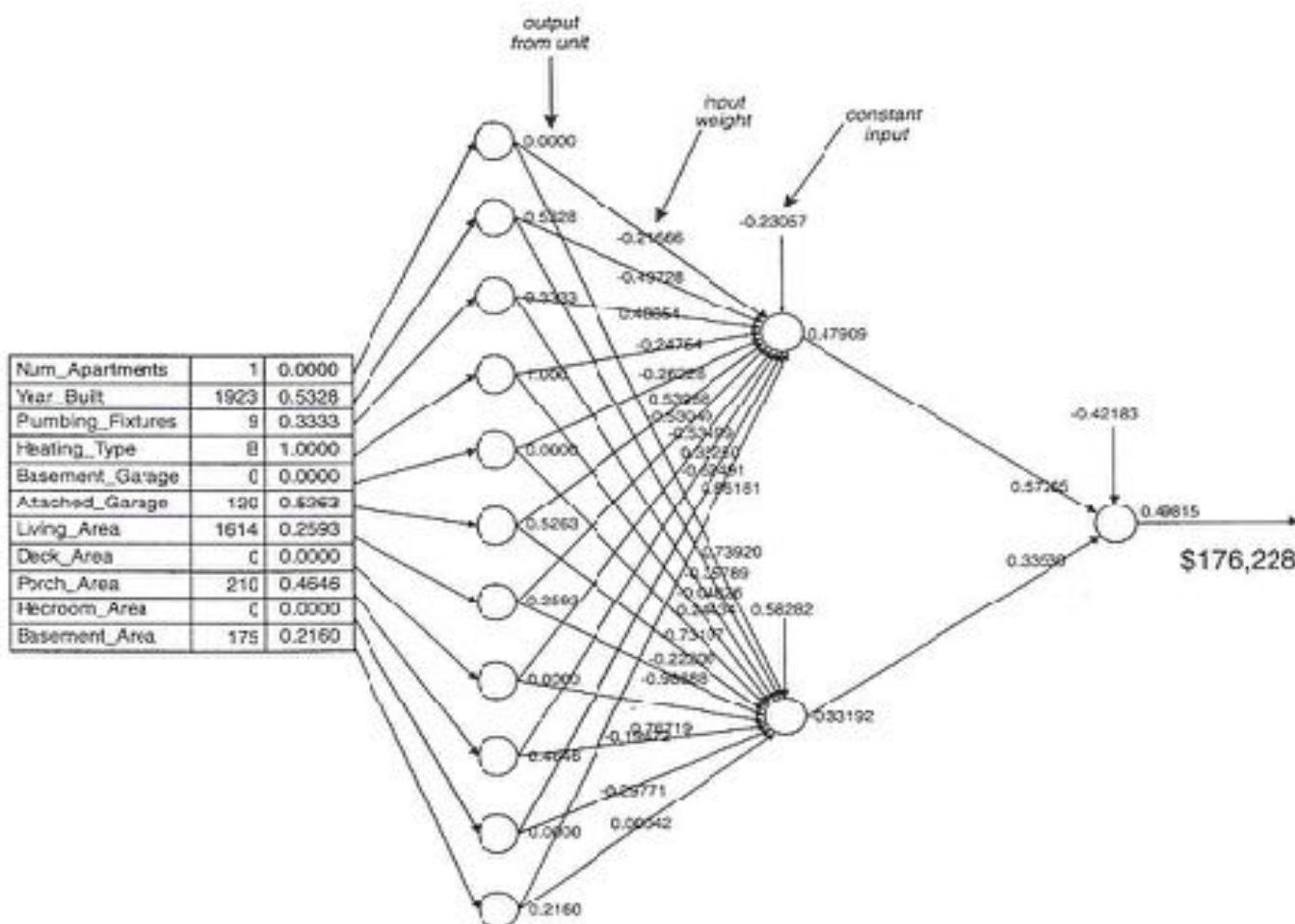
Three Main Components of a Neural Network

- **Activation function**
- **Network topology**
- **How the network is trained?**
 - Feed forward
 - Back propagation

Activation Function (1)

- **Combination function**
 - The function that merges all the input into a single value.
 - The most common combination function is the weighted sum.
- **Transfer function**
 - The function that transfers the value of the combination function to the output of the unit.

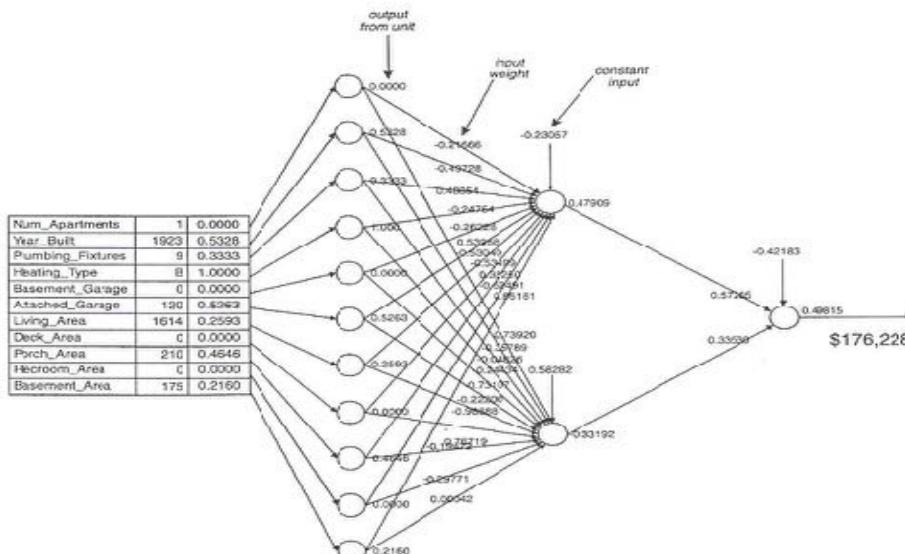
Feed-Forward Neural Networks (1)



Feed-Forward Neural Networks: Input Layer

- **Input layer**

- Each unit in the input layer is connected to exactly one source field.



Feed-Forward Neural Networks : Hidden layer (1)

- **Hidden layer**

- Each unit in the hidden layer is typically connected to all the units in the input layer.
- The units in the hidden layer calculate their output by multiplying the value of each input by its corresponding weight, adding them up, and applying the transfer function.

Feed-Forward Neural Networks: Hidden layer (2)

- **Hidden layer**

- A neural network can have any number of hidden layers, but in general **one** hidden layer is sufficient.
- The wider the layer (the more units it contains), the greater the capacity of the network to recognize patterns.

Feed-Forward Neural Networks : Output Layer (1)

- **Output layer**

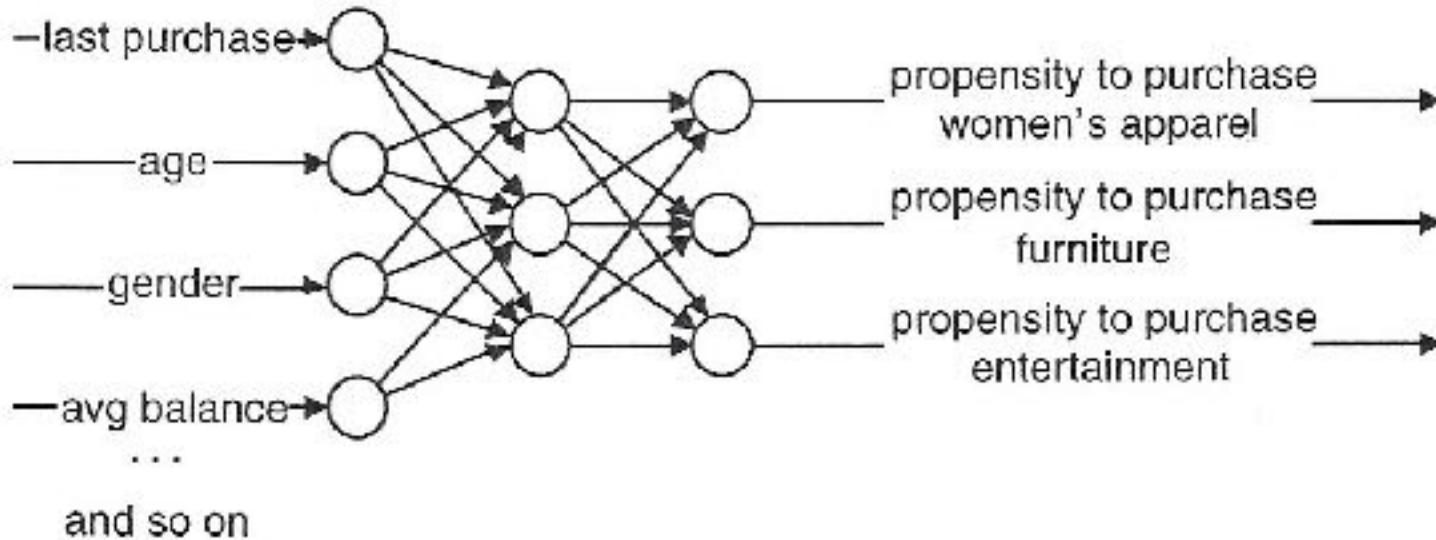
- It is fully connected to all the units in the hidden layer.
- Most of the time, the neural network is being used to calculate a single value, so there is only one unit in the output layer and the value.
- **We must map this value back to understand the output.**

Feed-Forward Neural Networks: Output Layer (2)

- **Output layer**

- It is possible for the output layer to have more than one unit.
- Eg. A department store chain wants to predict the likelihood that customers will be purchasing products from various departments, such as women's apparel, furniture, and entertainments.

Feed-Forward Neural Networks: Output Layer (3)



- After feeding the inputs for a customer into the network, the network calculates three values.

Back-Propagation Algorithm

- Assume the network is a fixed structure that corresponds to a directed graph.
- Learning corresponds to choosing a weight value for each edge in the graph.
- It attempts to minimize the squared error between the network output values and the target values for these outputs.
- It is the ANN learning technique that is most commonly used.

Back Propagation Neural Networks: Steps (1)

1. The network gets a training example and, using the existing weights in the network, it calculates the output.
2. Back propagation calculates the error by taking the difference between the calculated result and the expected (actual result).

Back Propagation Neural Networks: Steps (2)

3. The error is fed back through the network and the weights are adjusted to minimize the error.
4. After being shown enough training examples, the weights on the network no longer change significantly and the error no longer decreases. This is the point where training stops.

Back-Propagation Algorithm: Input

- Training example is a pair of the form $\langle x, t \rangle$, where x is the vector of network input values and t is the vector of target network output values
 - n is the learning rate
 - n_{in} is the number of network inputs
 - n_{hidden} is the number of units in the hidden layer
 - n_{out} is the number of output units
- The input from unit i into unit j is denoted x_{ji}
- The weight from unit i to unit j is denoted w_{ji}

Back-Propagation Algorithm: process (1)

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units
- Initialize all network weight to small random numbers
- Until the termination condition is met, Do repeat the steps

Back-Propagation Algorithm: process (2)

Propagate the input forward through the network:

1. Input the instance x to the network and compute the output o_u of every unit u in the network

Propagate the errors backward through the network:

2. For each network output unit k , calculate its error term e_k

$$e_k \leftarrow o_k (1 - o_k) (t_k - o_k)$$

Back-Propagation Algorithm: process

3. For each hidden unit h , calculate its error term e_h

$$e_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} e_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = n e_j x_{ji}$$

Termination conditions

- Stop after a fixed number of iterations
- Stop when the error on the training examples falls below some threshold
- Stop when the error on a separate validation set of examples meets some criterion

Hidden Layer Representations

- Training examples constrain the network inputs and outputs.
- The weight tuning procedure is free to set weights that define whatever hidden unit representation is most effective in minimizing the squared error.
- The ability of multilayer networks to automatically discover useful representation **at the hidden layers** is a key feature of ANN learning.

Heuristics for Using Feed-Forward & Back Propagation Networks

- One important decision is the **number of units in the hidden layer**. The more unit, the more patterns the network can recognize.
- We generally do not use hidden layers larger than the number of inputs.
- A good place to start for many problems is to experiment with one, two, and three nodes in the hidden layer.

Choosing the Training Set: Coverage of values for all Features

- In general, it is good to have several examples in the training set for each value of a categorical feature and for values throughout the ranges of ordered discrete and continuous features.

Choosing the Training Set: Number of Features

- The more features used as inputs into the networks, the larger the networks needs to be, increasing the risk of overfitting and increasing the size of the training set.
- The more features, the longer it takes the network to converge to a set of weights.
- In many cases, it is useful to calculate new variables that represent particular aspects of the business problems.

Choosing the Training Set: Size of Training Set

- The more features there are, the more training examples that are needed to get a good coverage of patterns in the data.
- Overfitting is guaranteed to happen when there are fewer training examples than there are weights in the network.

Choosing the Training Set: Number of Output

- The number of training examples for each possible output should be about the same.

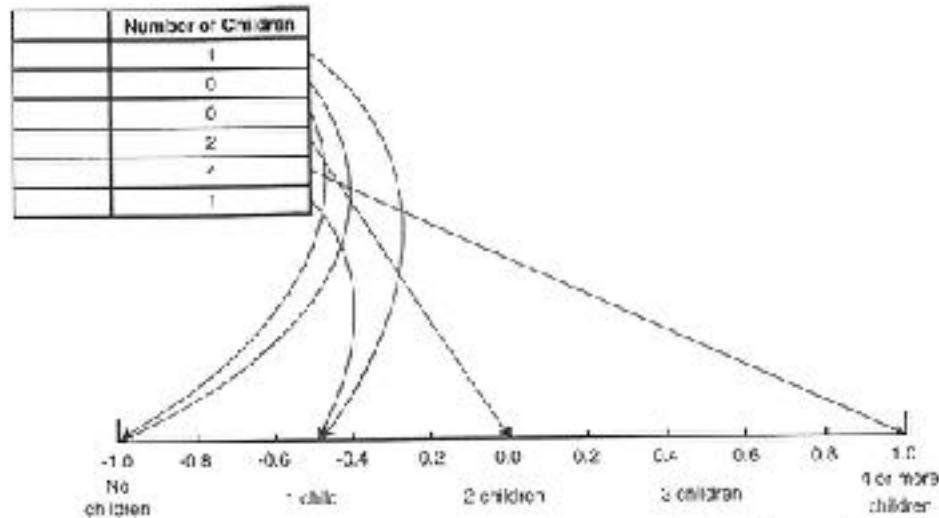
Preparing the Data: Features with Continuous Values

- The value should be scaled to be in a reasonable range.
- Plan for a larger range
- Reject out-of-range values
- Peg values lower than the minimum to the minimum and higher than the maximum to the maximum
- Map the minimum value to -0.9 and the maximum value to 0.9 instead of -1 and 1

Preparing the Data: Features with Ordered Discrete Values (1)

- First, count the number of different values and assign each a proportional fraction into some range.
- eg. If there are five distinct values, these get mapped to 0, 0.25, 0.50, 0.75, and 1.0

$$\begin{array}{rcl} 0 & \rightarrow & 0000 = 0/16 = 0.0000 \\ 1 & \rightarrow & 1000 = 8/16 = 0.5000 \\ 2 & \rightarrow & 1100 = 12/16 = 0.7500 \\ 3 & \rightarrow & 1110 = 14/16 = 0.8750 \end{array}$$



Preparing the Data: Features with Ordered Discrete Values (2)

- It is also possible to break a range into unequal parts.
- One example is called thermometer codes

$$\text{\# } 0 \rightarrow 0\ 0\ 0\ 0 = 0 / 16 = 0.0000$$

$$\text{\# } 1 \rightarrow 1\ 0\ 0\ 0 = 8 / 16 = 0.5000$$

$$\text{\# } 2 \rightarrow 1\ 1\ 0\ 0 = 12 / 16 = 0.7500$$

$$\text{\# } 3 \rightarrow 1\ 1\ 1\ 0 = 14 / 16 = 0.8750$$

It shows that the difference on one end of the scale is less significant than differences on the other end.

Preparing the Data: Features with Categorical Values (1)

- When working with categorical variables in neural networks, the mapping of variables into numbers **introduces an ordering** of the variables, which the neural network takes into account.
- The second way of handling categorical features is to break the categories into flags, one for each value.

Preparing the Data: Features with Categorical Values (2)

Gender	N coding			N-1 coding	
	Gender Male Flag	Gender Female Flag	Gender Unknown Flag	Gender Male Flag	Gender Female Flag
Male	+1.0	-1.0	-1.0	+1.0	-1.0
Female	-1.0	+1.0	-1.0	-1.0	+1.0
Unknown	-1.0	-1.0	+1.0	-1.0	-1.0

Interpreting the Results (1)

- When estimating a continuous value, often the output needs to be scaled back to the correct range.
- For binary or categorical output variables, the approach is still to take the inverse of the translation used for training the network.
- eg. If “churn” is given the value of 1 and “no churn” a value of -1 , the values near 1 represent churn and those near -1 represent no churn.

Interpreting the Results (2)

- When there are two outcomes, the meaning of the output depends on the training set used to train the network.
- The average value produced by the network during training is usually going to be close to the average value in the training set.

Interpreting the Results (3)

- One way to handle is, eg.
- If the training set had 50% churn and 50% no churn, the average value the network will produce from the training examples is going to be close to 0.0.
- Values higher than 0.0 are more like churn and those less than 0.0, less like churn.
- If the original training set had 10% churn, then the cutoff would more reasonable be –
- 0.8 rather than 0.0 (0.8 is 10% of the way from -1 to 1).

Interpreting the Results (4)

- For binary values, it is also possible to create a network that produces two output, one for each value.
- In this case, each output represents the strength of evidence that that category is the correct one. The chosen category would then be the one with the higher value.

How to Know What is Going on Inside a Neural Network

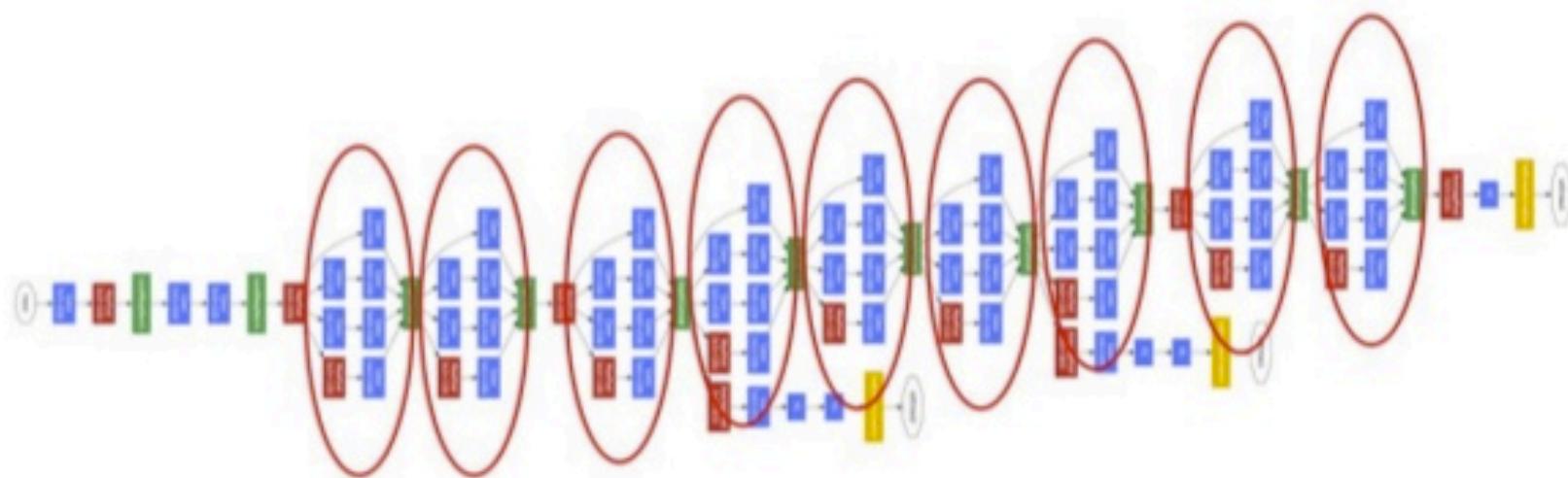
Sensitivity analysis uses the test set to determine how sensitive the output of the network is to each input :

1. Find the average value for each input.
2. Measure the output of the network when all inputs are at their average value.
3. Measure the output of the network when each input is modified, one at a time, to be at **its minimum and maximum values**.

Summary

- Neural networks can be used for both categorical and continuous inputs.
- Neural networks learn best when input fields have been mapped to the range between -1 and +1.
- Neural networks work best when there are only a few variables.
- When training a network, there is no guarantee that the resulting set of weights is optimal.
- A neural network cannot explain what it is doing.

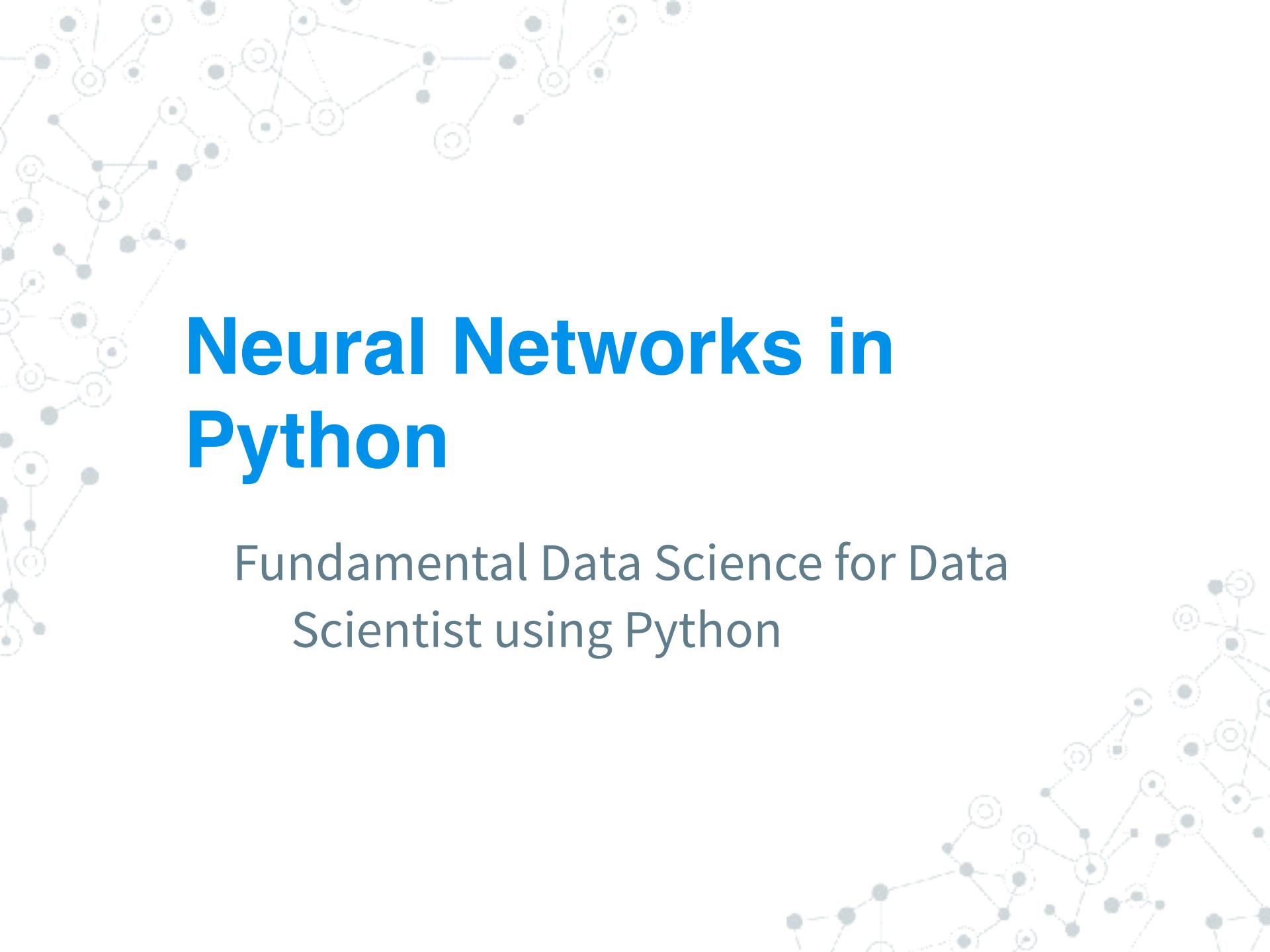
GoogLeNet (Inception)



9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other



Neural Networks in Python

Fundamental Data Science for Data
Scientist using Python

Neural Network

```
In [17]: iris.loc[iris['species']=='Iris-setosa', 'species']=0  
iris.loc[iris['species']=='Iris-versicolor', 'species']=1  
iris.loc[iris['species']=='Iris-virginica', 'species']=2  
iris = iris.apply(pd.to_numeric)  
  
# Change dataframe to array  
iris_array = iris.as_matrix()
```

Neural Network

```
import numpy as np  
  
In [19]: from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(iris.iloc[:, 0:4],  
                                                iris['species'],  
                                                train_size=0.7,  
                                                random_state=123)  
print("Labels for training and testing data")  
print(y_train)
```

Labels for training and testing data

```
114    2  
136    2  
53     1  
19     0  
38     0  
110    2  
23     0  
9      0
```

Neural Network

Scaling Data first

```
In [20]: from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
# Fit only to the training data  
scaler.fit(X_train)  
StandardScaler(copy=True, with_mean=True, with_std=True)  
# Now apply the transformations to the data:  
X_train = scaler.transform(X_train)  
X_test = scaler.transform(X_test)
```

```
In [22]: from sklearn.neural_network import MLPClassifier  
  
mlp = MLPClassifier(hidden_layer_sizes=(10),solver='sgd',learning_rate_init=0.01,max_iter=500)  
mlp.fit(X_train,y_train)  
  
Out[22]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,  
beta_2=0.999, early_stopping=False, epsilon=1e-08,  
hidden_layer_sizes=10, learning_rate='constant',  
learning_rate_init=0.01, max_iter=500, momentum=0.9,  
nesterovs_momentum=True, power_t=0.5, random_state=None,  
shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,  
verbose=False, warm_start=False)
```

```
In [24]: print("Predicting iris on the test set using K-NN")
t0 = time()
y_pred = mlp.predict(X_test)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Predicting iris on the test set using K-NN

done in 0.001s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	18
1	0.91	1.00	0.95	10
2	1.00	0.94	0.97	17
avg / total	0.98	0.98	0.98	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  1 16]]
```

Workshop 7 - Neural Networks

1. Use data from Workshop 2
2. Build Neural Networks Model to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- **Ensemble methods: Bagging and Boosting**
- Summary

Combining classifiers

- So far, we have only discussed individual classifiers, i.e., how to build them and use them.
- Can we combine multiple classifiers to produce a better classifier?
- Yes, sometimes
- We discuss two main algorithms:
 - **Bagging**
 - **Boosting**

Bagging

- Breiman, 1996
- Bootstrap Aggregating = Bagging
 - Application of **bootstrap sampling**
 - **Given:** set D containing m training examples
 - Create a sample $S[i]$ of D by drawing m examples at random with replacement from D
 - $S[i]$ of size m: expected to leave out 0.37 of examples from D

Bagging (cont...)

- Training
 - Create k bootstrap samples $S[1], S[2], \dots, S[k]$
 - Build a distinct classifier on each $S[i]$ to produce k classifiers, using the same learning algorithm.
- Testing
 - Classify each new instance by voting of the k classifiers (equal weights)

Bagging Example

Original	1	2	3	4	5	6	7	8
Training set 1	2	7	8	3	7	6	3	1
Training set 2	7	8	5	6	4	2	7	1
Training set 3	3	6	2	7	5	6	2	2
Training set 4	4	5	1	4	6	4	3	8

Bagging (cont ...)

- When does it help?
 - When learner is unstable
 - Small change to training set causes large change in the output classifier
 - True for decision trees, neural networks; not true for k -nearest neighbor, naïve Bayesian, class association rules
 - Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

Bagging Predictors, Leo Breiman, 1996

Boosting

- A family of methods:
 - We only study **AdaBoost** (Freund & Schapire, 1996)
- **Training**
 - Produce a sequence of classifiers (the same base learner)
 - Each classifier is dependent on the previous one, and focuses on the previous one's errors
 - Examples that are incorrectly predicted in previous classifiers are given higher weights
- **Testing**
 - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

AdaBoost

Weighted training set

(x_1, y_1, w_1)

(x_2, y_2, w_2)

...

(x_n, y_n, w_n)

Non-negative weights
sum to 1

Change weights



called a weaker classifier

- Build a classifier h_t whose accuracy on training set $> \frac{1}{2}$ (better than random)

AdaBoost algorithm

Algorithm AdaBoost.M1

Input: sequence of m examples $((x_1, y_1), \dots, (x_m, y_m))$
with labels $y_i \in Y = \{1, \dots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i .

Do for $t = 1, 2, \dots, T$:

1. Call **WeakLearn**, providing it with the distribution D_t .

2. Get back a hypothesis $h_t : X \rightarrow Y$.

3. Calculate the error of h_t : $\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$.

If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.

5. Update distribution D_{t+1} :

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$h_{\bar{f}(x)}(x) = \arg \max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}.$$

Does AdaBoost always work?

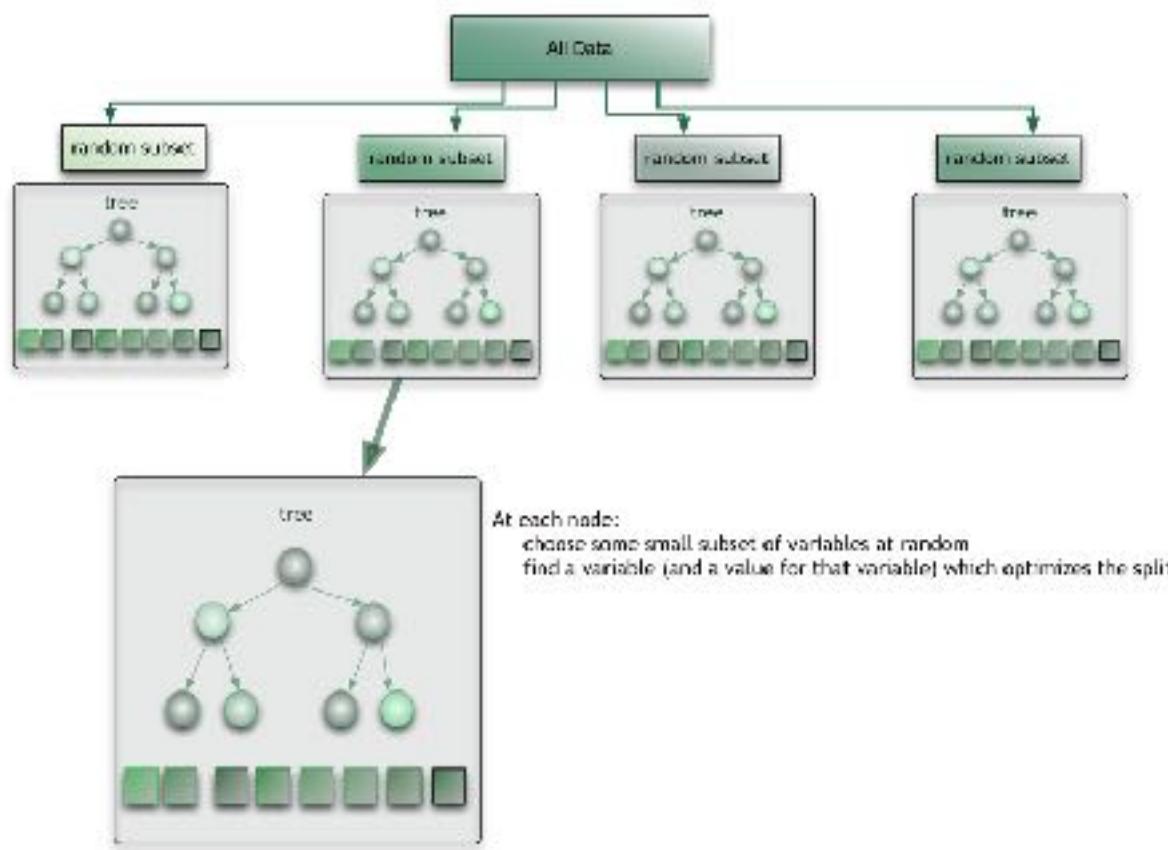
- The actual performance of boosting depends on the data and the base learner.
 - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
 - When the number of outliers is very large, the emphasis placed on the hard examples can hurt the performance.

Ensemble in Python

Fundamental Data Science for Data
Scientist using Python

Ensemble : Bagging

◎ Random Forest



Random Forest

- ◎ Here is how such a system is trained; for some number of trees T:
- ◎ 1) Sample N cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
- ◎ 2) At each node:
 - a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.
 - b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
 - c) At the next node, choose another m variables at random from all predictor variables and do the same.

Bagging



In [75]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Create a random forest Classifier. By convention, clf means 'Classifier'
clf = RandomForestClassifier(n_estimators=100, oob_score=True, random_state=123456)

# Train the Classifier to take the training features and learn how they relate
# to the training y (the species)
clf.fit(X_train, y_train)
```

Out[75]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                      oob_score=True, random_state=123456, verbose=0,
                      warm_start=False)
```



```
In [77]: print("Predicting iris on the test set")
t0 = time()
y_pred = clf.predict(X_test)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))
print(confusion_matrix(y_test, y_pred))
```

Predicting iris on the test set

done in 0.021s

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.77	1.00	0.87	10
Iris-virginica	1.00	0.82	0.90	17
avg / total	0.95	0.93	0.93	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  3 14]]
```

Boosting

```
In [81]: from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(n_estimators=100, random_state=1234)

clf.fit(X_train, y_train)
scores = cross_val_score(clf, X_train, y_train)
scores.mean()
```

Out[81]: 0.8951758481170246

```
In [82]: print("Predicting iris on the test set")
t0 = time()
y_pred = clf.predict(X_test)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=iris['species'].unique()))
print(confusion_matrix(y_test, y_pred))
```

Predicting iris on the test set

done in 0.020s

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.67	1.00	0.89	10
Iris-virginica	1.00	0.71	0.83	17
avg / total	0.93	0.89	0.89	45

```
[[18  0  0]
 [ 0 10  0]
 [ 0  5 12]]
```

Workshop 8 - Ensemble Method

1. Use data from Workshop 2
2. Train your model using either Bagging (Random Forest) or Boosting (AdaBoost) to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Machine Learning Summary

- Applications of supervised learning are in almost any field or domain.
- We studied 8 classification techniques.
- There are still many other methods, e.g.,
 - Bayesian networks
 - Neural networks
 - Genetic algorithms
 - Fuzzy classification
- This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.

Unsupervised Learning

Fundamental Data Science for Data
Scientist using Python

Supervised learning vs. unsupervised learning

- **Supervised learning:** discover patterns in the data that relate data attributes with a target (class) attribute.
 - These patterns are then utilized to predict the values of the target attribute in future data instances.
- **Unsupervised learning:** The data have no target attribute.
 - We want to explore the data to find some intrinsic structures in them.

Clustering

- Clustering is a technique for finding **similarity groups** in data, called **clusters**. i.e.,
 - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.
- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.
- Due to historical reasons, clustering is often considered synonymous with unsupervised learning.
 - In fact, association rule mining is also unsupervised
- This chapter focuses on clustering.

An illustration

- The data set has three natural groups of data points, i.e., 3 natural clusters.



What is clustering for?

- Let us see some real-life examples
- Example 1: groups people of similar sizes together to make “small”, “medium” and “large” T-Shirts.
 - Tailor-made for each person: too expensive
 - One-size-fits-all: does not fit all.
- Example 2: In marketing, segment customers according to their similarities
 - To do targeted marketing.

What is clustering for? (cont...)

- Example 3: Given a collection of text documents, we want to organize them according to their content similarities,
 - To produce a topic hierarchy
- In fact, clustering is one of the most utilized data mining techniques.
 - It has a long history, and used in almost every field, e.g., medicine, psychology, botany, sociology, biology, archeology, marketing, insurance, libraries, etc.
 - In recent years, due to the rapid increase of online documents, text clustering becomes important.

Aspects of clustering

- A clustering algorithm
 - Partitional clustering
 - Hierarchical clustering
 - ...
- A distance (similarity, or dissimilarity) function
- Clustering quality
 - Inter-clusters distance \Rightarrow maximized
 - Intra-clusters distance \Rightarrow minimized
- The **quality** of a clustering result depends on the algorithm, the distance function, and the application.

K-means clustering

- ◎ K-means is a **partitional clustering** algorithm
- ◎ Let the set of data points (or instances) D be
 - ◎ $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$,
 - where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ir})$ is a **vector** in a real-valued space $X \subseteq \mathbb{R}^r$, and r is the number of attributes (dimensions) in the data.
- ◎ The k -means algorithm partitions the given data into k clusters.
 - Each cluster has a cluster **center**, called **centroid**.
 - k is specified by the user

K-means algorithm

- Given k , the *k-means* algorithm works as follows:
 - 1) Randomly choose k data points (**seeds**) to be the initial **centroids**, cluster centers
 - 2) Assign each data point to the closest **centroid**
 - 3) Re-compute the **centroids** using the current cluster memberships.
 - 4) If a convergence criterion is not met, go to 2).

K-means algorithm – (cont ...)

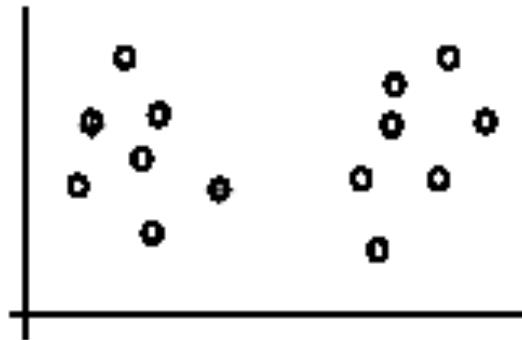
```
Algorithm k-means( $k$ ,  $D$ )
1   Choose  $k$  data points as the initial centroids (cluster centers)
2   repeat
3       for each data point  $\mathbf{x} \in D$  do
4           compute the distance from  $\mathbf{x}$  to each centroid;
5           assign  $\mathbf{x}$  to the closest centroid      // a centroid represents a cluster
6   endfor
7   re-compute the centroids using the current cluster memberships
8   until the stopping criterion is met
```

Stopping/convergence criterion

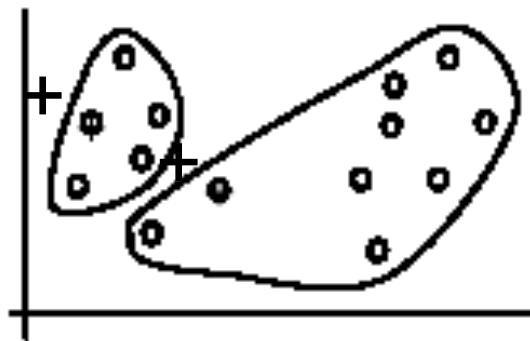
1. no (or minimum) re-assignments of data points to different clusters,
2. no (or minimum) change of centroids, or
3. minimum decrease in the **sum of squared error** (SSE),
 - C_i is the j th cluster, \mathbf{m}_j is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point \mathbf{x} and centroid \mathbf{m}_j .

$$SSE = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \quad (1)$$

An example



(A). Random selection of k centers

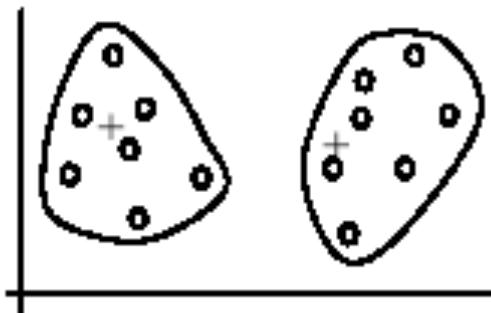


Iteration 1: (B). Cluster assignment

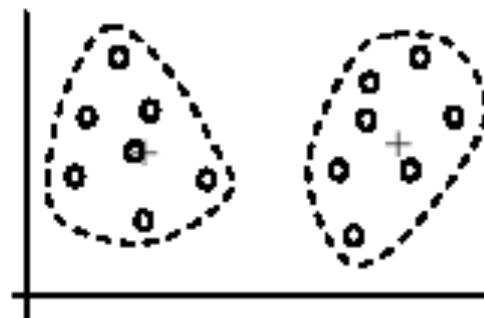


(C). Re-compute centroids

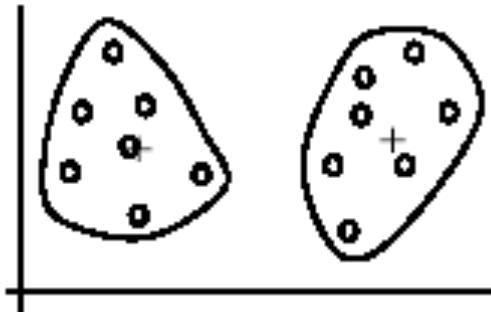
An example (cont ...)



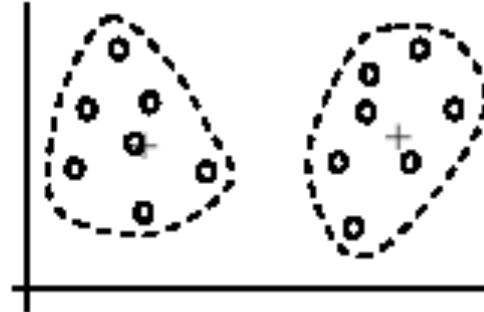
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

An example distance function

The k -means algorithm can be used for any application data set where the **mean** can be defined and computed. In the **Euclidean space**, the mean of a cluster is computed with:

$$\mathbf{m}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \quad (2)$$

where $|C_j|$ is the number of data points in cluster C_j . The distance from one data point \mathbf{x}_i to a mean (centroid) \mathbf{m}_j is computed with

$$\begin{aligned} dist(\mathbf{x}_i, \mathbf{m}_j) &= \| \mathbf{x}_i - \mathbf{m}_j \| \\ &= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \dots + (x_{ir} - m_{jr})^2} \end{aligned} \quad (3)$$

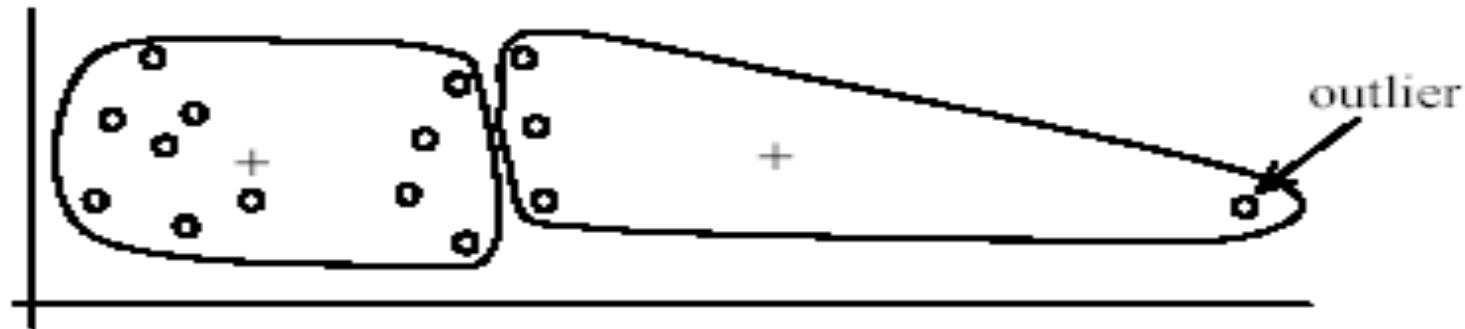
Strengths of k-means

- **Strengths:**
 - **Simple:** easy to understand and to implement
 - **Efficient:** Time complexity: $O(tkn)$,
 - where n is the number of data points,
 - k is the number of clusters, and
 - t is the number of iterations.
 - Since both k and t are small. k -means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a **local optimum** if SSE is used. The **global optimum** is hard to find due to complexity.

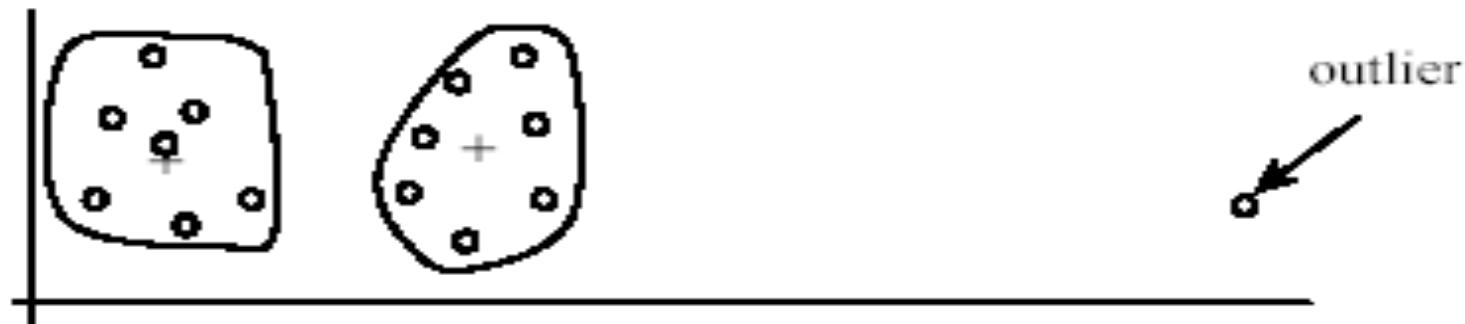
Weaknesses of k-means

- The algorithm is only applicable if the **mean** is defined.
 - For categorical data, **k-mode** - the centroid is represented by most frequent values.
- The user needs to specify **k**.
- The algorithm is sensitive to **outliers**
 - Outliers are data points that are very far away from other data points.
 - Outliers could be errors in the data recording or some special data points with very different values.

Weaknesses of k-means: Problems with outliers



(A): Undesirable clusters

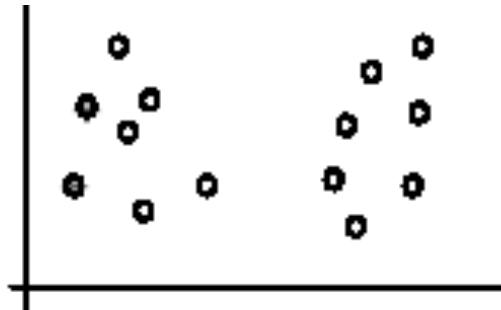


(B): Ideal clusters

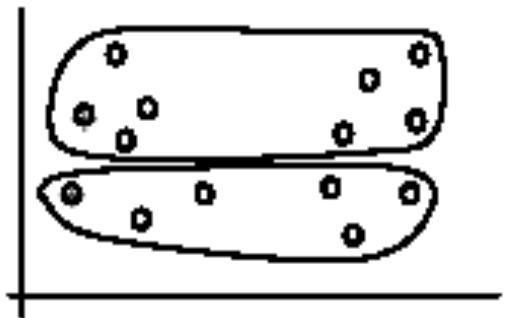
Weaknesses of k-means: To deal with outliers

- **One method** is to **remove some data points** in the clustering process that are much further away from the centroids than other data points.
 - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.
- **Another method** is to **perform random sampling**. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.
 - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

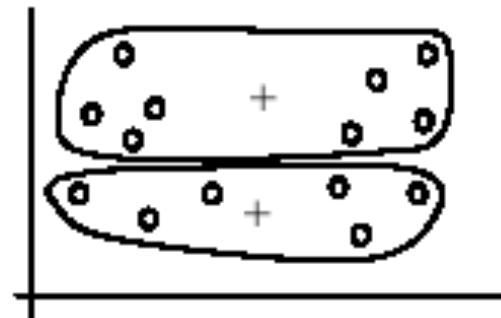
Weaknesses of k-means (cont ...)



(A). Random selection of seeds (centroids)



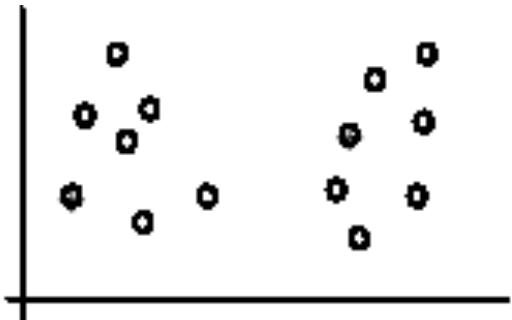
(B). Iteration 1



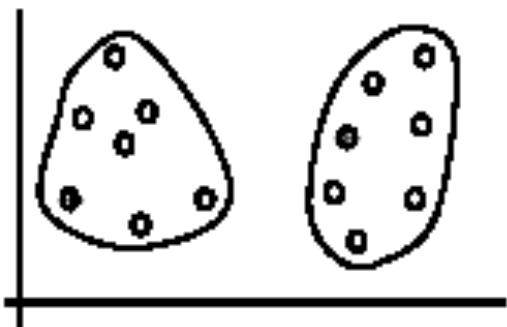
(C). Iteration 2

Weaknesses of k-means (cont ...)

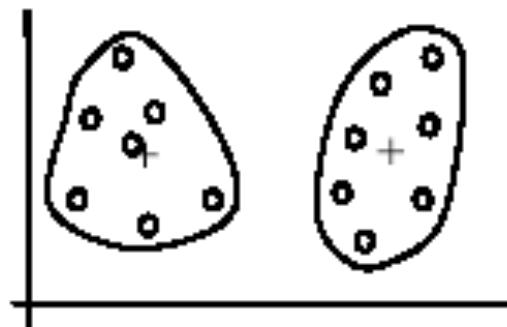
- There are some methods to help choose good seeds



(A). Random selection of k seeds (centroids)



(B). Iteration 1



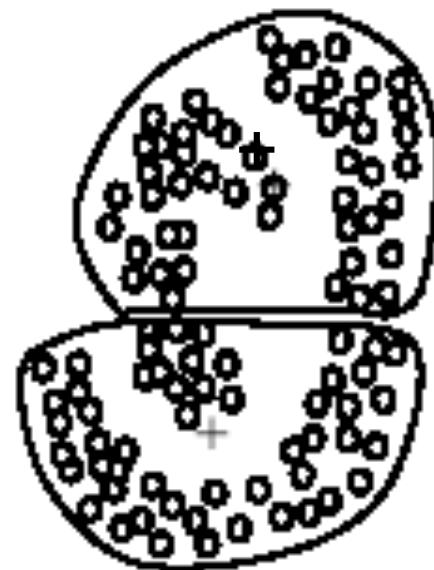
(C). Iteration 2

Weaknesses of k-means (cont ...)

- The k -means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).



(A): Two natural clusters



(B): k -means clusters

K-means summary

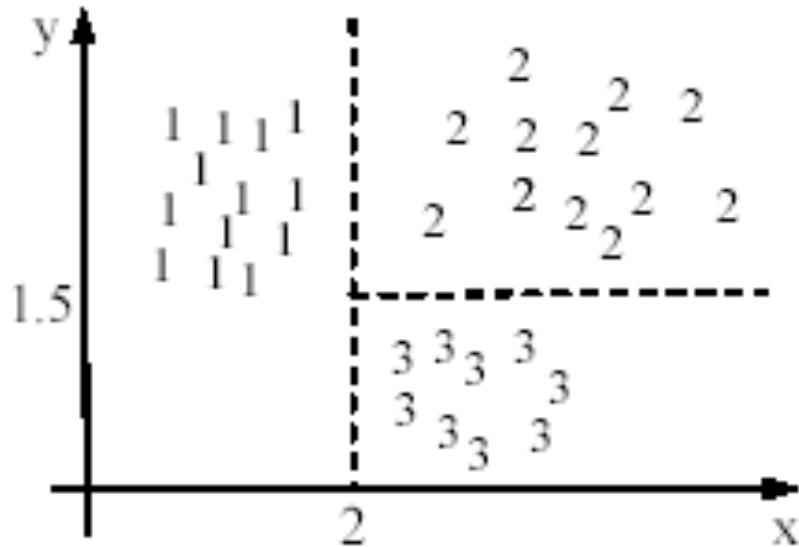
- Despite weaknesses, k-means is still the most popular algorithm due to its simplicity, efficiency and
 - other clustering algorithms have their own lists of weaknesses.
- No clear evidence that any other clustering algorithm performs better in general
 - although they may be more suitable for some specific types of data or applications.
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Common ways to represent clusters

- Use the centroid of each cluster to represent the cluster.
 - compute the radius and
 - standard deviation of the cluster to determine its spread in each dimension
- The centroid representation alone works well if the clusters are of the hyper-spherical shape.
- If clusters are elongated or are of other shapes, centroids are not sufficient

Using classification model

- All the data points in a cluster have the same class label, e.g., they
 - run a supervised learning classification model.



$x \leq 2 \rightarrow$ cluster 1

$x > 2, y > 1.5 \rightarrow$ cluster 2

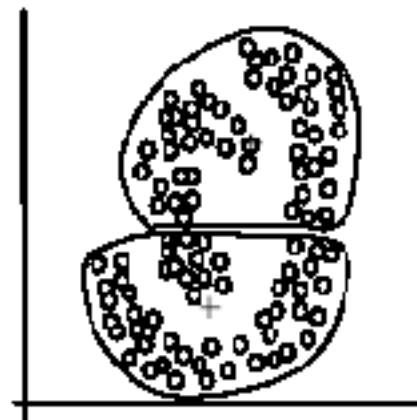
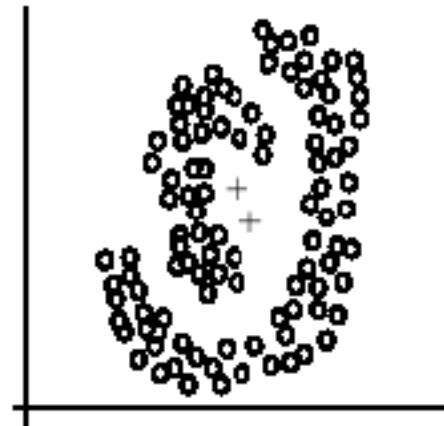
$x > 2, y \leq 1.5 \rightarrow$ cluster 3

Use frequent values to represent cluster

- This method is mainly for clustering of categorical data (e.g., k -modes clustering).
- Main method used in text clustering, where a small set of frequent words in each cluster is selected to represent the cluster.

Clusters of arbitrary shapes

- Hyper-elliptical and hyper-spherical clusters are usually easy to represent, using their centroid together with spreads.
- **Irregular shape clusters are hard to represent.** They may not be useful in some applications.
 - Using centroids are not suitable (upper figure) in general
 - K-means clusters may be more useful (lower figure), e.g., for making 2 size T-shirts.



k-Mean in Python

Fundamental Data Science for Data
Scientist using Python

K-Means Clustering

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

```
In [16]: from sklearn.cluster import KMeans  
  
x = iris.iloc[:, [0, 1, 2, 3]].values  
  
x
```

```
In [19]: #Applying kmeans to the dataset / Creating the kmeans classifier
```

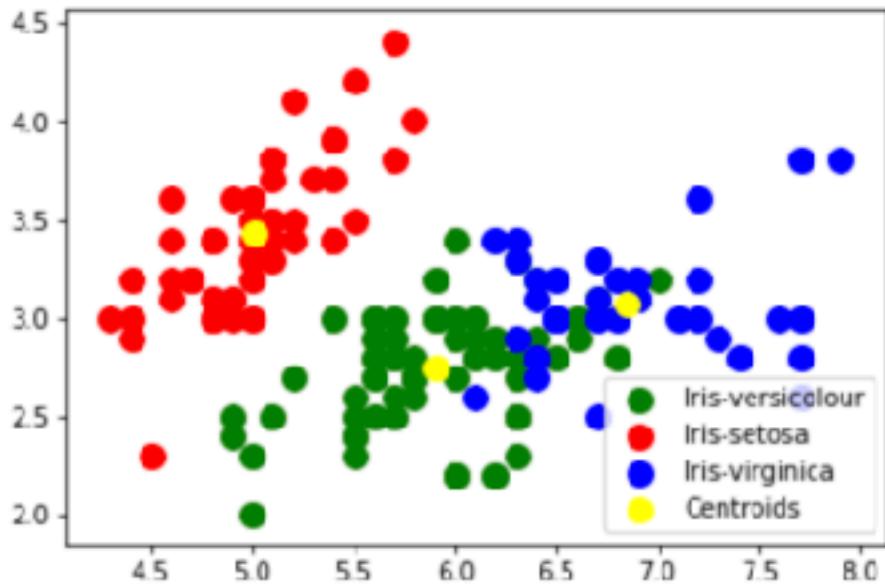
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

```
[0]: #Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100,
            c = 'green', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100,
            c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100,
            c = 'blue', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100,
            c = 'yellow', label = 'Centroids')

plt.legend()
```

```
[0]: <matplotlib.legend.Legend at 0x1177e4dd8>
```



Another round

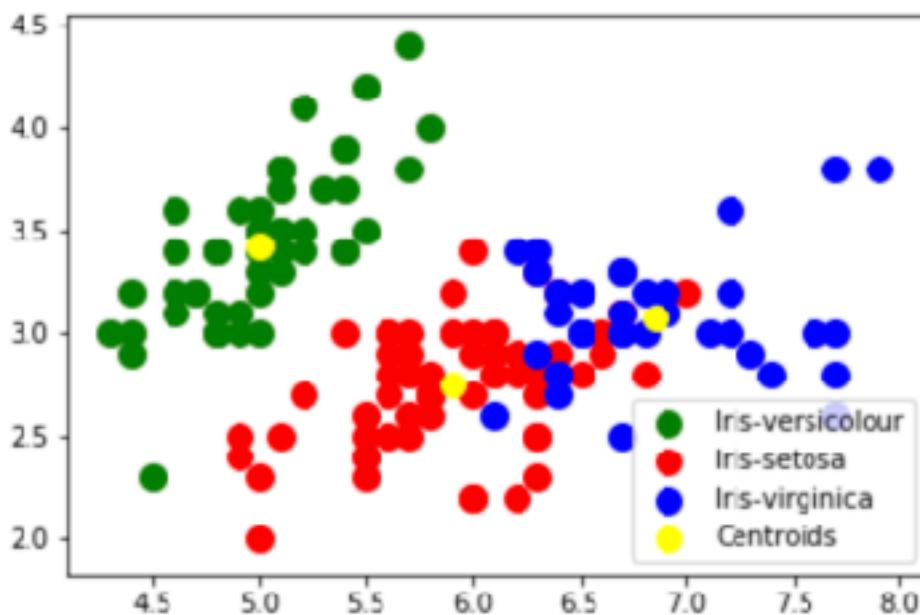
```
[1]: #Applying kmeans to the dataset / Creating the kmeans classifier  
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10,  
                 random_state = 12345)  
y_kmeans = kmeans.fit_predict(x)
```

```
2]: #Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100,
            c = 'green', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100,
            c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100,
            c = 'blue', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100,
            c = 'yellow', label = 'Centroids')

plt.legend()
```

```
2]: <matplotlib.legend.Legend at 0x119a94780>
```

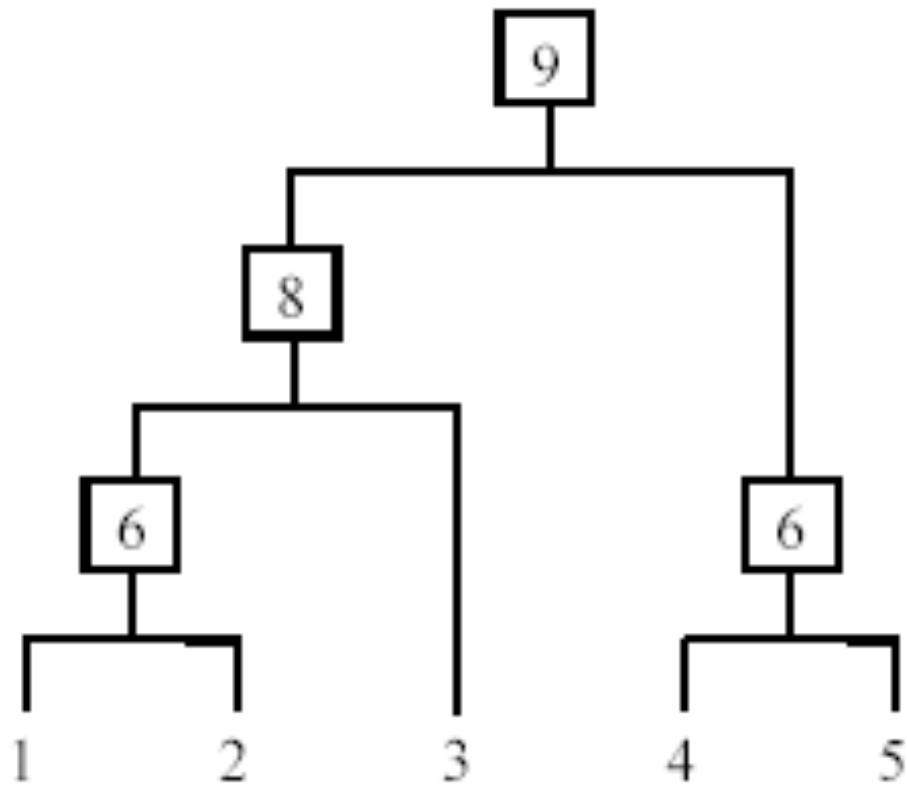


Workshop 9 - k-Mean Clustering

1. Use data from Workshop 2
2. Perform k-Mean clustering to divide data into k groups

Hierarchical Clustering

- Produce a nested sequence of clusters, a **tree**, also called **Dendrogram**.



Types of hierarchical clustering

- **Agglomerative (bottom up) clustering:** It builds the dendrogram (tree) from the bottom level, and
 - merges the most similar (or nearest) pair of clusters
 - stops when all the data points are merged into a single cluster (i.e., the root cluster).
- **Divisive (top down) clustering:** It starts with all data points in one cluster, the root.
 - Splits the root into a set of child clusters. Each child cluster is recursively divided further
 - stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

Agglomerative clustering

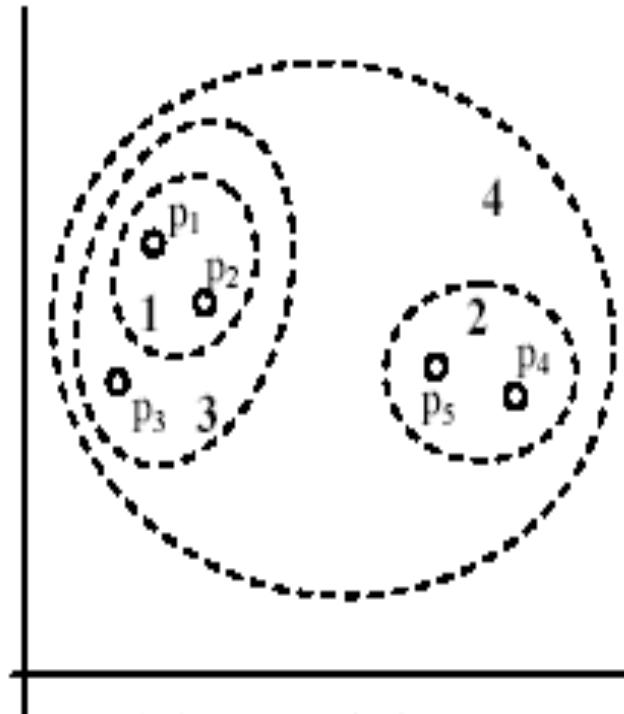
- It is more popular than divisive methods.
- At the beginning, each data point forms a cluster (also called a node).
- Merge nodes/clusters that have the least distance.
- Go on merging
- Eventually all nodes belong to one cluster

Agglomerative clustering algorithm

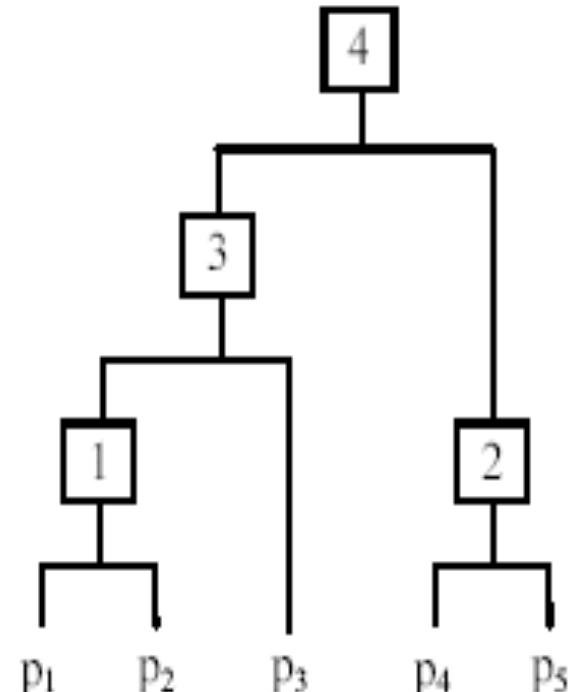
Algorithm Agglomerative(D)

- 1 Make each data point in the data set D a cluster,
- 2 Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in D$;
- 2 **repeat**
- 3 find two clusters that are nearest to each other;
- 4 merge the two clusters form a new cluster c ;
- 5 compute the distance from c to all other clusters;
- 12 **until** there is only one cluster left

An example: working of the algorithm



(A). Nested clusters



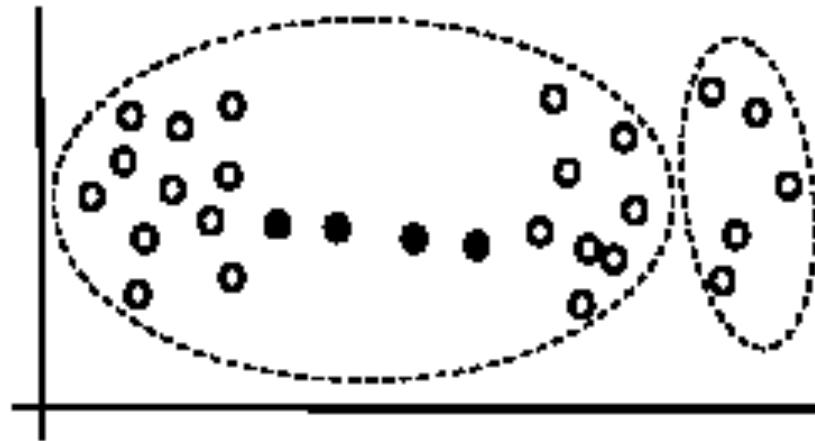
(B) Dendrogram

Measuring the distance of two clusters

- A few ways to measure distances of two clusters.
- Results in different variations of the algorithm.
 - Single link
 - Complete link
 - Average link
 - Centroids
 - ...

Single link method

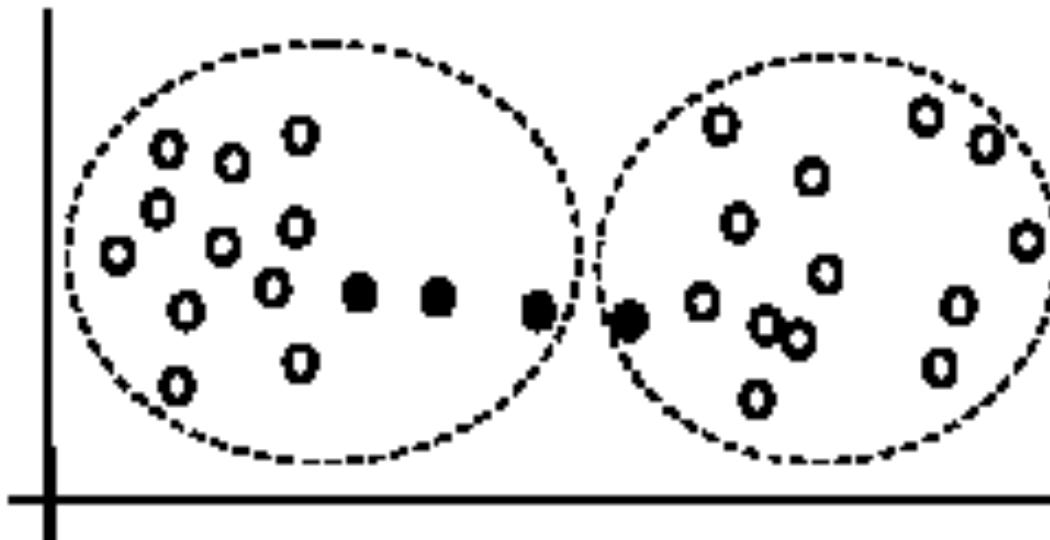
- The distance between two clusters is the distance between two **closest data points** in the two clusters, one data point from each cluster.
- It can find arbitrarily shaped clusters, but
 - It may cause the undesirable “**chain effect**” by noisy points



Two natural clusters are split into two

Complete link method

- The distance between two clusters is the distance of two **furthest** data points in the two clusters.
- It is sensitive to outliers because they are far away



Average link and centroid methods

- **Average link:** A compromise between
 - the sensitivity of complete-link clustering to outliers and
 - the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.
 - In this method, the distance between two clusters is the average distance of all pair-wise distances between the data points in two clusters.
- **Centroid method:** In this method, the distance between two clusters is the distance between their centroids

Distance functions

- Key to clustering. “**similarity**” and “**dissimilarity**” can also commonly used terms.
- There are numerous distance functions for
 - Different types of data
 - Numeric data
 - Nominal data
 - Different specific applications

Distance functions for numeric attributes

- Most commonly used functions are
 - **Euclidean distance** and
 - **Manhattan (city block) distance**
- We denote distance with: $dist(\mathbf{x}_i, \mathbf{x}_j)$, where \mathbf{x}_i and \mathbf{x}_j are data points (vectors)
- They are special cases of **Minkowski distance**. h is positive integer.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \left((x_{i1} - x_{j1})^h + (x_{i2} - x_{j2})^h + \dots + (x_{ir} - x_{jr})^h \right)^{\frac{1}{h}}$$

Euclidean distance and Manhattan distance

- If $h = 2$, it is the **Euclidean distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2}$$

- If $h = 1$, it is the **Manhattan distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ir} - x_{jr}|$$

- **Weighted Euclidean distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \dots + w_r(x_{ir} - x_{jr})^2}$$

Squared distance and Chebychev distance

- **Squared Euclidean distance:** to place progressively greater weight on data points that are further apart.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2$$

- **Chebychev distance:** one wants to define two data points as "different" if they are different on any one of the attributes.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \dots, |x_{ir} - x_{jr}|)$$

Hierarchical Clustering in Python

Fundamental Data Science for Data
Scientist using Python

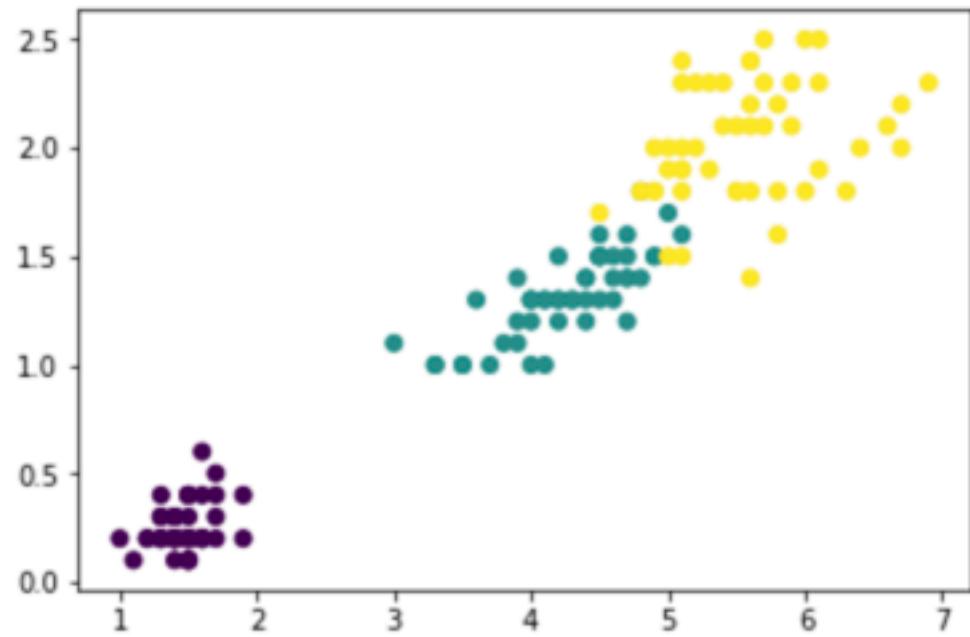
Hierarchical Clustering

- ◎ Compute distance between every pairs of point/cluster.
 - ◎ (a) Distance between point is just using the distance function.
 - ◎ (b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).
 - ◎ (c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.
- ◎ Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains

```
In [2]: from sklearn import datasets
```

```
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
n_samples, n_features = X.shape

plt.scatter(X[:, 0], X[:, 1], c=y);
```

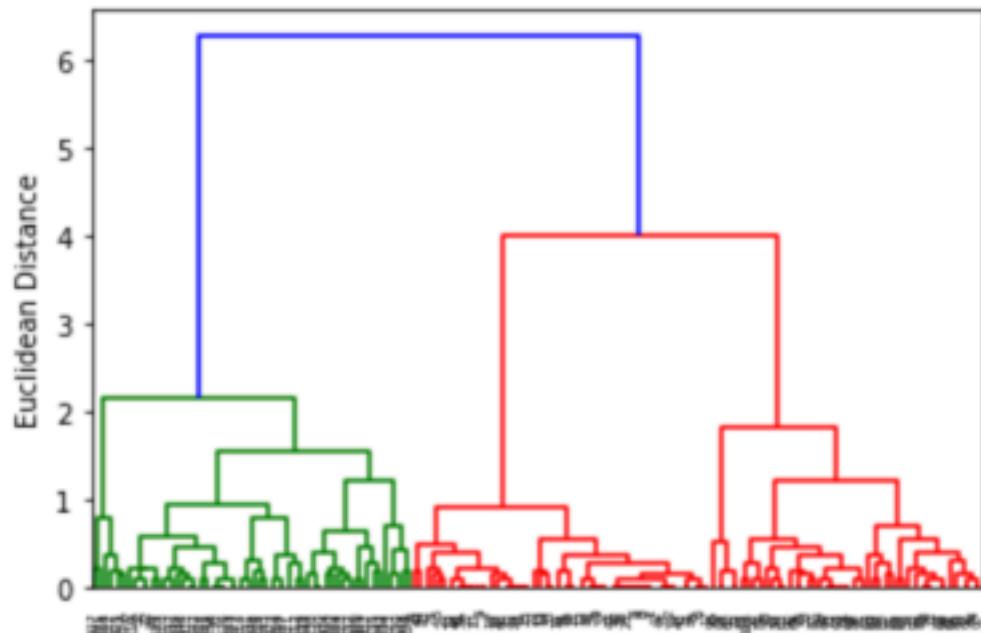


```
In [3]: from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram

clusters = linkage(X,
                    metric='euclidean',
                    method='complete')

dendr = dendrogram(clusters)

plt.ylabel('Euclidean Distance');
```



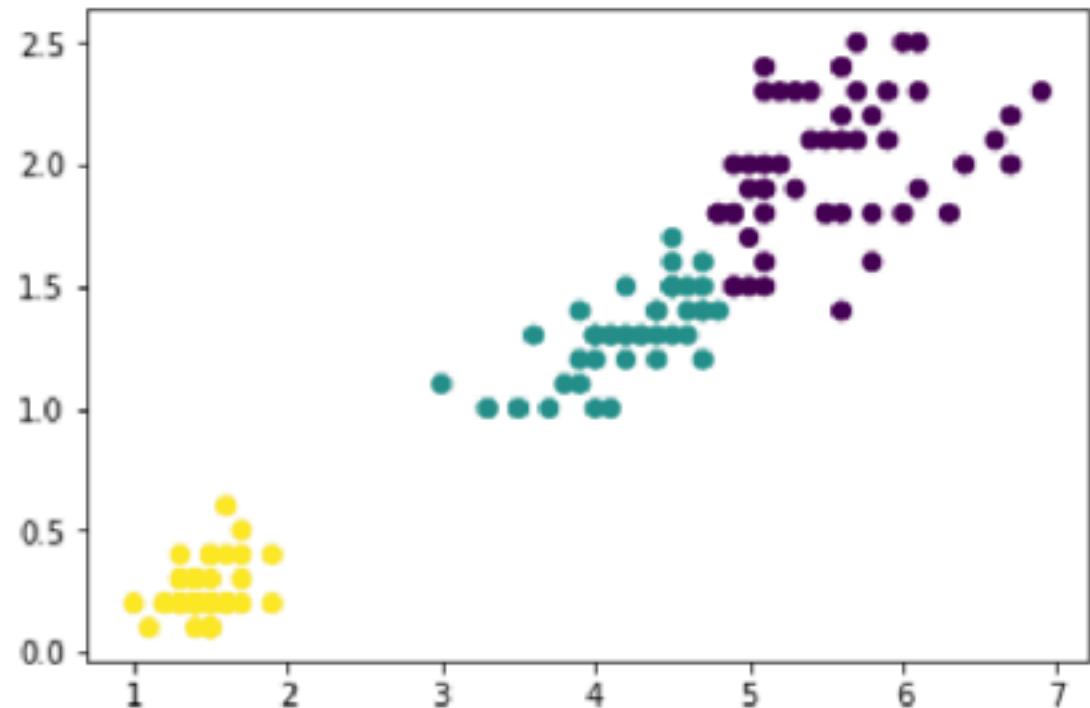
It's possible to prune the result tree.

```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=3,
                             affinity='euclidean',
                             linkage='complete')

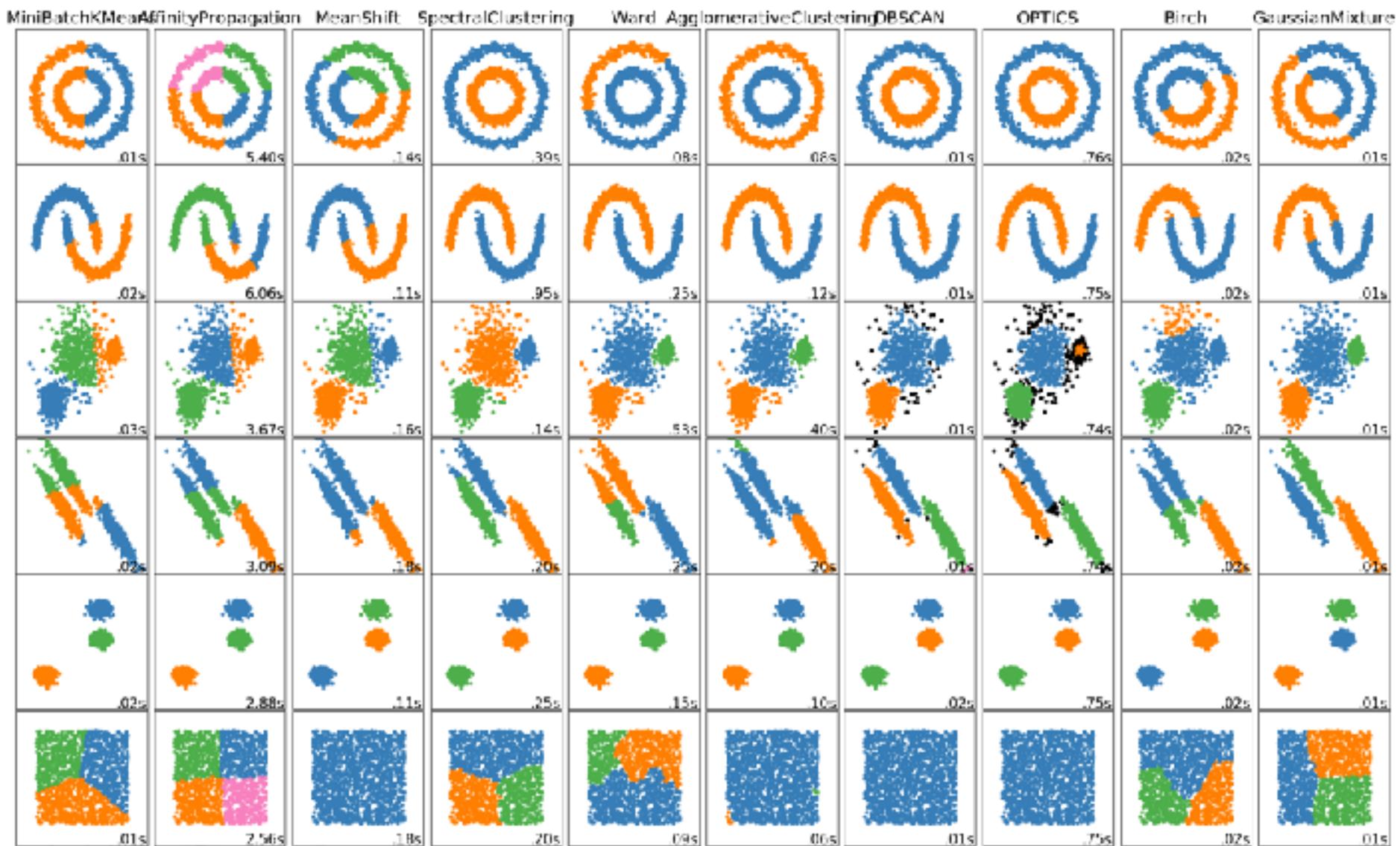
prediction = ac.fit_predict(X)
print('Cluster labels: %s\n' % prediction)
```

```
In [5]: plt.scatter(x[:, 0], x[:, 1], c=prediction);
```



Workshop 10 - Hierarchical Clustering

1. Use data from Workshop 2
2. Perform Hierarchical clustering to divide data into k groups



How to choose a clustering algorithm

- Clustering research has a long history. A vast collection of algorithms are available.
 - We only introduced several main algorithms.
- Choosing the “best” algorithm is a challenge.
 - Every algorithm has limitations and works well with certain data distributions.
 - It is very hard, if not impossible, to know what distribution the application data follow. The data may not fully follow any “ideal” structure or distribution required by the algorithms.
 - One also needs to decide how to standardize the data, to choose a suitable distance function and to select other parameter values.

Choose a clustering algorithm (cont ...)

- Due to these complexities, the common practice is to
 - run several algorithms using different distance functions and parameter settings, and
 - then carefully analyze and compare the results.
- The interpretation of the results must be based on insight into the meaning of the original data together with knowledge of the algorithms used.
- Clustering is highly **application dependent** and to certain extent **subjective** (personal preferences).

Cluster Evaluation: hard problem

- The quality of a clustering is very hard to evaluate because
 - We do not know the correct clusters
- Some methods are used:
 - User inspection
 - Study centroids, and spreads
 - Rules from a decision tree.
 - For text documents, one can read some documents in clusters.

Cluster evaluation: ground truth

- We use some labeled data (for classification)
- **Assumption:** Each class is a cluster.
- After clustering, a confusion matrix is constructed.
From the matrix, we compute various measurements,
entropy, purity, precision, recall and F-score.
 - Let the classes in the data D be $C = (c_1, c_2, \dots, c_k)$.
The clustering method produces k clusters, which divides D into k disjoint subsets, D_1, D_2, \dots, D_k .

Evaluation measures: Entropy

Entropy: For each cluster, we can measure its entropy as follows:

$$\text{entropy}(D_i) = -\sum_{j=1}^k \Pr_i(c_j) \log_2 \Pr_i(c_j), \quad (29)$$

where $\Pr_i(c_j)$ is the proportion of class c_j data points in cluster i or D_i . The total entropy of the whole clustering (which considers all clusters) is

$$\text{entropy}_{\text{total}}(D) = \sum_{i=1}^k \frac{|D_i|}{|D|} \times \text{entropy}(D_i) \quad (30)$$

Evaluation measures: purity

Purity: This again measures the extent that a cluster contains only one class of data. The purity of each cluster is computed with

$$purity(D_i) = \max_j(\Pr_i(c_j)) \quad (31)$$

The total purity of the whole clustering (considering all clusters) is

$$purity_{total}(D) = \sum_{i=1}^k \frac{|D_i|}{|D|} \times purity(D_i) \quad (32)$$

Evaluation based on internal information

- **Intra-cluster cohesion** (compactness):
 - Cohesion measures how near the data points in a cluster are to the cluster centroid.
 - Sum of squared error (SSE) is a commonly used measure.
- **Inter-cluster separation** (isolation):
 - Separation means that different cluster centroids should be far away from one another.
- In most applications, expert judgments are still the key.

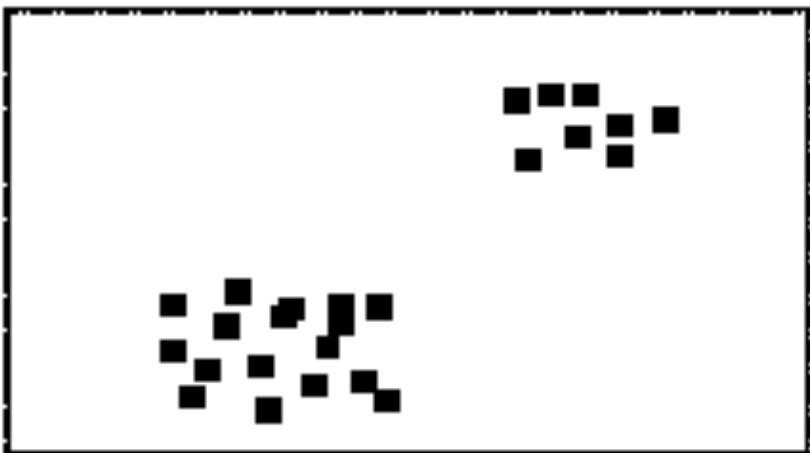
Indirect evaluation

- In some applications, clustering is **not the primary task**, but used to help perform another task.
- We can use the performance on the primary task to compare clustering methods.
- For instance, in an application, the primary task is to provide recommendations on book purchasing to online shoppers.
 - If we can cluster books according to their features, we might be able to provide better recommendations.
 - We can evaluate different clustering algorithms based on how well they help with the recommendation task.
 - Here, we assume that the recommendation can be reliably evaluated.

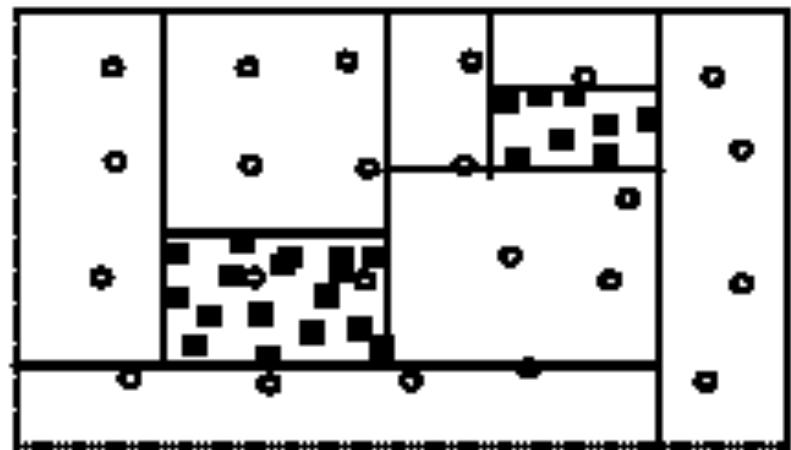
Supervised learning for unsupervised learning

- Decision tree algorithm is not directly applicable.
 - it needs at least two classes of data.
 - A clustering data set has no class label for each data point.
- The problem can be dealt with by a simple idea.
 - Regard each point in the data set to have a class label Y .
 - Assume that the data space is uniformly distributed with another type of points, called **non-existing points**. We give them the class, N .
- With the N points added, the problem of partitioning the data space into data and empty regions becomes a supervised classification problem.

An example



(A): The original data space



(B). Partitioning with added
 N points

- A decision tree method is used for partitioning in (B).

Can it done without adding N points?

- Yes.
- Physically adding N points increases the size of the data and thus the running time.
- More importantly: it is unlikely that we can have points truly uniformly distributed in a high dimensional space as we would need an exponential number of points.
- Fortunately, no need to physically add any N points.
 - We can compute them when needed

Characteristics of the approach

- It provides representations of the resulting data and empty regions in terms of **hyper-rectangles**, or **rules**.
- It detects outliers automatically. Outliers are data points in an empty region.
- It may not use all attributes in the data just as in a normal decision tree for supervised learning.
 - It can automatically determine what attributes are useful. Subspace clustering ...
- **Drawback:** data regions of irregular shapes are hard to handle since decision tree learning only generates hyper-rectangles (formed by axis-parallel hyper-planes), which are rules.

Building the Tree

- The main computation in decision tree building is to evaluate **entropy** (for **information gain**):

$$\text{entropy}(D) = - \sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

- Can it be evaluated without adding N points? Yes.
- $\Pr(c_j)$ is the probability of class c_j in data set D , and $|C|$ is the number of classes, Y and N (2 classes).
 - To compute $\Pr(c_j)$, we only need the number of Y (data) points and the number of N (non-existing) points.
 - We already have Y (or data) points, and we can compute the number of N points on the fly. Simple: as we assume that the N points are uniformly distributed in the space.

Summary

- Clustering is has along history and still active
 - There are a huge number of clustering algorithms
 - More are still coming every year.
- We only introduced several main algorithms. There are many others, e.g.,
 - density based algorithm, sub-space clustering, scale-up methods, neural networks based methods, fuzzy clustering, co-clustering, etc.
- Clustering is hard to evaluate, but very useful in practice. This partially explains why there are still a large number of clustering algorithms being devised every year.
- Clustering is highly application dependent and to some extent subjective.

Recommendation System

- Concept
- Building Recommendation System



Recommendation System Topics

- Basic concepts
- Apriori algorithm
- Summary

Association rule mining

- Proposed by **Agrawal et al in 1993**.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

•

• Bread → Milk [sup = 5%, conf = 100%]

The model: data

- $I = \{i_1, i_2, \dots, i_m\}$: a set of *items*.
- Transaction t :
 - t a set of items, and $t \subseteq I$.
- Transaction Database T : a set of transactions
 $T = \{t_1, t_2, \dots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:
 - $t_1: \{\text{bread, cheese, milk}\}$
 - $t_2: \{\text{apple, eggs, salt, yogurt}\}$
 - ...
 - $t_n: \{\text{biscuit, eggs, milk}\}$
- Concepts:
 - An *item*: an item/article in a basket
 - I : the set of all items sold in the store
 - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
 - A *transactional dataset*: A set of transactions

Transaction data: a set of documents

- A text document data set. Each document is treated as a “bag” of keywords

- doc1: Student, Teach, School
- doc2: Student, School
- doc3: Teach, School, City, Game
- doc4: Baseball, Basketball
- doc5: Basketball, Player, Spectator
- doc6: Baseball, Coach, Game, Team
- doc7: Basketball, Team, City, Game

The model: rules

- A transaction t contains X , a set of items (**itemset**) in I , if $X \subseteq t$.
- An **association rule** is an implication of the form:
 - $X \rightarrow Y$, where $X, Y \subset I$, and $X \cap Y = \emptyset$
- An **itemset** is a set of items.
 - E.g., $X = \{\text{milk, bread, cereal}\}$ is an itemset.
- A **k -itemset** is an itemset with k items.
 - E.g., $\{\text{milk, bread, cereal}\}$ is a 3-itemset

Rule strength measures

- **Support:** The rule holds with support sup in T (the transaction data set) if $sup\%$ of transactions contain $X \cup Y$.
 - $sup = \Pr(X \cup Y)$.
- **Confidence:** The rule holds in T with confidence $conf$ if $conf\%$ of transactions that contain X also contain Y .
 - $conf = \Pr(Y | X)$
- An association rule is a pattern that states when X occurs, Y occurs with certain probability.

Support and Confidence

- Support count: The support count of an itemset X , denoted by $X.count$, in a data set T is the number of transactions in T that contain X . Assume T has n transactions.
- Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).
- **Key Features**
 - Completeness: find all rules.
 - No target item(s) on the right-hand-side
 - Mining with data on **hard disk** (not in memory)

An example

- Transaction data

- Assume:

- ◎ $\text{minsup} = 30\%$
- ◎ $\text{minconf} = 80\%$

- An example **frequent itemset**:

- ◎ $\{\text{Chicken, Clothes, Milk}\}$ [sup = 3/7]

- **Association rules** from the itemset:

- ◎ Clothes \rightarrow Milk, Chicken [sup = 3/7, conf = 3/3]

- ◎ ...

- ...

Clothes, Chicken \rightarrow Milk, [sup = 3/7, conf = 3/3]

- t1: Beef, Chicken, Milk
- t2: Beef, Cheese
- t3: Cheese, Boots
- t4: Beef, Chicken, Cheese
- t5: Beef, Chicken, Clothes, Cheese, Milk
- t6: Chicken, Clothes, Milk
- t7: Chicken, Milk, Clothes

Transaction data representation

- A simplistic view of shopping baskets,
- Some important information not considered. E.g,
 - the quantity of each item purchased and
 - the price paid.

Many mining algorithms

- **There are a large number of them!!**
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.
 - Given a transaction data set T , and a minimum support and a minimum confident, the set of association rules existing in T is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study only one: **the Apriori Algorithm**

Recommendation System Topics

- Basic concepts
- Apriori algorithm
- Summary

The Apriori algorithm

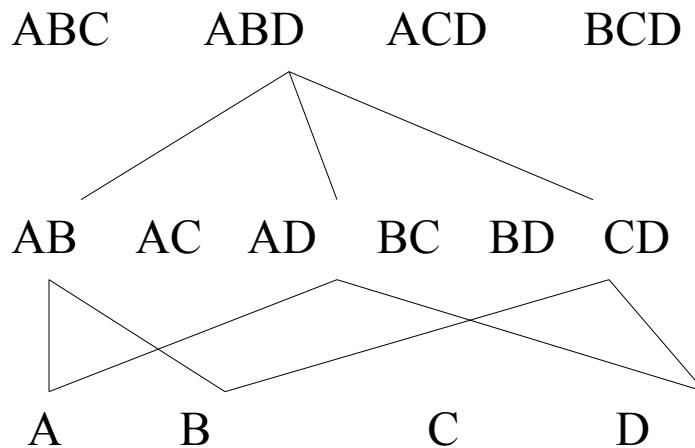
- **Probably the best known algorithm**
- **Two steps:**
 - Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets).
 - Use frequent itemsets to **generate rules**.

- E.g., a frequent itemset
 - {Chicken, Clothes, Milk} [sup = 3/7]
 - and one rule from the frequent itemset

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

Step 1: Mining all frequent itemsets

- A **frequent itemset** is an itemset whose support is $\geq \text{minsup}$.
- **Key idea: The apriori property (downward closure property):** any subsets of a frequent itemset are also frequent itemsets



The Algorithm

- Iterative algo. (also called level-wise search): Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.
 - In each iteration k , only consider itemsets that contain some $k-1$ frequent itemset.
- Find frequent itemsets of size 1: F_1
- From $k = 2$
 - C_k = candidates of size k : those itemsets of size k that could be frequent, given F_{k-1}
 - F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once).

Example – Finding frequent itemsets

TID	Items
T100	1, 3, 4
T200	2, 3, 5
T300	1, 2, 3, 5
T400	2, 5

itemset:count

1. scan T \rightarrow C₁: {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

\rightarrow F₁: {1}:2, {2}:3, {3}:3, {5}:3

\rightarrow C₂: {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. scan T \rightarrow C₂: {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

\rightarrow F₂: {1,3}:2, {2,3}:2, {2,5}:3, {3,5}:2

\rightarrow C₃: {2, 3, 5}

3. scan T \rightarrow C₃: {2, 3, 5}:2 \rightarrow F₃: {2, 3, 5}

Details: ordering of items

- The items in I are sorted in **lexicographic order** (which is a total order).
- The order is used throughout the algorithm in each itemset.
- $\{w[1], w[2], \dots, w[k]\}$ represents a k -itemset w consisting of items $w[1], w[2], \dots, w[k]$, where $w[1] < w[2] < \dots < w[k]$ according to the total order.

Details: the algorithm

- **Algorithm Apriori(T)**

```
C1 ← init-pass( $T$ );  
F1 ← { $f \mid f \in C_1, f.\text{count}/n \geq \text{minsup}$ }; // n: no. of transactions in T  
for ( $k = 2; F_{k-1} \neq \emptyset; k++$ ) do  
    Ck ← candidate-gen(Fk-1);  
    for each transaction  $t \in T$  do  
        for each candidate  $c \in C_k$  do  
            if  $c$  is contained in  $t$  then  
                c.count++;  
            end  
        end  
    end  
    Fk ← { $c \in C_k \mid c.\text{count}/n \geq \text{minsup}$ }  
end  
return  $F \leftarrow \bigcup_k F_k$ ;
```

Apriori candidate generation

- The **candidate-gen** function takes F_{k-1} and returns a **superset** (called the candidates) of the set of all **frequent k -itemsets**. It has two steps
 - join step*: Generate all possible candidate itemsets C_k of length k
 - prune step*: Remove those candidates in C_k that cannot be frequent.

Candidate-gen function

```
○ Function candidate-gen( $F_{k-1}$ )
○    $C_k \leftarrow \emptyset;$ 
○   forall  $f_1, f_2 \in F_{k-1}$ 
○     with  $f_1 = \{i_1, \dots, i_{k-2}, i_{k-1}\}$ 
○     and  $f_2 = \{i'_1, \dots, i'_{k-2}, i'_{k-1}\}$ 
○     and  $i_{k-1} < i'_{k-1}$  do
○        $c \leftarrow \{i_1, \dots, i_{k-1}, i'_{k-1}\};$            // join  $f_1$  and  $f_2$ 
○        $C_k \leftarrow C_k \cup \{c\};$ 
○       for each  $(k-1)$ -subset  $s$  of  $c$  do
○         if ( $s \notin F_{k-1}$ ) then
○           delete  $c$  from  $C_k;$            // prune
○         end
○       end
○     return  $C_k;$ 
```

An example

- $F_3 = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\},$
 - ◎ $\{1, 3, 5\}, \{2, 3, 4\}\}$
- After join
 - $C_4 = \{\{1, 2, 3, 4\}, \{1, 3, 4, 5\}\}$
- After pruning:
 - $C_4 = \{\{1, 2, 3, 4\}\}$
 - because $\{1, 4, 5\}$ is not in F_3 ($\{1, 3, 4, 5\}$ is removed)

Step 2: Generating rules from frequent itemsets

- Frequent itemsets \neq association rules
- One more step is needed to generate association rules
- For each frequent itemset X ,
 - ◎ For each proper nonempty subset A of X ,
 - Let $B = X - A$
 - $A \rightarrow B$ is an association rule if
 - $\text{Confidence}(A \rightarrow B) \geq \text{minconf}$,
 - $\text{support}(A \rightarrow B) = \text{support}(A \cup B) = \text{support}(X)$
 - $\text{confidence}(A \rightarrow B) = \text{support}(A \cup B) / \text{support}(A)$

Generating rules: an example

- Suppose $\{2,3,4\}$ is frequent, with sup=50%
 - Proper nonempty subsets: $\{2,3\}$, $\{2,4\}$, $\{3,4\}$, $\{2\}$, $\{3\}$, $\{4\}$, with sup=50%, 50%, 75%, 75%, 75%, 75% respectively
 - These generate these association rules:
 - $2,3 \rightarrow 4$, confidence=100%
 - $2,4 \rightarrow 3$, confidence=100%
 - $3,4 \rightarrow 2$, confidence=67%
 - $2 \rightarrow 3,4$, confidence=67%
 - $3 \rightarrow 2,4$, confidence=67%
 - $4 \rightarrow 2,3$, confidence=67%
 - All rules have support = 50%

Generating rules: summary

- To recap, in order to obtain $A \rightarrow B$, we need to have $\text{support}(A \cup B)$ and $\text{support}(A)$
- All the required information for confidence computation has already been recorded in itemset generation. No need to see the data T any more.
- This step is not as time-consuming as frequent itemsets generation.

On Apriori Algorithm

- Seems to be very expensive
- Level-wise search
- $K =$ the size of the largest itemset
- It makes at most K passes over data
- In practice, K is bounded (10).
- The algorithm is very fast. Under some conditions, all rules can be found in **linear time**.
- Scale up to large data sets

Metrics of Association Rules

- Support
- Confidence
- Lift
- Leverage
- Conviction

Metrics of Association Rules

- ◎ The currently supported metrics for evaluating association rules and setting selection thresholds are listed below. Given a rule " $A \rightarrow C$ ", A stands for antecedent and C stands for consequent.

support

$$\text{support}(A \rightarrow C) = \text{support}(A \cup C), \quad \text{range: } [0, 1]$$

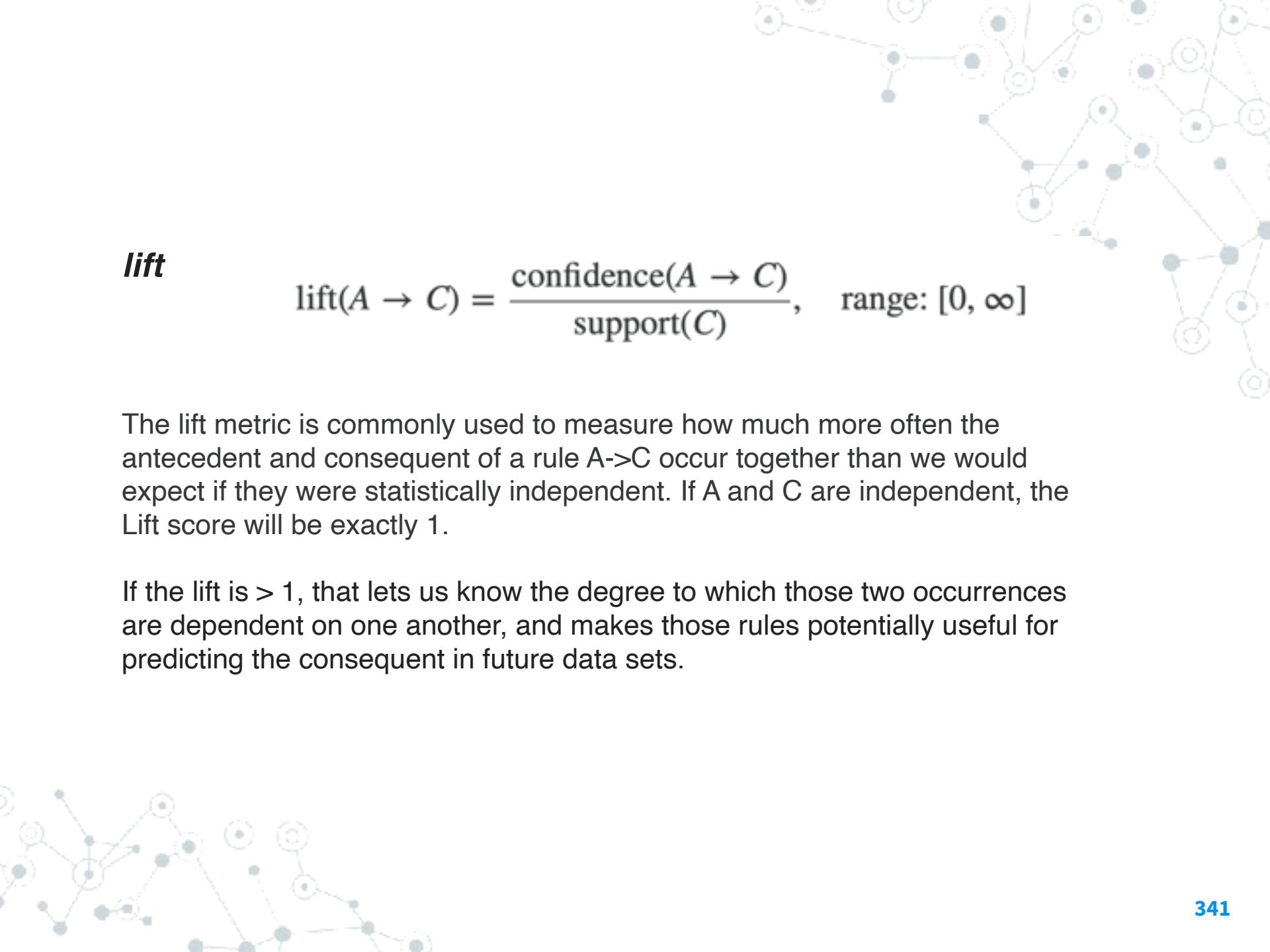
The support metric is defined for itemsets, not association rules. The table produced by the association rule mining algorithm contains three different support metrics: 'antecedent support', 'consequent support', and 'support'. Here, 'antecedent support' computes the proportion of transactions that contain the antecedent A , and 'consequent support' computes the support for the itemset of the consequent C . The 'support' metric then computes the support of the combined itemset $A \cup C$ -- note that 'support' depends on 'antecedent support' and 'consequent support' via $\min(\text{antecedent support}, \text{consequent support})$.

Typically, support is used to measure the abundance or frequency (often interpreted as significance or importance) of an itemset in a database. We refer to an itemset as a "frequent itemset" if its support is larger than a specified minimum-support threshold. Note that in general, due to the *downward closure* property, all subsets of a frequent itemset are also frequent.

confidence

$$\text{confidence}(A \rightarrow C) = \frac{\text{support}(A \rightarrow C)}{\text{support}(A)}, \quad \text{range: } [0, 1]$$

The confidence of a rule A->C is the probability of seeing the consequent in a transaction given that it also contains the antecedent. Note that the metric is not symmetric or directed; for instance, the confidence for A->C is different than the confidence for C->A. The confidence is 1 (maximal) for a rule A->C if the consequent and antecedent always occur together.



lift

$$\text{lift}(A \rightarrow C) = \frac{\text{confidence}(A \rightarrow C)}{\text{support}(C)}, \quad \text{range: } [0, \infty]$$

The lift metric is commonly used to measure how much more often the antecedent and consequent of a rule A->C occur together than we would expect if they were statistically independent. If A and C are independent, the Lift score will be exactly 1.

If the lift is > 1, that lets us know the degree to which those two occurrences are dependent on one another, and makes those rules potentially useful for predicting the consequent in future data sets.

leverage

$$\text{leverage}(A \rightarrow C) = \text{support}(A \rightarrow C) - \text{support}(A) \times \text{support}(C), \quad \text{range: } [-1, 1]$$

Leverage computes the difference between the observed frequency of A and C appearing together and the frequency that would be expected if A and C were independent. A leverage value of 0 indicates independence.

conviction

$$\text{conviction}(A \rightarrow C) = \frac{1 - \text{support}(C)}{1 - \text{confidence}(A \rightarrow C)}, \quad \text{range: } [0, \infty]$$

A high conviction value means that the consequent is highly depending on the antecedent. For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to $1 - 1$) for which the conviction score is defined as 'inf'. Similar to lift, if items are independent, the conviction is 1.

Summary

- Association rule mining has been extensively studied in the data mining community.
- There are many efficient algorithms and model variations.
- Other related work includes
 - Multi-level or generalized rule mining
 - Constrained rule mining
 - Incremental rule mining
 - Maximal frequent itemset mining
 - Numeric association rule mining
 - Rule interestingness and visualization
 - Parallel algorithms

...

Building Recommendation System in Python



```
In [1]: # Importing the libraries
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: # Data Loading
```

```
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)

# Adding all customers into a list of lists
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
```

```
In [3]: dataset.head(5)
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole wheat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	9
1	burgers	meatballs		eggs		NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney		NaN		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water		milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN

```
In [5]: # Training Apriori on the dataset
# The hyperparameters chosen on this training are:
# min_support = items bought more than 3 times a day * 7 days (week) / 7500 customers = 0.0028
# min_confidence: at least 20%, min_lift = minimum of 3 (less than that is too low)

from apyori import apriori
rules = apriori(transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2)
```

```
In [6]: # Visualising the results
results = list(rules)
```

```
In [7]: lift = []
association = []
for i in range (0, len(results)):
    lift.append(results[:len(results)][i][2][0][3])
    association.append(list(results[:len(results)][i][0]))
```

Visualizing results in a dataframe

```
In [8]: rank = pd.DataFrame([association, lift]).T  
rank.columns = ['Association', 'Lift']
```

```
In [9]: # Show top 10 higher lift scores  
rank.sort_values('Lift', ascending=False).head(10)
```

Out[9]:

	Association	Lift
128	[nan, whole wheat pasta, olive oil, mineral wa...	6.11586
58	[whole wheat pasta, olive oil, mineral water]	6.11586
96	[mineral water, soup, milk, frozen vegetables]	5.48441
146	[frczen vegetables, nan, soup, mineral water, ...	5.48441
28	[honey, nan, fromage blanc]	5.16427
3	[honey, fromage blanc]	5.16427
16	[chicken, nan, light cream]	4.84395
0	[chicken, light cream]	4.84395
2	[escalope, pasta]	4.70081
26	[nan, escalope, pasta]	4.70081

Workshop 11 - Build Recommendation System

Load ratings csv file

In [2]:
ratings_df = pd.read_csv("../input/ratings_small.csv")

In [3]:
ratings_df.shape

Out[3]:
(1000004, 4)

In [4]:
ratings_df.head()

Out[4]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1098	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759165
4	1	1172	4.0	1260759205

Load Movies metadata csv file

In [5]:

```
movies_df = pd.read_csv("../input/movies_metadata.csv")
```

In [7]:

```
movies_df.head()
```

out[7]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	popularity
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...}	300000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...']}	http://toystory.disney.com/toy-story	852	tt0114709	en	7.3
1	False	NaN	650000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...']}	NaN	3844	tt0113497	en	7.1
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...}	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...}]	NaN	15802	tt0113228	en	6.6



In [12]:

```
ratings_df = pd.merge(ratings_df,movies_df[['title','id']],left_on='movieId',right_on='id')
ratings_df.head()
```

Out[12]:

	userId	movieId	rating	timestamp	title	id
0	1	1371	2.5	1260759135	Rocky III	1371
1	4	1371	4.0	949810302	Rocky III	1371
2	7	1371	3.0	851868160	Rocky III	1371
3	19	1371	4.0	855193404	Rocky III	1371
4	21	1371	3.0	853652263	Rocky III	1371

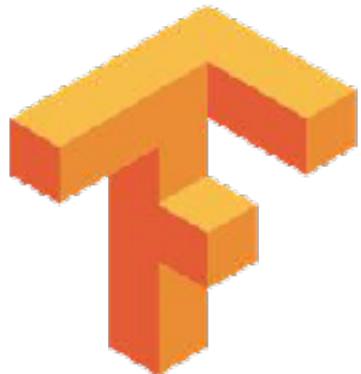


Workshop 11

- ◎ Use Apriori to build movie recommendation system

Deep Learning with TensorFlow & Keras

Deep Learning Framework



theano



Caffe



Overview: TensorFlow

What is TensorFlow?



- URL: <https://www.tensorflow.org/>
- Released under the open source license on November 9, 2015
- Current version 1.2
- Open source software library for numerical computation using data flow graphs
- Originally developed by Google Brain Team to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well
- TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch.

Most popular on Github

tensorflow / tensorflow

Watch 5,498 Star 61,869 Fork 29,849

Code Issues 714 Pull requests 57 Projects 0 Insights

Computation using data flow graphs for scalable machine learning <http://tensorflow.org>

tensorflow machine-learning python deep-learning deep-neural-networks neural-network ml distributed

19,049 commits 16 branches 33 releases 917 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

AnishShah committed with drpngx [issue #10835] Negative axis support for gradient of reduce_prod (#11019) 3 hours ago Latest commit ea79ba4 3 hours ago

tensorflow [issue #10835] Negative axis support for gradient of reduce_prod (#11019) 3 hours ago

third_party Add python import library on Windows (#10980) a day ago

tools Create tf_env collect.sh 13 days ago

<https://github.com/tensorflow/tensorflow>

TensorFlow architecture

- Core in C++
 - Low overhead
- Different front ends for specifying/driving the computation
 - Python, C++, R and many more

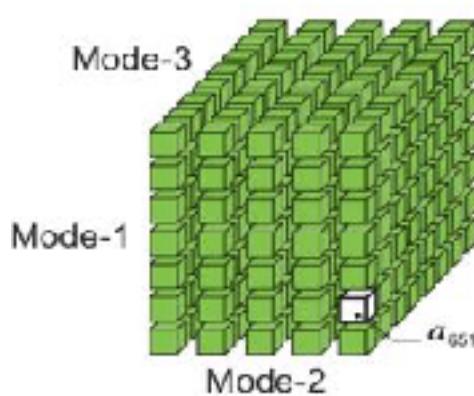


CPU - GPU

- In TensorFlow, the supported device types are CPU and GPU. They are represented as strings. For example:
 - "/cpu:0": The CPU of your machine.
 - "/gpu:0": The GPU of your machine, if you have one.
 - "/gpu:1": The second GPU of your machine, etc.

```
# Creates a graph.
with tf.device('/gpu:2'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

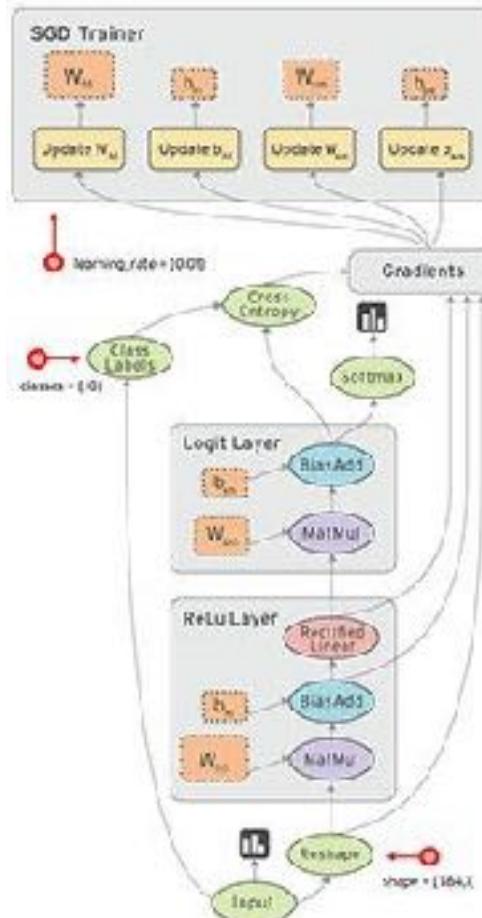
What is a Tensor?



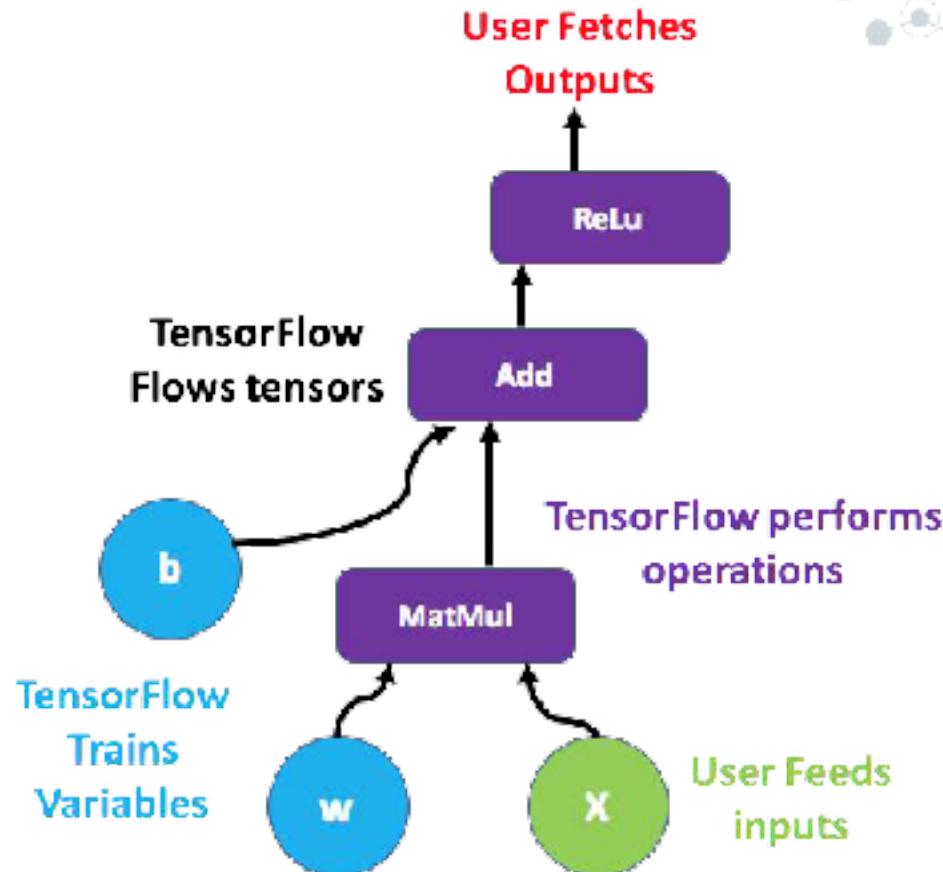
Dimensions	Example	Terminology									
1	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr></table>	0	1	2	Vector						
0	1	2									
2	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	Matrix
0	1	2									
3	4	5									
6	7	8									
3	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	3D Array (3 rd order Tensor)
0	1	2									
3	4	5									
6	7	8									
N		ND Array									

Why it is called TensorFlow?

- TensorFlow is based on computation data flow graph
- TensorFlow separates definition of computations from their execution

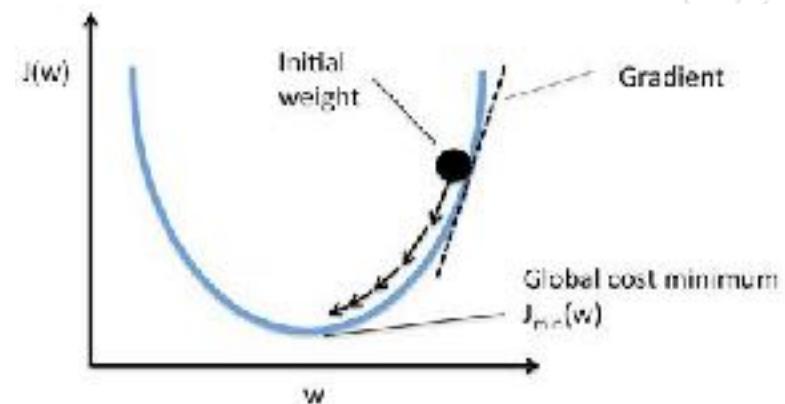


TensorFlow Graph

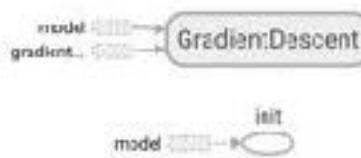
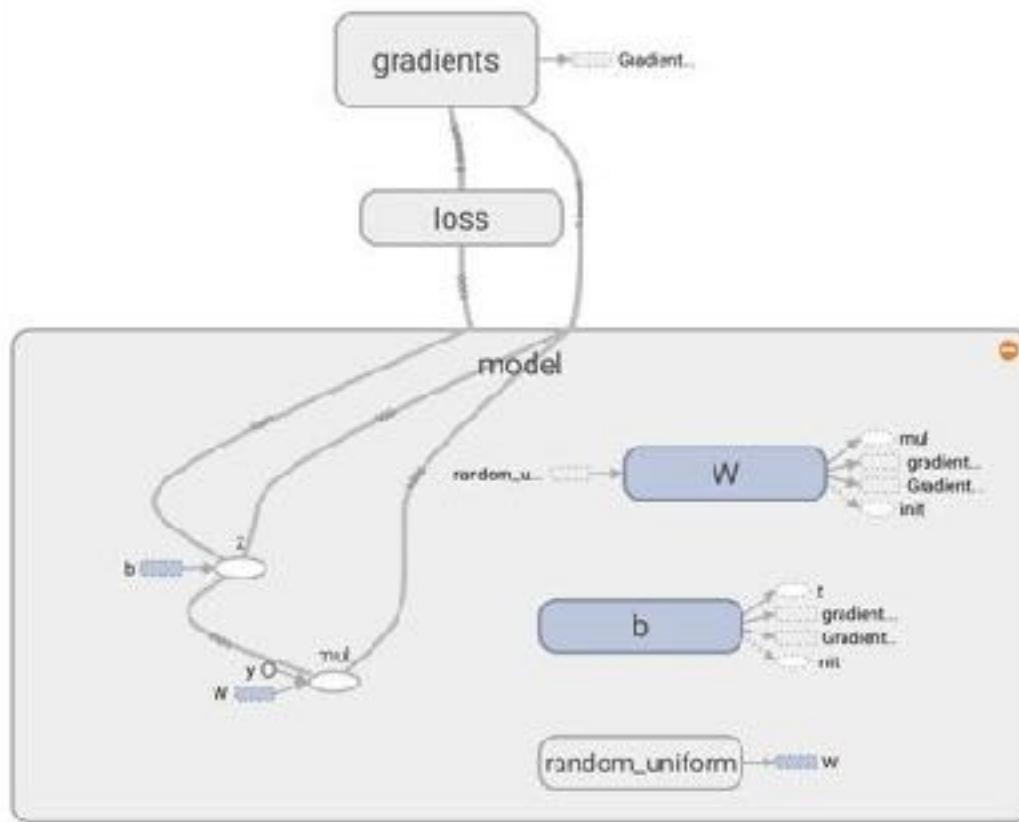


Learn Parameters: Optimization

- The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables.
- A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.
- TensorFlow provides functions to compute the derivatives for a given TensorFlow computation graph, adding operations to the graph.

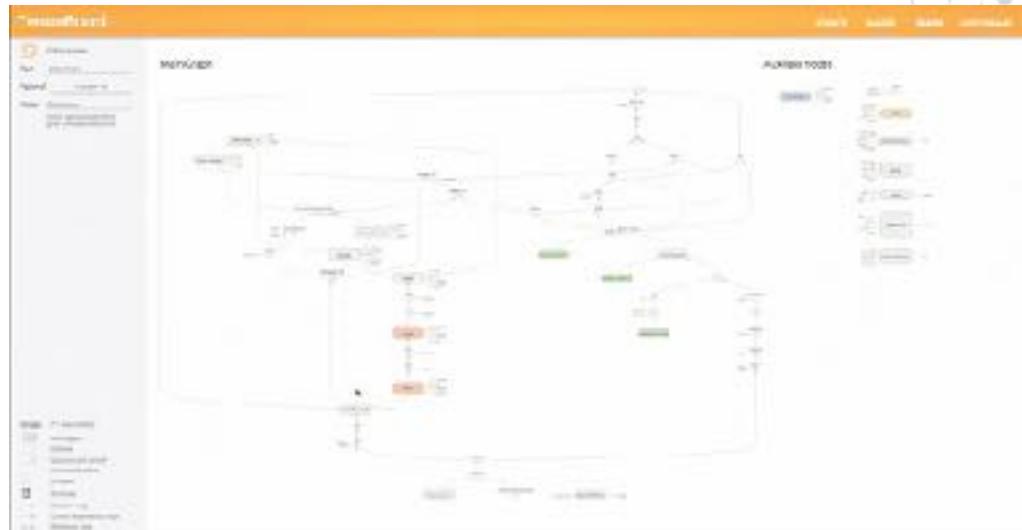


Learn Parameters: Optimization



TensorBoard

- Visualize your TensorFlow graph
- Plot quantitative metrics about the execution of your graph
- Show additional data like images that pass through it



TensorFlow Models

<https://github.com/tensorflow/models>

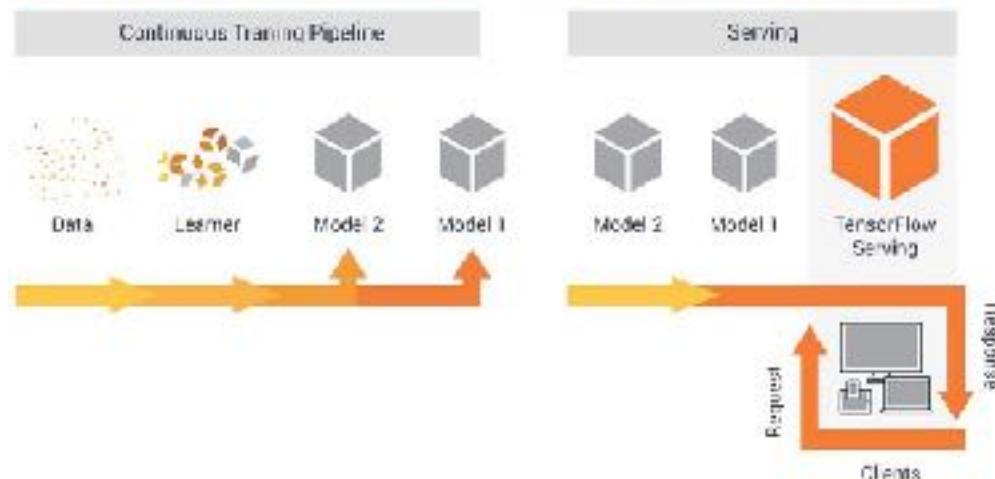
Models

- [adversarial_crypto](#): protecting communications with adversarial neural cryptography.
- [adversarial_text](#): semi-supervised sequence learning with adversarial training.
- [attention_ocr](#): a model for real-world image text extraction.
- [autoencoder](#): various autoencoders.
- [cognitive_mapping_and_planning](#): implementation of a spatial memory based mapping and planning architecture for visual navigation.
- [compression](#): compressing and decompressing images using a pre-trained Residual GRU network.
- [differential_privacy](#): privacy-preserving student models from multiple teachers.
- [domain_adaptation](#): domain separation networks.
- [im2txt](#): image-to-text neural network for image captioning.
- [inception](#): deep convolutional networks for computer vision.

TensorFlow Serving

- Flexible, high-performance serving system for machine learning models, designed for production environments.
- Easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs

Serve models in production with TensorFlow Serving



Code

```
import tensorflow as tf
# build a linear model where y
= w * x + b
w = tf.Variable([0.2], tf.float32,
b = tf.Variable([0.3], tf.float32,
name='bias')
X = tf.placeholder(tf.float32,
name="X")
Y = tf.placeholder(tf.float32,
# the training values for
x =
y =
([-1., -2., -3., -4.,
# define the linear
linear_model =
w*X+b
# define the loss
square_delta =
loss =
tf.reduce_sum(square del
#set the learning rate and
learning_rate =
training_epoch =
1000
```

```
# optimizer =
train = .Optimizer(learning_r
optimizer.minimize(loss
# start a
init =
tf.global_variables_initializer()
with tf.Session() as sess:
sess.run(
(init)
for i in
    sess.run(train,
    feed_dict={X:x, Y:y})
# evaluate training
curr_w, curr_b, curr_loss = sess.run([w, b,
print('w: %f b: %f loss: %f'%(curr_w,
curr_b, curr_loss))
```

TensorFlow: Installation

- pip3 install tensorflow

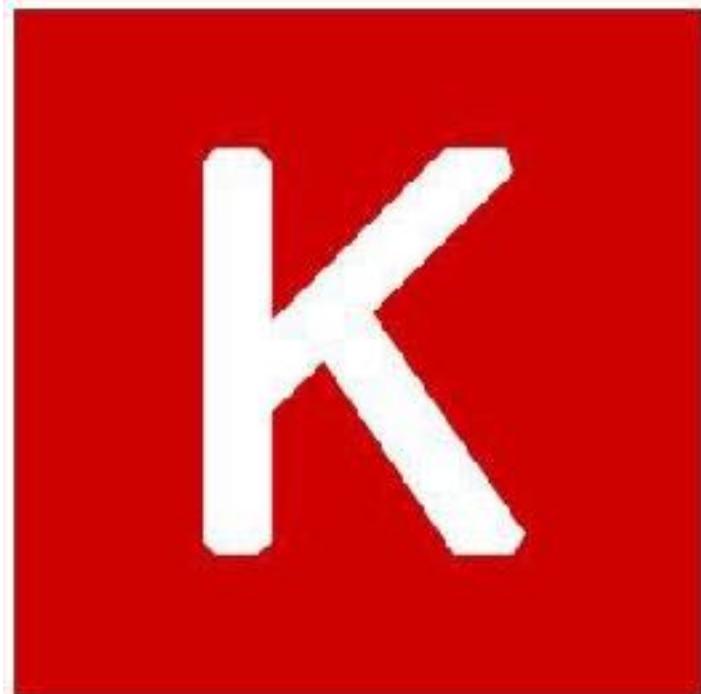
or

- **Anaconda:** conda install -c conda-forge tensorflow

Overview: Keras

Keras

- <https://keras.io/>
- Minimalist, highly modular neural networks library
- Written in Python
- Capable of running on top of either TensorFlow/Theano and CNTK
- Developed with a focus on enabling fast experimentation



Guiding Principles

- Modularity
 - A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible
- Minimalism
 - Each module should be kept short and simple
- Easy extensibility
 - New modules can be easily added and extended
- Python
 - Supports Python

General Design

- General idea is to based on layers and their input/output
 - Prepare your inputs and output tensors
 - Create first layer to handle input tensor
 - Create output layer to handle targets
 - Build virtually any model you like in between

Layers

- Keras has a number of pre-built layers. Notable examples include:
 - Regular dense, MLP type

```
keras.layers.core.Dense(units, activation=None,  
use_bias=True, kernel_initializer='glorot_uniform',  
bias_initializer='zeros', kernel_regularizer=None,  
bias_regularizer=None, activity_regularizer=None,  
kernel_constraint=None, bias_constraint=None)
```
 - Recurrent layers, LTSM, GRU, etc

```
keras.layers.recurrent.Recurrent(return_sequences=False,  
stateful=False, unroll=False,  
implementation=0)
```

Layers

- 1D Convolutional layers

```
keras.layers.convolutional.Conv1D(filters, kernel_size, strides=1,  
padding='valid', dilation_rate=1, activation=None, use_bias=True,  
kernel_initializer='glorot_uniform', bias_initializer='zeros',  
kernel_regularizer=None, bias_regularizer=None,  
activity_regularizer=None, kernel_constraint=None,  
bias_constraint=None)
```

- 2D Convolutional layers

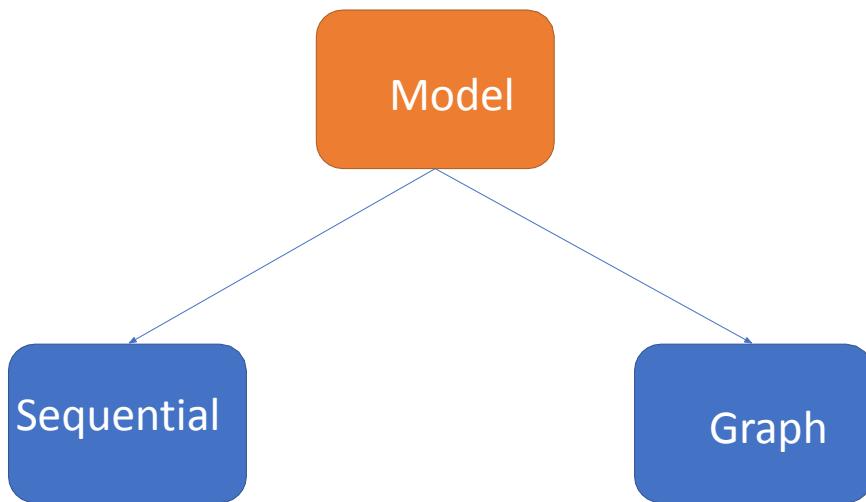
```
keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1),  
padding='valid', data_format=None, dilation_rate=(1, 1),  
activation=None, use_bias=True, kernel_initializer='glorot_uniform',  
bias_initializer='zeros', kernel_regularizer=None,  
bias_regularizer=None, activity_regularizer=None,  
kernel_constraint=None, bias_constraint=None)
```

Why use Keras?

- Easy and fast prototyping (through total modularity, minimalism, and extensibility).
- Supports both convolutional networks and recurrent networks and combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).
- Runs seamlessly on CPU and GPU.

Keras Code examples

- The core data structure of Keras is a **model**
- Model → a way to organize layers



Code-snippet

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense
from keras.optimizers import SGD

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid')) model.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)

history = model.fit(X, y, nb_epoch=10000, batch_size=4)
```

Iris classification using Keras

Import Library

In [1]:

```
import numpy
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

Using TensorFlow backend.

In [2]:

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [5]:

```
# load dataset
dataframe = pandas.read_csv("iris.csv")
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]
```

In [6]:

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
```



Define Neural Network

4 inputs -> [8 hidden nodes] -> 3 outputs

```
In [7]: # define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```



```
In [8]: estimator = KerasClassifier(build_fn=baseline_model, epochs=200, batch_size=5, verbose=0)
```

```
In [9]: kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
```

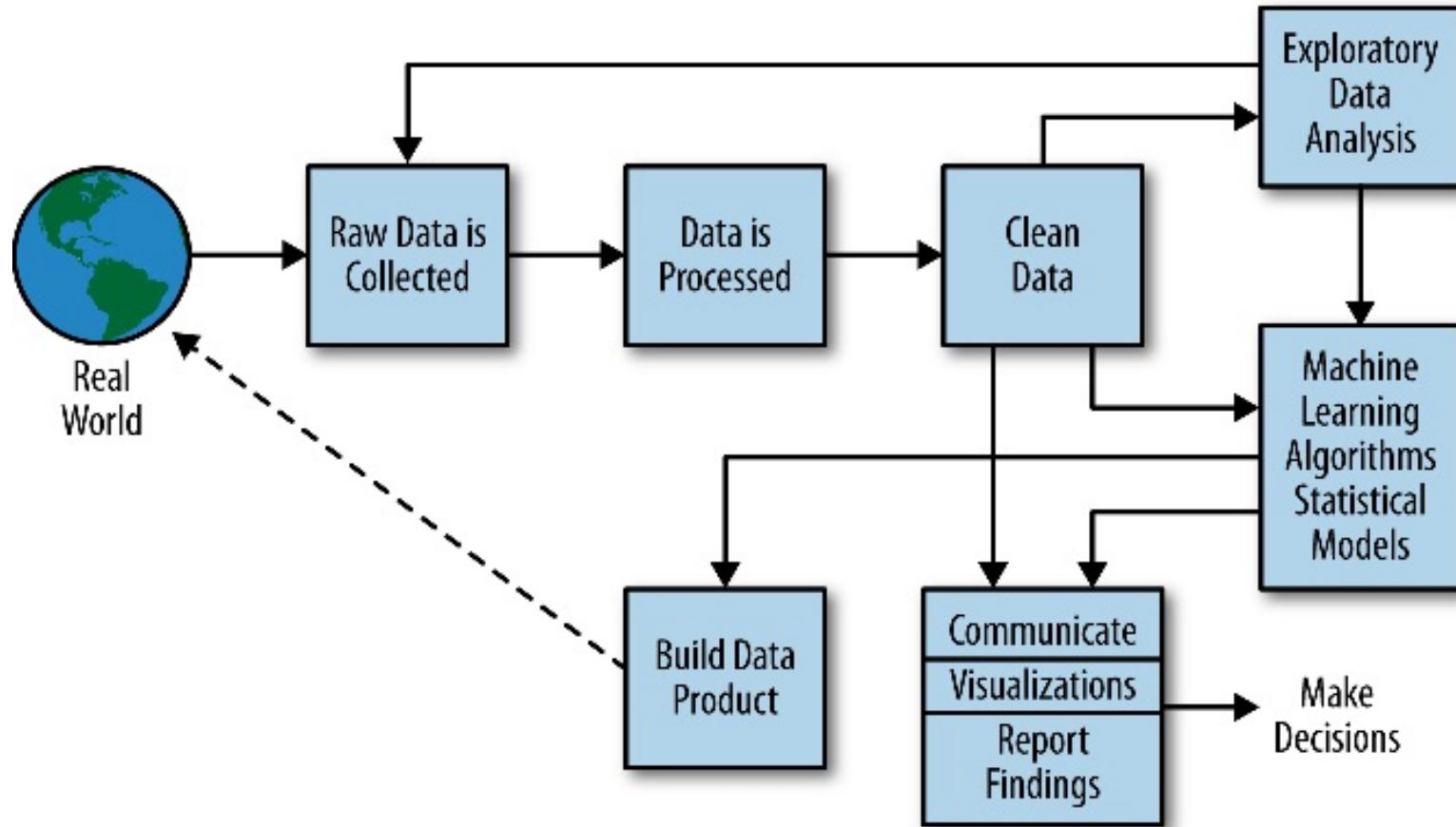
```
In [10]: results = cross_val_score(estimator, X, dummy_y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Baseline: 97.33% (4.42%)
```

Workshop 12 - Deep Learning with Keras

1. Use data from Workshop 2
2. Train your model using Keras.
3. Perform performance evaluation by creating confusion matrix

Conclusion





Contact

- ◎ Contact :
- ◎ Email: veerasak.kritsanaphan@gmail.com
- ◎ Twitter: @veerasakk