WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

For the Change Makers

Advanced Programming for Data Science

Day 2 (18th May)
Information Systems and Management
Warwick Business School
Term 3, 2018-2019

Agenda

- Day 1:
  - Data collection: web scraping using Requests and BeautifulSoup.
  - Data visualization: Seaborn
  - Data cleaning and manipulation: Numpy and Pandas.
- Day 2:
  - Data Processing
  - Conventional machine learning: clustering using Scikit-Learn
  - Deep neural network: classification using Keras
  - Case study

A simple html example

```
<!DOCTYPE html>
<html>
<head>
<title>A simple html example</title>
</head>
<body>
<p class = "name" id="first1"> Zhewei </p>
<p class = "name" id="last1"> Zhang </p>
</body>
</html>
```

- Most tags will be used in pairs with opening tag and closing tag, i.e. <head> and </head>.
- Content between a pair of opening and closing tags will be displayed accordingly, which is also called element in a html file.

Web scraping in two steps

1. Access the webpage and download the html file.
    - **Requests**
    - https://2.python-requests.org/en/master/user/quickstart/

2. Extract elements (data) from a webpage, which can be located by the tags and attributes.

    - **BeautifulSoup**
    - https://www.crummy.com/software/BeautifulSoup/

Fetch the webpage with Requests

- Requests is a HTTP library for Python that help you to make http request to the web server and fetch the webpage.
- You send a request to the web server asking for certain webpages.
- You can make **get** and **post** request.

```
import requests
url = "test.html"
result = requests.get(url)
```

Parsing your result with BeautifulSoup

- You need to parse the text retrieved by Requests: convert the plain text into structured data.
- BeautifulSoup is a parser library to create such tree-structured data by parsing HTML or XML files.

```
from bs4 import BeautifulSoup
url_html = result.text
url_content = BeautifulSoup(url_html, 'html.parser')
```

Navigate the parsed data

- Parsed result will be return as a BeautifulSoup object that has a range of methods to help you navigate and search the data.
    1. Using tag names directly.

    url_content.head # return the first matching element.

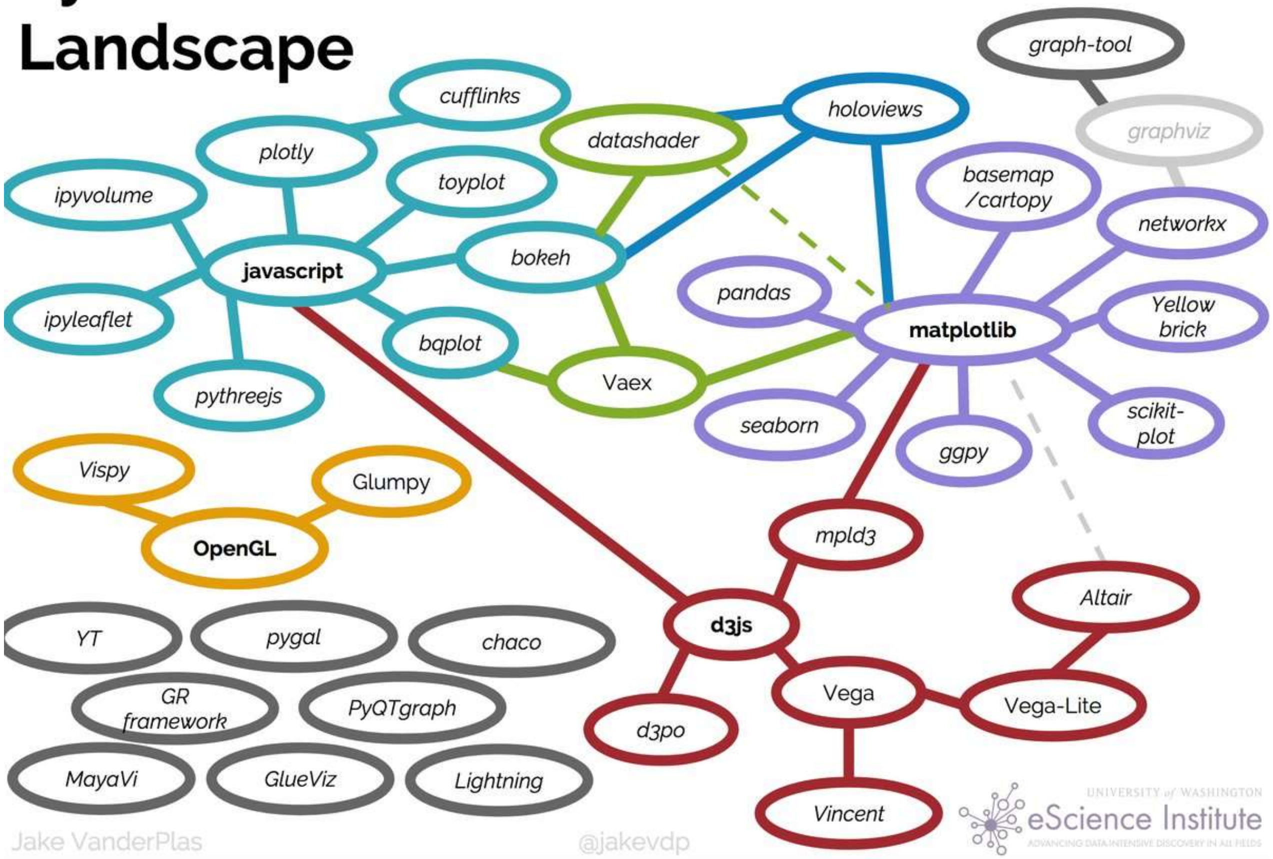
    2. Using *find()* and *find_all()*

    url_content.find_all('p') # find_all() returns a list

    3. Using select()

    url_content.select() # use CSS selectors instead of HTML tags.

Python's Visualization Landscape



- relplot(x,y,hue,style,size,data) #scalar,line
- catplot(x,y,data,hue) #box, count
- distplot(x)
- jointplot(x,y,data)
- pairplot(data)

Create DataFrame from files

- Pandas can read data directly from a wide range of file formats, such as csv, Excel, JSON, SQL database, Stata, SAS, etc. We will focus on csv files in this class.
- Use *read_csv()* function. Filename is the only required argument.

```
df_titan = pd.read_csv('titanic_train.csv')
```

- ❖ Many optional arguments can be passed when importing data.
- ❖ https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Key parameters of read_csv()

- **delimiter**: comma by default, set when necessary.
- **header**: row to be used as the column headers, and the start of data. 0 by default. Set to None if no header.

```
df_titan = pd.read_csv('titanic_train.csv',header=None)
```

- **names**: a list of column names to be used instead header.

```
df_titan = pd.read_csv('titanic_train.csv',header=None,names=[1,2])
```

- **index_col**: specify a column to be used for row index.

```
df_titan = pd.read_csv('titanic_train.csv',index_col=0)
```

Row index can also be specified with column name

```
df_titan = pd.read_csv('titanic_train.csv',index_col='PassengerId')
```

Key parameters of read_csv()

- **usecols**: import selected columns by passing a list of column **index or names**.

```
columns = [2,3,4] #columns = ['Pclass', 'Name', 'Sex']
df_titan = pd.read_csv('titanic_train.csv',usecols=columns)
```

- **skiprows**: specify the number of first n rows to skip.
- **nrows**: specify the total number of rows to read. Useful when reading large files.

```
df_titan = pd.read_csv('titanic_train.csv',skiprows=3, nrows=10)
```

Key parameters of read_csv()

- na_values: list of **strings** to be treated as NaN. Most common ones can be detected automatically, such as #N/A, n/a, null, etc.

```
missing = ['not available', 'missing']
```

```
df_titan = pd.read_csv('titanic_train.csv',na_values=missing)
```

Column selection and deletion

- Column selection using the column label:

`>>>print(df_titan['Age'])` #column label as the key, return a series

`>>>print(df_titan[['Age']])` #return a dataframe

- Add new column with label, similar as adding new item to a dictionary:

`>>> df_titan['f'] = pd.Series([10,10])` #series/list/narray

- New column can be added by calculating existing columns:

`>>> df_titan['g'] = df_titan['b'] + df_titan['f']` #NaN if one cell is Nan.

- del to delete a column:

`>>>del df_titan['g']`

Row Selection

- Row selection by passing row **labels** to loc[] method. The row will be returned as a series or a dataframe:

`>>> df_titan.loc[1]` #column labels will be used as row index.

- Multiple rows can be selected:

`>>> df_titan.loc[['a','c','e']]` #a **list** of indexes/labels, returns a dataframe

`>>> df_titan.loc['a':'c']` #slice, both start and end included.

- Column labels can be provided to filter the results:

`>>> df_titan.loc[['a','c','e'],'A']`

- Select rows with Boolean list indicating whether to be selected:

`>>> df_titan.loc[[True,False,False,True,False]]`#same length as row.

- Select rows with Boolean expression:

`>>> df_titan.loc[df8['B'] > 2]`

`>>> df_titan.loc[df8['B'] > 2,'A']`# only display column A

Add rows

- Add new rows with append() method at the end of a dataframe.

```
>>> df_titan.append([99,99]) #existing dataframe not updated.
>>> df_titan.append(pd.DataFrame([99,99]))
```

- Python will align matching columns.

```
>>> df_titan.append(pd.DataFrame([99,99],columns=['A','B']))
>>> df_titan.append(pd.DataFrame([[99,99]],columns=['A','B']))
```

- You can reset row labels by setting ignore_index=True (default is False).
- df_titan.append(pd.DataFrame([[99,99]],columns=['A','B']), ignore_index=True)

Row deletion

- You can remove rows by drop() method with a list of labels.

`df_titan.drop([4])` #same as df8.drop(4)

`df_titan.drop([0:2])` #error [0:2] is not a list. Alternatively, df8 = df8[2:].

`df_titan.drop([0,1,2])`

`df_titan.drop(range(3))` #existing dataframe not updated

`df_titan = df8.drop([4])`

`df_titan.drop([5],inplace=True)` # existing dataframe updated.

Important dataframe attributes

- .index returns a "list" of row **indexes** (numeric positions rather than labels).
- `df_titan.index`
- `df_titan.loc[df_titan['B'] > 2].index`
- Very handy to remove rows based on condition.
- `df_titan.drop(df_titan.loc[df_titan['B'] > 2].index)`

Scalar operation

- Basic arithmetic and Boolean operations with scalar data are element-wise.
- `df_titan *2` #broadcasting
- `df_titan.add(2)`
- `df_titan >2`
- `df_titan['B']*2`

Python Operator	Pandas Method(s)
+	add()
-	sub(), subtract()
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

More operations

- Arithmetic and Boolean operations with another list or Series will be performed based on matching labels (columns).
- `d10 = pd.DataFrame([[1,2,3],[3,4,5],[5,6,7]])`
- `d10 - [1,2,3]` #default to compare by column.
- `d10 > [3,3,3]`
- `d10 - [1,2]`#error, different length
- `d10 - pd.Series([1,2])` # NaN for no-match column.
- `d8 - pd.Series([1,2])`

Data processing

Explorer Dataset

- Dimensions of the dataset.

`df_titan.shape` # quick view of the dimensions.

`df_titan.size` # total number of measurements in your data.

`df_titan.info()` # count and datatype for each column

- Preview your data with head(n) and tail(n), n is 5 by default.

`df8.head(3)` # first 3 rows; `tail(3)` for last 3 rows.

Descriptive statistics

- The **`describe()`** method computes a summary (NaN will be excluded) of statistics pertaining to the DataFrame columns. Summary is also a DataFrame.

```
df_titan.describe()
```

- You may pass proper value to argument include:
  - **number** – default, only summarizes Numeric columns.
  - **object** – only summarizes String columns.
  - **all** – Summarizes all columns together (Should not pass it as a list value).

```
df_titan.describe(include='all')
```

Descriptive statistics

- You can get descriptive measurements using methods like: sum(),min(),max(),mean(),count(),median(),std(), and more. All measurements are returned as Series.

```
df_titan.mean()
```

- By default, these statistics are for each column. You can get row-based statistics by specifying axis=1.

```
df_titan.mean(axis=1)# default axis = 0
```

- To get the unique values and the corresponding counts for a column:

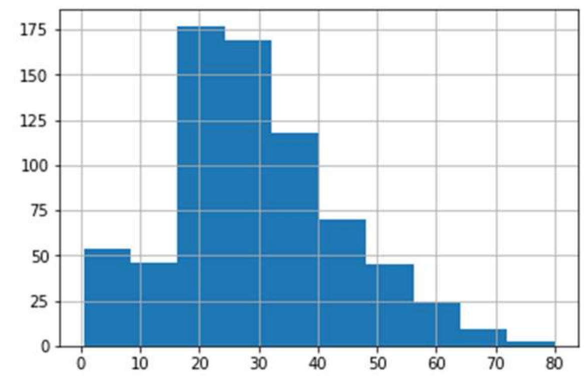
```
df_titan['Pclass'].value_counts() #only works for individual column.
```

Exploring your data with simple visualization

- Pandas offer some functions for basic plotting.
- Histogram

```
df_titan.hist()
```

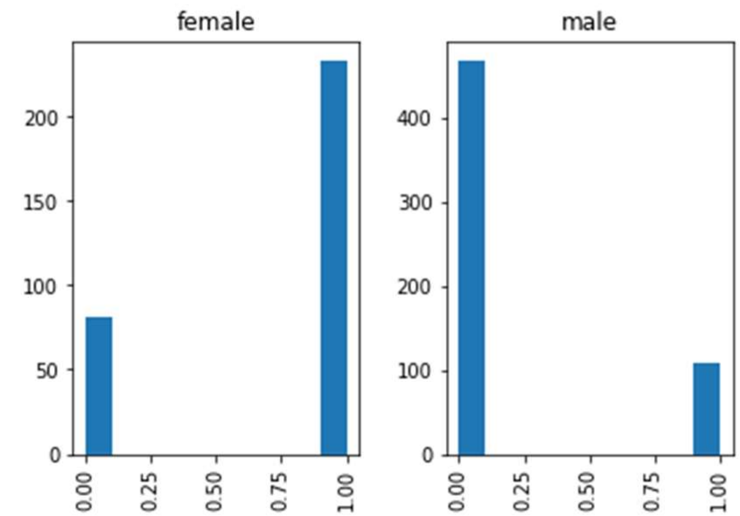
```
df_titan.hist('Age')
```



- Line
- Pie
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

- Instead of showing a single distribution, you can split based on another column.

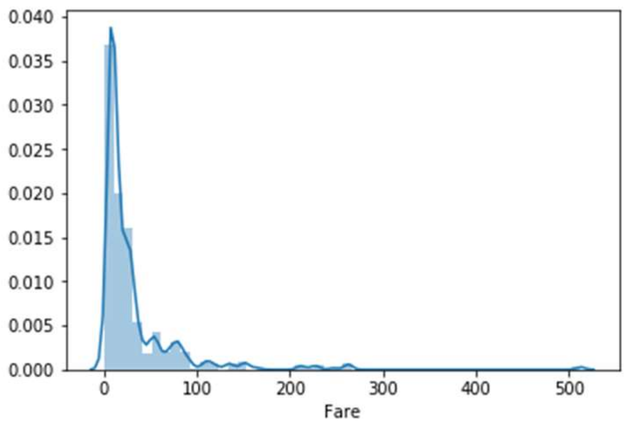
```
df_titan.hist("Survived", by="Sex")
```

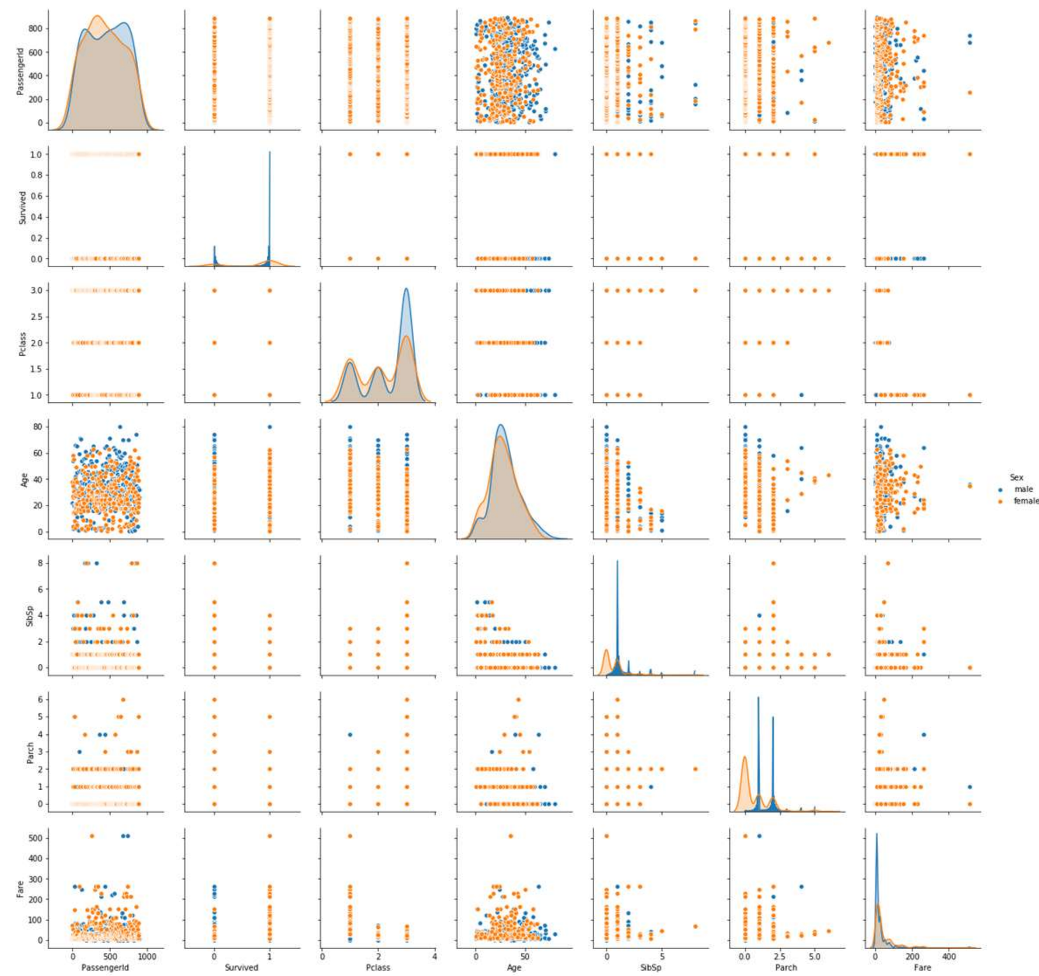


Visualization with Seaborn

- Pandas dataframe can be used directly by Seaborn as dataset, if there is no NaN value.

```
import seaborn as sns
sns.distplot(df_titan['Age'])
sns.distplot(df_titan['Fare'])
sns.distplot(df_titan['Fare'],
hue='Sex')
```

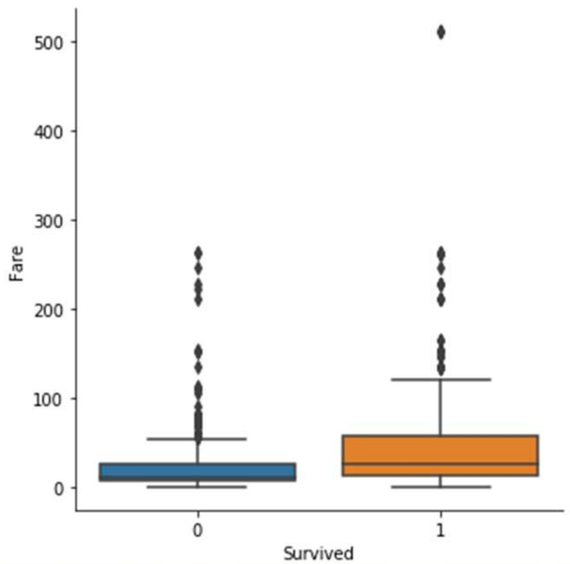


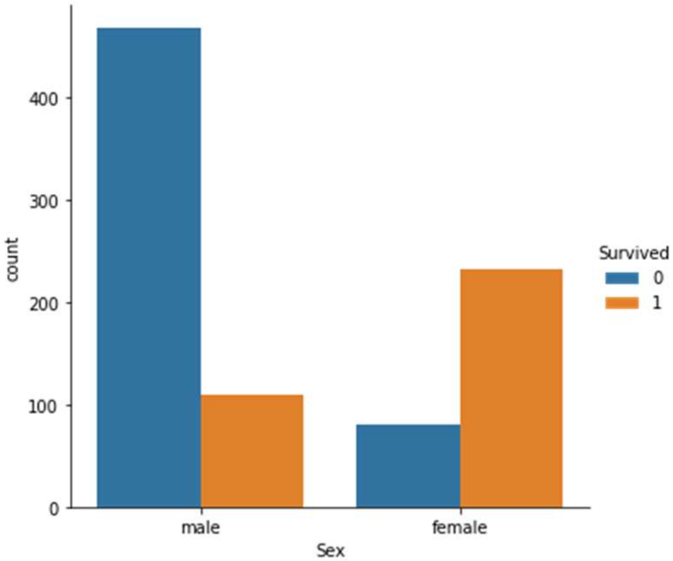


PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare
Sex
male
female

Exercise

- Can you do a quick check if people paying higher fare was more likely to survive? Was female passenger more likely to survive?





Detect missing data

In most cases, you will have missing data issue in your dataset.

Check if there is any missing data

- `df_titan.describe(include='all')` #missing data in Age and Cabin.
- `df_titan.isnull().sum()` # another trick to find out missing data.

Reasons for missing data

1. Missing Completely at Random(MCAR): The missingness has nothing to do with any other factors.
   - ❖ Age is missing due to neglect.
2. Missing at Random(MCR): The missingness is caused by other items been measured but has nothing to do with the the its own measurement.
   - ❖ Age is missing because female don't like to report their ages.
3. Missing Not at Random (MNAR): the missingness is caused by its own measurement.
   - ❖ Age is missing because elder people don't like to report their ages.

Strategies to deal with missing data

- Delete data with missing value, with **CAUTION**.
- Removing rows with missing data by using dropna().

`df_titan.dropna()` # old dataframe will not be replaced automatically.

- Removing columns with missing data by specifying axis=1

`df_titan.dropna(axis=1)` # age and cabin dropped.

- Set threshhold to remove.

`df_titan.dropna(thresh=11)` # keep rows with at least 11 non-missing data. i.e. rows missing both age and cabin get dropped.

Strategies to deal with missing data

- Impute missing data. Very tricky and a lot of consideration. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3668100/

1. Impute with a constant with `fillna()`.

`df_titan.fillna(1)` #fill all NaN with 1.

`df_titan.fillna({'Age':20,'Cabin':'B96'})` # different value for two columns.

2. Impute with mean/median/mode.

`df_titan.fillna({'Age':df_titan['Age'].mean(),'Cabin':'B96'})` #replace with mean.

3. Impute with statistic estimation (will discuss later).

Exercise

- Replace all missing ages with mean and drop the Cabin column.

```
df_titan = df_titan.fillna({'Age':df_titan['Age'].mean()}).dropna(axis=1)
```

Data Cleaning

- Remove irrelevant columns (variables) using drop(columns = [column names]).

df_titan.drop(columns = [Ticket']) # remove Ticket column from data.

- Sometimes, you may also want to remove outliers from you data.
    - For example, remove values outside 2 std of mean for normally distributed variables.

- top = df_titan['Age'].mean()+2*df_titan['Age'].std()
- bot = df_titan['Age'].mean()-2*df_titan['Age'].std()
- df_titan[df_titan['Age']>top].info()
- df_titan.drop[df_titan[df_titan['Age']>top].index]

Split multi-value columns

- Sometimes, you may want to split one column into multiple ones.
  - ➢ Datetime -> Year, Month, Date, Hour, Mins, Secs.
  - ➢ Name -> Title, First Name, Last Name.
  - ➢ Email -> Username, Domain.
- Regular expression can often do the trick.
  - Series.str can be used to access the values of the series as strings and apply several methods to it.
  - extract() is a Series.str method to capture groups in the regex pat as columns in a DataFrame.

Example

- Extract title and last name from column Name as new columns.

```
df_titan.Name.str.extract('\s(\w+)\.')
df_titan['Title'] = df_titan.Name.str.extract('\s(\w+)\.')
df_titan.Name.str.extract('^(\w+),')# NaN
df_titan.Name.str.extract('^([\w\s]+),') #NaN
df_titan.Name.str.extract('^([\w\s\']+),')
df_titan['LastName'] = df_titan.Name.str.extract('^(\D+),')
```

Derive and transform columns

- Sometimes, you may want to create new columns derived from existing ones or transform existing ones.

➢ Normalization and standardization.

➢ Log transformation.

➢ Continuous to categorial.

➢ Dummy variables.

One-hot encoding

- get_dummies(column) is a Pandas function used to create dummy variables columns based on unique values in current column. This process is also called one-hot encoding. This function returns a DataFrame.
- pd.get_dummies(df_titan['Sex'])
- df_titan[['Female','Male']] = pd.get_dummies(df_titan['Sex'])

- Normalization
- df_titan['FareNor'] =(df_titan['Fare']-df_titan['Fare'].mean())/df_titan['Fare'].std()
- Log transformation
- df_titan['FareLog'] = np.log(df_titan['Fare']) # zero division

Continuous to categorical

- We can group a range of continuous values into a category by using cut(**column**,**catergory**,labels) function. For category, you can pass three types of values:
    1. An integer: defines the number of equal-width categories.
    2. sequence of scalars : Defines the category boundaries allowing for non-uniform width.
    3. IntervalIndex : Defines the exact categories to be used.
- pd.cut(df_titan['Age'],3) # 3 groups.
- pd.cut(df_titan['Age'],[0,19,61,100]) # 3 groups with boundaries.
- df_titan['AgeGroup'] = pd.cut(df_titan['Age'], [0,19,61,100], labels = ['Minor', 'Adult','Elder'])

Derived columns

- Instead of differentiating parent/children and sibling/spouse, we are only interested in family relationships.
- df_titan['Family'] = df_titan["Parch"] + df_titan["SibSp"]
- df_titan.loc[df_titan['Family'] > 0, 'Family'] = 1
- df_titan.loc[df_titan['Family'] == 0, 'Family'] = 0

Final results

```
def data_clean1(dataframe):
    dataframe = pd.read_csv('titanic_train.csv',index_col='PassengerId')
    dataframe = dataframe.fillna({'Age':dataframe['Age'].mean()}).dropna(axis=1)
    dataframe = dataframe.drop(columns = ['Ticket'])
    top = dataframe['Age'].mean()+2*dataframe['Age'].std()
    bot = dataframe['Age'].mean()-2*dataframe['Age'].std()
    dataframe = dataframe.drop(df_titan[dataframe['Age']>top].index)
    dataframe = dataframe.drop(df_titan[dataframe['Age']<bot].index)
    dataframe['Title'] = dataframe.Name.str.extract('\s(\w+)\.')
    dataframe[['Female','Male']] = pd.get_dummies(dataframe['Sex'])
    dataframe['FareNor'] =(dataframe['Fare']-dataframe['Fare'].mean())/dataframe['Fare'].std()
    dataframe['AgeGroup'] = pd.cut(dataframe['Age'], [0,19,61,100], labels = ['Minor', 'Adult','Elder'])
    dataframe['Family'] = dataframe["Parch"] + dataframe["SibSp"]
    dataframe.loc[dataframe['Family'] > 0, 'Family'] = 1
    dataframe.loc[dataframe['Family'] == 0, 'Family'] = 0
    dataframe = dataframe.drop(columns = ['SibSp','Parch','Sex'])
    return dataframe
```

2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	71.2833	Mrs	1	0	0.795347	Adult	1
3	1	3	Heikkinen, Miss. Laina	26.0	7.9250	Miss	1	0	-0.470569	Adult	0

What Is Machine Learning?

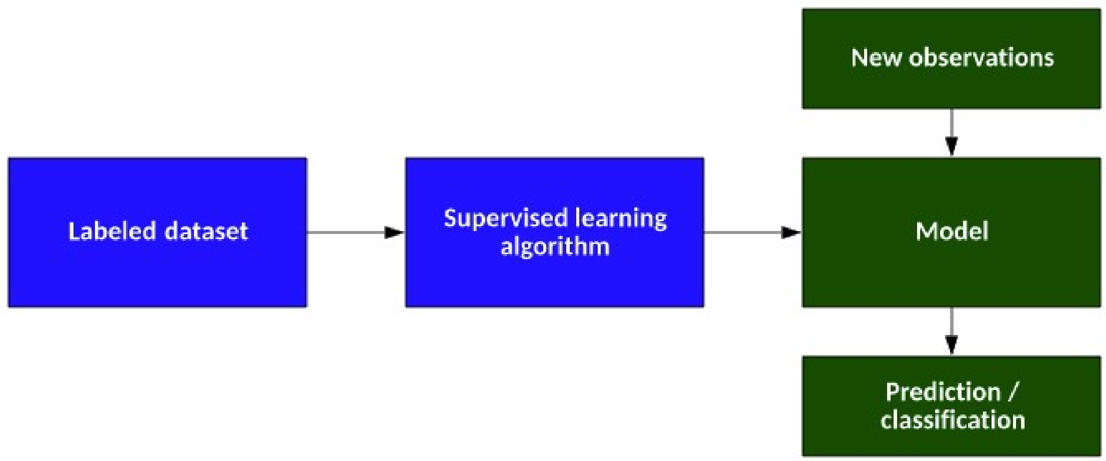
Categories of machine learning

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
  - Clustering
  - Dimensionality reduction

Supervised learning

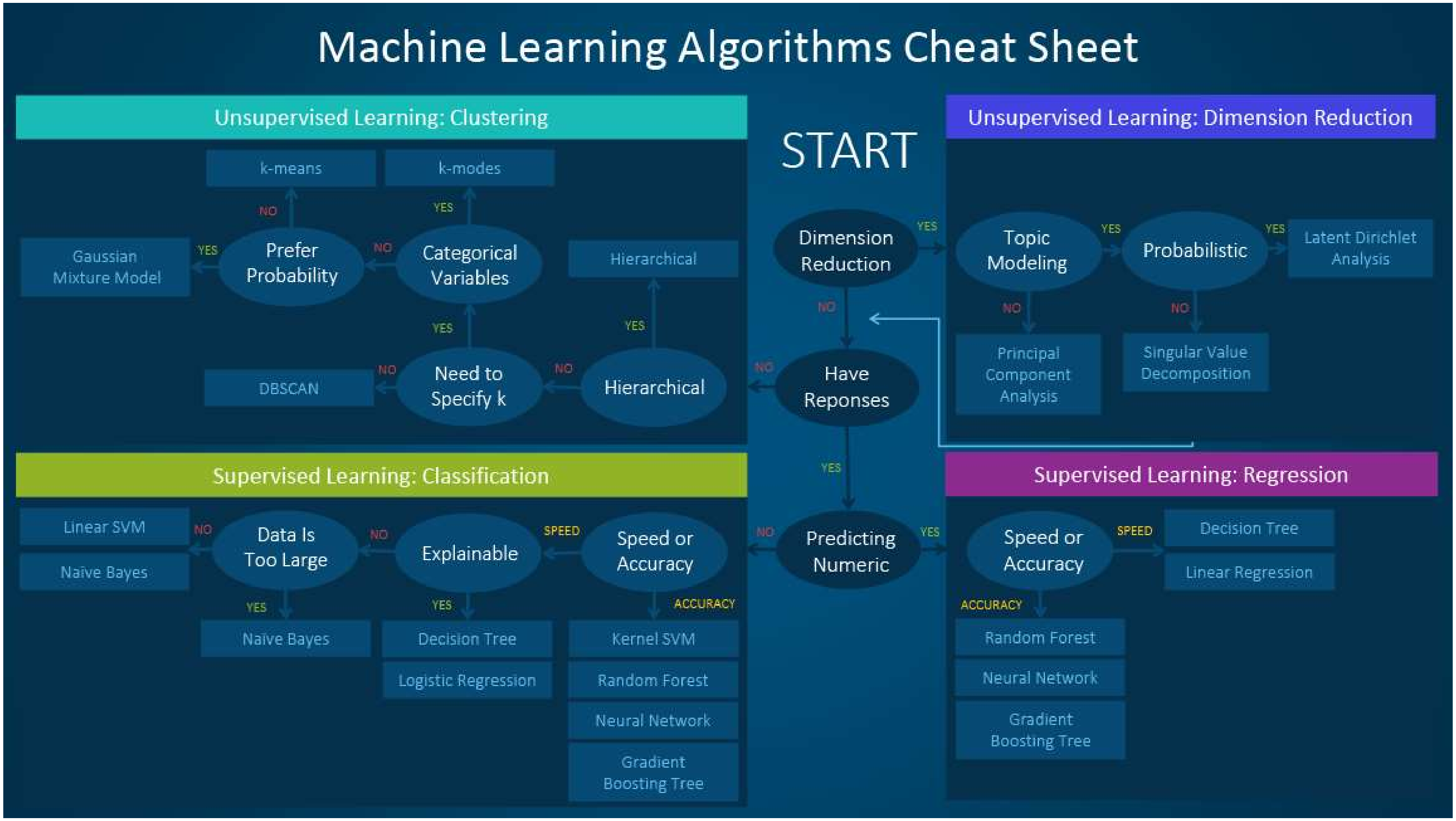
- Model training process that takes in data samples and associated outputs (known as labels or responses) to learn the relationship or mapping between inputs x and corresponding outputs y.
- This learned knowledge (model) can then be used in the future to predict an output y’ for any new input data sample x’.
- So called "supervised" as the model learns on data samples where the desired output responses/labels are already known beforehand in the training phase.
- Best for predictive tasks.

X	Y
...	...
...	...
...	...
...	...
...	...
...	...
...	...

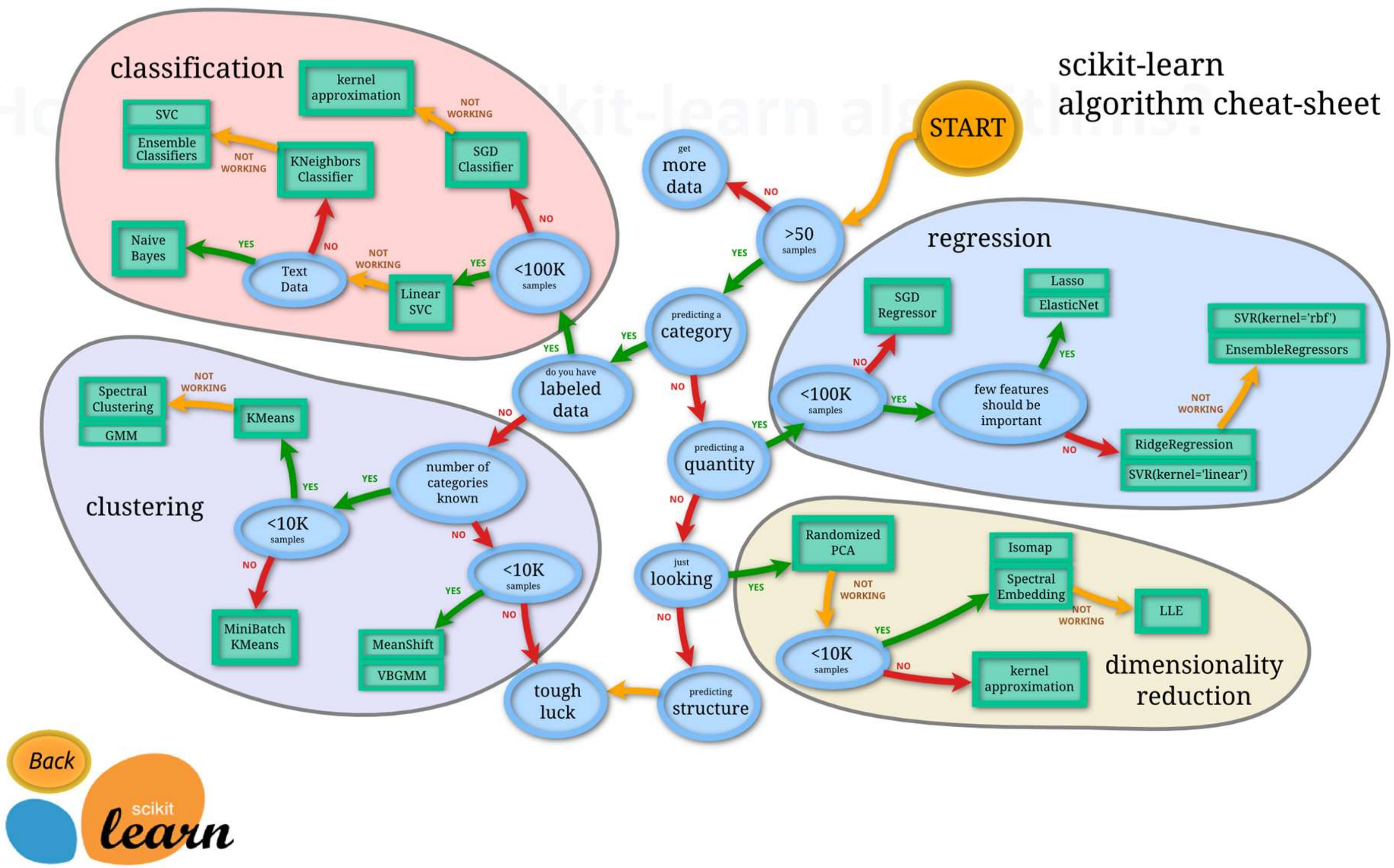


Labeled dataset
Supervised learning algorithm
New observations
Model
Prediction / classification

Unsupervised learning



https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/



https://scikit-learn.org/stable/tutorial/machine_learning_map/

scikit-learn

- Simple and efficient tools for data mining and data analysis
- Built on NumPy, SciPy, and matplotlib
  1. Classification
  2. Regression
  3. Clustering
  4. Dimensionality reduction.
  5. Model selection
  6. Preprocessing

scikit-learn

- Simple and efficient tools for data mining and data analysis
- Built on NumPy, SciPy, and matplotlib
    1. *Classification (deep learning in keras)*
    2. *Regression (deep learning in keras)*
    3. **Clustering**
    4. Dimensionality reduction
    5. **Model selection**
    6. **Preprocessing**

Preprocessing-cont.

- The **sklearn.preprocessing** package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

Imputation of missing values

- The SimpleImputer class provides basic strategies for imputing missing values.
- Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent)
- Key parameters:
  1. missing_values : number, string, np.nan (default) or None
  2. strategy : string, such as "mean" (default), "median", "most_frequent", "constant", optional
  3. fill_value : string or numerical value, optional (default=None)

Two-step imputation

- Step 1: create the imputation transformer

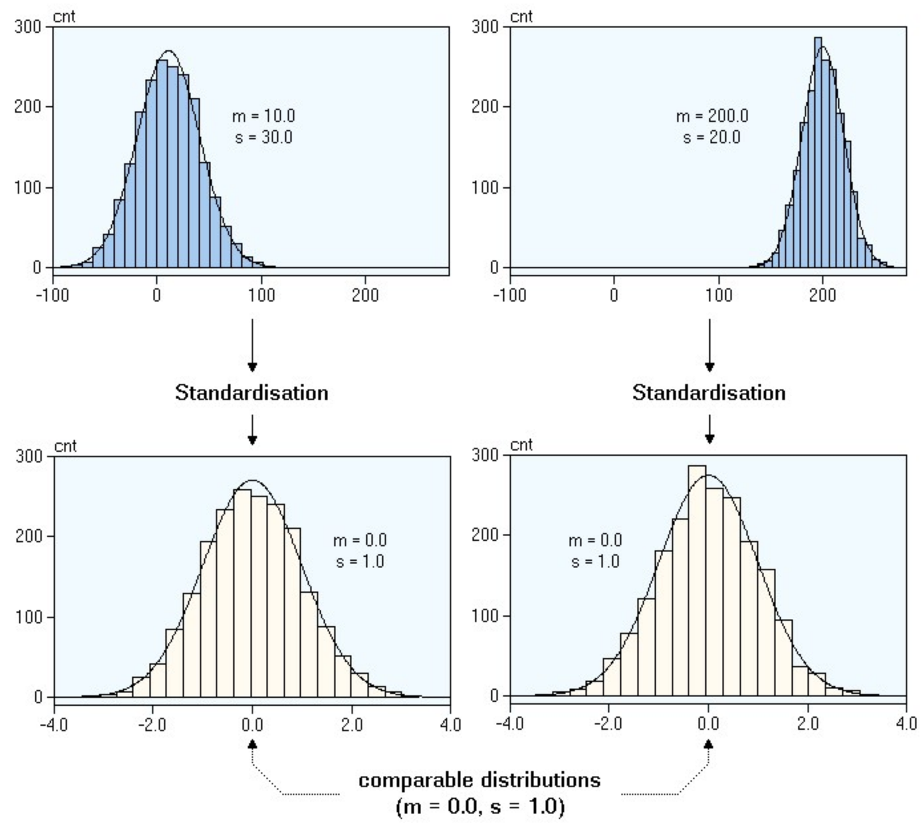
```
from sklearn.impute import SimpleImputer # import the imputer class
imp = SimpleImputer(strategy='mean') # set the imputer
```

- Step 2: apply the transformer using method `fit_transform()`, the data needs to be a dataframe/array

```
df_titan['Age'] = imp.fit_transform(df_titan[['Age']])
df_titan['Age'] = imp.fit_transform(np.array(df_titan['Age']).reshape(-1,1))
```

Deal with features in different scales

- Standardization is a statistic approach to transform your data so that they'll have the properties of a Gaussian distribution with mean of 0 and standard deviation of 1. It is in fact normalize your data.
- Scaling or rescaling (normalization) is a algebra approach to transforms your data into a range between 0 and 1 (or -1 to 1). It is in fact standardize your data.
- They both aim to solve the issue of difference scales in multivariate problems. e.g. age and fare in Titanic case.


cnt
300
200
100
0
-100
0
100
200
m = 10.0
s = 30.0
m = 200.0
s = 20.0
Standardisation
Standardisation
m = 0.0
s = 1.0
m = 0.0
s = 1.0
-4.0
-2.0
0.0
2.0
4.0
comparable distributions
(m = 0.0, s = 1.0)

Standardization

- Step 1: create the transformer StandardScaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() # set the standardization transformer.
```

- Step 2: apply the imputer to the data using method fit_transform()

```
df_titan['Age'] = scaler.fit_transform(df_titan[['Age']])
df_titan = scaler.fit_transform(df_titan)# if all numeric
```

- StandardScaler therefore cannot guarantee balanced feature scales in the presence of outliers.

Scaling

- MinMaxScaler (others like MaxAbsScaler, RobustScaler)

X_scaled = scale * X + min - X.min(axis=0) * scale

where scale = (max - min) / (X.max(axis=0) - X.min(axis=0))

MinMaxScaler transformation

- Step 1: create the transformer MinMaxScaler

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler() # set the standardization transformer.

- Step 2: apply the imputer to the data using method fit_transform()

df_titan['Age'] = scaler.fit_transform(df_titan[['Age']])

df_titan = scaler.fit_transform(df_titan)# if all numeric

- The same process can be applied with MaxAbsScaler and RobustScaler

Handling categorical features

- Most machine learning algorithms only work well with numeric input. Therefore, we need to convert strings (categorical features) into numbers.
- Convert each value in a column to a number.

	Course	Mark
1	Management	72
2	Accounting	68
3	Finance	78
4	Accounting	65
5	Management	65

	Course	Mark
1	0	72
2	1	68
3	2	78
4	1	65
5	0	65

	Management	Accounting	Finance	Mark
1	1	0	0	72
2	0	1	0	68
3	0	0	1	78
4	0	1	0	65
5	1	0	0	65

- Step 1: create the transformer OrdinalEncoder

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder () # set the standardization transformer.
```

- Step 2: apply the imputer to the data using method fit_transform()

```
df_titan['Sex'] = enc.fit_transform(df_titan[['Sex']])
df_titan = scaler.fit_transform(df_titan)# if all numeric
```

- Step 1: create the transformer OneHotEncoder

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder () # set the standardization transformer.

- Step 2: apply the imputer to the data using method fit_transform()

df_titan['Sex'] = enc.fit_transform(df_titan[['Sex']])

df_titan = enc.fit_transform(df_titan)# cannot handel missing value and returns a numpy array instead of pandas dataframe.

df_titan = pd.DataFrame(enc.fit_transform(df_titan))

Model Computation in sklearn

- Generally, the task of machine learning is done in two steps with SK-Learn:

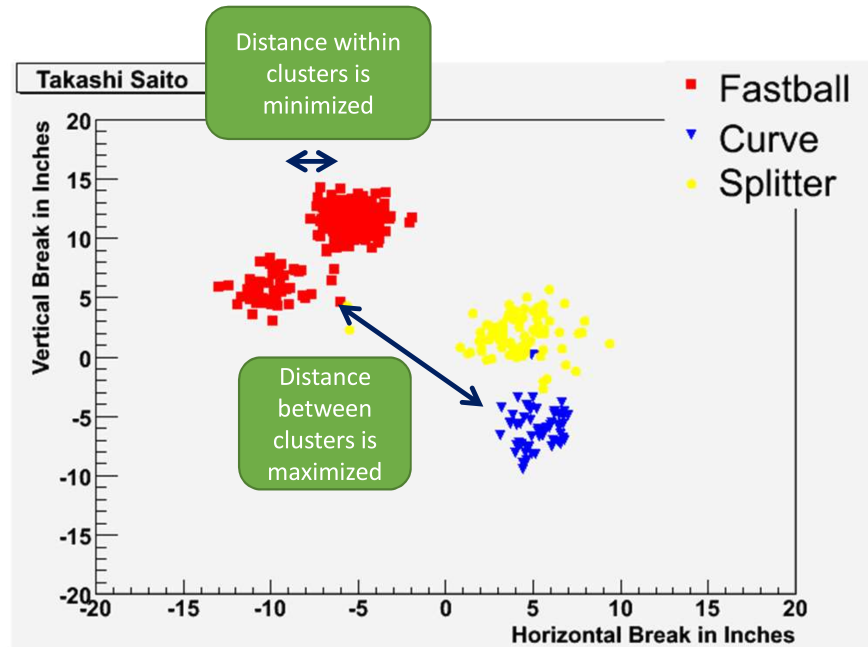
1. Learning (computing): it is usually done with the `fit()` method, which is to compute the model's parameters based on your training data.
2. Predicting: it is usually done with the `predict()` method, which is to make the prediction based on the computed model from previous step.
2. *Transformation: it is usually done with the `transform()` method, which is to convert your data to certain form based on the paratermeter generated from*

- In most cases, you can combine the two steps using `fit_predict()` method (or `fit_transform()` like our earlier examples).

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

What is Clustering?

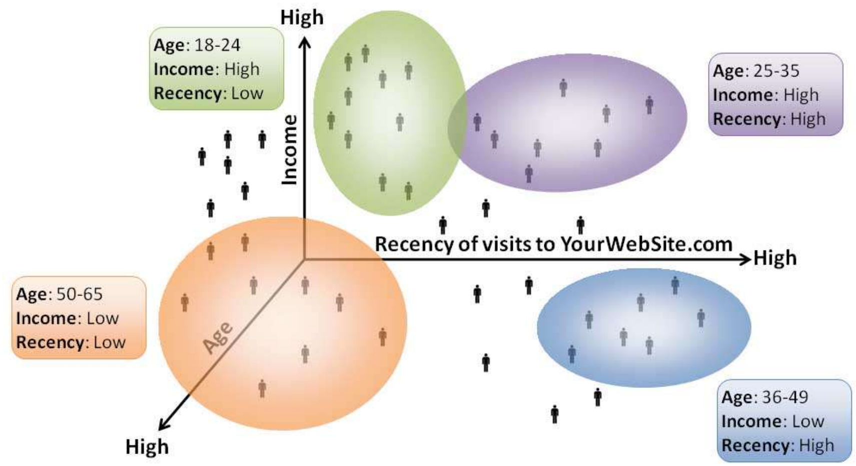
- Grouping data so that elements within a group will be
  - Similar (or related) to one another
  - Dissimilar (or unrelated) from elements in other groups



http://www.baseball.bornbybits.com/blog/uploaded_images/Takashi_Saito-703616.gif

Clustering

- Used to determine distinct groups of data
- Based on data across multiple dimensions
- Uses
  - Customer segmentation
  - Identifying patient care groups
  - Performance of business sectors



Here you have four clusters of web site visitors.

What does this tell you?

from http://www.datadrivesmedia.com/two-ways-performance-increases-targeting-precision-and-response-rates/

Applications

Understanding

- Group related documents for browsing
- Create groups of similar customers
- Discover which stocks have similar price fluctuations

Summarization

- Reduce the size of large data sets
- Those similar groups can be treated as a single data point

Even more examples

Marketing

- Discover distinct customer groups for targeted promotions

Insurance

- Finding “good customers” (low claim costs, reliable premium payments)

Healthcare

- Find patients with high-risk behaviors

What cluster analysis is NOT

Manual ("supervised") classification

- People simply place items into categories

Simple segmentation

- Dividing students into groups by last name

Main idea:

The clusters must come from the data, not from external specifications.

Creating the "buckets" beforehand is categorization, but not clustering.

Clusters can be ambiguous

How many clusters?

6

2

4

The difference is the threshold you set.
How distinct must a cluster be to be it's own cluster?

Two clustering techniques

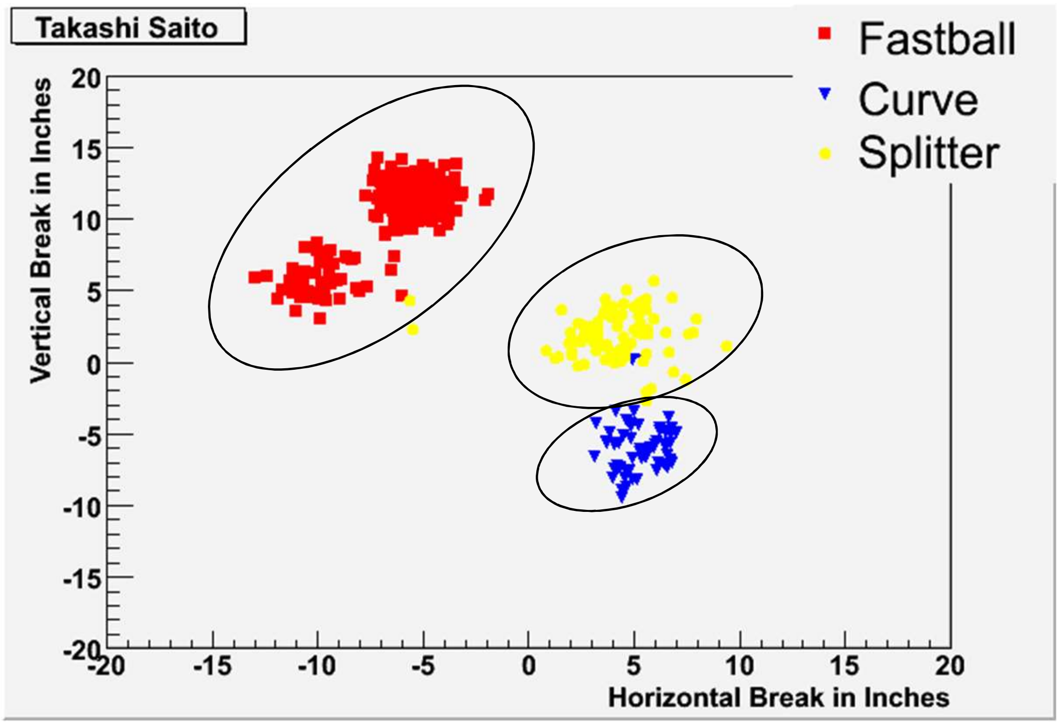
Partition

- Non-overlapping subsets (clusters) such that each data object is in exactly one subset

Hierarchical

- Set of nested clusters organized as a hierarchical tree

Partitional Clustering

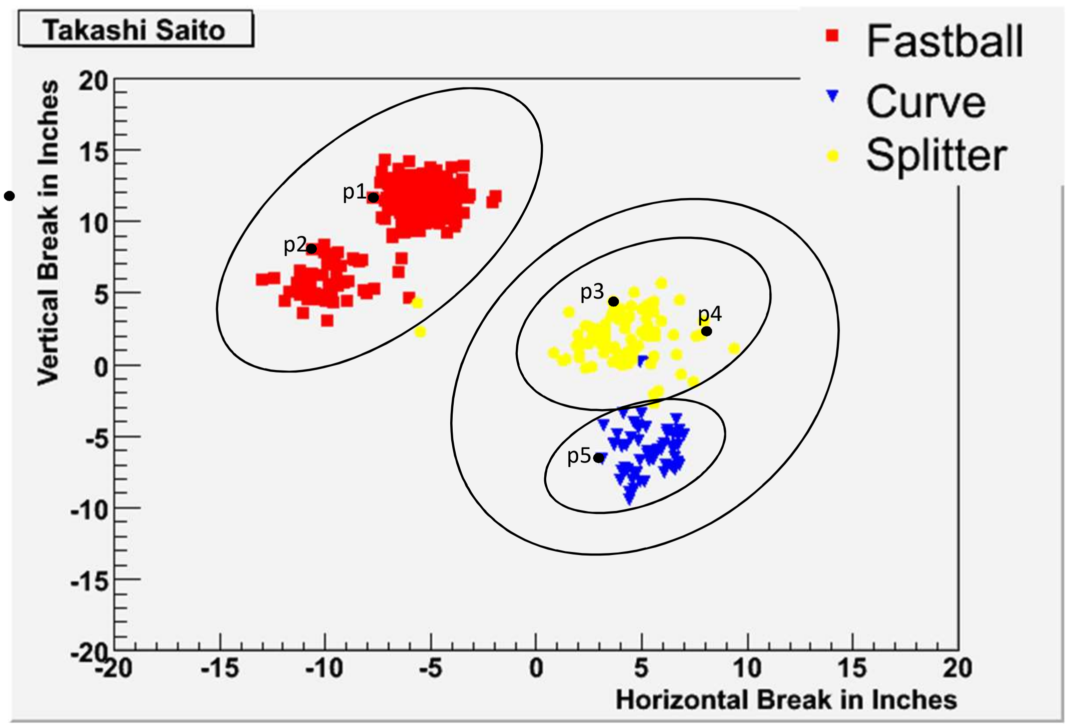


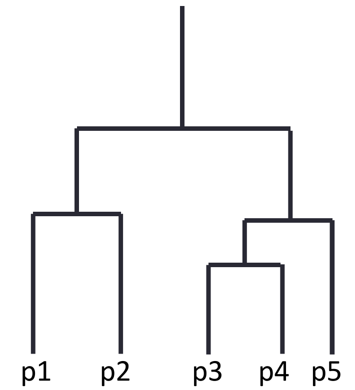
Three distinct groups emerge, but…

…some curveballs behave more like splitters.

…some splitters look more like fastballs.

Hierarchical Clustering

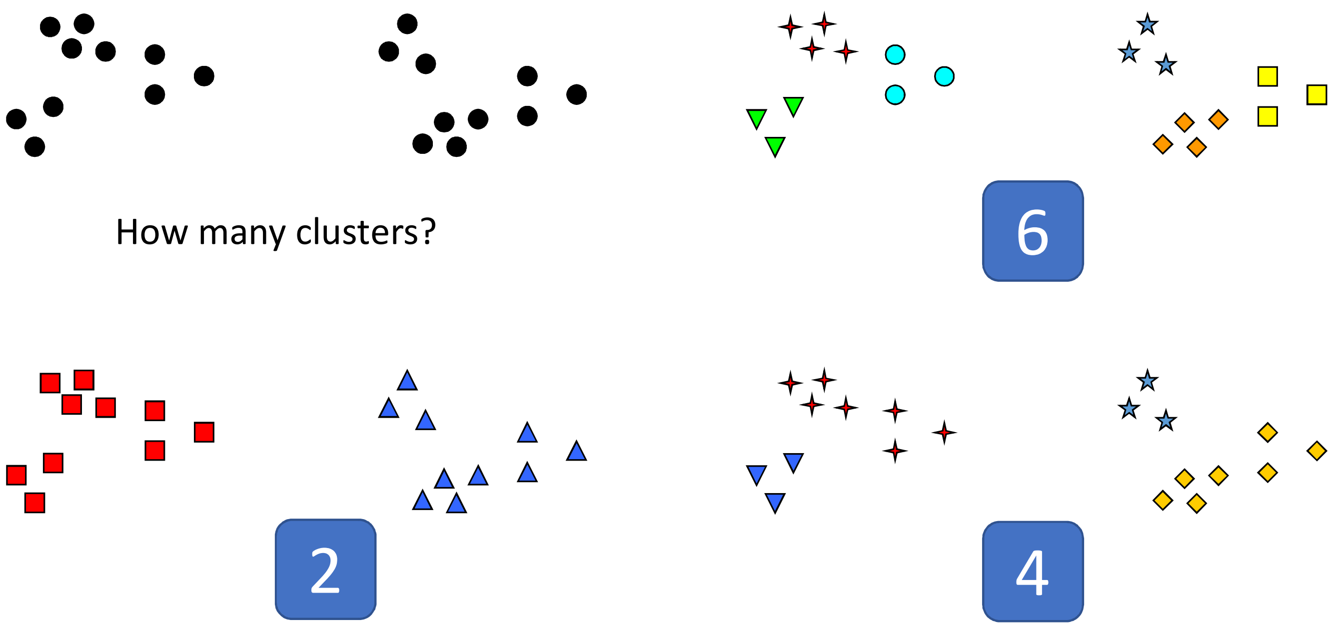




This is a dendrogram

Tree diagram used to represent clusters

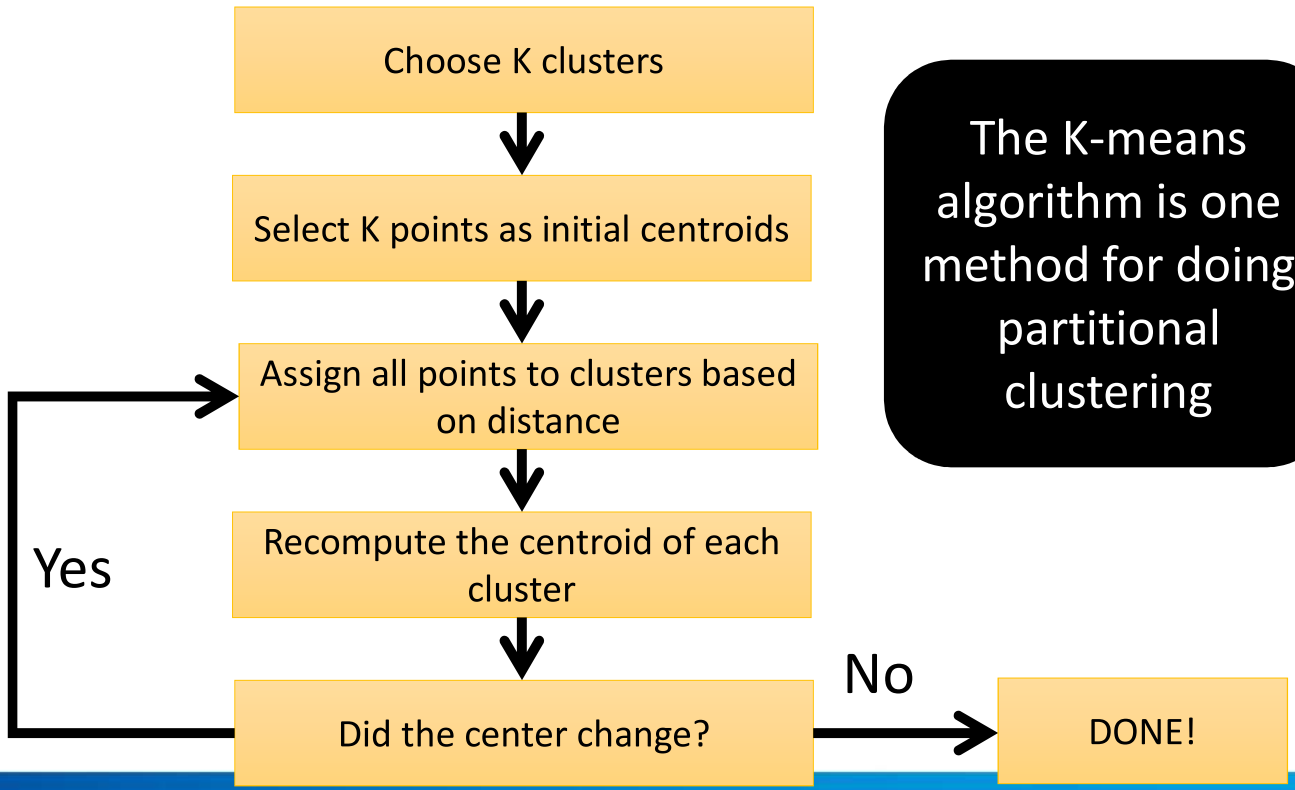
Clusters can be ambiguous



The difference is the threshold you set.
How distinct must a cluster be to be it’s own cluster?

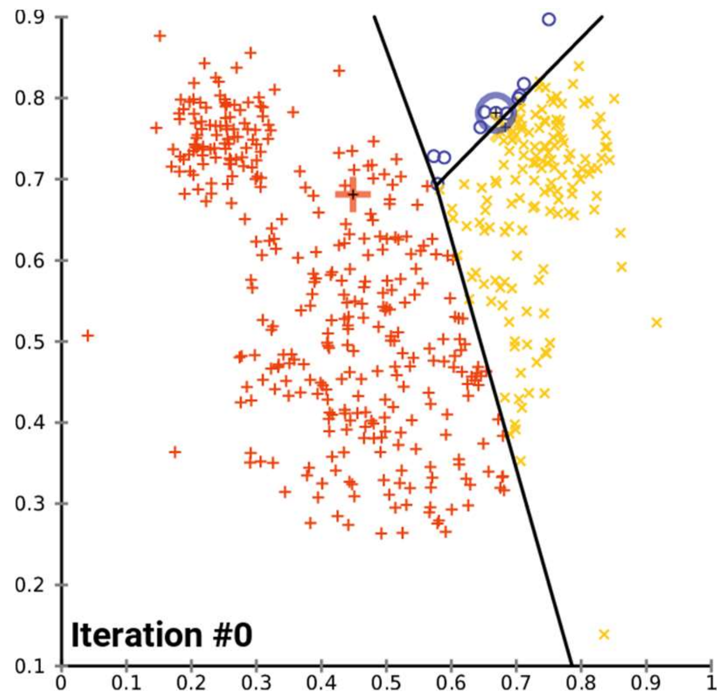
adapted from Tan, Steinbach, and Kumar. Introduction to Data Mining (2004)

K-means (partitional)



The K-means algorithm is one method for doing partitional clustering

K-Mean clustering



Choosing the initial centroids

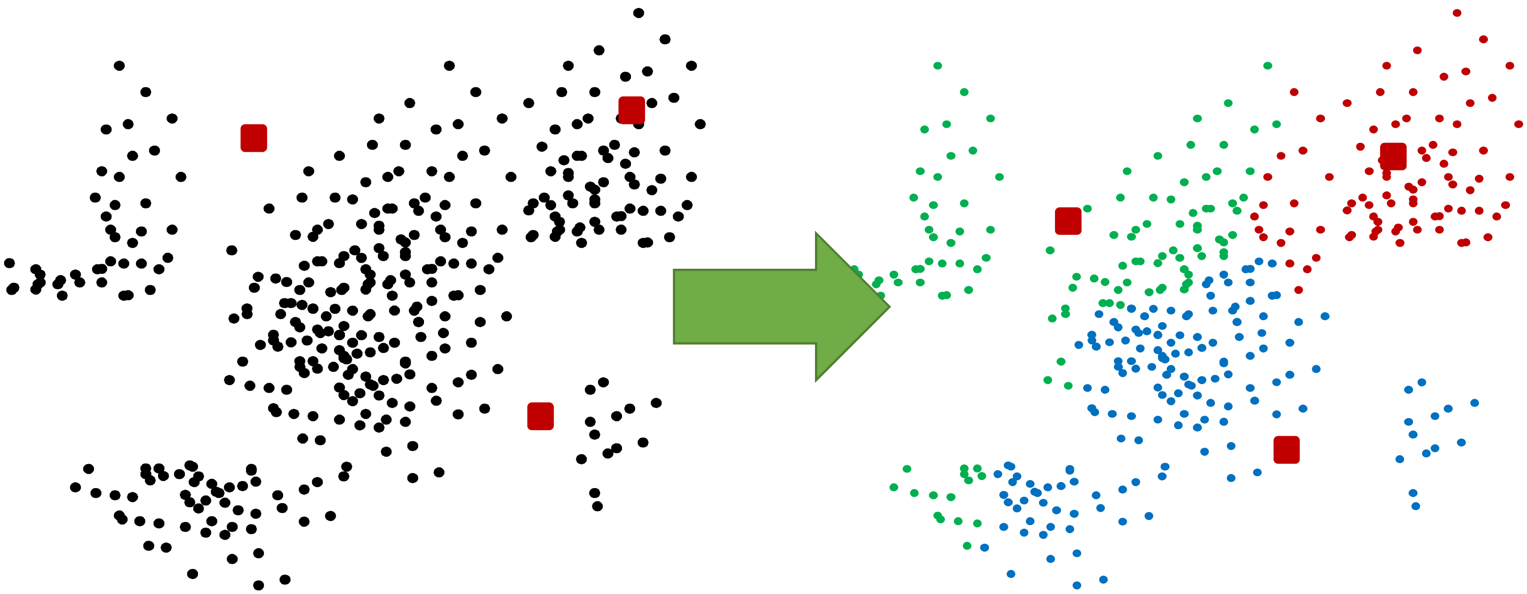
It matters

- Choosing the right number
- Choosing the right initial location

Bad choices create bad groupings

- They won’t make sense within the context of the problem
- Unrelated data points will be included in the same group

Example of Poor Initialization



This may "work" mathematically but the clusters don't make much sense.

Limitations of K-Means Clustering

K-Means gives unreliable results when

- Clusters vary widely in size
- Clusters vary widely in density
- Clusters are not in rounded shapes
- The data set has a lot of outliers

The clusters may never make sense.
In that case, the data may just not be well-suited for clustering!

Scikit-learn for K-Mean clustering

- K-Mean clustering can be with scikit-klearn easily :

```
from sklearn.cluster import KMeans
```

- You need to first initialize the classifier by creating a kmean classification model object:

```
kmeans = KMeans(n_clusters=k)
```

n_clusters is the only required argument to specify the number of clusters you want.

Important arguments

- Besides the number of clusters, there are few important arguments you should be aware of:
- algorithm: default “auto”, “full” or “elkan”.
- max_iter: a number, default 300. Specify the maximum number of iterations of the k-means algorithm for a single run. Depending on your dataset, it may never converge.
- n_jobs: default "None" or number. Specify how many concurrent processes/threads should be used for parallelized routines. None means to use 1, -1 means to use all processor.

- `kmeans = KMeans(algorithm='auto', max_iter=300, n_clusters=4, n_jobs=1)`
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

The Iris example

```
import pandas as pd
import numpy as np
import seaborn as sns
iris = sns.load_dataset('iris')
```

Let's do some exploration first.

Since clustering works with unlabelled data, we need to first remove the label "species" from our dataset:

```
iris_train = iris[['sepal_length','sepal_width','petal_length','petal_width']]
columns = iris.columns[0:4]
iris_train = iris[columns]
```

- Initialize a basic k-mean classifier. KMeans() creates a classifier object.

kmeans = KMeans(n_clusters=3, max_iter = 100)

- Compute the model's parameters based on the "training" data. fit() performs the learning process and returns a computed classifier.

kmeans.fit(iris_train)

- Use the computed classifier to "predict" the classification of your data. It returns a series of predicted labels.

prediction = kmeans.predict(iris_train)

Evaluation

- Since clustering is an unsupervised learning, there is no "right" answer to be tested against. You are not trying to predict if one observation is accurately classified into a specific group. E.g. row 1 belongs to setosa. Instead, you are grouping similar observations into the same group, whether this group is setosa or not is not relevant.
- Therefore, if you have some observations with known groups, the evaluation of clustering can be done with V-measure.
- If you have no observation with known groups, then the evaluation can be done with Silhouette Coefficient

V-measure

- V-measure is an score by calculating the mean of homogeneity and completeness. The measurements are computed by comparing the memberships of known groups to memberships of predicted groups.
- homogeneity: each cluster contains only members of a single class.
- completeness: all members of a given class are assigned to the same cluster.
- from sklearn import metrics
- metrics.homogeneity_score(labels_true, labels_pred)
- metrics.completeness_score(labels_true, labels_pred)
- metrics.v_measure_score(labels_true, labels_pred)

```
from sklearn import metrics
metrics.homogeneity_score(iris['species'], iris['predicted']) #0.75148
metrics.completeness_score(iris['species'], iris['predicted']) #0.76498
metrics.v_measure_score(iris['species'], iris['predicted']) #0.75817
```

- Bounded scores: 0.0 is as bad as it can be, 1.0 is a perfect score.
- No assumption is made on the cluster structure

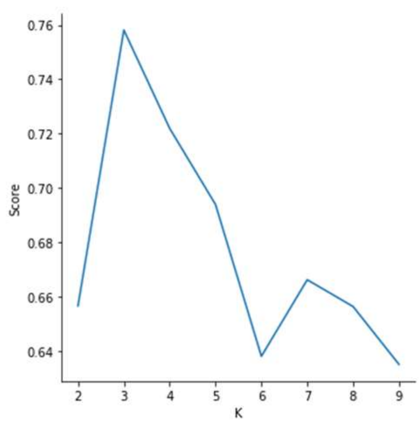
Silhouette Coefficient

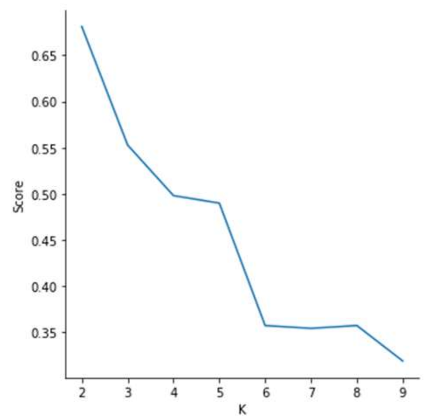
- Silhouette Coefficient can be used when there is no known labels. It only relies on assessing the distance between the observations within the same group and across different groups.
  - **a**: The mean distance between a sample and all other points in the same class. (similarity within the same cluster)
  - **b**: The mean distance between a sample and all other points in the *next nearest cluster*. (dissimilarity between different clusters)
  - s=(b-a)/max(a,b)
- metrics.silhouette_score(X, labels, metric='euclidean')

- from sklearn import metrics
- metrics.silhouette_score(iris_train, prediction, metric='euclidean') #0.55281
- The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering.

Finding the right number of clusters

- No golden rules.
- Reality and cost.
- Comparing the evaluation metrics.





Supervised learning: regression vs. classification

- A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:
- What is the value of a house in California?
- What is the probability that a user will click on this ad?
- A classification model predicts discrete values. For example, classification models make predictions that answer questions like the following:
- Is a given email message spam or not spam?
- Is this an image of a dog, a cat, or a hamster?

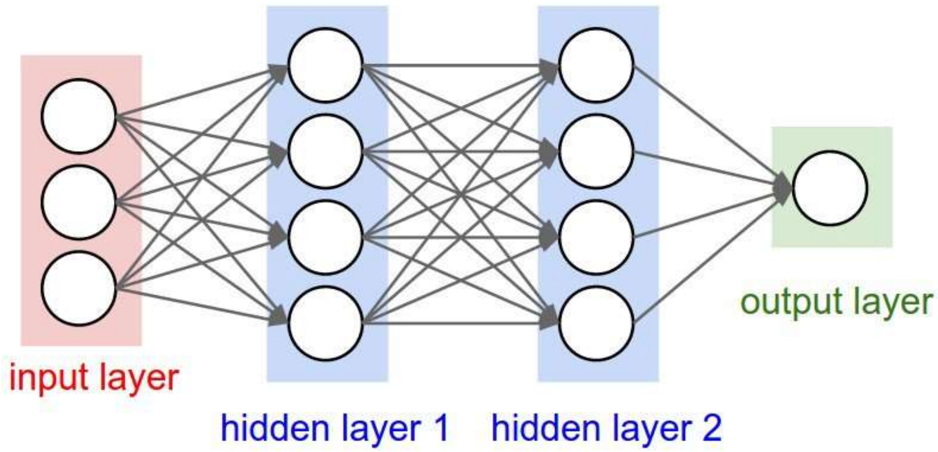
First "deep" neural network

```
from sklearn import datasets
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
dataset = datasets.load_iris()
train = dataset.data
labels = dataset.target
labels = pd.get_dummies(labels)
network = Sequential()
network.add(Dense(8, activation='relu', input_dim=4))
network.add(Dense(3, activation='softmax'))
network.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
network.fit(train,labels,epochs=100,batch_size=5)
```

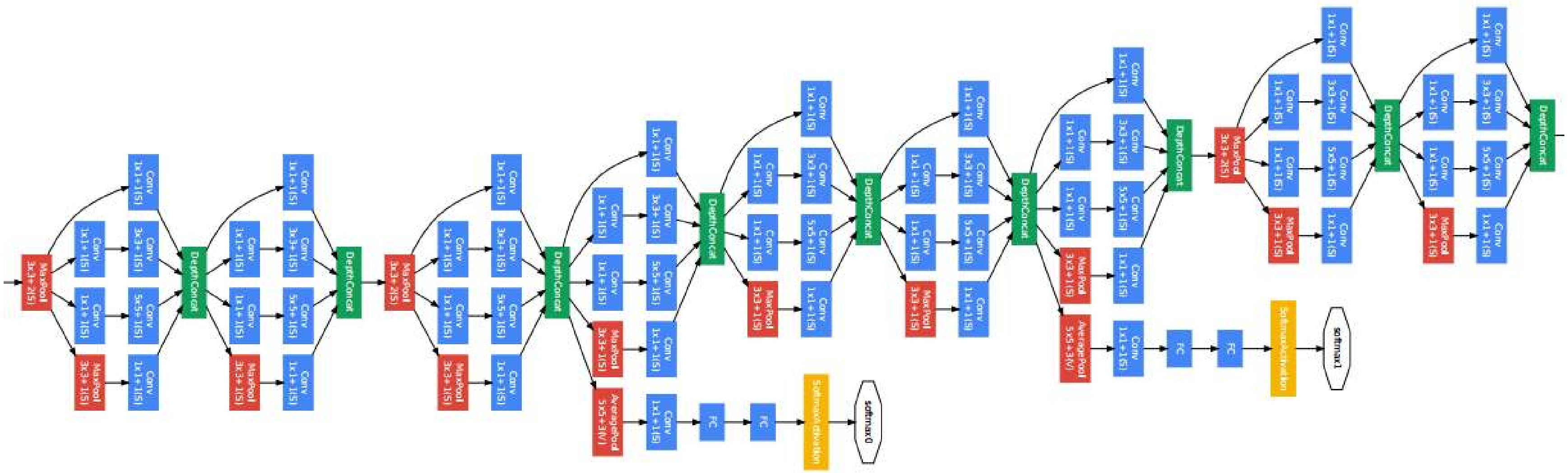
```
network = Sequential()
network.add(Dense(8, activation='relu', input_dim=4))
network.add(Dense(3, activation='softmax'))
network.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
network.fit(train,labels,epochs=100,batch_size=5)
```

network = Sequential()

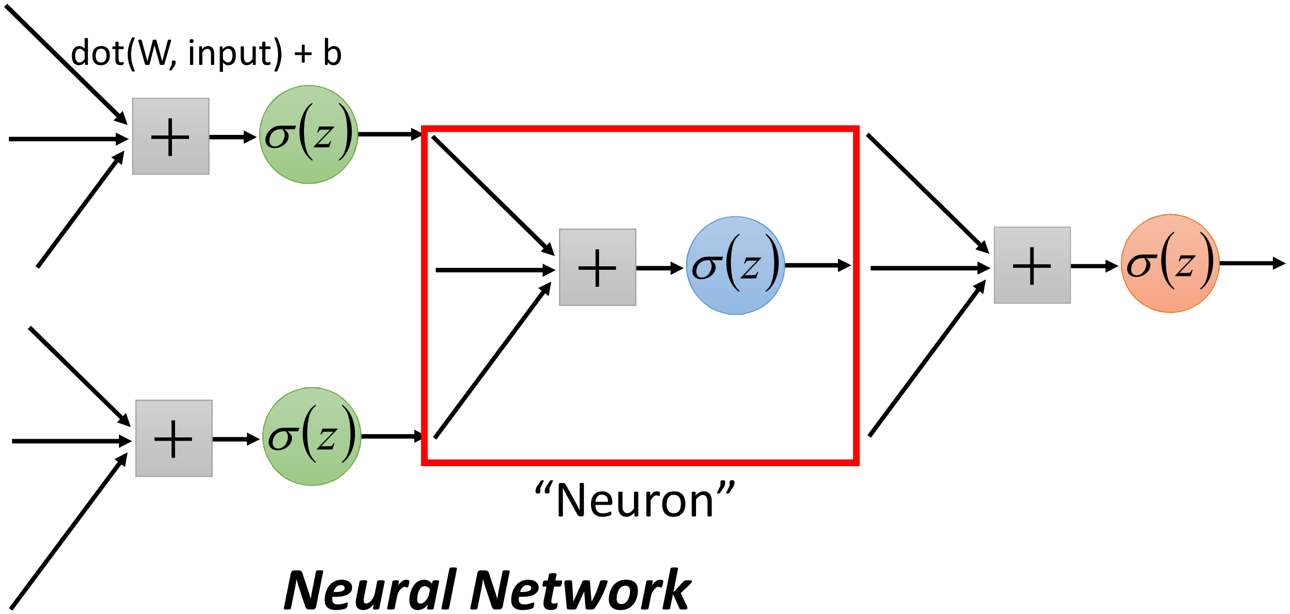
- A deep neural network is basically a series of layers of "neurons".
- A 3-layer fully connected network.



https://towardsdatascience.com/coding-neural-network-forward-propagation-and-backpropagtion-ccf8cf369f76



A close look of neural network



Neural Network

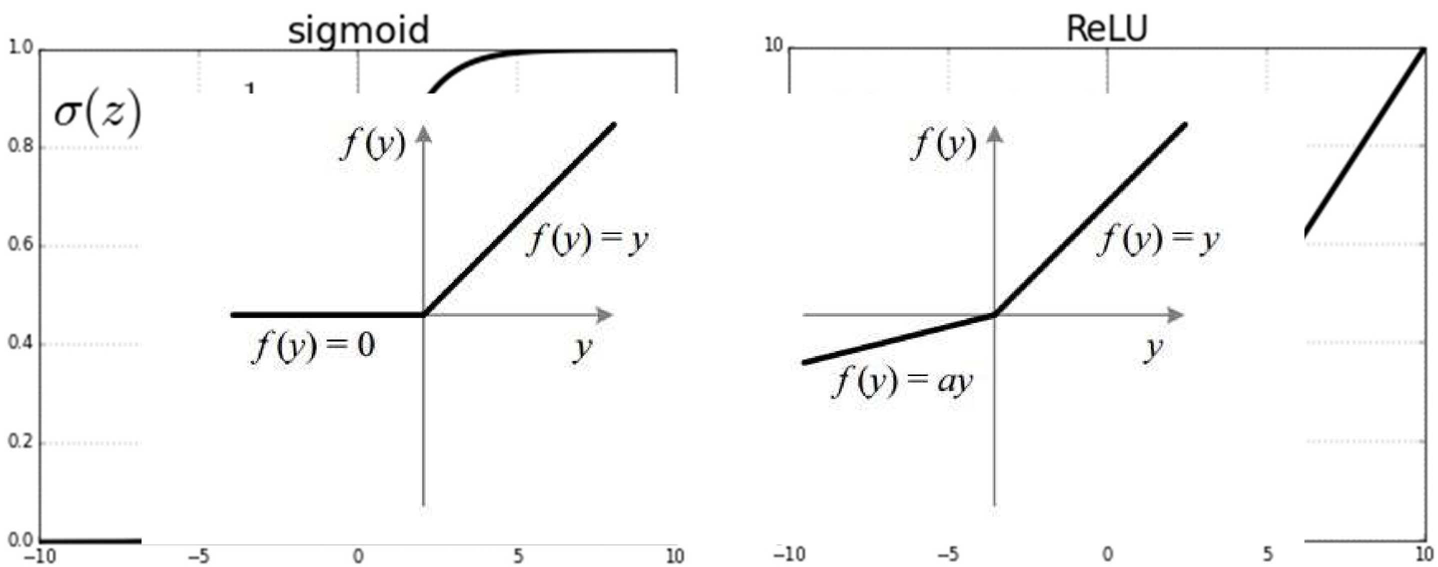
network.add(Dense(8, activation='relu', input_dim=4))

- Creating a deep neural network is like building lego set.
- You can different layers sequentially.
- We add our first layer, which is a dense layer, with a 4 dimensional input, 8 dimensional output, and an activation function of "relu".
  - Dense means it is a densely-connected (also called "fully-connected") neural layer.
  - It takes input from our original training set, which is a 4-dimensional data.
  - It outputs an 8-dimensional result for next layer.

Activation function

- Activation function is the function to determine the final outcome of the neuron.
- It decides if the neuron is activated or not.
- Therefore, activation function is a non-linear function that generally transforms the linear operation output into a value between 0 to 1.
- Why?
- Stacking of linear functions is still a linear function, which defeat the purpose of multiple layers.

Common activation functions

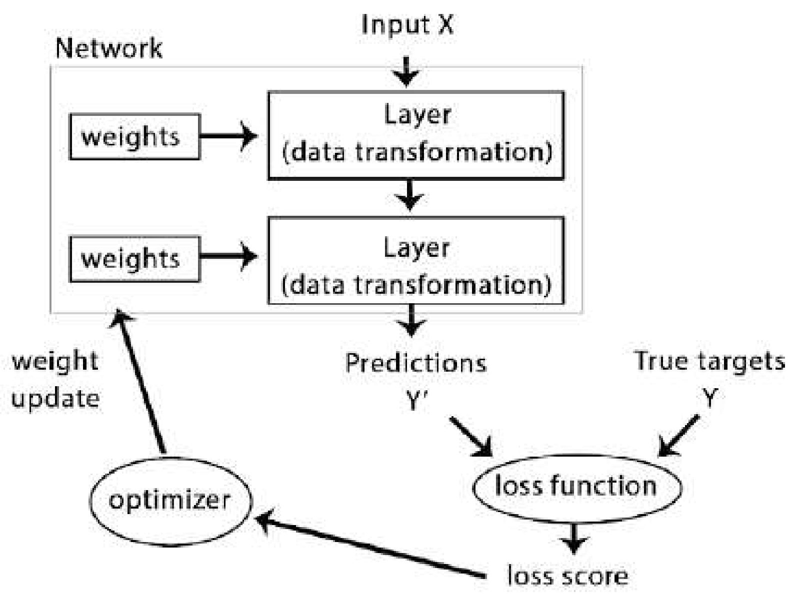


network.add(Dense(3, activation='softmax'))

- We further add the second layer, which is also the last layer, the output layer.
  - It is still a dense layer.
  - From the second layer, you don't need to specify the input shape as the model will automatically determine it from the previous layer.
  - Since this is the last layer, we aim to produce 3 groups of results. Thus the output dimension is 3.
  - We use an activation function 'softmax', which is normally only used in the last layer for classification problems.

network.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

- Overall flow



Loss function

- Loss function is a function used to assess how well does the model perform in terms of predicting the expected results. Most commonly used include:
  1. MSE (Mean Square Error)
  2. Binary_crossentropy
  3. Categorical_crossentropy

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multi-class, single-label classification	`softmax`	`categorical_crossentropy`
Multi-class, multi-label classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	`None`	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Optimizer

Learning: Weight update

For each neuron, output = relu(dot(W, input) + b). Initially, W and b will be assigned randomly. They will then get updated based on the loss and optimizer.

gradient-based optimization

- Develop a model that does better than a baseline.
    1. Choice of the last-layer activation.
    2. Choice of loss function.
    3. Choice of optimization configuration.
- Scale up: develop a model that overfits
    1. Add layers.
    2. Make your layers bigger.
    3. Train for more epochs.
- Regularize your model and tune your hyperparameters.
- Dropout
- Different architectures.
- Different hyperparameters.
- Iterate feature engineering.

Solutions to overfitting

- Getting more training data.
- Reducing the capacity of the network.
- Adding weight regularization.
- Adding dropout.

Regularization

- Reduce the size of network.
  - Number of learnable parameters in the model.
  1. Number of layers.
  2. Number of units per layer
  3. Start with simpler network and increase the complexity.

- ADDING WEIGHT REGULARIZATION
  - For the same number of parameters, the simpler models are less likely to overfit.
  - adding cost to weights, forcing its weights to only take small values

Adding dropout

- Dropout, applied to a layer, consists of randomly "dropping out" (i.e. setting to zero) a number of output features of the layer during training.
- [0.2, 0.5, 1.3, 0.8, 1.1] -> [0, 0.5, 1.3, 0, 1.1]
- Dropout rate: 0.2 to 0.5 generally.
- At test time, no units are dropped out, and instead the layer’s output values are scaled down by a factor equal to the dropout rate,

Scikit-Learn

encoding

Train/validation split

K Fold

Keras Wrapper

- Classifier

Keras Wrapper

- Model search

Workflow of machine learning

- Define the problem and assemble a dataset.
  - Input and output.
  - Machine learning problem
- Pick a measure of success
  - Accuracy for balanced
  - Precision recall for unbalanced.
- Decide on an evaluation protocol
  - Hold-out validation set.
  - K-fold cross validation.
  - Iterated K-fold validation.
- Prepare your data
  - Formated as tensors.
  - Scale to small values.
  - Normalization.
  - Feature engineering