EPL451: Data Mining on the Web – Lab 5

University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01

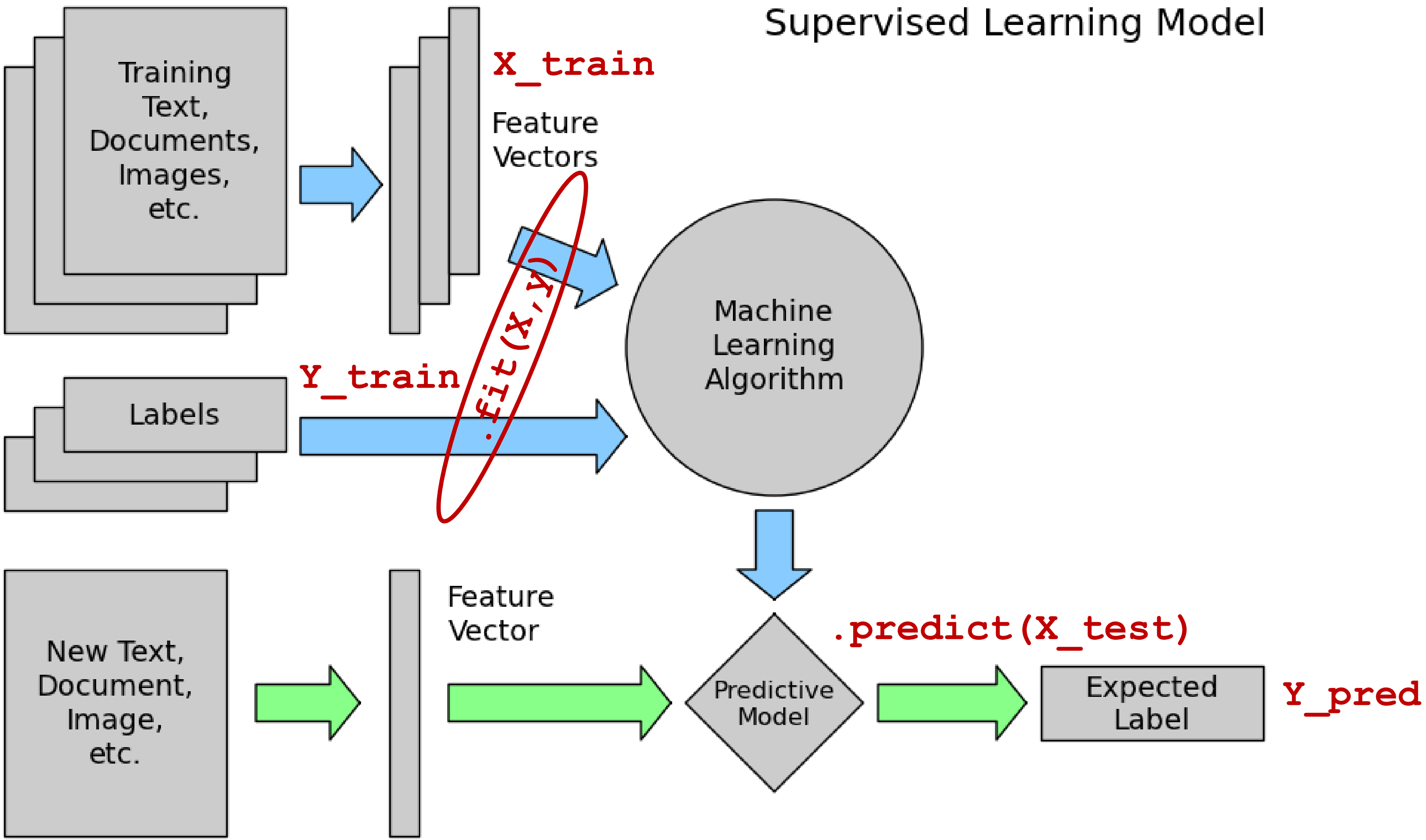
Predictive modeling techniques

- IBM reported in June 2012 that 90% of data available created in the past 2 yrs [Ref]
- Predictive modeling techniques help translate vast amount of data into value
  - Examples: Neural Networks (NNs), Support Vector Machines (SVMs), regression models, classification techniques, clustering techniques, …
- Data + Predictive Modeling Technique → Predictive Model
  - Learning/training phase:
    - Past (historical) data are used to train one or more predictive modelling techniques
    - Goal: find a mapping between a set of input variables (features) and an output (target) variable using training data (in some cases no output is available – see supervised vs unsupervised learning)
  - Validation/testing phase:
    - Select the best performing modeling technique using validation data
    - Estimate the accuracy of the selected technique using test data
  - Prediction (application) phase:
    - Apply predictive model to real-world input data with and predict output
  - Split initial dataset into 3 smaller datasets:
    - Usually: Train – Validation – Test : 60% – 20% – 20%

Supervised learning

- You have input variables (X) and an output variable (y) and you use a technique to learn the mapping function from the input to output
  - Majority of predictive techniques use supervised learning
- Supervised learning problems can be further grouped into:
  - **Classification problems**: A classification problem is when the output variable is a category, such as “disease” or “no disease” (binary classification) and “red” or “blue” or “green” (multiclass classification)
    - Popular techniques: Logistic Regression (binary classification), Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machine (SVM), Naïve Bayes, Gaussian Naïve Bayes
  - **Regression problems**: A regression problem is when the output variable is a real value, such as “price” or “weight”
    - Popular techniques: Linear Regression, Non-linear Regression, Support Vector Regression (SVR), Random Forests

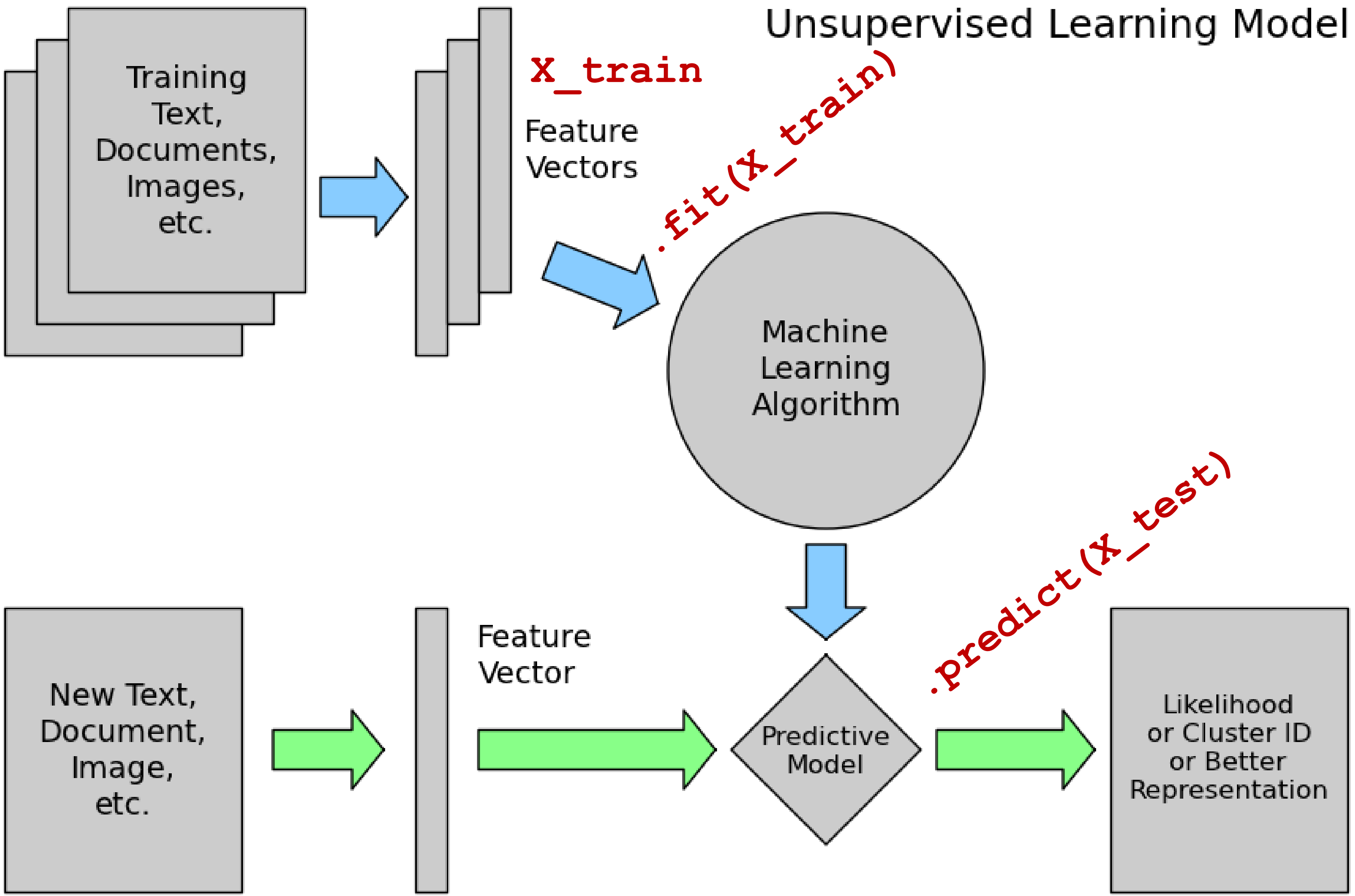
Supervised learning



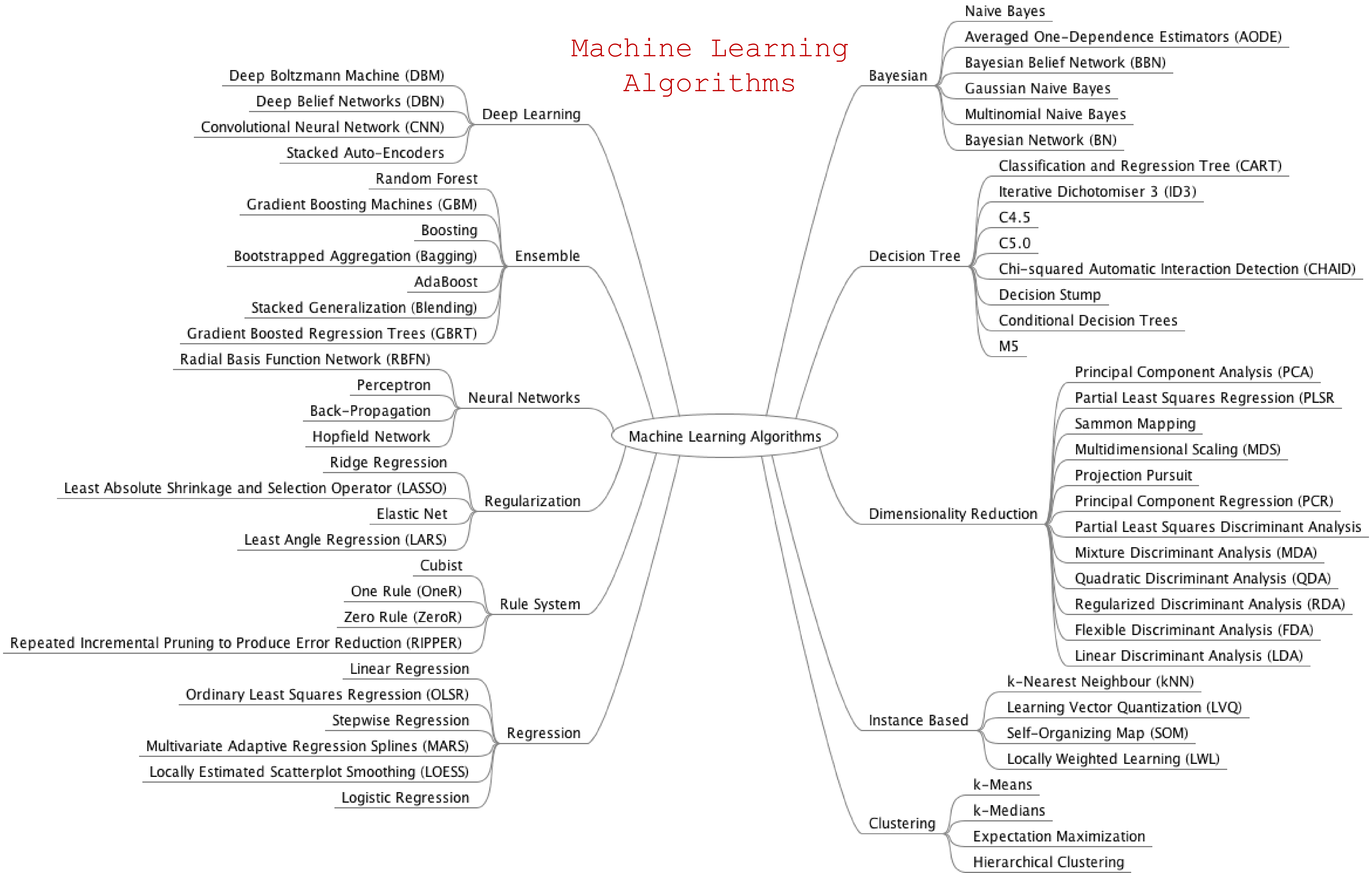
Unsupervised learning

- You only have input data (X) and no corresponding output variables
  - no mapping from input to output data
- Goal: model the underlying structure or distribution in the data in order to learn more about the data
- Unsupervised learning problems can be further grouped into:
  - **Clustering problems**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
    - Popular techniques: k-means
  - **Association problems**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y
    - Popular techniques: Apriori algorithm

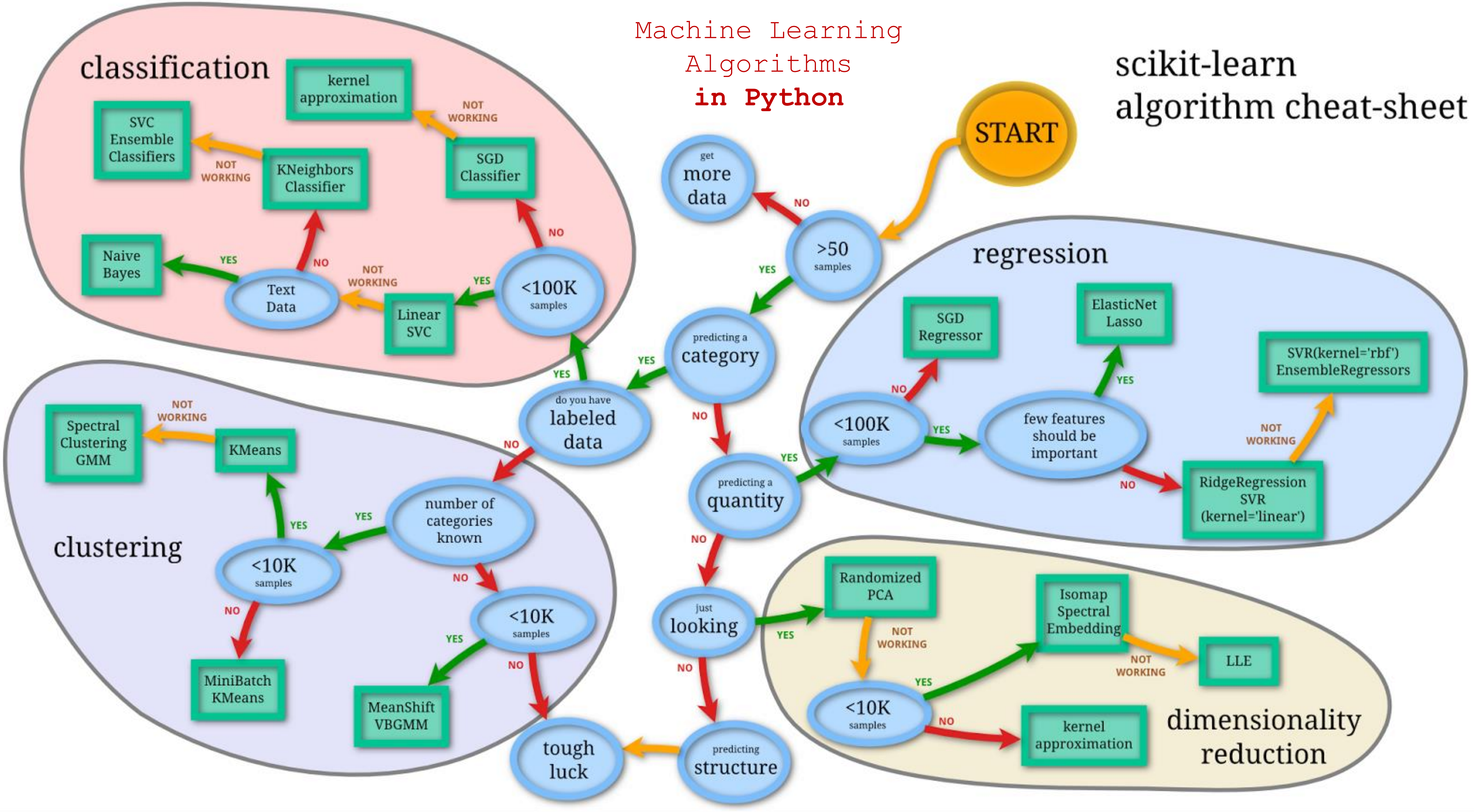
Unsupervised learning



Machine Learning Algorithms



Machine Learning Algorithms in Python

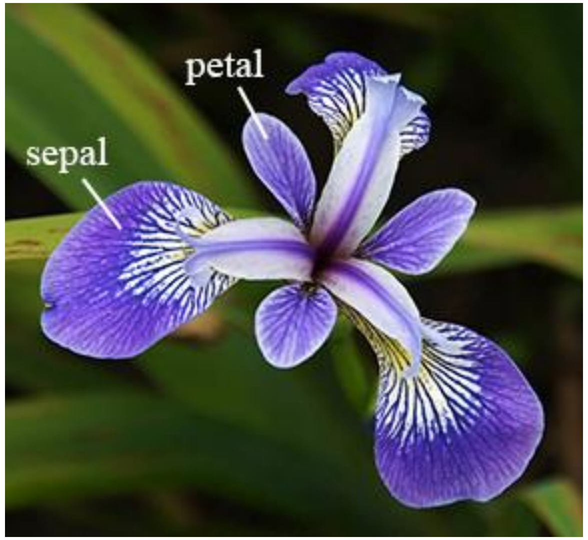


Predictive Modelling: Supervised task

Classification problem: classify *Iris* flowers into three related species

Dataset: The iris dataset

- iris flowers dataset
  – “hello world” dataset in machine learning and statistics
- Small dataset with 150 observations of iris flowers
  – each observation has 4 columns of measurements (or variables or features) of the flowers (in centimeters)
  – 5th column is the species (class) of the flower observed
  – all observed flowers belong to one of three species




```
sepal-length,sepal-width,petal-length,petal-width,class
5.1,3.5,1.4,0.2,Iris-setosa
5.9,3.0,4.2,1.5,Iris-versicolor
5.8,2.7,5.1,1.9,Iris-virginica
4.6,3.1,1.5,0.2,Iris-setosa
```

- More info: https://en.wikipedia.org/wiki/Iris_flower_data_set

Dataset Overview

- Features:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm

- Target classes (labels):
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

Get to know your data!

```
# Load dataset iris data.csv
# create dataframe object
dataset = pandas.read_csv("iris_data.csv")
```

- Take a look at the data a few different ways:
  - Dimensions of the dataset.
  - Peek at the data itself.
  - Statistical summary of all features.
  - Breakdown of the data by the class variable.

Summarize the dataset

- Dimensions of Dataset

```
# shape
print(dataset.shape)
```

```
(150, 5)
```

- Peek at the Data

```
# head
print(dataset.head(20))
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

- Data type for each column

```
print(dataset.dtypes)
```

```
sepal-length     float64
sepal-width      float64
petal-length     float64
petal-width      float64
class             object
```

- Statistical summary

```
# descriptions
print(dataset.describe())
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

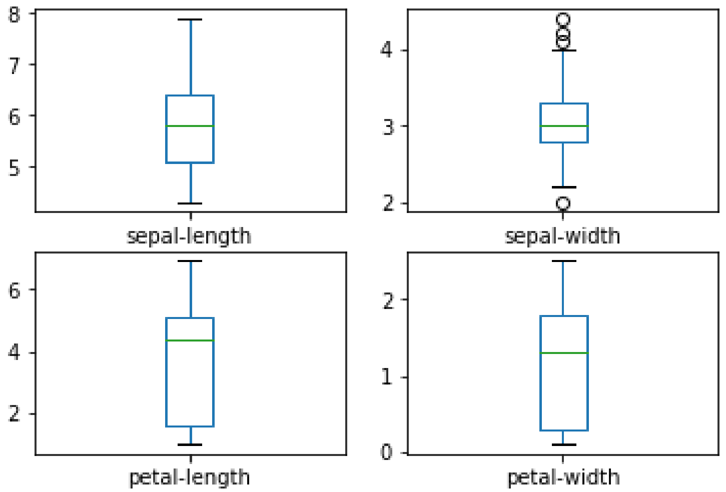
- Class Distribution

```
# class distribution
# i.e. how many flowers from each class
print(dataset.groupby('class').size())
```

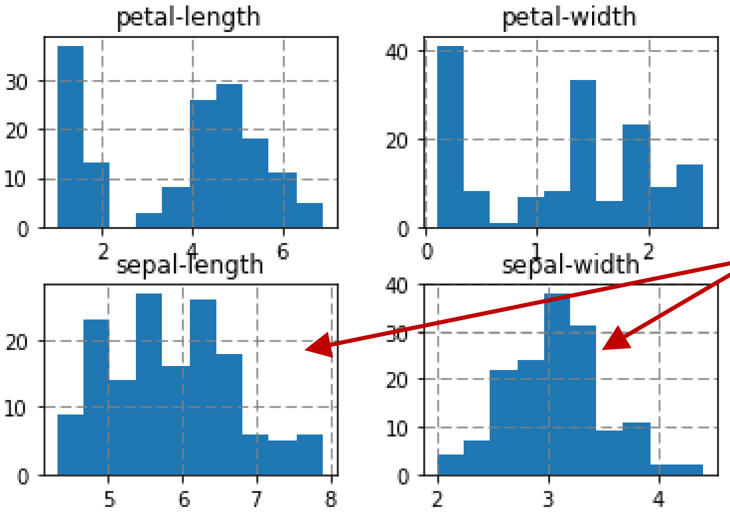
Data Visualization: Univariate Plots

- Two types of plots:
  - Univariate plots to better understand each feature (statistics/distribution)
  - Multivariate plots to better understand the relationships between features
- Univariate (single-feature) Plots

```
# box and whisker plots
dataset.plot(kind='box', subplots=True,
layout=(2,2), sharex=False, sharey=False)
plt.show()
```




```
# histograms
dataset.hist()
plt.show()
```



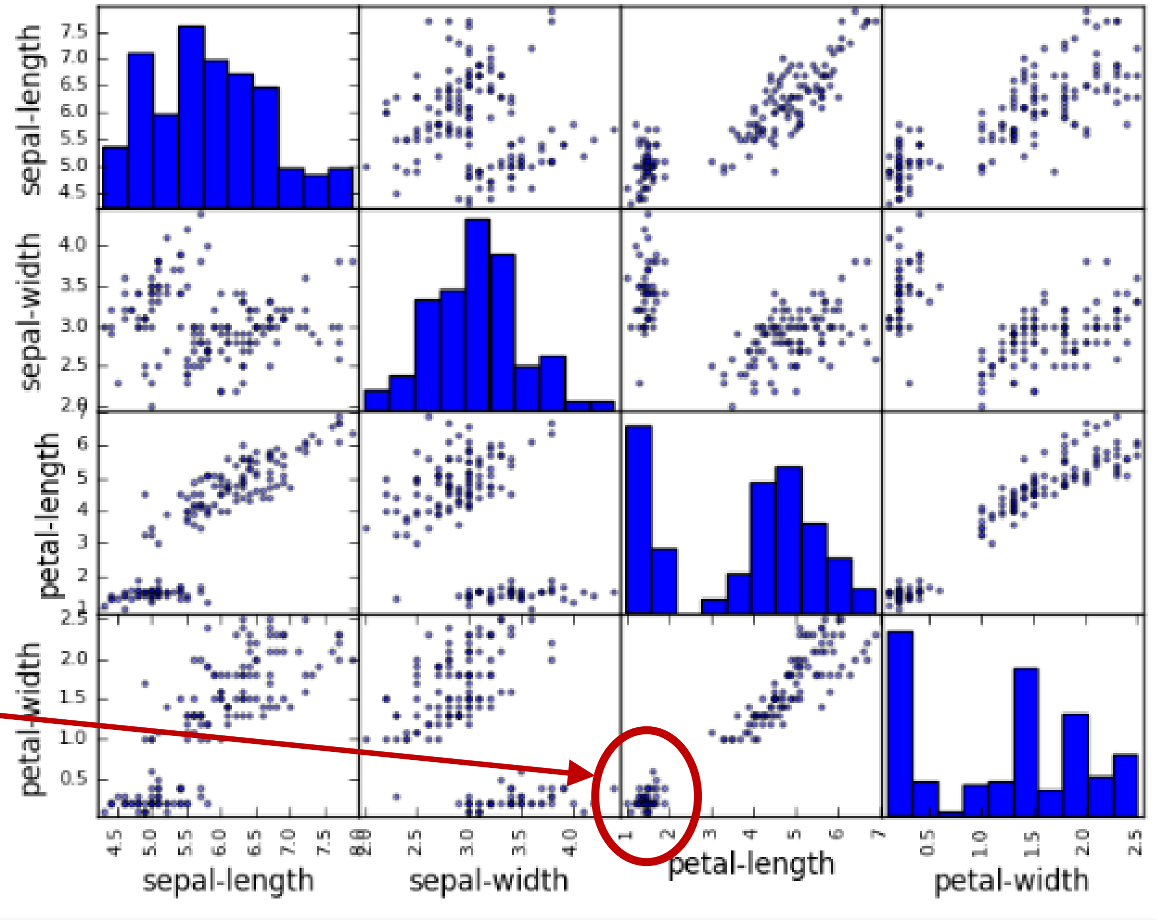
2 features seem to follow the Gaussian distribution: we can use algorithms that can exploit this assumption

Data Visualization: Multivariate (multi-feature) Plots

- Look at the interactions/correlations between features
- Scatter plots

```
# scatter plot matrix
scatter_matrix(dataset)
plt.show()
```

  - Note the diagonal grouping of some pairs of attributes. This suggests a high correlation and a predictable relationship.
    - Small petal-length values are highly correlated to small petal-width values
  - Scatter plots work well up to three dimensions



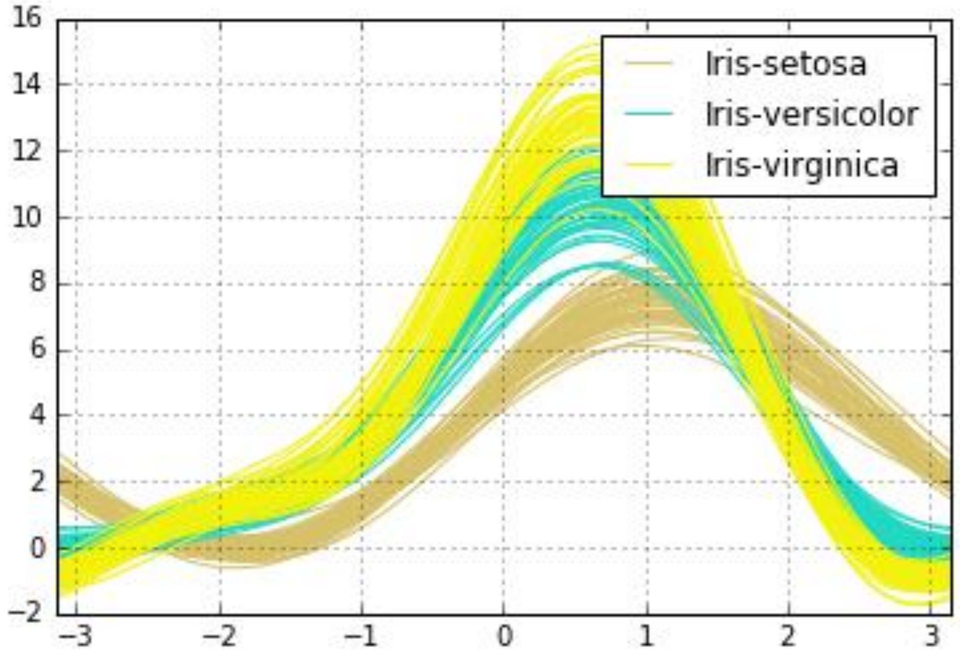
Scattplot Matrix

Data Visualization: Multivariate (multi-feature) Plots

- Andrew curves:
  - representing multivariate data by curves
  - useful tool for separating multivariate observation into groups that can not easily be distinguished in a tabular presentation
  - each multivariate observation (each line of file) $X_i = (X_{i,1}, X_{i,2}, \ldots, X_{i,p})$, here p=4, is transformed (Fourier series transformation) into a curve as follows:

$$f_i(t) = \begin{cases} \frac{X_{i,1}}{\sqrt{2}} + X_{i,2}\sin(t) + X_{i,3}\cos(t) + \ldots + X_{i,p-1}\sin(\frac{p-1}{2}t) + X_{i,p}\cos(\frac{p-1}{2}t) & \text{for } p \text{ odd} \\ \frac{X_{i,1}}{\sqrt{2}} + X_{i,2}\sin(t) + X_{i,3}\cos(t) + \ldots + X_{i,p}\sin(\frac{p}{2}t) & \text{for } p \text{ even} \end{cases}$$

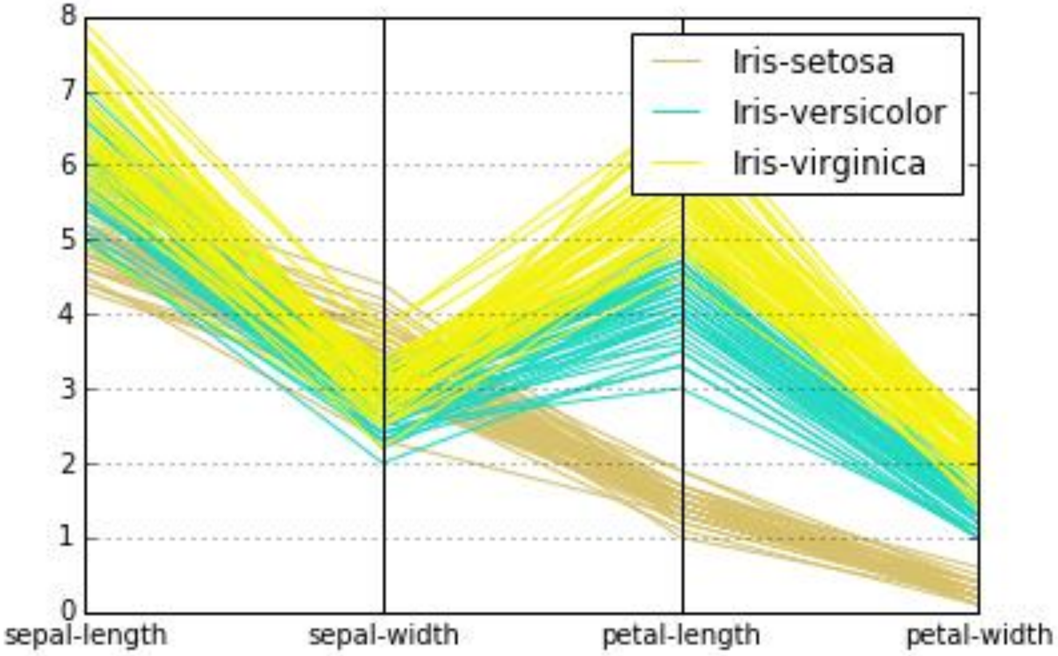
```
# andrews curves
andrews_curves(dataset, 'class')
plt.show()
```



Data Visualization: Multivariate (multi-feature) Plots

- Parallel coordinates
  - allows to see clusters in data and to estimate other statistics visually
  - each multivariate observation is represented (in parallel) by connected line segments
  - each vertical line represents one feature
  - points that tend to cluster will appear closer together

```
# parallel coordinates
parallel_coordinates(dataset,
'class')
plt.show()
```



Build Predictive Models for Classification

- Split dataset into training/validation/test datasets → 60/20/20
- Build different models to predict species from flower measurements
  - K-Nearest Neighbors (KNN), Support Vector Machines (SVM)
- Select the best model

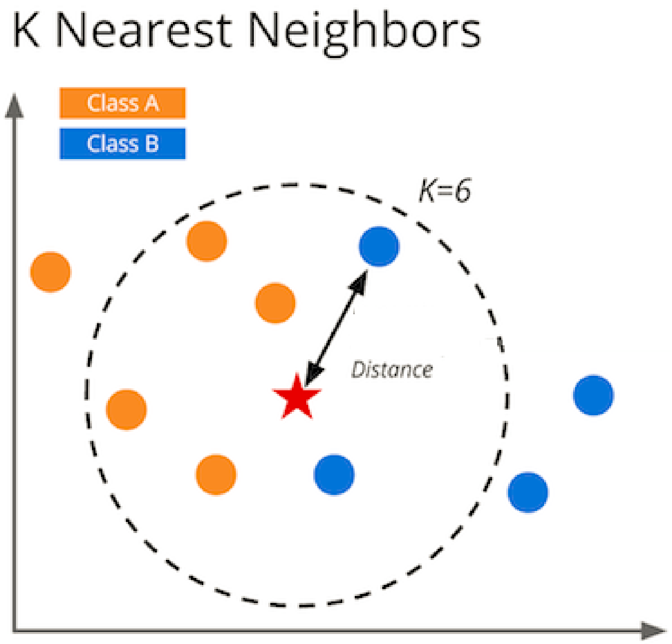
Dataset Splitting

```
array = dataset.values
X = array[:,0:4] # features
Y = array[:,4]    # target

# Split-out dataset
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.4)                    # test = 40%, train = 60%
x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5)        # test = 20%, validation = 20%
```

Build model: K-Nearest Neighbors

- Learning phase: training dataset is imported in the algorithm
- Prediction phase: new data instance is classified to the class with the highest frequency from the K-most similar (close) instances
  - Similarity measure: euclidean/minkowski distance
  - Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance.
    - In Iris classification problem (class can be 'Iris-setosa‘, or 'Iris-virginica' or 'Iris-versicolor'):



  - $p(\text{Iris−setosa}) = \dfrac{\text{count}(\text{Iris−setosa})}{\text{count}(\text{Iris−setosa}) + \text{count}(\text{Iris−virginica}) + \text{count}(\text{Iris−versicolor})}$
  - Class with the highest probability is chosen

K-Nearest Neighbors in Python

- Python implementation: `sklearn.neighbors.KNeighborsClassifier()` class
- Create knn model and run (fit) to train model

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5).fit(x_train, y_train)
```

- Predict the correct flower class on validation dataset

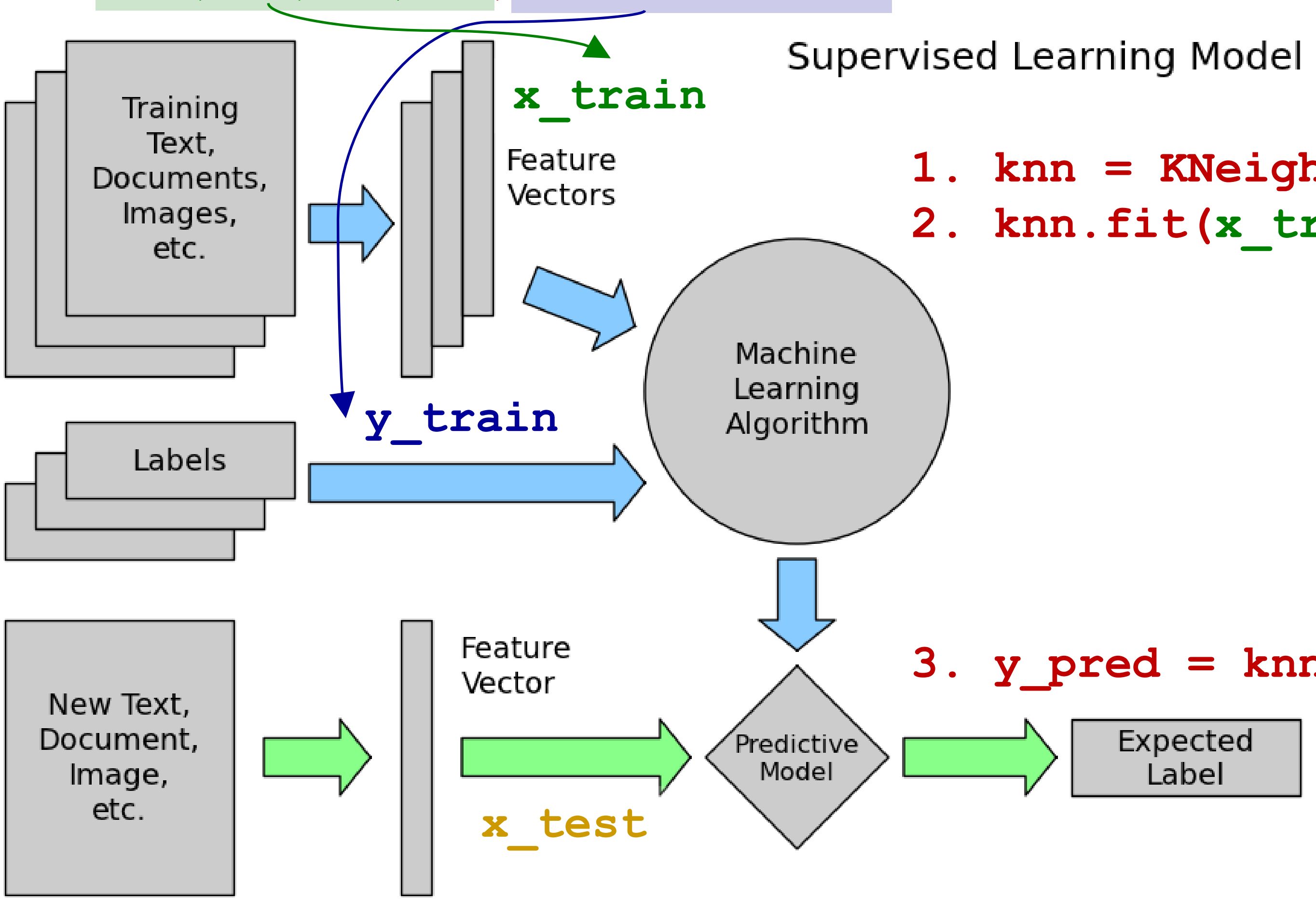
```
y_pred_knn = knn.predict(x_val)
```

- Estimate accuracy of knn model on validation dataset

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_val, y_pred_knn)
```

K-Nearest Neighbors in Python

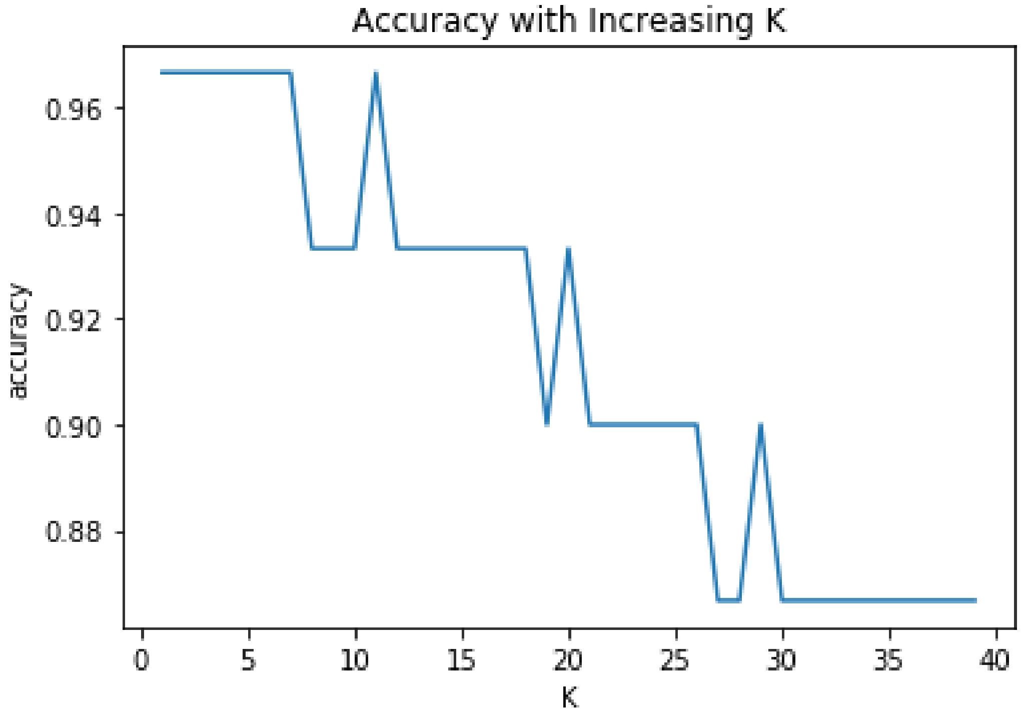
```
sepal-length,sepal-width,petal-length,petal-width,class
5.1,3.5,1.4,0.2,Iris-setosa
5.9,3.0,4.2,1.5,Iris-versicolor
5.8,2.7,5.1,1.9,Iris-virginica
4.6,3.1,1.5,0.2,Iris-setosa
```




```
1. knn = KNeighborsClassifier()
2. knn.fit(x_train,y_train)
3. y_pred = knn.predict(x_test)
```

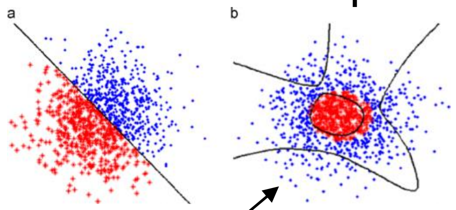
Choosing the right K

- Run the *KNN* classification algorithm for a range of *K* values [1, 40] and plot the results

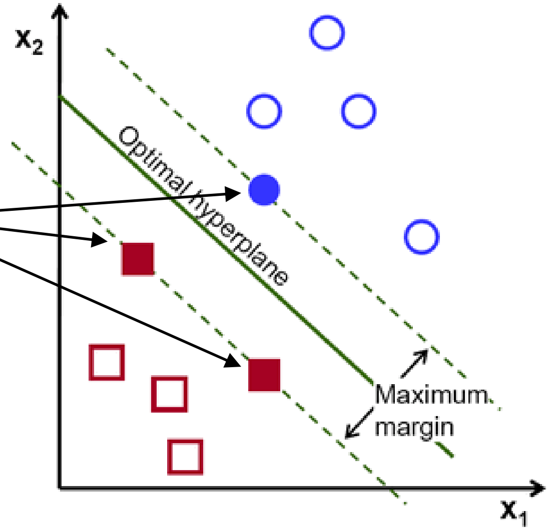


Build model: Support Vector Machine

- Basic idea of support vector machines
  - Find optimal hyperplane for linearly separable patterns
  - Extend to patterns that are not linearly separable by transformations (kernel functions) of original data to map into new space



- Support vectors
  - Data points that lie closest to the decision surface
  - They are the most difficult to classify
  - They have direct bearing on the optimum location of the decision surface



- SVMs maximize the margin around the separating hyperplane
- The decision function is fully specified by a subset of training samples, the support vectors

Support Vector Machines in Python

- Python implementation: `sklearn.svm.SVC()` class
- Create svm model and run (fit) to train model

```
from sklearn.svm import SVC
svm = SVC().fit(x_train, y_train)
```

- Predict the correct flower class on validation dataset

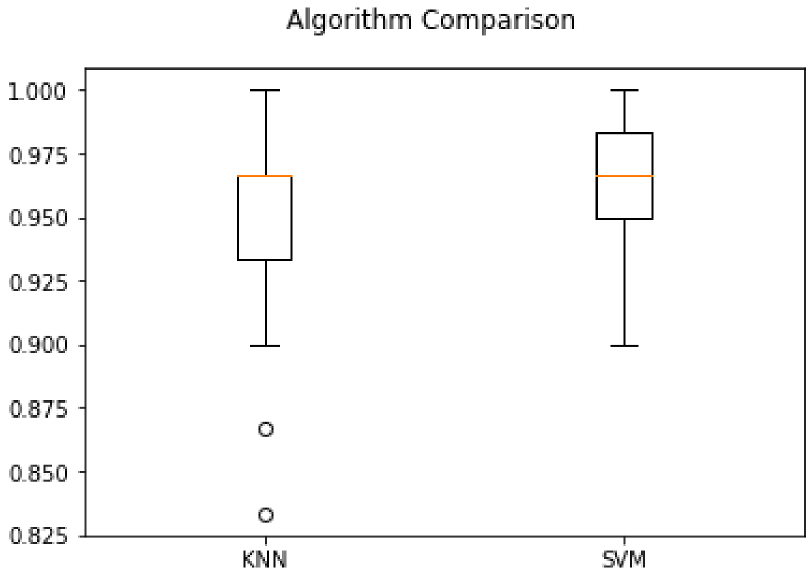
```
y_pred_svm = svm.predict(x_val)
```

- Estimate accuracy of knn model on validation dataset

```
accuracy = accuracy_score(y_val, y_pred_svm)
```

Comparative Study [Task 1]

- Run the *KNN* classification algorithm for 50 times using the best *K* value and find the mean and standard deviation of accuracy values
- Run the *SVM* classification algorithm for 50 times and find the mean and standard deviation of accuracy values
- Plot a boxplot



Predictive Modelling: Unsupervised task

Clustering problem: cluster fleet drivers into different behavior groups

Dataset

- Includes 4000 drivers
- Each observation has 3 columns:
  - `Driver_ID`
  - `Distance_Feature`: mean distance driven per day
  - `Speeding_Feature`: mean percentage of time a driver was >5 mph over the speed limit
- No notion of groups (labels)
- Load dataset

```
Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28.0
3423313212,52.53,25.0
3423313724,64.54,27.0
```

```
dataset = pd.read_csv(“fleet_data.csv")
```

K-Means

- Used to find groups which have not been explicitly labeled in data to:
  - confirm business assumptions about what types of groups exist or
  - identify unknown groups in complex data sets
- Python implementation: `sklearn.cluster.Kmeans()` class
- Run algorithm to define groups (clusters)

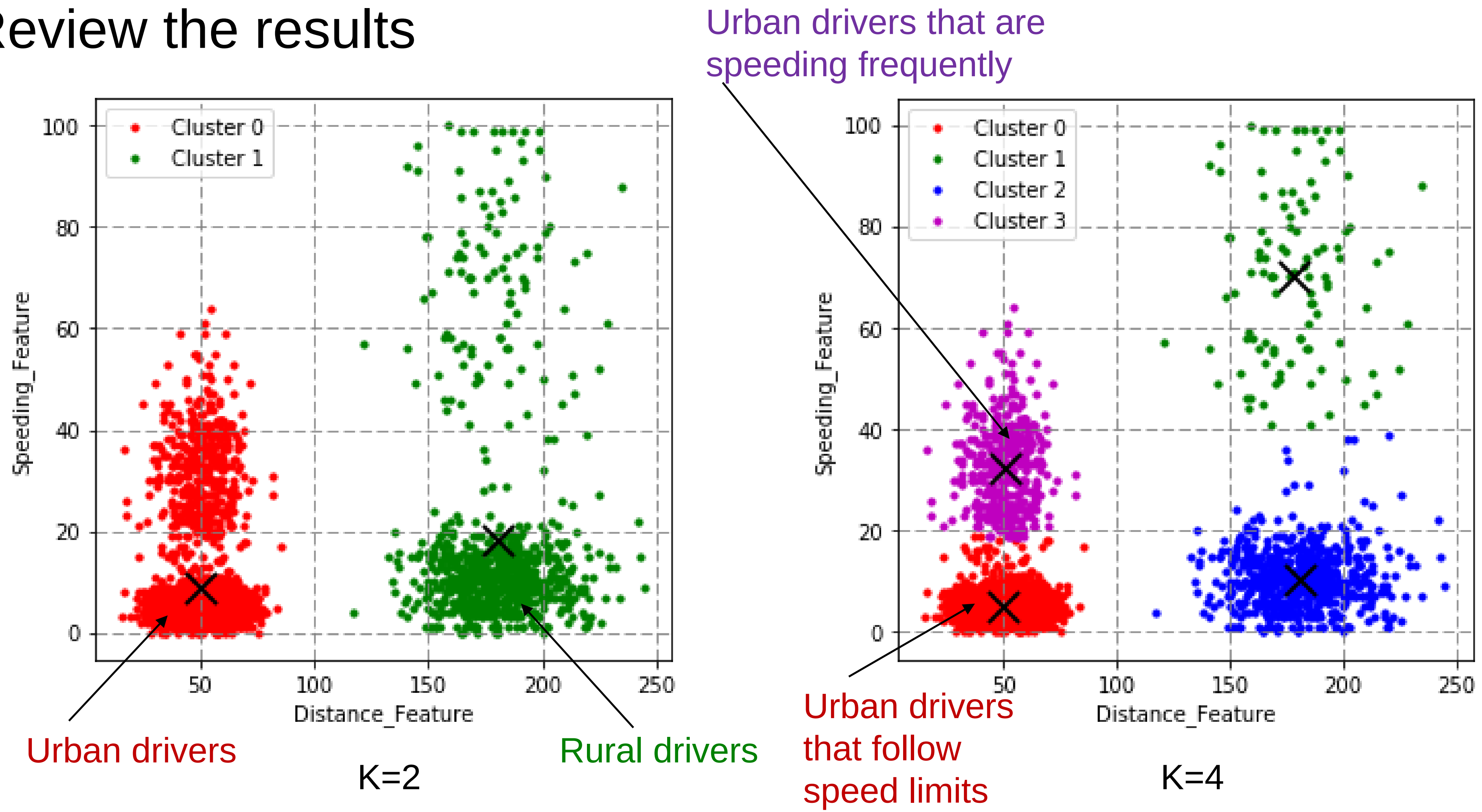
```
from sklearn.cluster import KMeans
X = dataset.values[:,1:]
kmeans = KMeans(n_clusters=2).fit(X)
print(kmeans.labels_)
print(kmeans.centroids_)
```

- Assign new data to the correct group

```
new_data = …
y_pred = kmeans.predict(new_data)
```

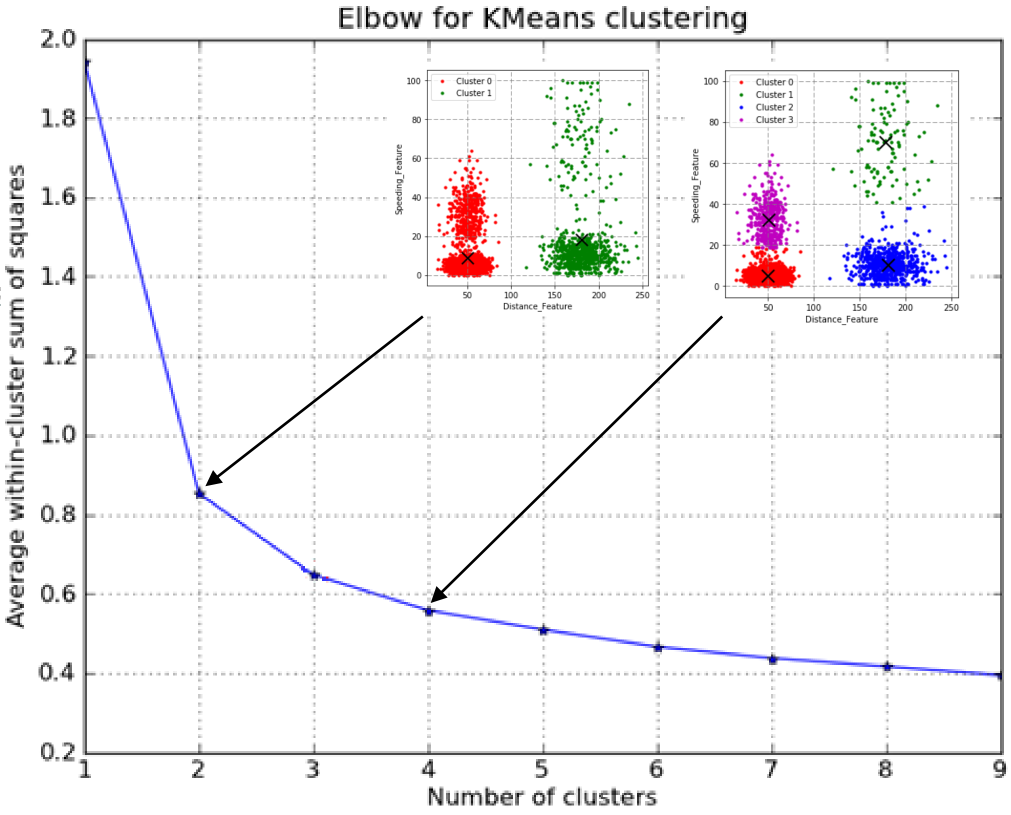
Choosing the right K (number of clusters)

- Run the *K*-means clustering algorithm for a range of *K* values
- Review the results



Choosing the right K (number of clusters)

- Difficult to visualize clusters in high-dimensional (over 3D) data
- There is no method for determining the exact value of *K*
- An estimation can be obtained using the following techniques:
  - Find mean distance between data points and their cluster centroid
    - Since increasing the number of clusters will always reduce the distance to data points, increasing *K* will *always* decrease this metric, to the extreme of reaching zero when *K* is the same as the number of data points. Thus, this metric cannot be used as the sole target.
    - Instead, mean distance to the centroid as a function of *K* is plotted and the **"elbow point"**, where the rate of decrease sharply shifts, can be used to roughly determine *K*.



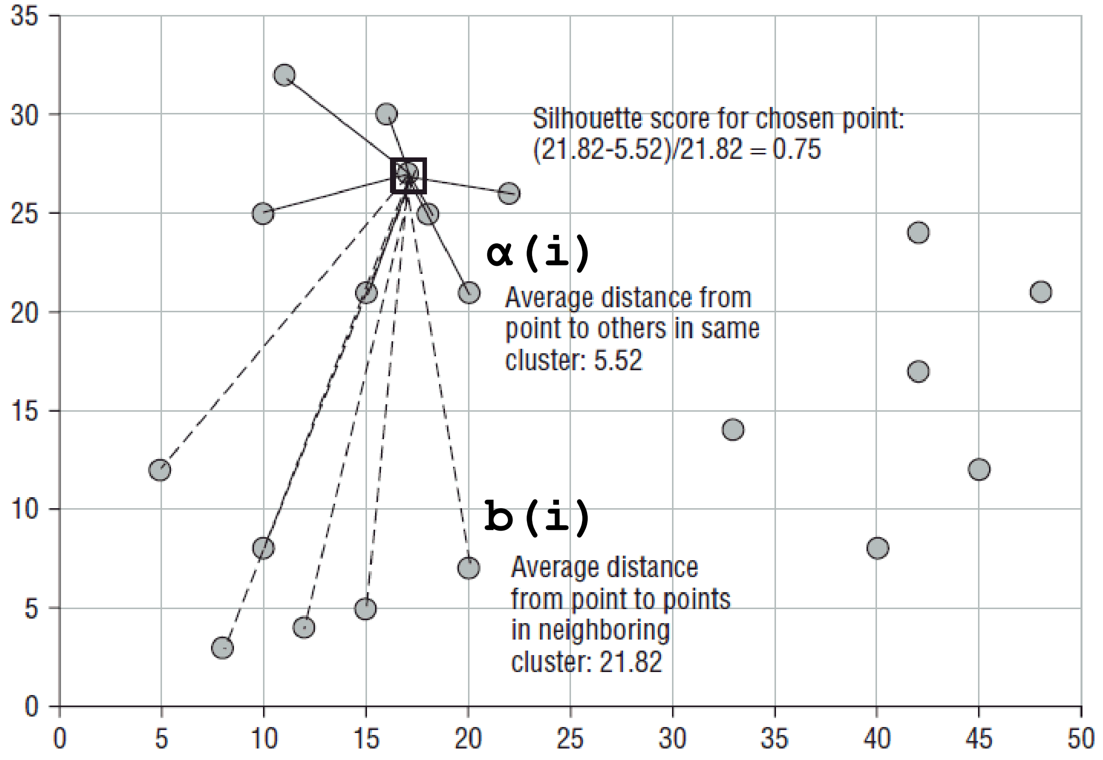
Choosing the right K (number of clusters)

– Silhouette analysis

- measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation)
- Silhouette score for data point i :

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i}$$

- ranges from -1 to 1
  – high value indicates that the data point is well matched to its own cluster and poorly matched to neighboring clusters
- Find mean value of silhouette score of all data points => if most objects have a high value, then mean value is close to 1 and the clustering configuration is appropriate



Task 2

- Goal: Build a classifier to detect wine types
- Given dataset contains data of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars.
- Analysis determined the quantities of 13 constituents found in each of the three types of wines.

```
class,alcohol,malic_acid,ash,alcalinity_of_ash,magnesium,total_...
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
```

- Answer the questions:
  - How many wines of each type are there in the dataset?
  - Which classification algorithm gives better accuracy?
    - you may try K-Nearest Neighbors, Support Vector Machine, Linear Discriminant Analysis, Gaussian Naïve Bayes, Decision Trees since all of them are executed with the same way

APPENDIX: Useful Python Libraries

- numpy – powerful N-dimensional array object & useful linear algebra, Fourier transform, and random number capabilities
- scipy – collection of mathematical algorithms and convenience functions built on the NumPy
- matplotlib – plotting library
- pandas – data analysis library (powerful dataframe object) built on the NumPy
- sklearn – machine learning library

- NumPy / SciPy / Pandas Cheat Sheet
- Pandas DataFrame Object Cheat Sheet

Pandas Library

- Involves very neat functions to load datasets in various formats, which can be browsed by typing pd.read_*
  - e.g. **read_csv()**, read_json(), read_sql_table(), read_excel(), read_html(), …
  - data stored in DataFrame data structure

- DataFrame stores entries in tabular form, with:
  - an index (`dataframe.index`) for the rows
  - multiple columns for the different variables (`dataframe.columns`)
  - each column can be retrieved by dictionary-like notation (`dataframe['column_name']`) or by attribute (`dataframe.column_name`)
  - method `head(n)` displays first n rows of data (default = 5)
  - method `apply(f)` allows to apply a function to each column or row.