Python Machine Learning with Scikit Learn Training v6

Trainer: Tu Anqi

TINFOTECH

Website:www.tertiarycourses.com.sg
Email: enquiry@tertiaryinfotech.com

About the Trainer

Tu Anqi

Tu Anqi is a scholar and dean's lister with double degree in Business and Computer Science at NTU. Her areas of expertise lies in Python, R Programming, MySQL, Numpy, Pandas, Matplotlib, Tensorflow and Keras. She has been working as a Data Analyst at ViSenze Pte Ltd, a visual search and recognition startup.

Having strong interests in computer vision, data analytics and machine learning, she has competed and won the top rankings in several data competitions and hackathons. As the current Technical Director of NTU Open Source Society, she conducts regular workshops related to data analytics and data science

Let's Know Each Other...

Say a bit about yourself

- Name
- What Industry you are from?

Do you have any prior knowledge in programming and Python

- Why do you want to learn Scikit-Learn

Prerequisite

- Basic Python is assumed in this course.
- No Machine Learning knowledge is required

Agenda

Topic 1 Overview of Machine Learning and Scikit Learn

- Introduction to Machine Learning
- Supervised vs Unsupervised Learnings
- Machine Learning Applications and Case Studies
- What is Scikit Learn
- Installing Scikit-Learn

Topic 2 Classification

- What is Classification
- Classification Algorithms
- Classification Workflow
- Confusion Matrix
- Binary Classification Metrics
- ROC and AUC

Agenda

Topic 3 Regression

- What is Regression?
- Regression Algorithms
- Regression Workflow
- Regression Metrics
- Overfitting and Regularizations

Topic 4 Clustering

- What is Clustering
- K-Means Clustering
- Silhouette Analysis
- Dendrogram and Hierarchical Clustering

Agenda

Topic 5 Principal Component Analysis

- Curse of Dimensionality Issue
- What is Principal Component Analysis (PCA)
- Feature Reduction with PCA

Exercise Files

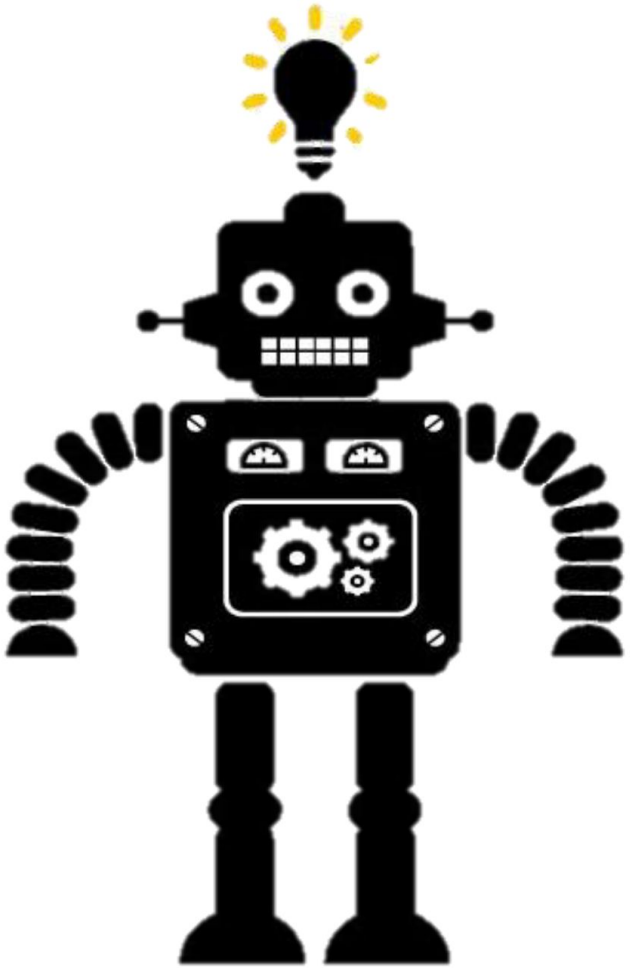
Download the exercise file from

https://github.com/anqitu/SklearnTraining

Topic 1
Overview of Machine Learning and Scikit Learn

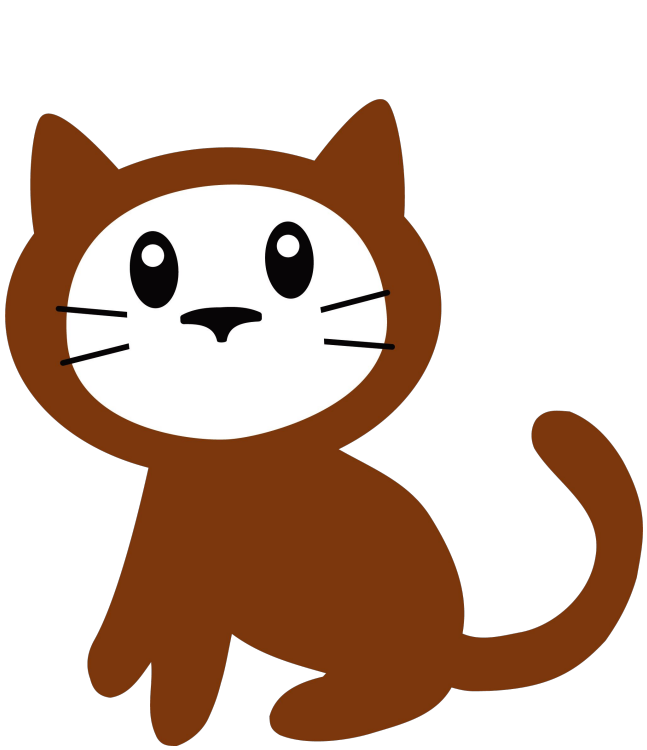
What is Machine Learning?

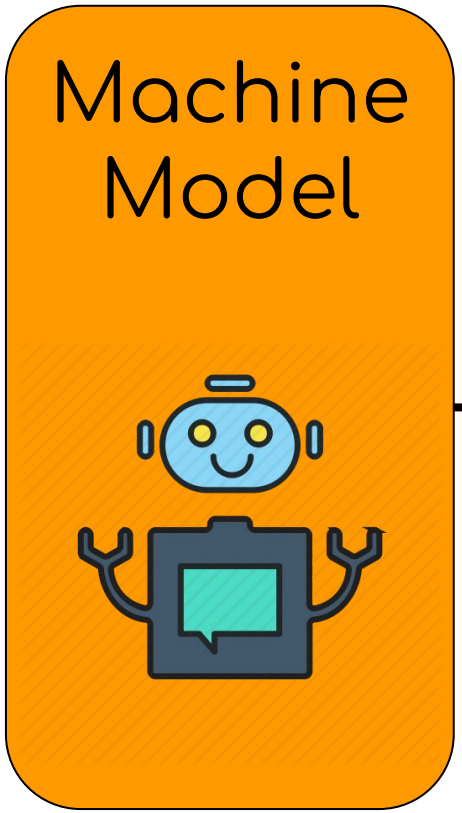
Machine learning allows computers to learn and infer from data

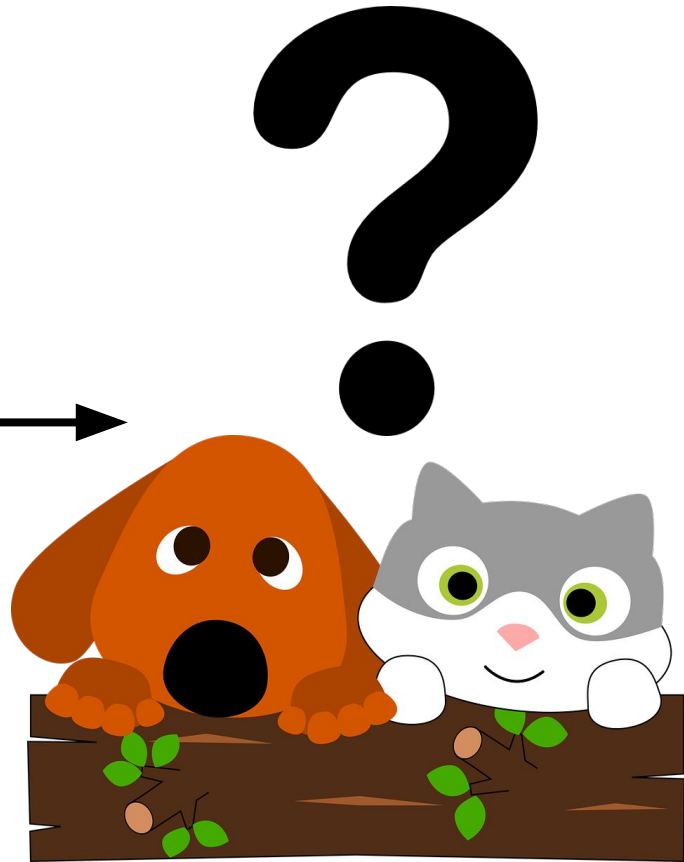


What is Machine Learning?

After you trained the machine to recognize dog or cat, then machine is able to tell your the answer when you give it an image

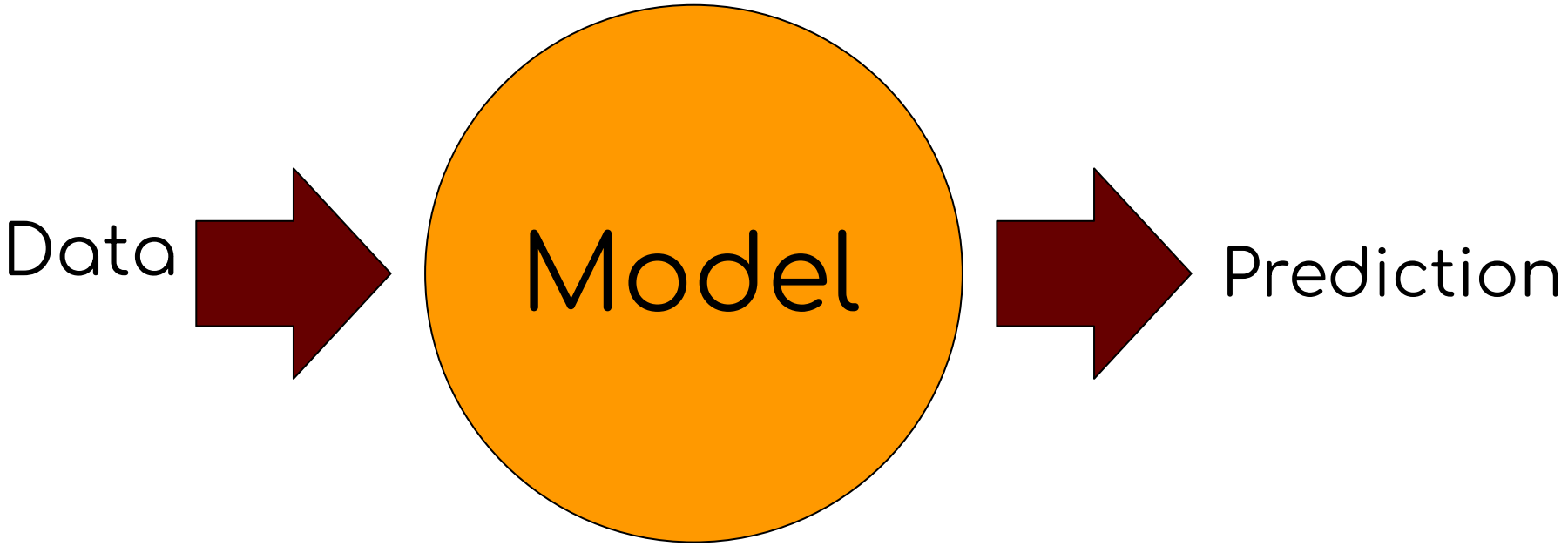






What is Machine Learning

- Machine Learning use test data to train a model
- The trained model can then used to predict the output given new data



Machine Learning in our Daily LIfe

Spam Filtering

Web Search

Postal Mail Routing

Fraud Detection

Movie Recommendations

Vehicle Driver Assistance

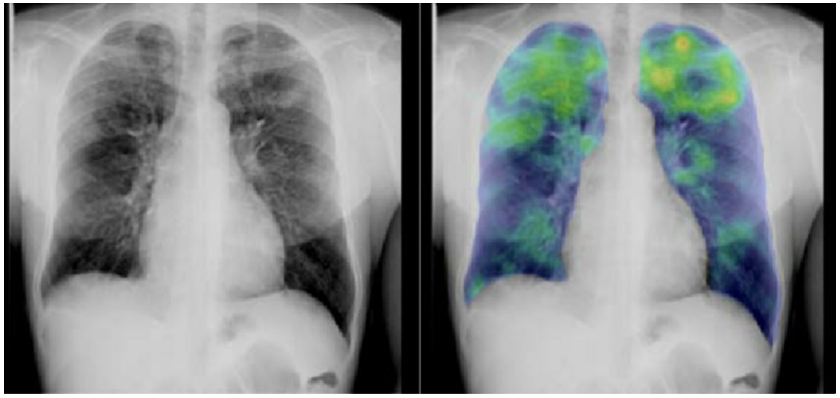
Web Advertisement

Social Networks

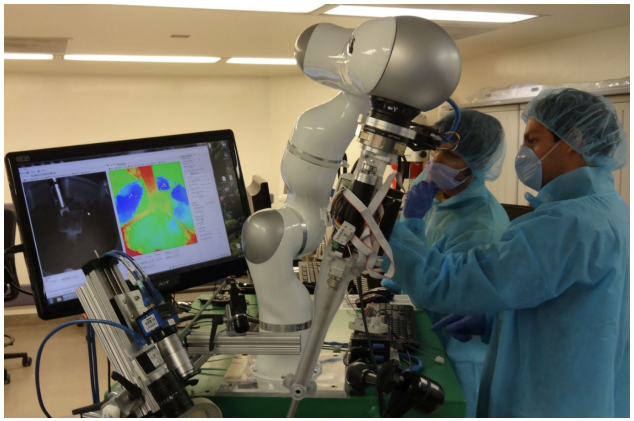
Speech Recognition

AI Applications on Health Care

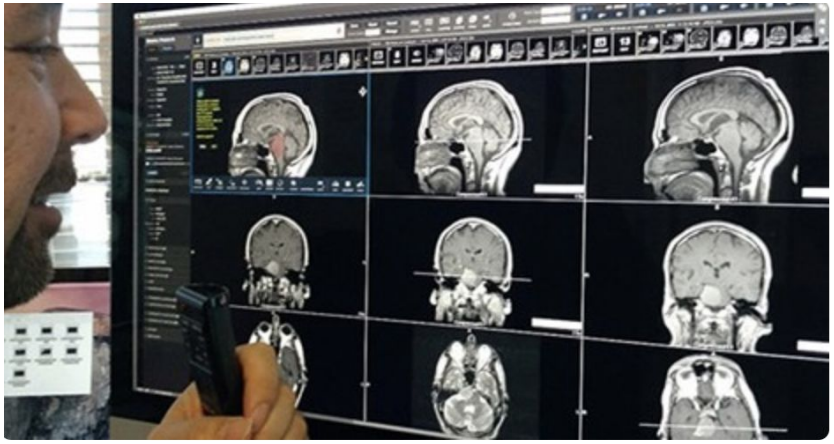
Identifying tuberculosis



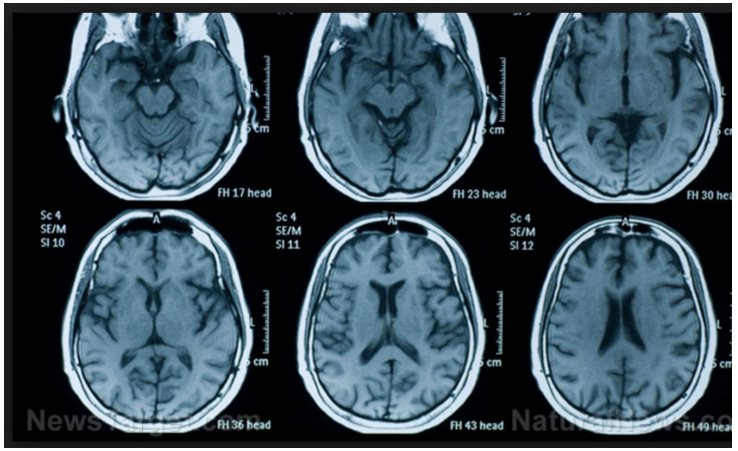
Robotics-assisted surgery



Detecting brain bleeds



Detecting Alzheimer's disease

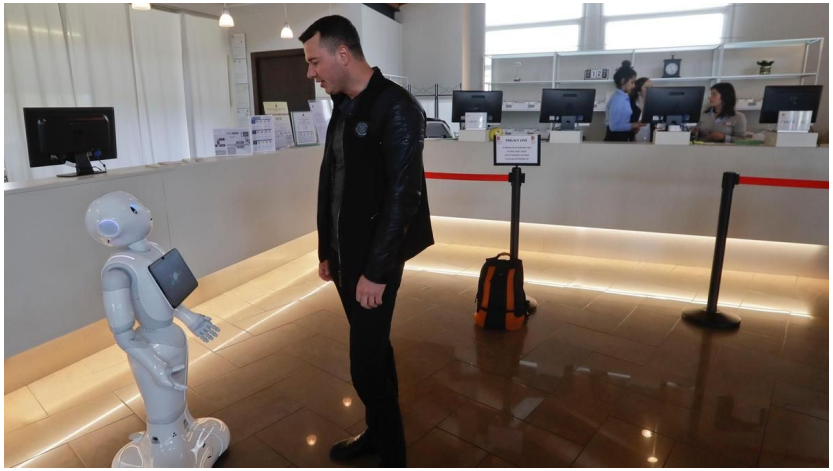


AI Applications on Retail/Logistics

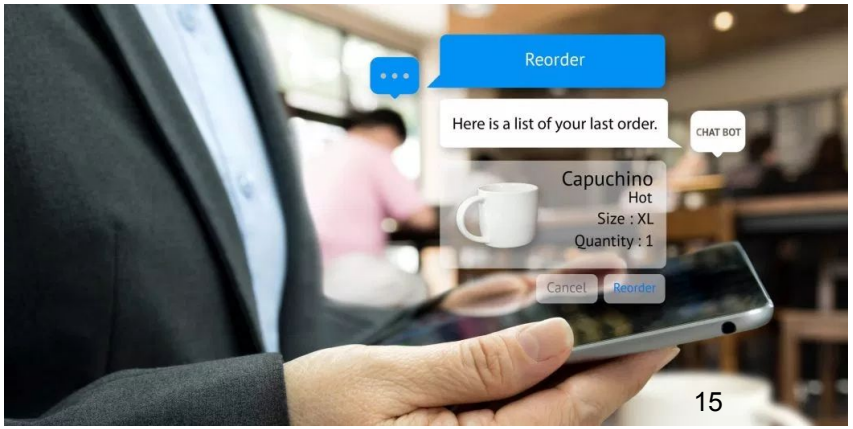
Unmanned Store: Amazon Go

Unmanned store :阿里巴巴无人超市

Frontdesk Robots



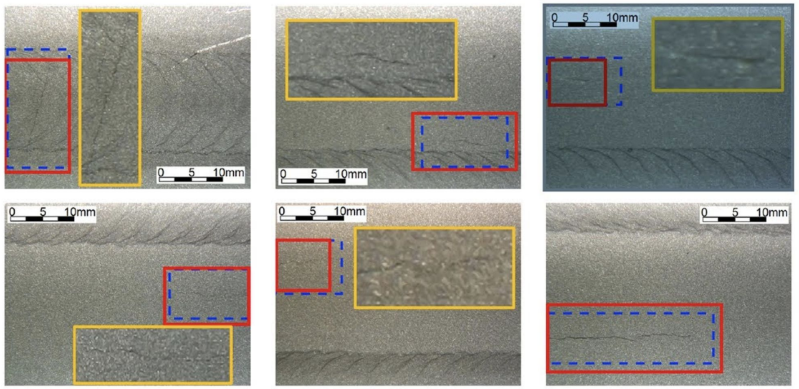
Chatbot for retail/services



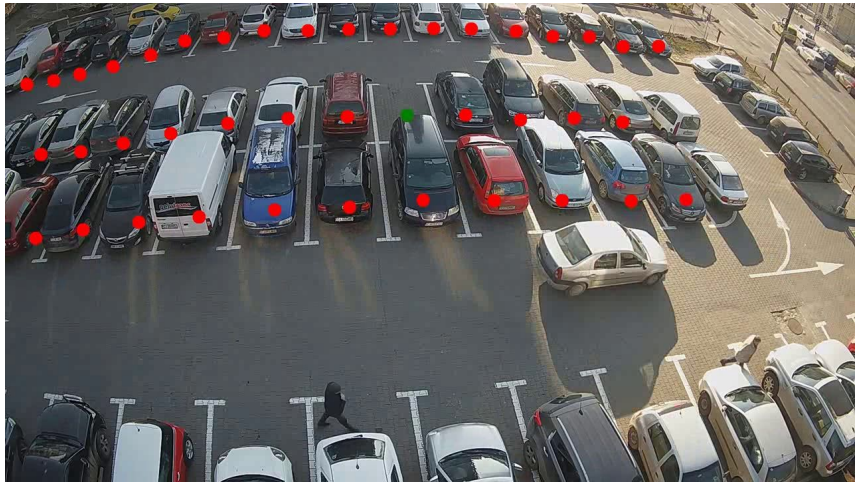
AI Applications on Security/Mfg

Face recognition
identifying criminals

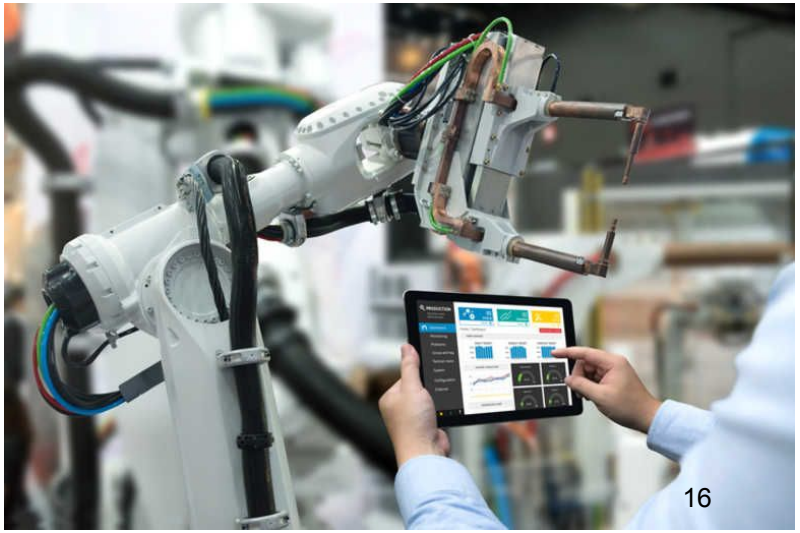
Defect detection



Parking Spotter



Robots in Mfg



Types of Machine Learning

Supervised Learning

Data Points have known outcome

Unsupervised Learning

Data Points have unknown outcome

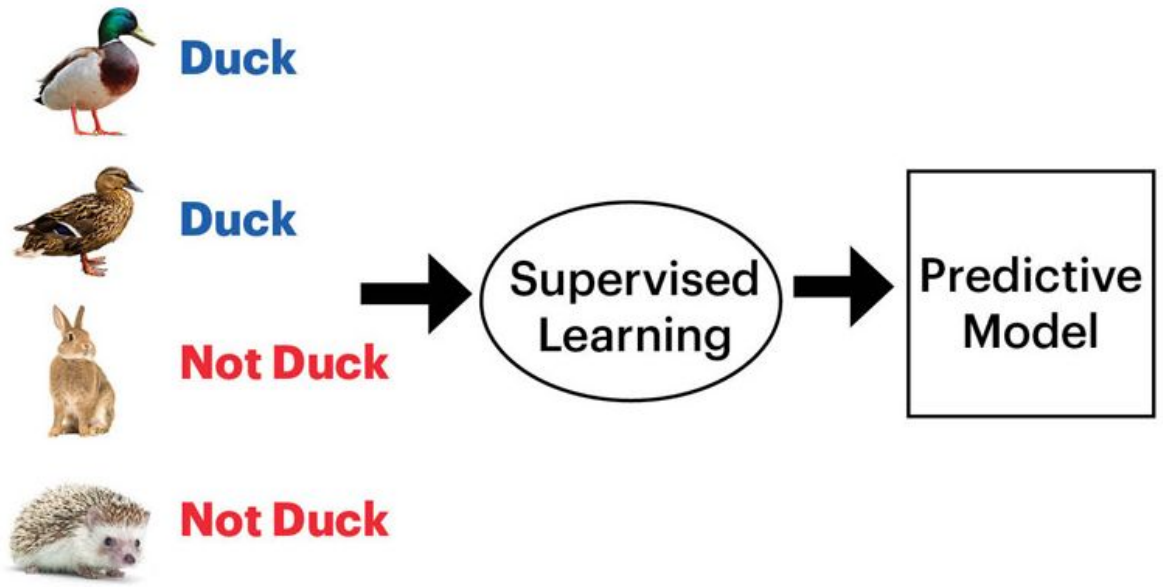
Supervised Learning

Apple

Orange

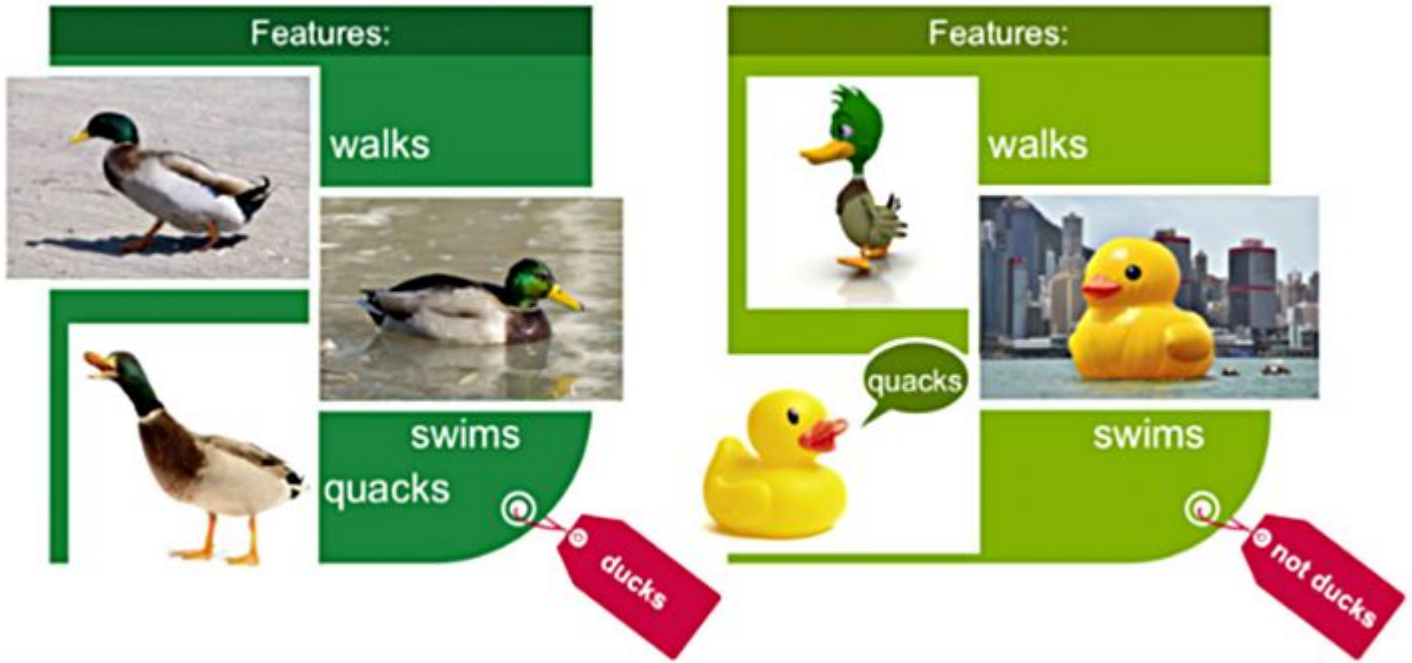
Banana

Supervised Learning



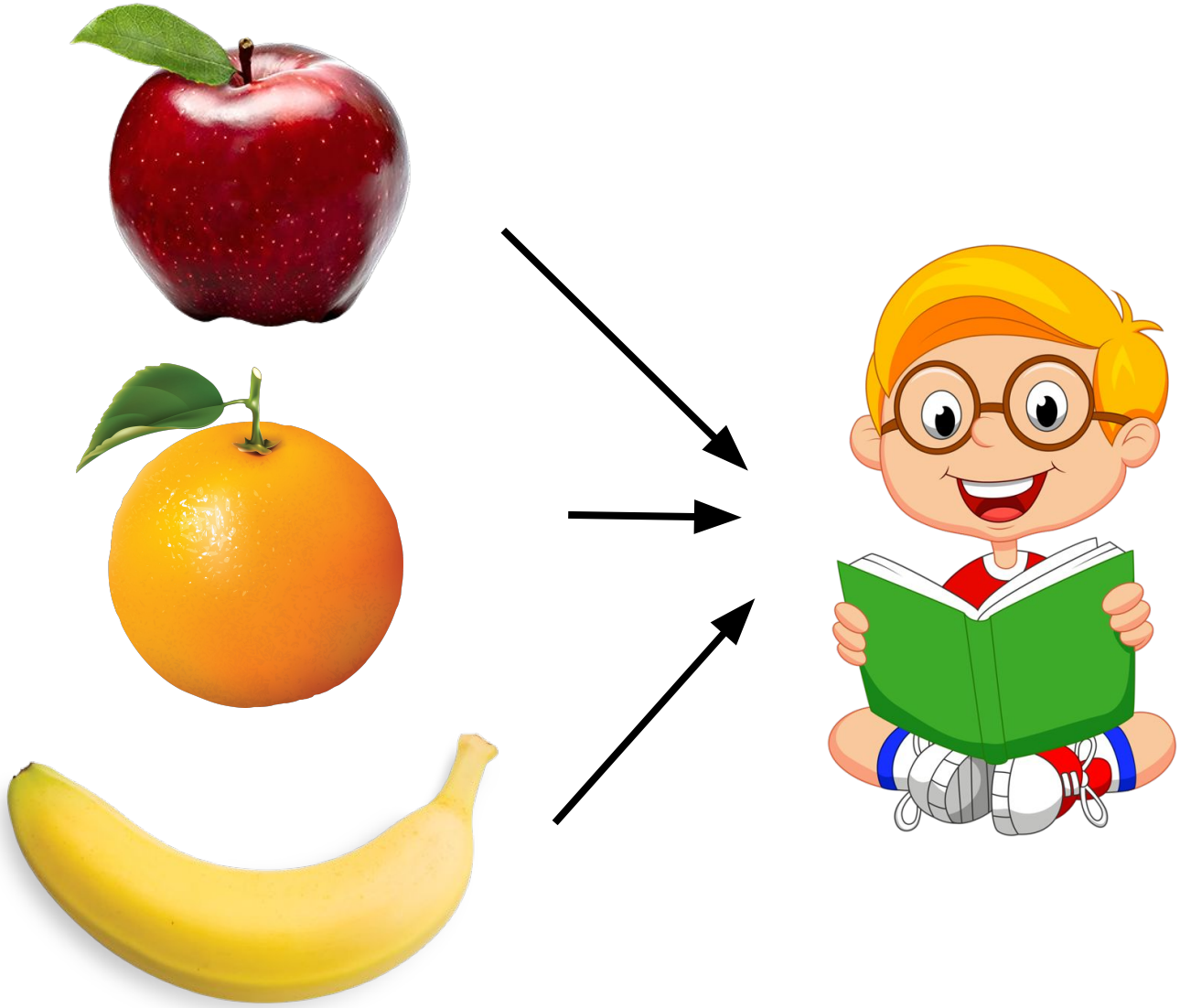
- The algorithm learns patterns from labelled data and then makes predictions and try to label new data

Supervised Learning

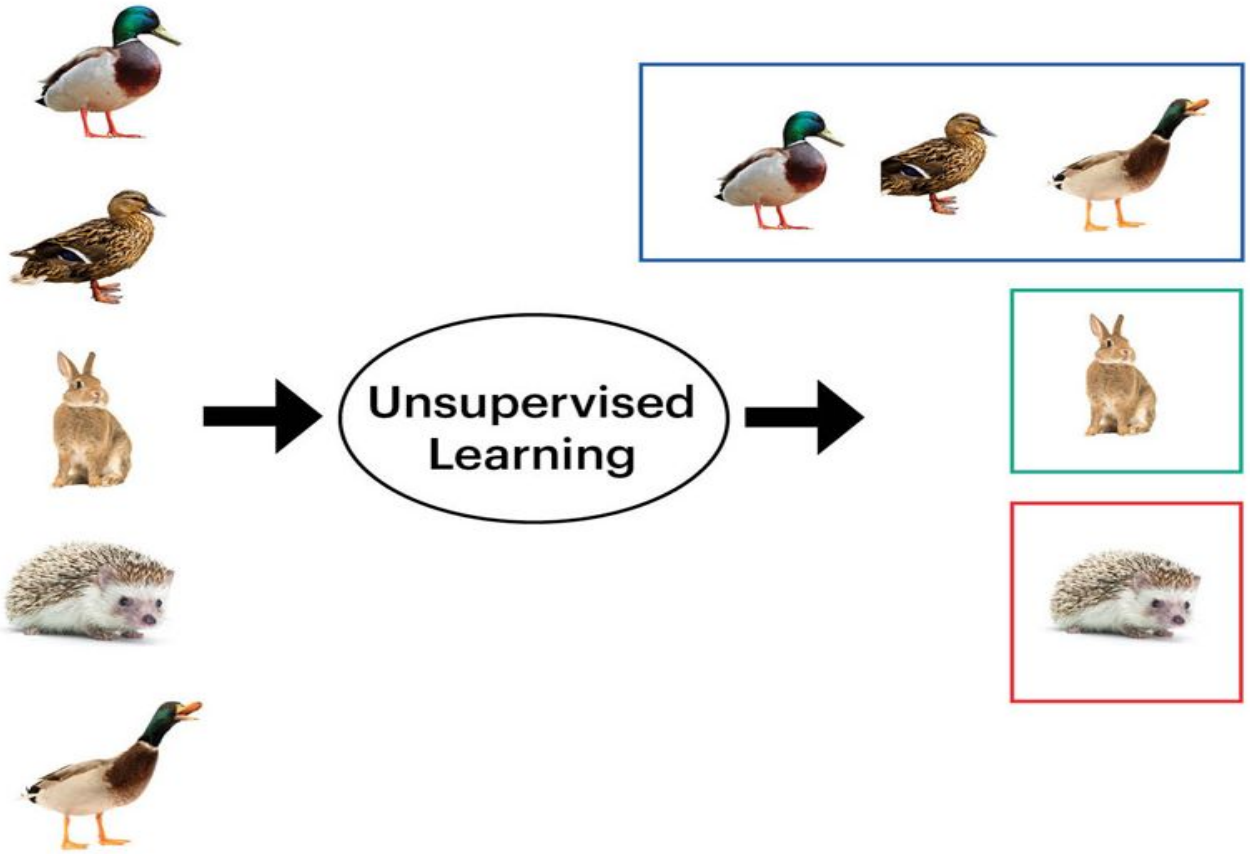


Uses labelled training data to improve programs future actions. Reproduces known knowledge. Learn by example approach. Supervised learning needs to be given examples of what is “good” and what is “bad”

UnSupervised Learning



UnSupervised Learning



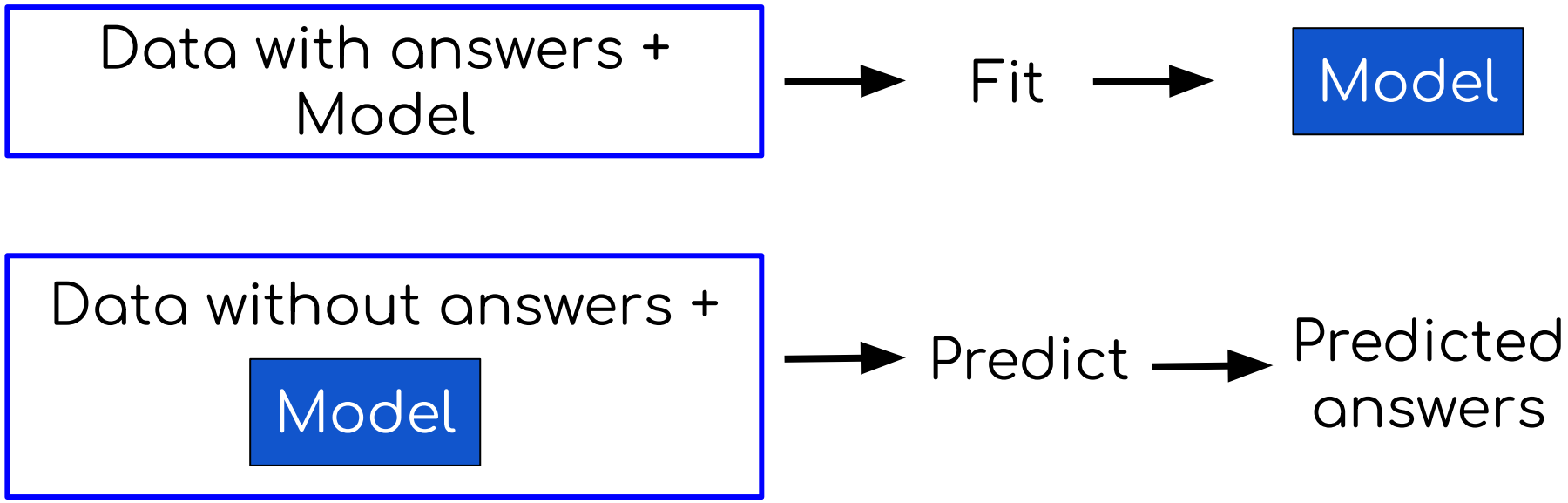
Uses unlabelled data, correct classes are not known. Interpret and groups the input data only

Types of Supervised Learning

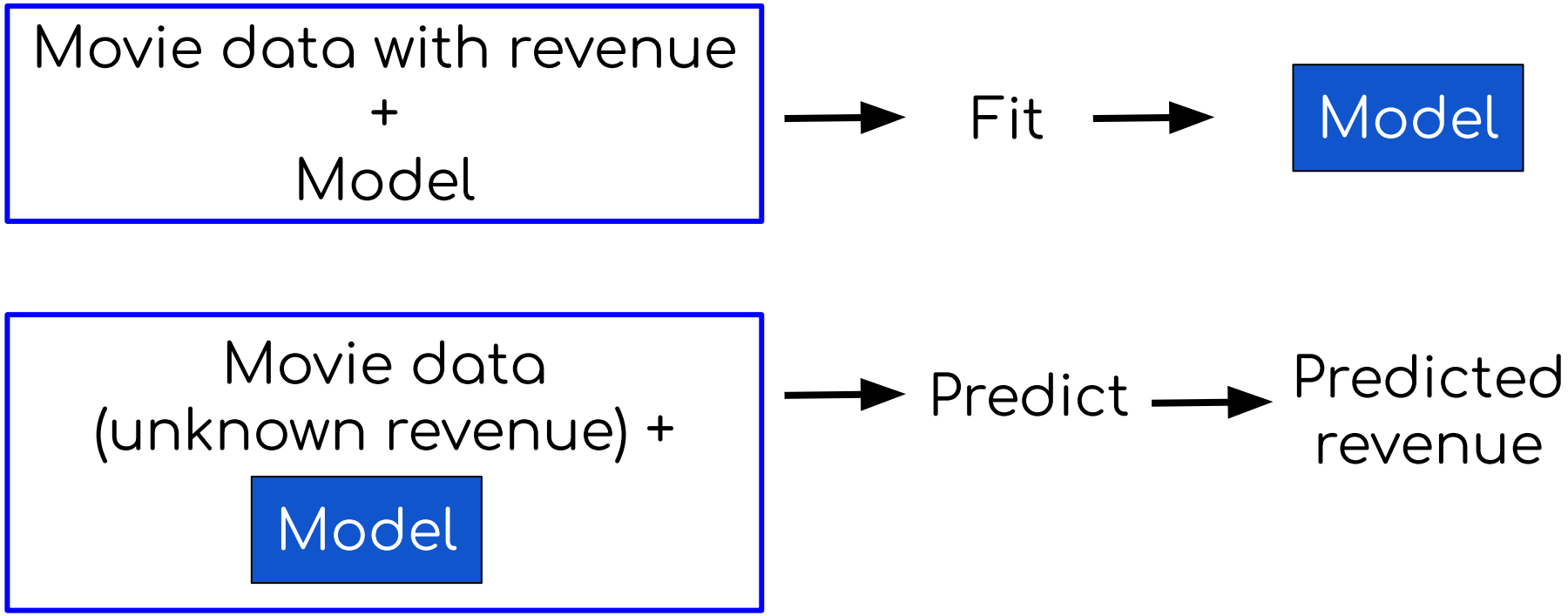
Regression — Outcome is continuous (numerical)

Classification — Outcome is a category

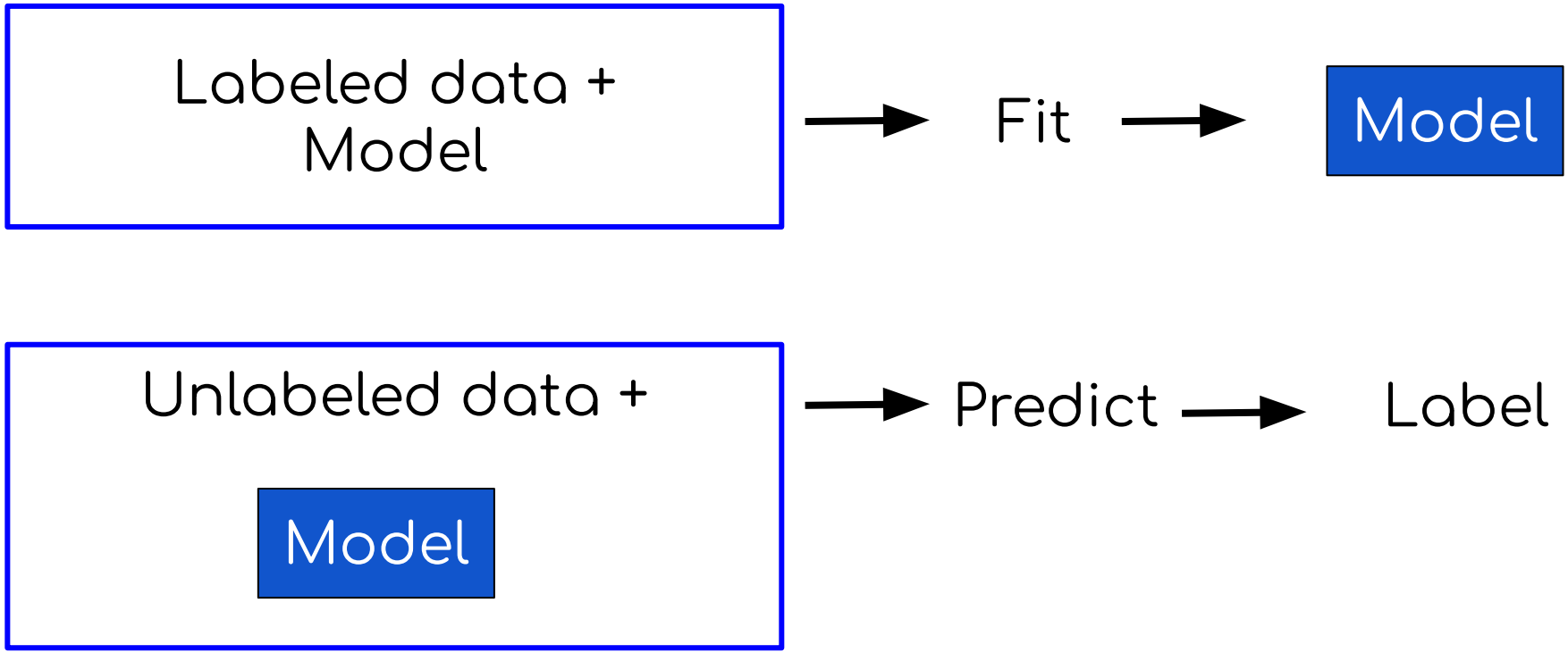
Supervised Learning Overview



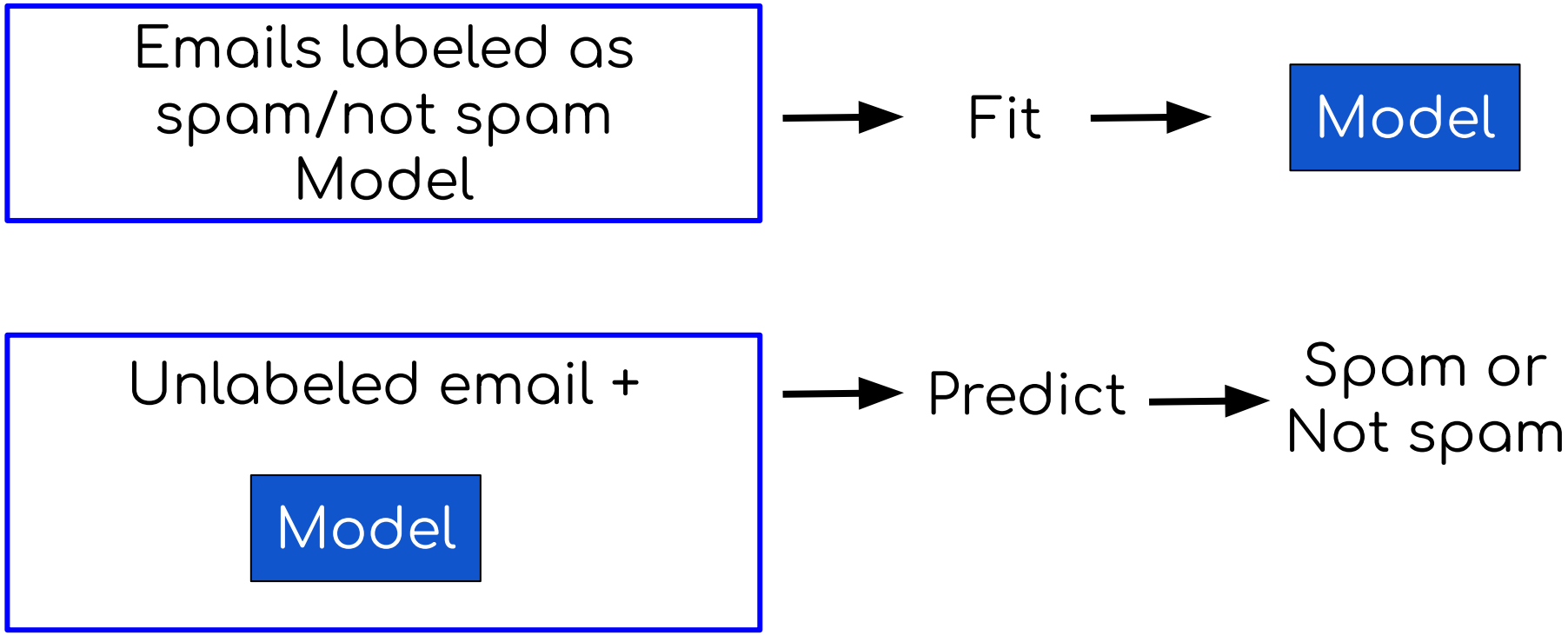
Regression: Numerical Answers



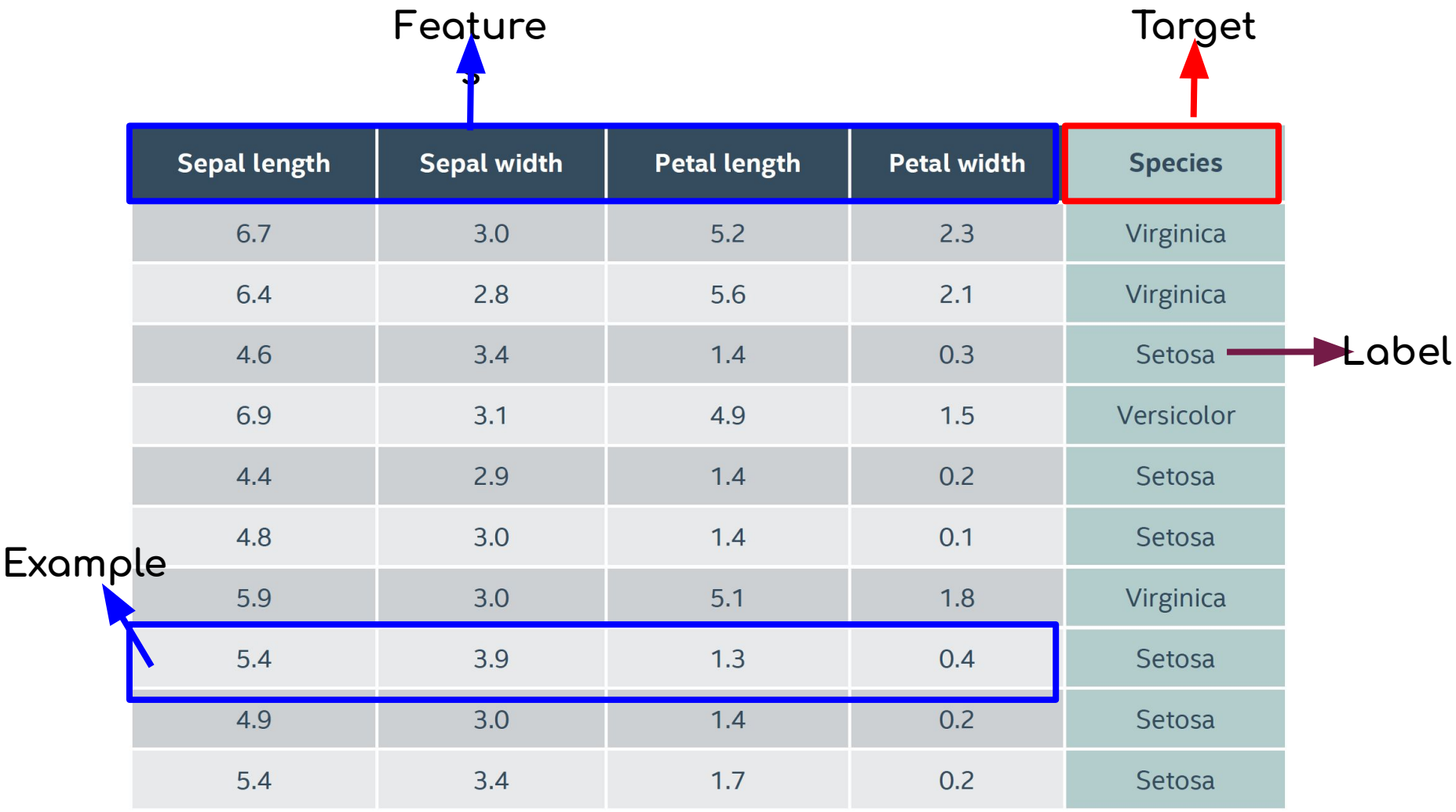
Classification: Categorical Answers



Classification: Categorical Answers



Machine Learning Vocabulary



Sepal length	Sepal width	Petal length	Petal width	Species
6.7	3.0	5.2	2.3	Virginica
6.4	2.8	5.6	2.1	Virginica
4.6	3.4	1.4	0.3	Setosa
6.9	3.1	4.9	1.5	Versicolor
4.4	2.9	1.4	0.2	Setosa
4.8	3.0	1.4	0.1	Setosa
5.9	3.0	5.1	1.8	Virginica
5.4	3.9	1.3	0.4	Setosa
4.9	3.0	1.4	0.2	Setosa
5.4	3.4	1.7	0.2	Setosa

Machine Learning Vocabulary

- Target: Predicted category or value of the data (column to predict)
- Features: properties of the data used for prediction (non-target columns)
- Example: a single data point within the data (one row)
- Label: the target value for a single data point.

Machine Learning Frameworks

PyTorch

The Microsoft Cognitive Toolkit

scikit learn

mxnet

GLUON

PaddlePaddle

Chainer

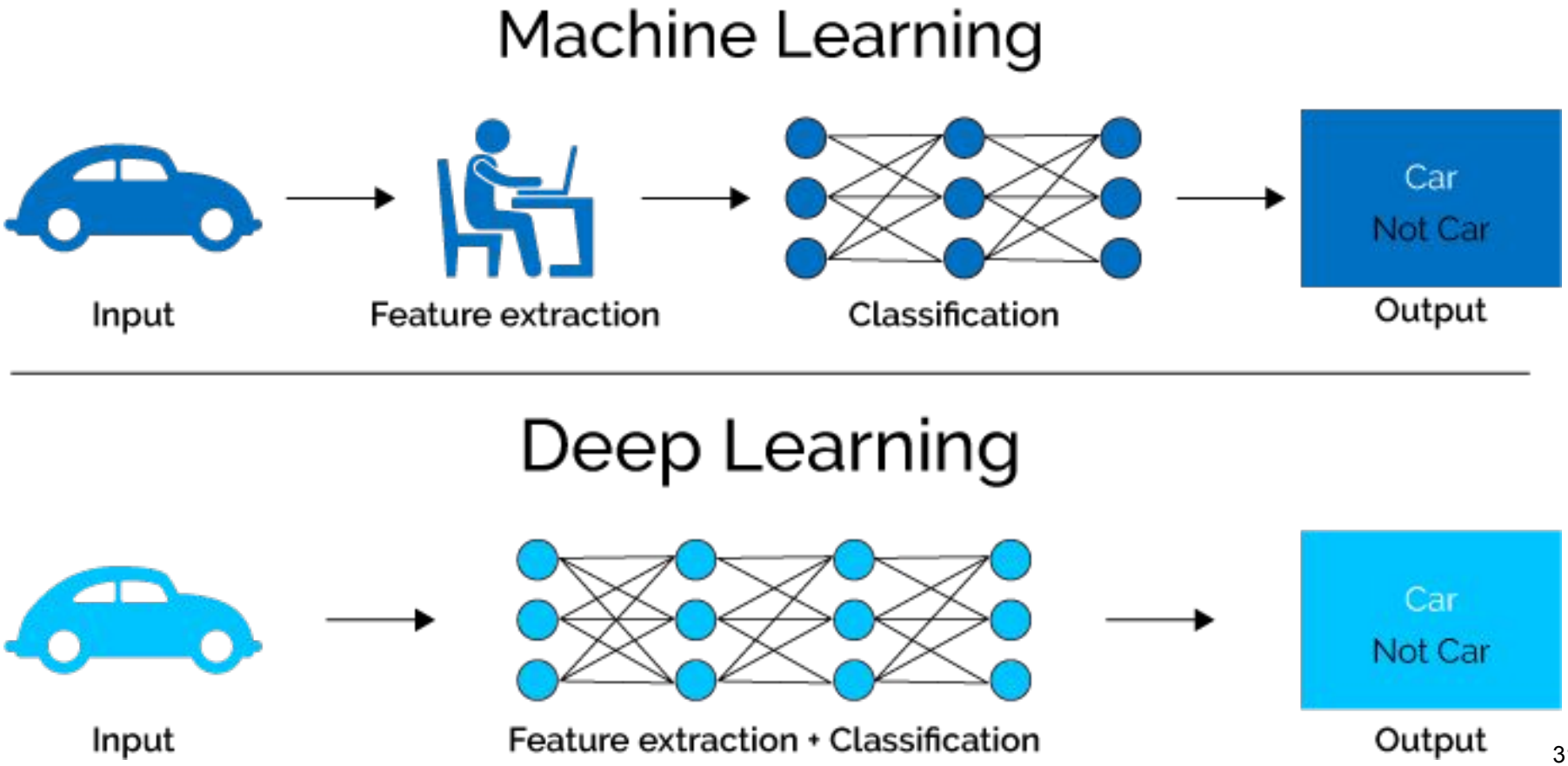
Caffe

torch

theano

Machine Learning vs Deep Learning

Deep Learning are neural network based and consists of millions of parameters to train.



What is Scikits Learn

- Scikit-learn (http://scikit-learn.org) is an open source Python library of popular machine learning algorithms
- It started in 2007 as a Google Summer of Code project
- It is built upon NumPy and SciPy

Scikits Learn Applications

- **Classifications**
  - Identifying to which category an object belongs to.
  - Spam detection, Image recognition.
- **Regression**
  - Predicting a continuous-valued attribute associated with an object.
  - Drug response, Stock prices.
- **Clustering**
  - Automatic grouping of similar objects into sets.
  - Customer segmentation, Grouping experiment outcomes

Scikits Learn Applications

- **Dimensional Reduction**
  - Reducing the number of random variables to consider.
  - Visualization, Increased efficiency
- **Model Selection**
  - Comparing, validating and choosing parameters and models.
  - Improved accuracy via parameter tuning
- **Preprocessing**
  - Feature extraction and normalization.
  - Transforming input data such as text for use with machine learning algorithms.

Install Scikit Learn

You can install scikit learn by typing the following command to terminal or CMD Prompt

pip install sklearn

pip3 install sklearn

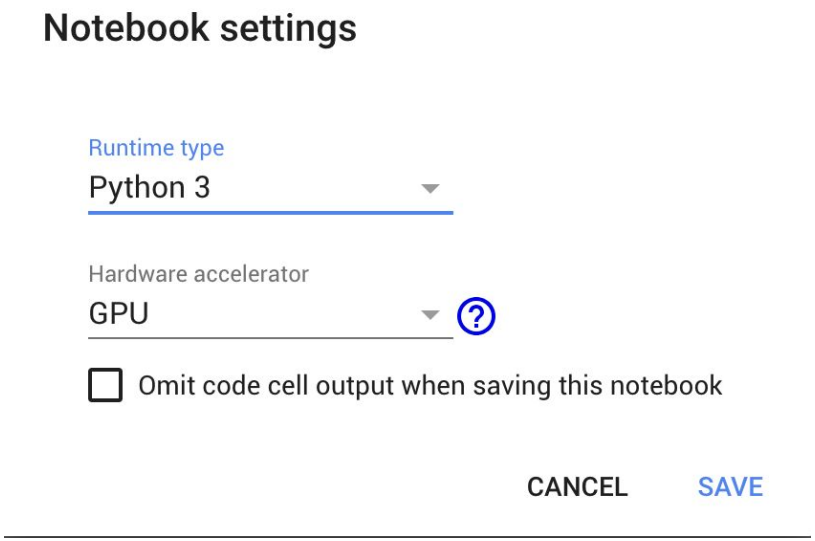
Google Colab for Scikit Learn

- Google Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.
- With Colaboratory you can write and execute Python and Scikit Learn code, save and share your analyses
- You can also access computing resources such as GPU and TPU for free
- You can access Google Colab by typing

  https://colab.research.google.com

Google Colab Setup

To use GPU, Goto Edit->Notebook Setting, set hardware accelerator to GPU



Access Google Drive from Colab

To mount google drive locally, refer to this link
https://colab.research.google.com/notebooks/io.ipynb

```
from google.colab import drive
drive.mount('/content/gdrive')
```

To access your Google Drive :

```
ls  '/content/gdrive/My Drive/'
```

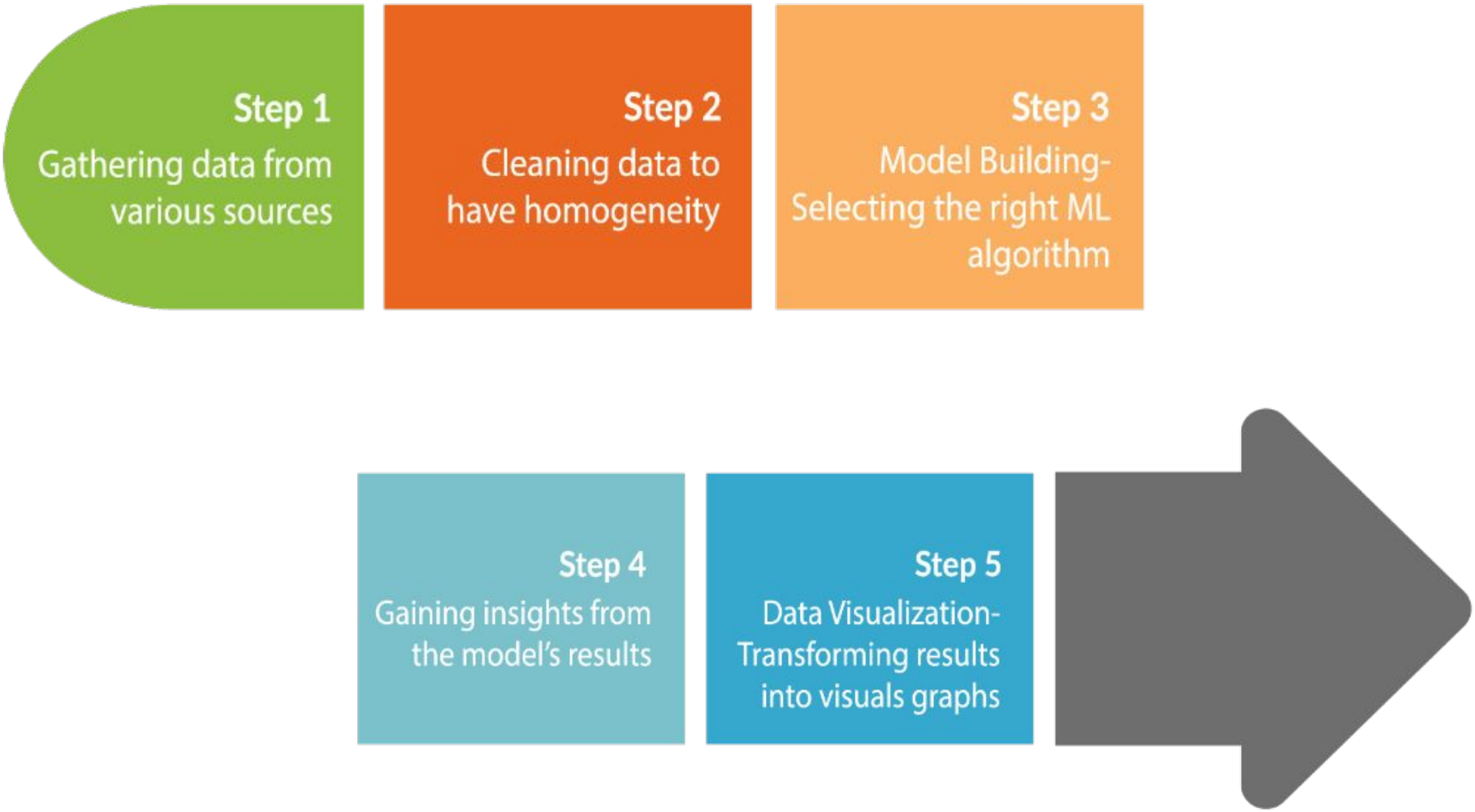
Checking your installation

You can check your scikit learn installation by typing the following command to terminal or CMD Prompt

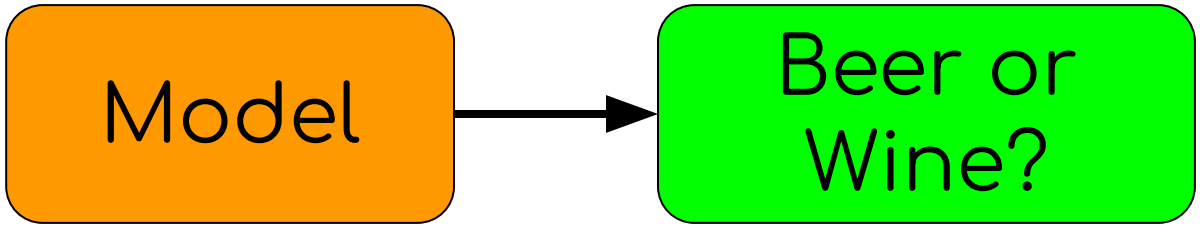
```
import sklearn
```

Topic 2
Classification

Machine Learning Flow



Supervised Learning Demo



Determine the Features

Color

Alcohol Level

Data Gathering

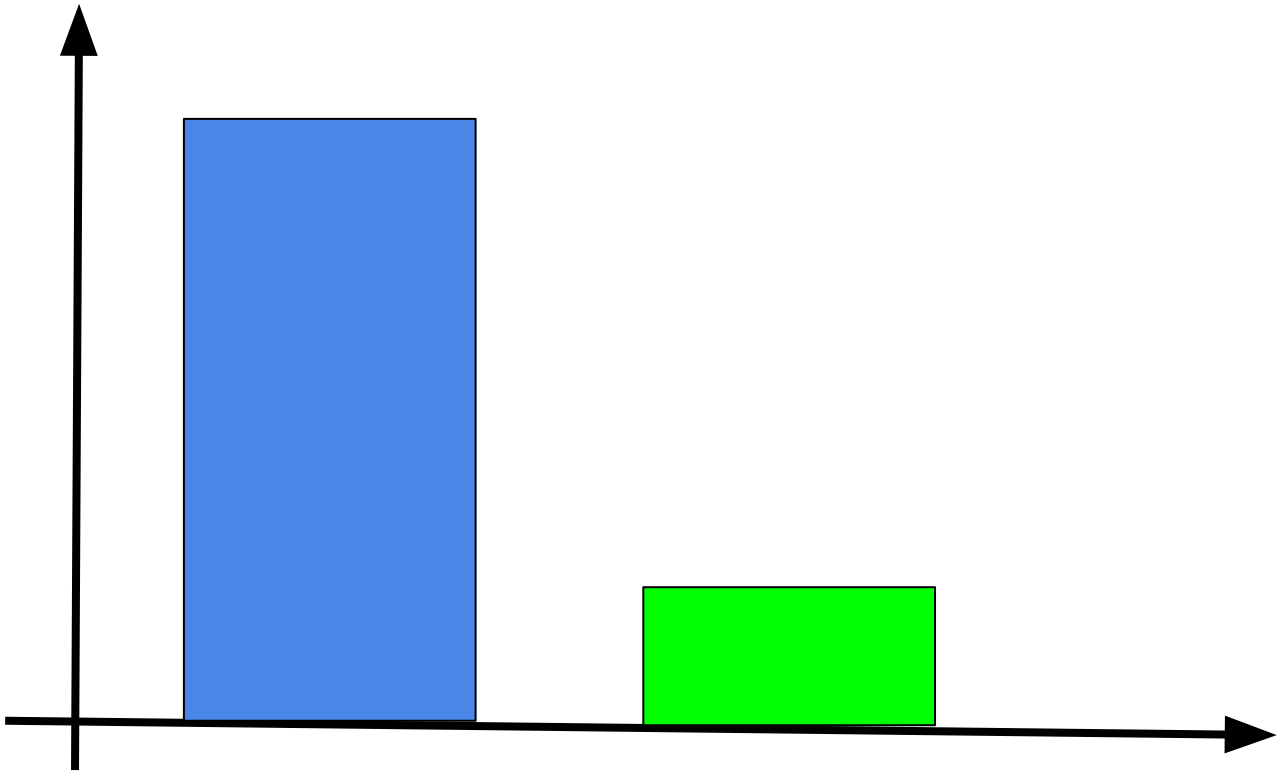
Color

Alcohol Level

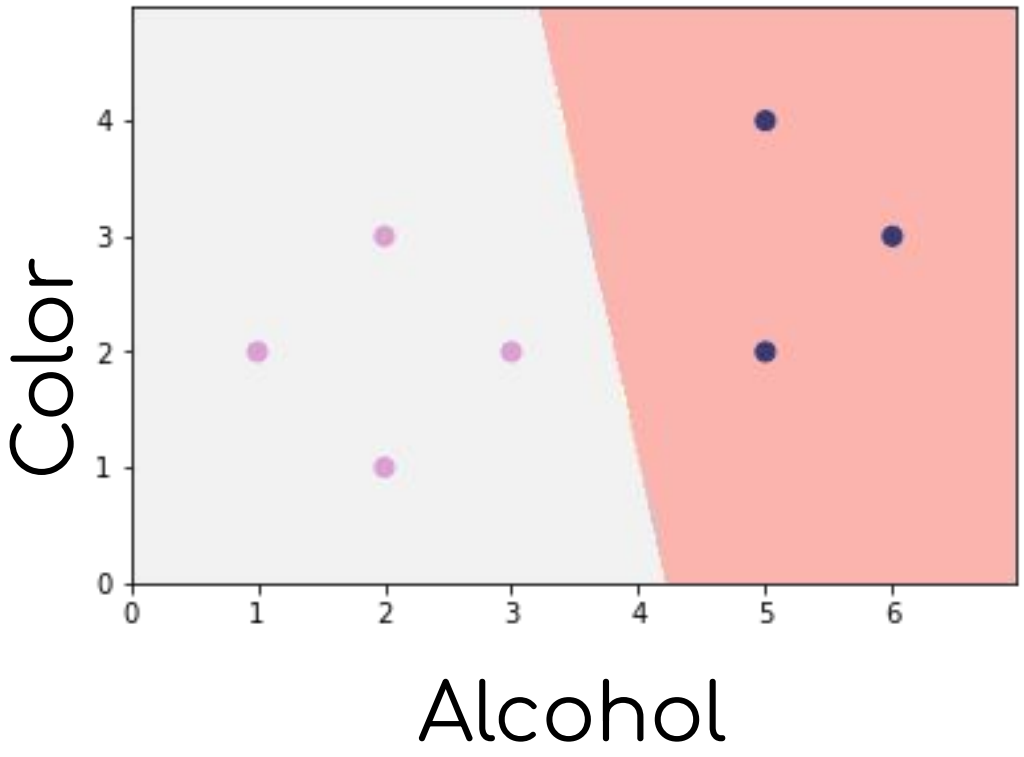
Data Collection

Color (nm)	Alcohol (%)	Wine or Beer?
610	5	Beer
559	14	Wine
693	14	Wine

Training and Testing Datasets



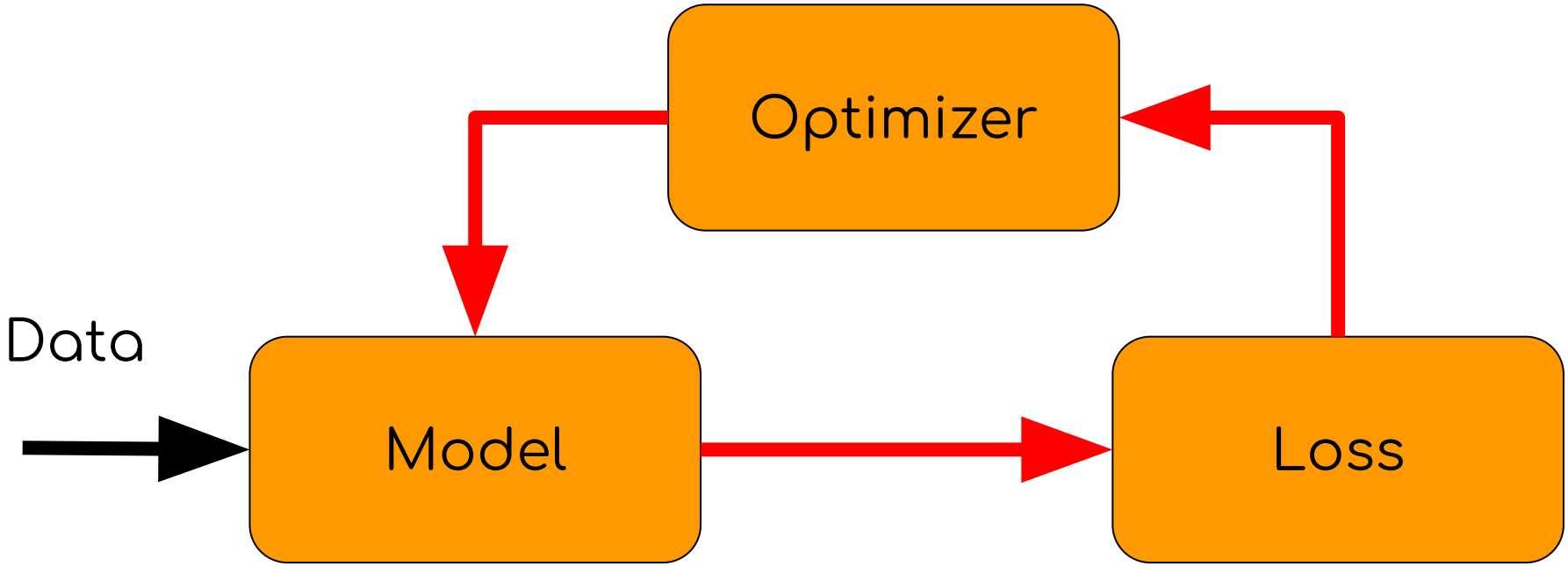
Model



Choose Your Model

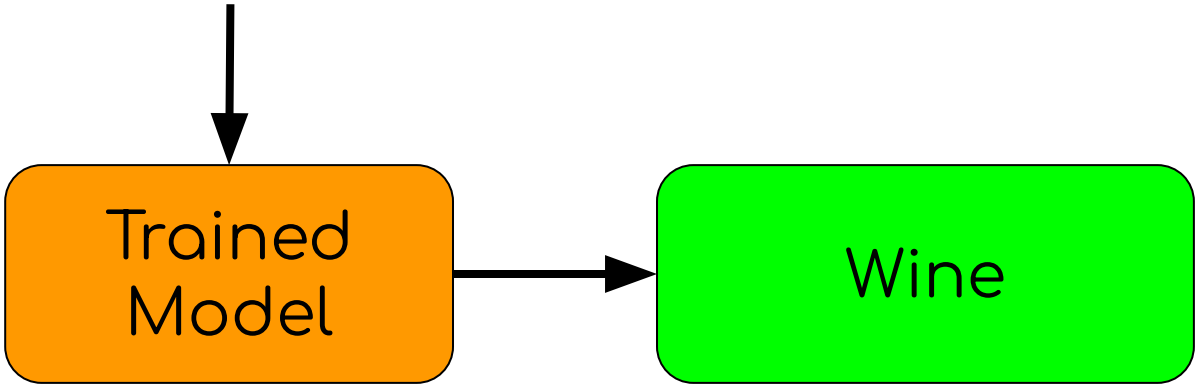
- KNN
- SVM
- Naive Bayes
- Decision Tree
- Random Forest
- Ensemble
- Neural Network
- Many others

Training



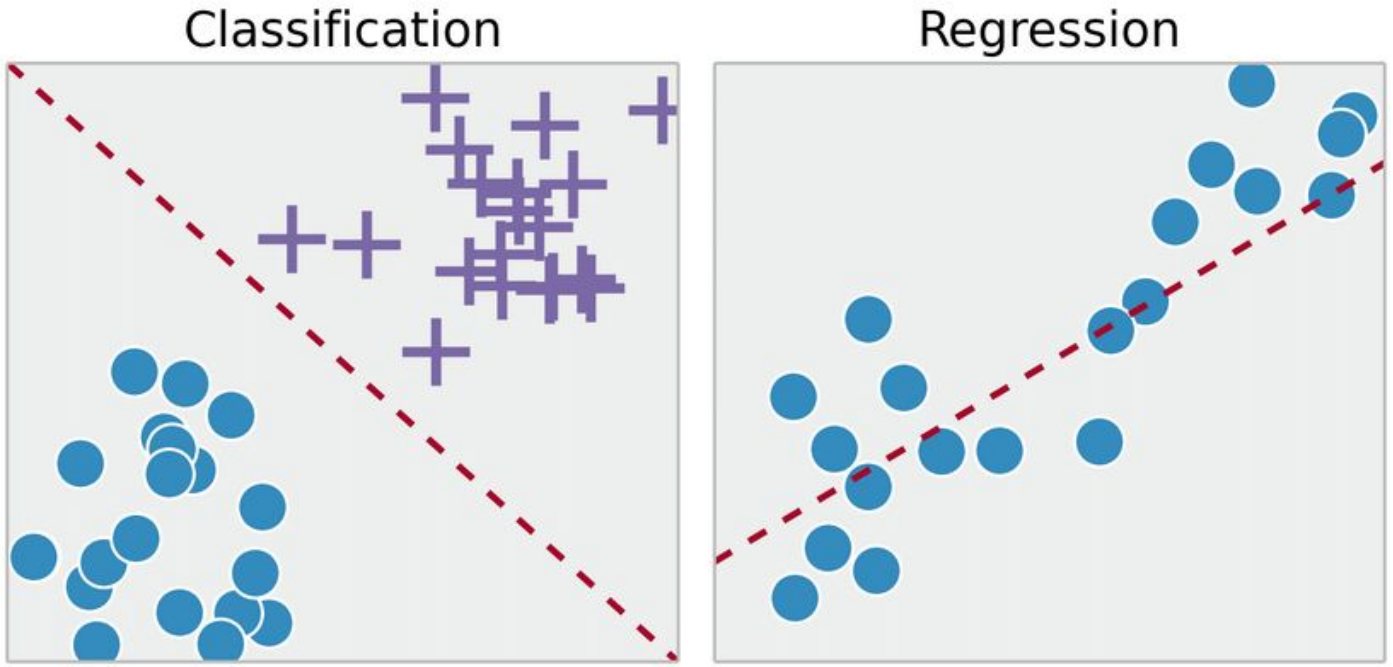
Prediction

Color: 660nm
Alcohol: 12%



What is Classification?

- Classification is to identifying to which category an object belongs to.
- Classification is a supervised learning method. During training, labels/classes are provided.

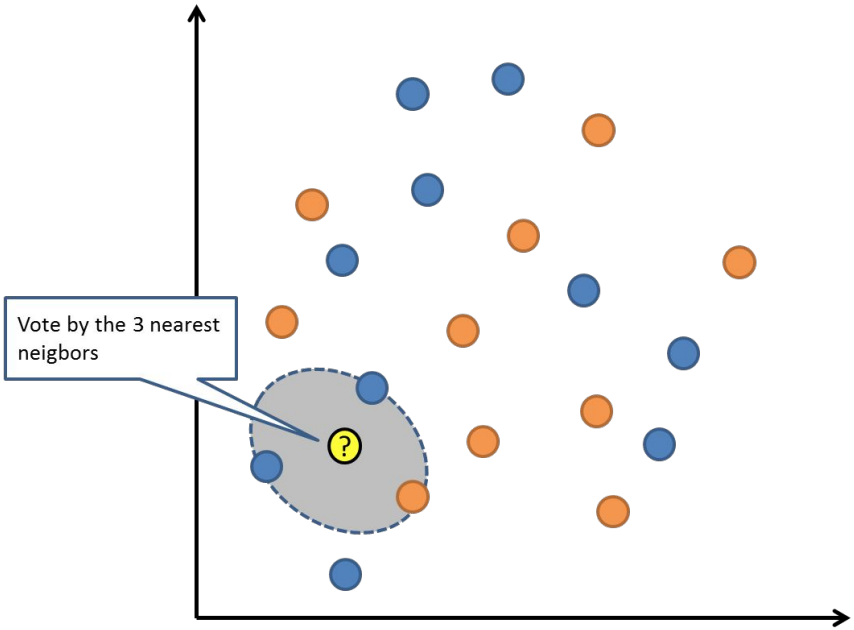


Classification Algorithms

- K Nearest Neighbors
- Logistic Regression
- Support Vector Machine
- Gaussian Naive Bayes
- Stochastic Gradient Descent
- Decision Tree
- Ensemble Methods

K Nearest Neighbors (KNN)

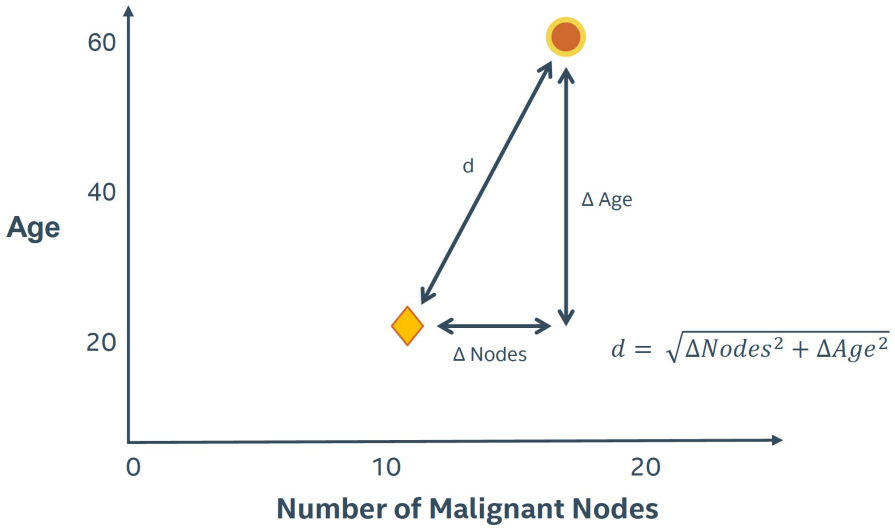
- A simple majority vote of the nearest neighbors of each point.
- Training data is the model. Fitting is fast just store data. However Prediction can be slow because lots of distances to measure
- Decision boundary is flexible



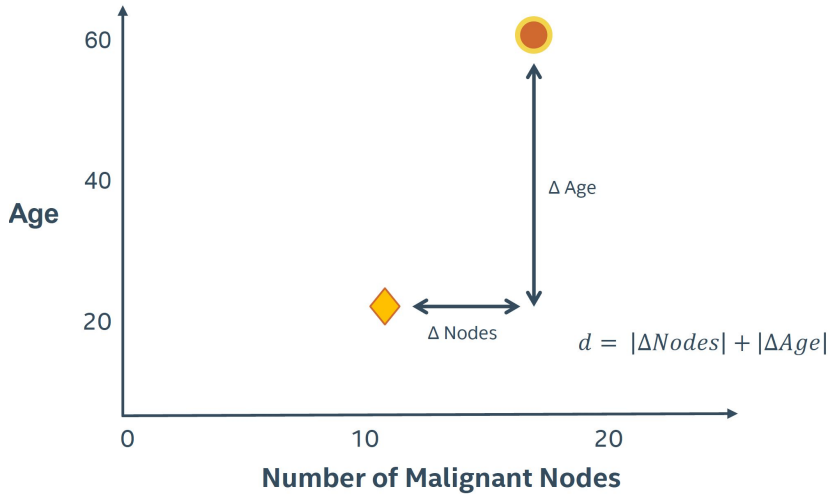
Distance Measure for KNN

- KNN depends on the distance measure
- There are two major distance measures:
  - Euclidean (L2)
  - Manhattan (L1)

Euclidean (L2)



Manhattan (L1)



KNN Classifier

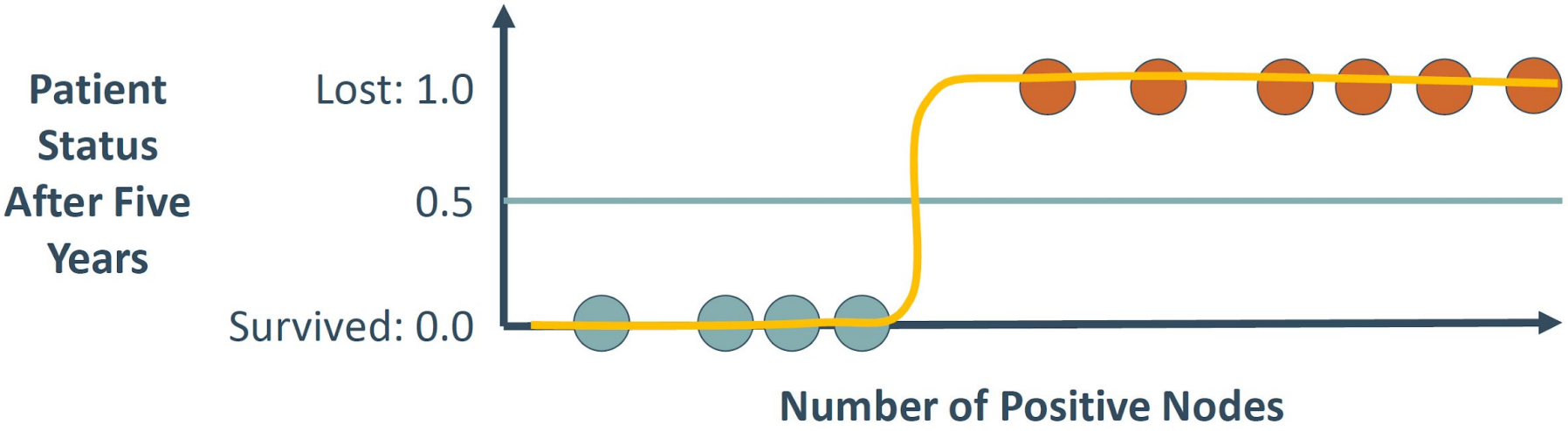
```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
```

Parameters:

```
KNeighborsClassifier(n_neighbors=5,
weights='uniform', algorithm='auto', leaf_size=30, p=2,
metric='minkowski', metric_params=None,
n_jobs=None, **kwargs)
```

Logistic Regression

- Fit the logistic function to the data
- Fitting can be slow must find best parameters
- Prediction is fast calculate expected value
- Decision boundary is simple, less flexible



$$y_\beta(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\varepsilon)}}$$

Logistic Regression Classifier

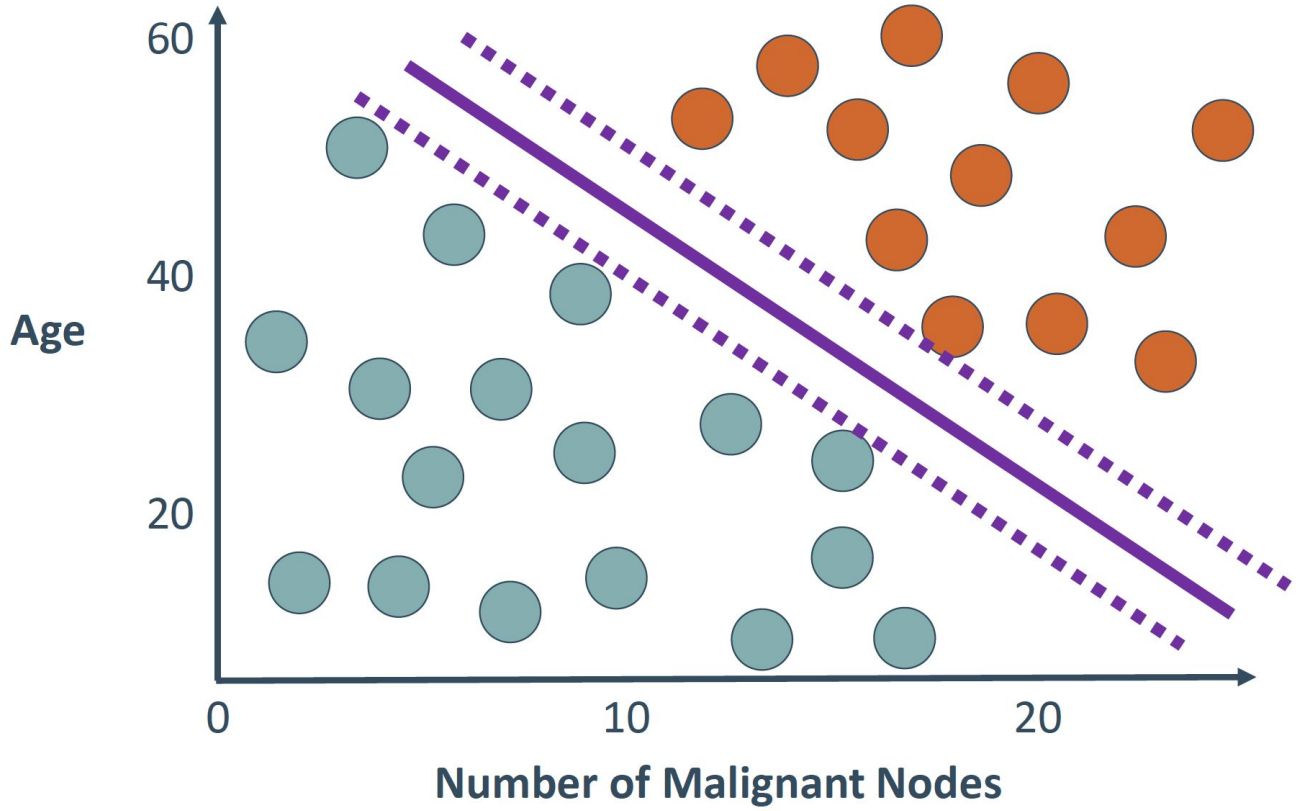
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
```

Default values:

```
LogisticRegression(penalty='l2', dual=False, tol=0.0001,
C=1.0, fit_intercept=True, intercept_scaling=1,
class_weight=None, random_state=None, solver='lbfgs',
max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)[source]
```

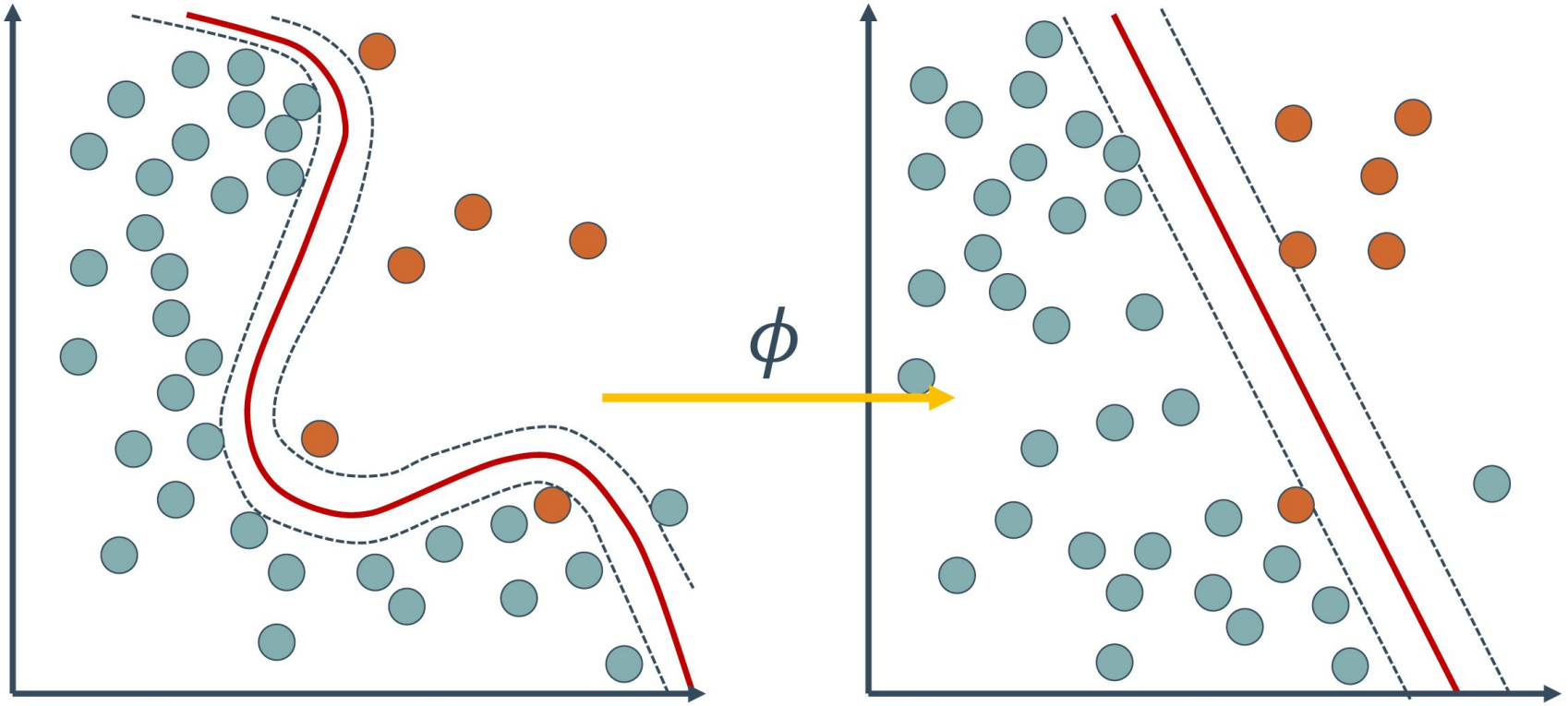
Support Vector Machine (SVM)

Identify a linear hyperplane (boundary) with maximum distance apart



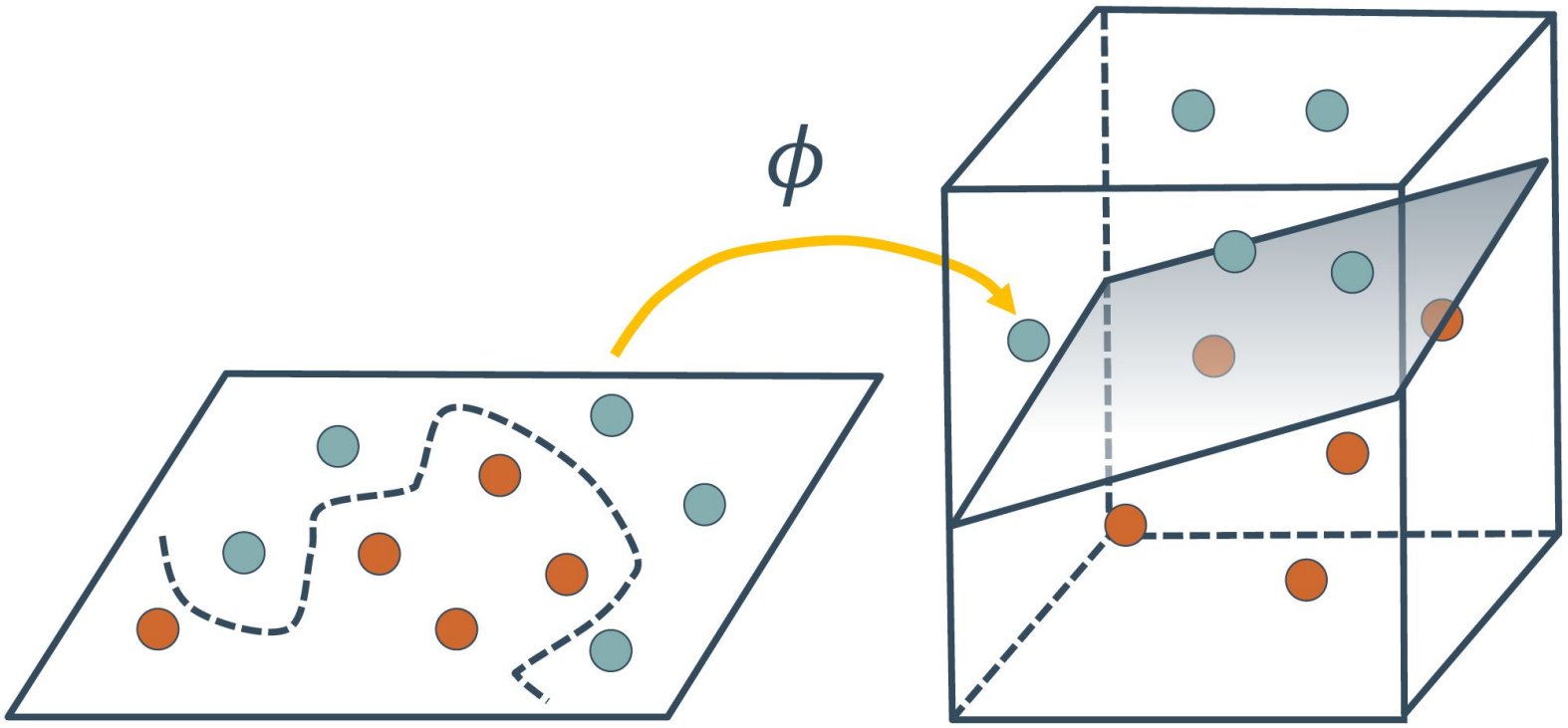
Non-Linear Boundary in SVM

Non-linear data can be made linear with higher dimension using Kernel trick



Kernel Trick

Transform data so that it is linearly separable



Kernels for SVM

The following 4 kernels are available in Scikit Learn:

- linear
- poly
- rbf (default)
- sigmoid

SVM Classifier

```
from sklearn import svm
clf = svm.SVC()
```

Default values:

```
SVC(C=1.0, kernel='rbf', degree=3, gamma='scale',
coef0=0.0, shrinking=True, probability=False, tol=0.001,
cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr',
break_ties=False, random_state=None)
```

Gaussian Naive Bayes (GNB)

- Naive Bayes is a conditional probability model.
- Given the feature vector x, it predict the probability for class C
- GNB assumes each feature is Gaussian distributed

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})} \qquad \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Example of GNB

Problem: Classify whether a given person is a male or a female based on the measured features.

The features include height, weight, and foot size

Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Assuming Gaussian distributed, the feature mean and height are computed as follows

Person	mean (height)	variance (height)	mean (weight)	variance (weight)	mean (foot size)	variance (foot size)
male	5.855	$3.5033*10^{-2}$	176.25	$1.2292*10^{2}$	11.25	$9.1667*10^{-1}$
female	5.4175	$9.7225*10^{-2}$	132.5	$5.5833*10^{2}$	7.5	1.6667

Example of GNB

Below is a sample to be classified as male or female.

Person	height (feet)	weight (lbs)	foot size(inches)
sample	6	130	8

posterior of male ~ 6.19×10^{-9}

posterior of female ~ 5.37×10^{-4}

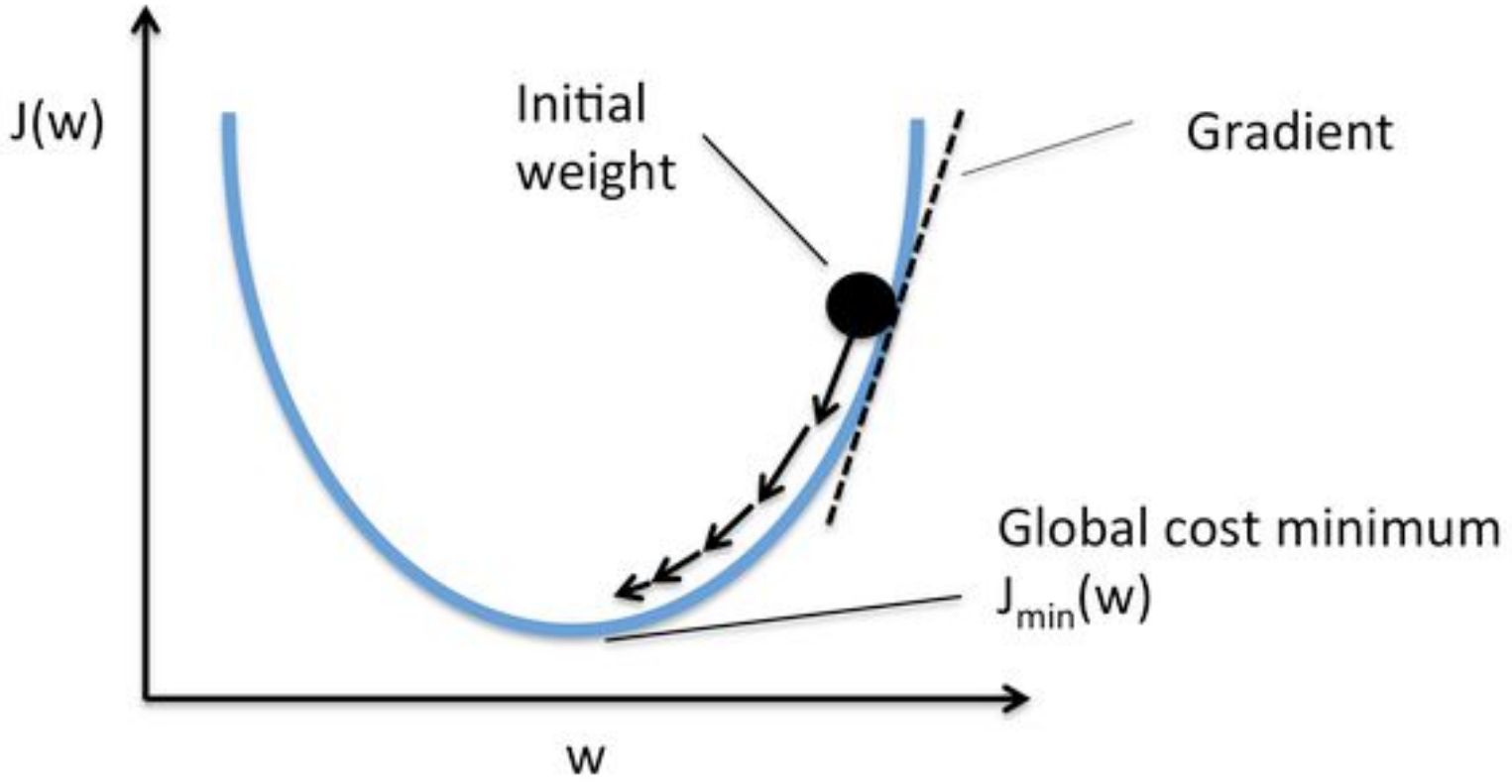
Therefore the person is likely to be female.

GND Classifier

```
from sklearn.naive_bayes import GauassianNB()
clf = GaussianNB()
```

Stochastic Gradient Descent

Stochastic gradient descent (SGD) performs a parameter update for each training using the negative gradient

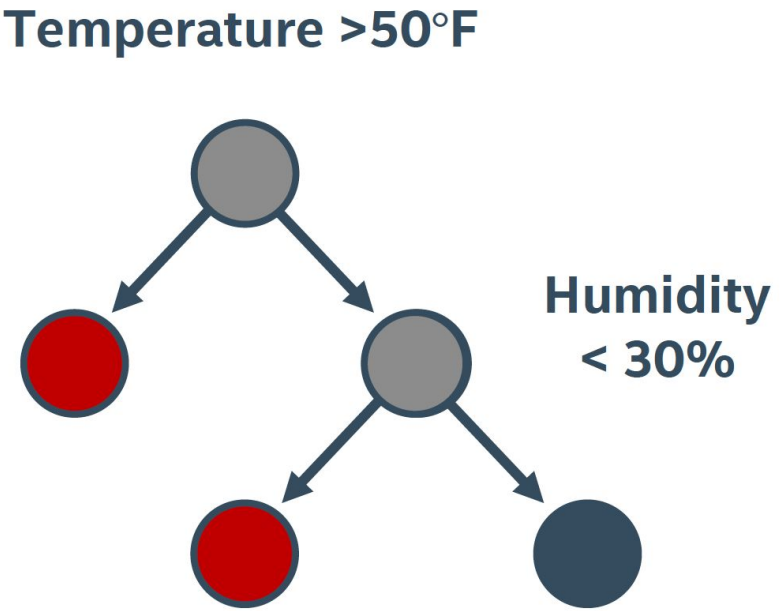


SGD Classifier

```
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier()
```

Decision Tree

- Decision tree is easy to interpret and implement
- Heterogeneous input data allowed, preprocessing
- required
- However, decision trees tend to overfit
- Pruning helps reduce variance to a point. Often not significant for model to generalize well



Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

Iris Flower Dataset

setosa (0)

versicolor (1)

virginica (2)

Iris flower dataset, introduced in 1936 by Sir Ronald Fisher

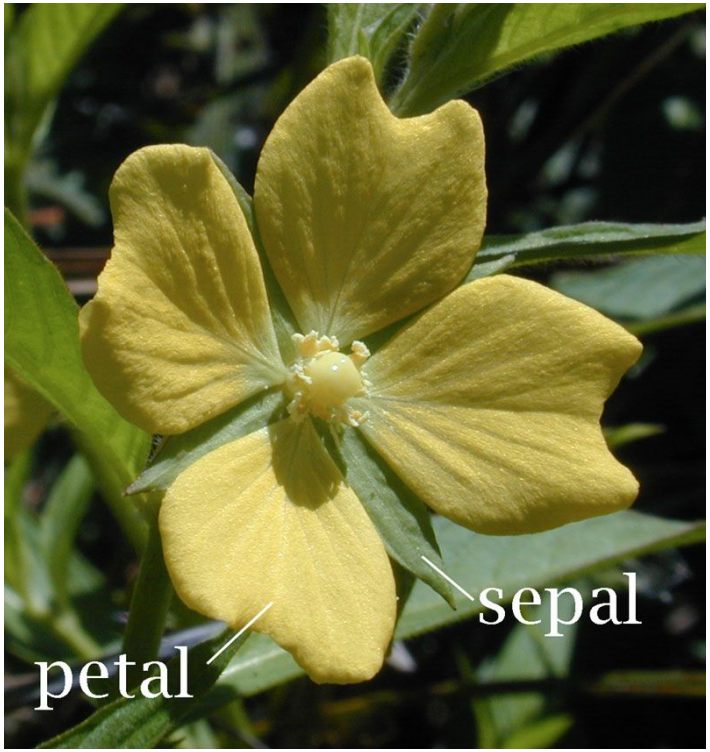
Iris Flower Dataset

Features in the Iris dataset:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Target classes to predict:

- setosa : 0
- versicolor : 1
- virginica : 2



Step 1 Prepare the Data

```
import pandas as pd


dataset_path =
"https://raw.githubusercontent.com/pandas-dev/pandas/master/pandas/tests/data/iris.csv"


X = pd.read_csv(dataset_path)


# Drop any missing data
X = X.dropna()
```

Extract Features and Target

```
y = X.pop('Name')


# Encode the Labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

Split and Randomize Data

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(
                                                X, y,
                                                test_size=0.3,
                                                random_state=0)
```

Normalize/Scale the Features

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 2 Define the Model

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=5)
```

Step 3 Train the Model

```
clf.fit(X_train,y_train)
```

Step 4 Evaluate the Model

```
clf.score(X_test,y_test)
```

Step 5 Save the Model

```
joblib.dump(clf, 'mymodel.pkl')
```

Step 6 Load the Model for Inference

```
from sklearn.externals import joblib
clf2 = joblib.load('iris.pkl')

import numpy as np

X_new = np.array([[6.7,3.1,4.7,1.5]])
y = clf2.predict(X_new)
label = {0:'sentosa',1:'versicolor',2:'virginica'}
print('The flower is ',label[y[0]])
```

Ex: Classification

Compare the following classifiers on iris dataset and evaluate which one is the best and worst classifier

- KNN
- Logistic Regression
- SVM
- GND
- SGD
- DT

What is your guess?

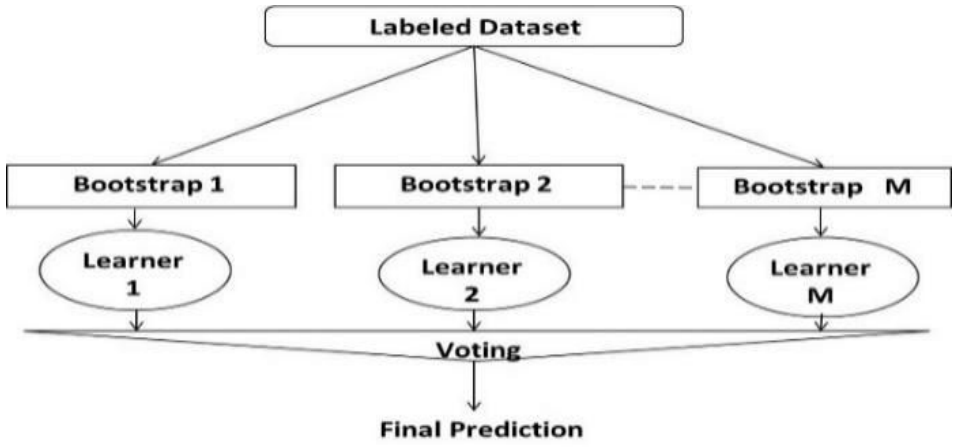
Time: 10 mins

Ensemble Methods

The main principle behind Ensemble methods is that a group of "weak learners" can come together to form a "strong learner".

There are 3 types of Ensemble methods

- **Bagging** (Bootstrap Aggregating) creates separate samples of the training dataset and creates a classifier for each sample. The results of these multiple classifiers are then combined (such as averaged or majority voting).



Ensemble Methods

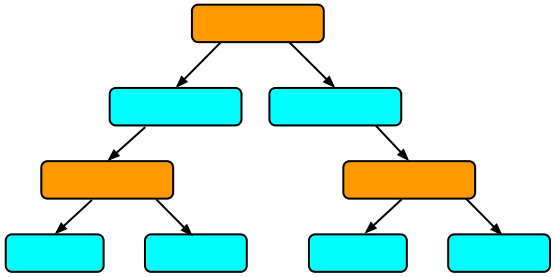
- **Boosting** starts out with a base classifier that is prepared on the training data. A second classifier is then created behind it to focus on the instances in the training data that the first classifier got wrong. The process continues to add classifiers until a limit is reached in the number of models or accuracy

- **Stacking** starts out with a set of base-level classifiers and train a meta-level classifier to combine the outputs of the base-level classifiers

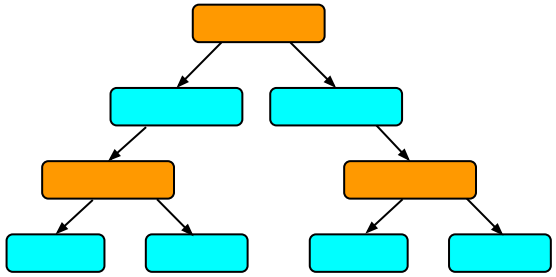
Random Forest

Random Forest classifier is a bagging ensemble method based on lots of decision trees with random selection of subsets of training samples

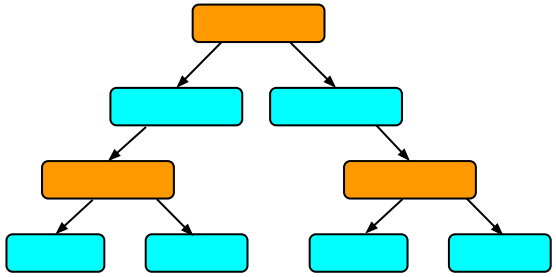
Tree 1



Tree 2



Tree 3



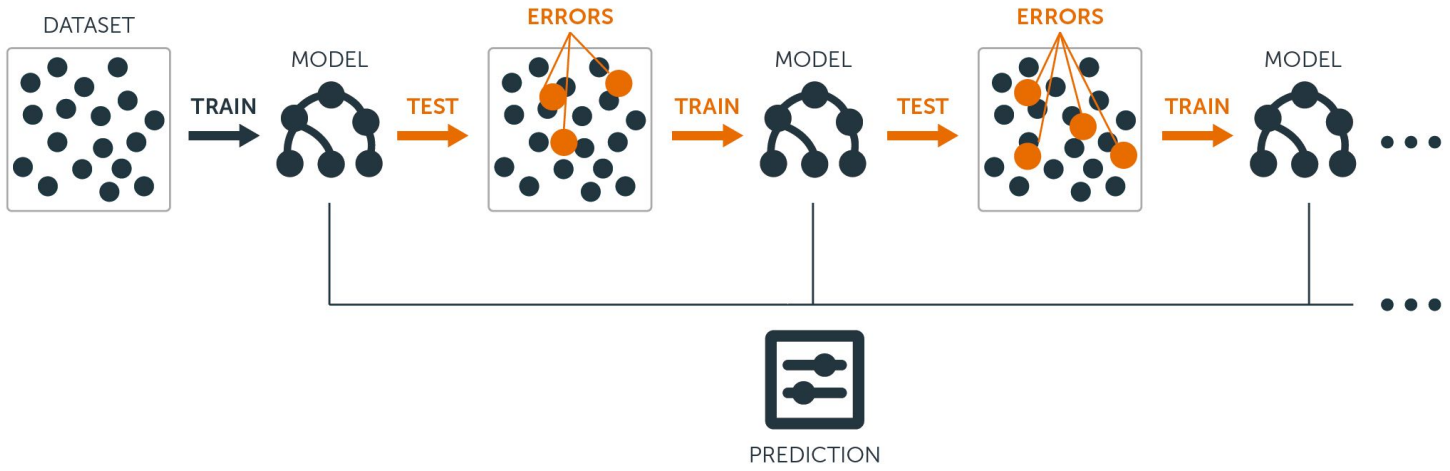
The result is based on the majority votes from all the decision trees

Random Forest Classifier

```
from sklearn.ensemble import
RandomForestClassifier


clf = RandomForestClassifier()
```

Gradient Boosting

- Gradient Boosting is a ensemble boosting method that "boosting" many weak predictive models into a strong one, in the form of ensemble of weak models.
- Boosting utilizes different loss functions At each stage, the margin is determined for each point. Margin is positive for correctly classified points and negative for misclassifications. Value of loss function is calculated from margin



Gradient Boosting Classifier

```
from sklearn.ensemble import
GradientBoostingClassifier


clf = GradientBoostingClassifier()
```

Bagging vs Boosting

Bagging

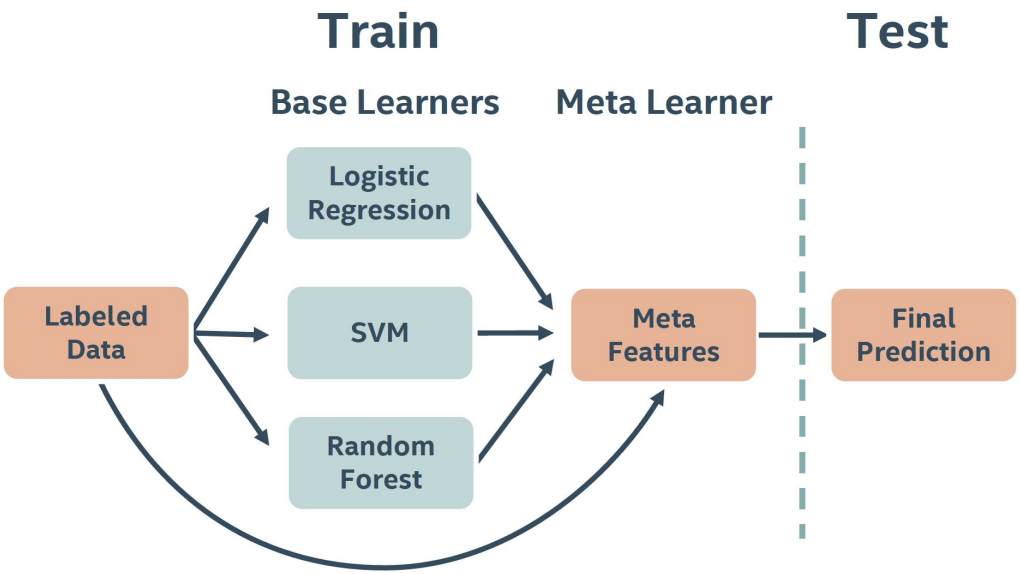
- Bootstrapped samples
- Base trees created independently
- Only data points considered
- No weighting used
- Excess trees will not overfit

Boosting

- Fit entire data set
- Base trees created successively
- Use residuals from previous models
- Up weight misclassified points
- Beware of overfitting

Stacking

- Models of any kind combined to create stacked model
- Output of base learners can be combined via majority vote or weighted
- Additional hold out data needed if meta learner parameters are used
- Be aware of increasing model complexity



Stacking Classifier

```
from sklearn.ensemble import VotingClassifier

clf = VotingClassifier(estimators list, voting='hard')
```

Ex: Ensemble Methods

Load the wine quality data from the following path

dataset_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"

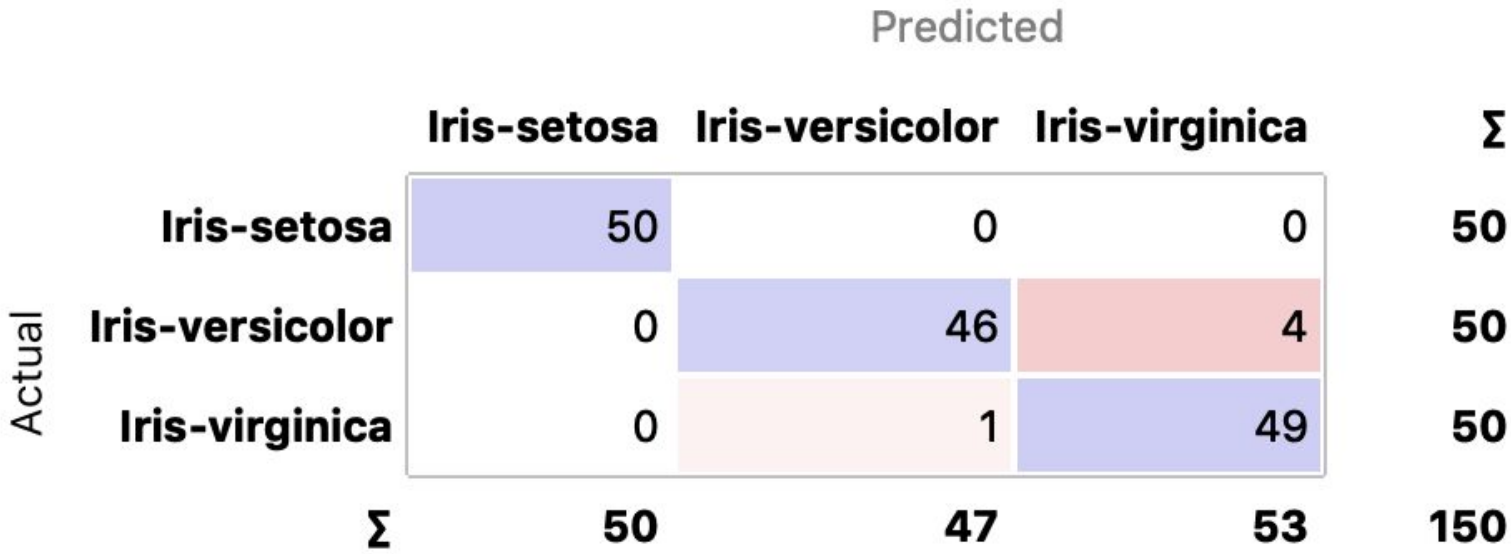
X = pd.read_csv(dataset_path,sep=';')

Determine which methods yield the best score

- Decision Tree
- Random Forest
- Gradient Boost
- Stacking

Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.



Confusion Matrix

```
from sklearn.metrics import confusion_matrix
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
```

	ant (predict)	bird (predict)	cat (predict)
ant (true)	2	0	0
bird (true)	0	0	1
cat (true)	1	0	2

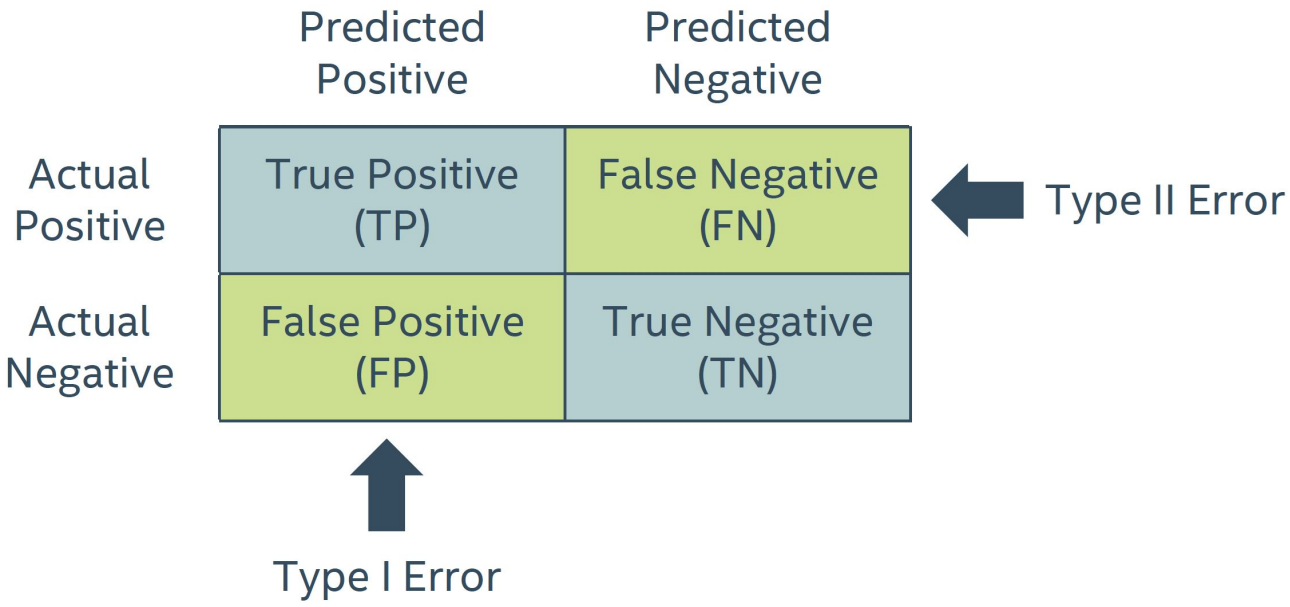
Ex: Confusion Matrix

Compare the confusion matrix for iris data for the following classifiers:

- KNN
- Logistic Regression
- SVM
- GND
- SGD
- DT

Binary Classification

- In many predictive analysis, we are interested in YES/NO analysis such as spam/not-spam, health/not-healthy etc.
- In this case, we can form a confusion matrix of 2 columns and 2 rows



Accuracy

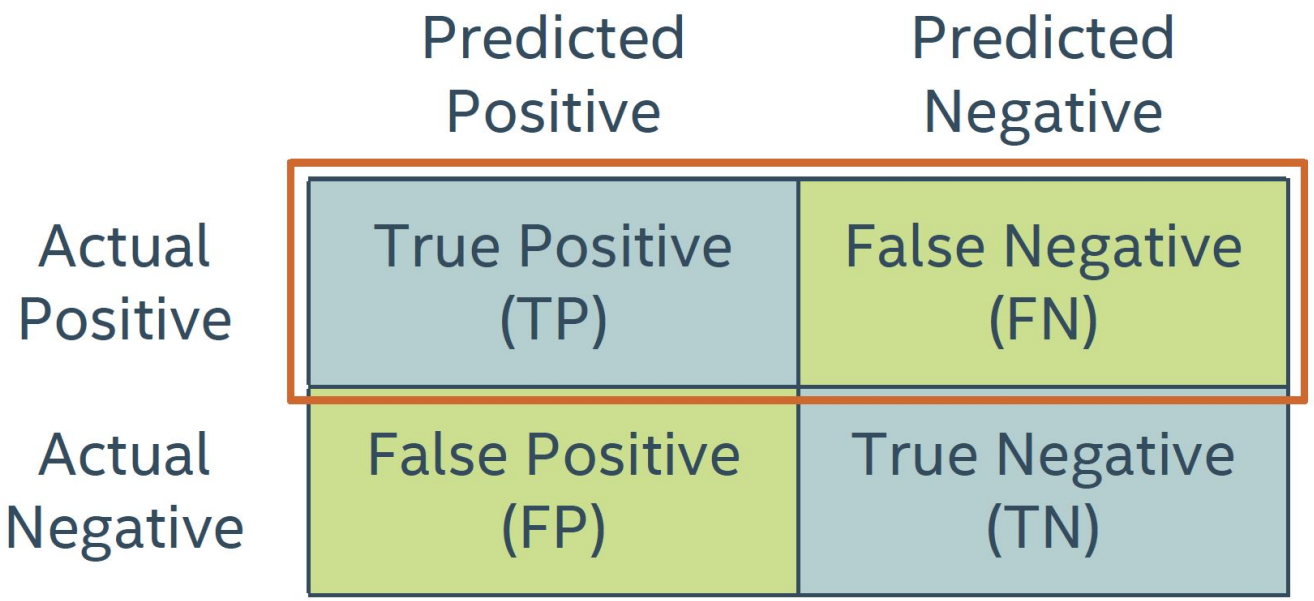
Accuracy is the percentage of correct hits.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

Recall or Sensitivity

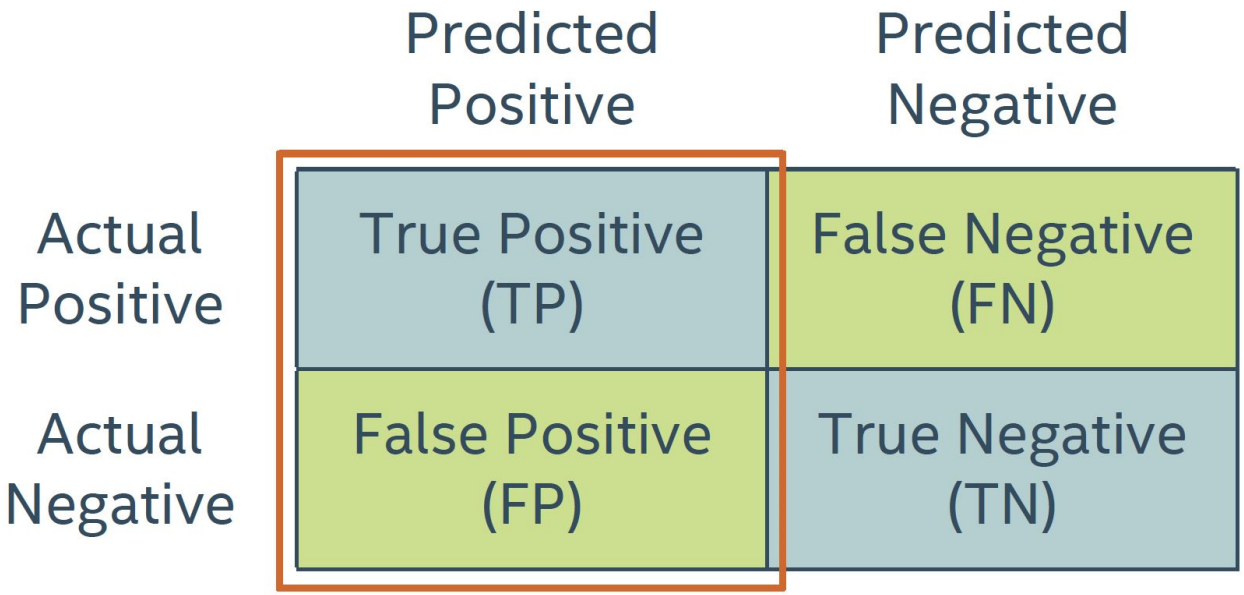
Recall or Sensitivity is to identify all the positive instances



$$\text{Recall or Sensitivity} = \frac{TP}{TP + FN}$$

Precision

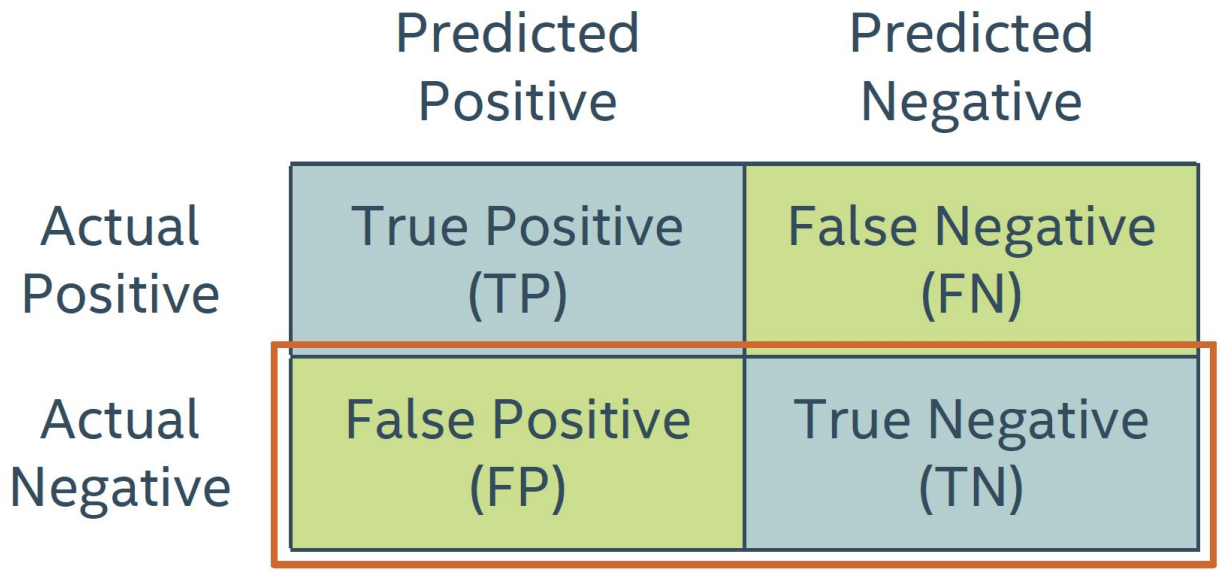
Precision is to identify only the positive instances



$$\text{Precision} = \frac{TP}{TP + FP}$$

Specificity

Specificity is to identify only all the negative instances to detect false alarm.



$$\text{Specificity} = \frac{TN}{FP + TN}$$

F1 Score

- For error measurement of a particular model, need to check all the four metrics
- However, it is more convenient to check a single parameter using F1 score

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall or Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{FP + TN}$$

$$F1 = 2\,\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Individual Metrics for Binary Classification

```
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score

from sklearn import neighbors
clf = neighbors.KNeighborsClassifier()
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
print('Accuracy = ',accuracy_score(y_test,y_pred))
print('Precision = ',precision_score(y_test,y_pred))
print('Recall = ',recall_score(y_test,y_pred))
print('F1 Score = ',f1_score(y_test,y_pred))
```

Classification Report

The classification report ishow the precision, recall and f1 score

```
from sklearn.metrics import classification_report
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]

classification_report(y_true, y_pred, labels=["ant", "bird", "cat"])
```

	precision	recall	f1-score	support
ant	0.67	1.00	0.80	2
bird	0.00	0.00	0.00	1
cat	0.67	0.67	0.67	3

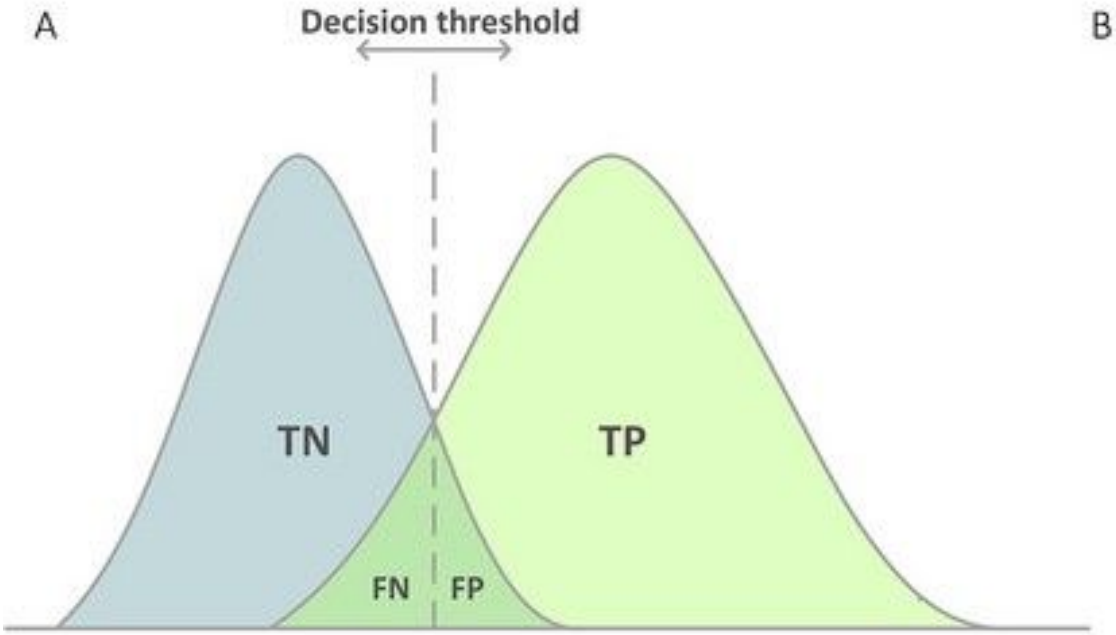
Ex: Classification Report

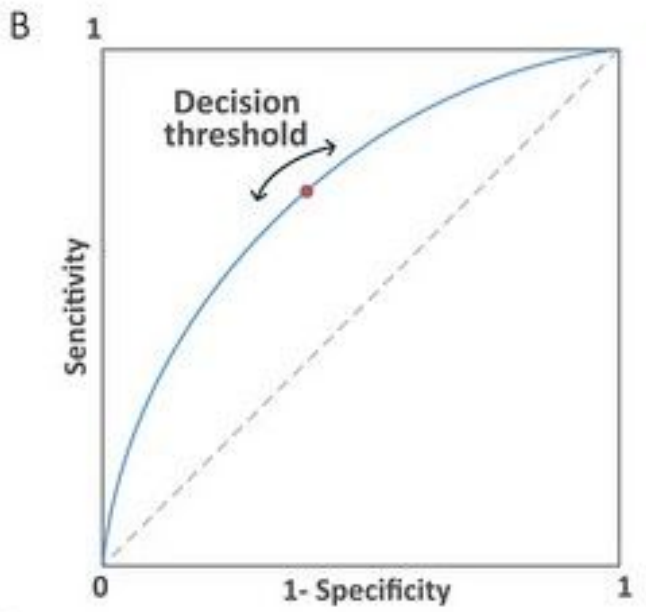
Compare the accuracy and F1 Score for iris data for the following classifiers:

- KNN
- Logistic Regression
- SVM
- GND
- SGD
- DT

Fine Tuning Binary Classification

It is tempting to assume that the classification threshold (decision threshold) should always be 0.5, but thresholds are problem-dependent, and are therefore values that you must tune.



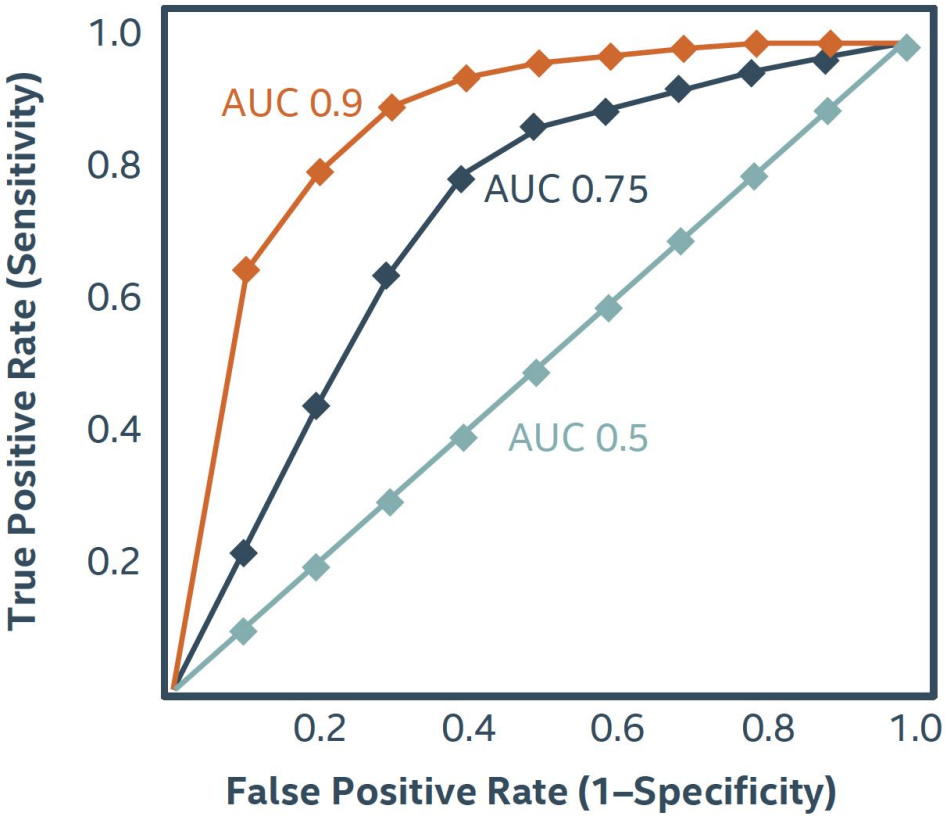


Receiver Operating Characteristics (ROC)

- Receiver Operating Characteristics (ROC) can be used to evaluate all the possible thresholds
- ROC is a plot of True vs False Positive Rates.

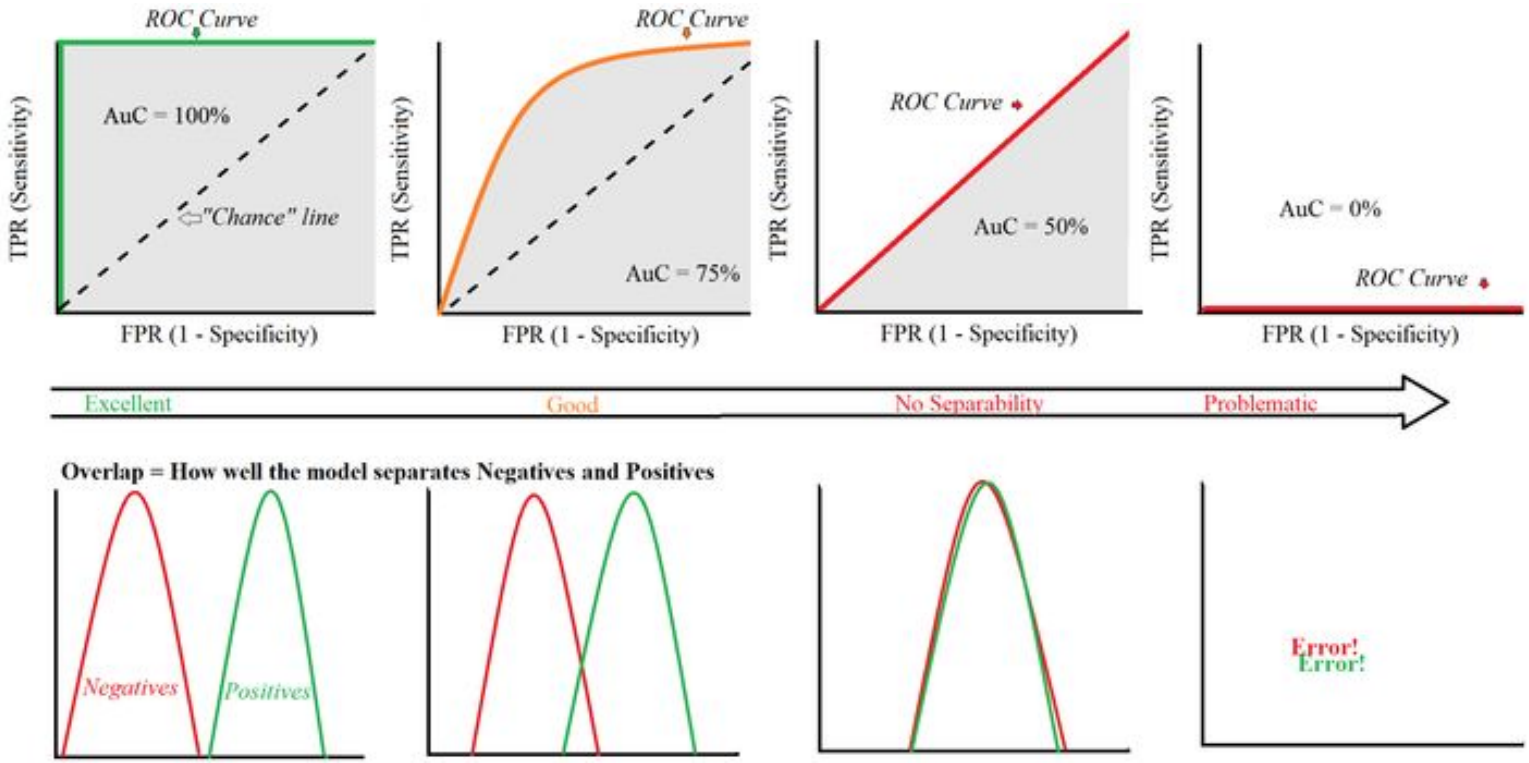
Area Under Curve (AUC)

- AUC measures the area under the ROC curve.
- It is a measure of how good is the binary classification model



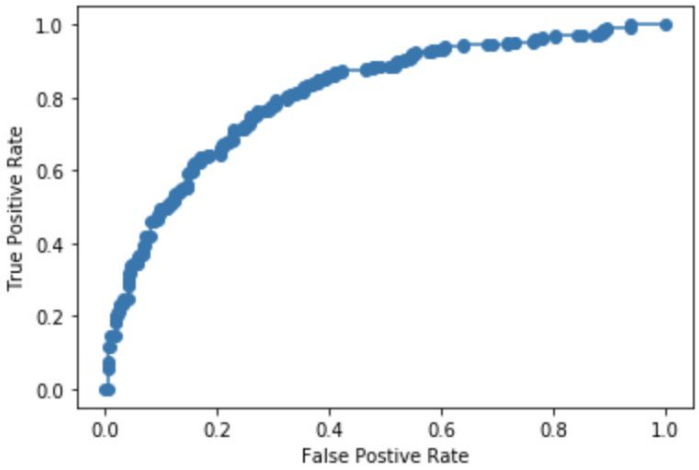
ROC and AUC

With a ROC curve, you're trying to find a good model that optimizes the trade off between the False Positive Rate (FPR) and True Positive Rate (TPR). What counts here is how much area is under the curve (Area under the Curve = AuC).



ROC and AUC

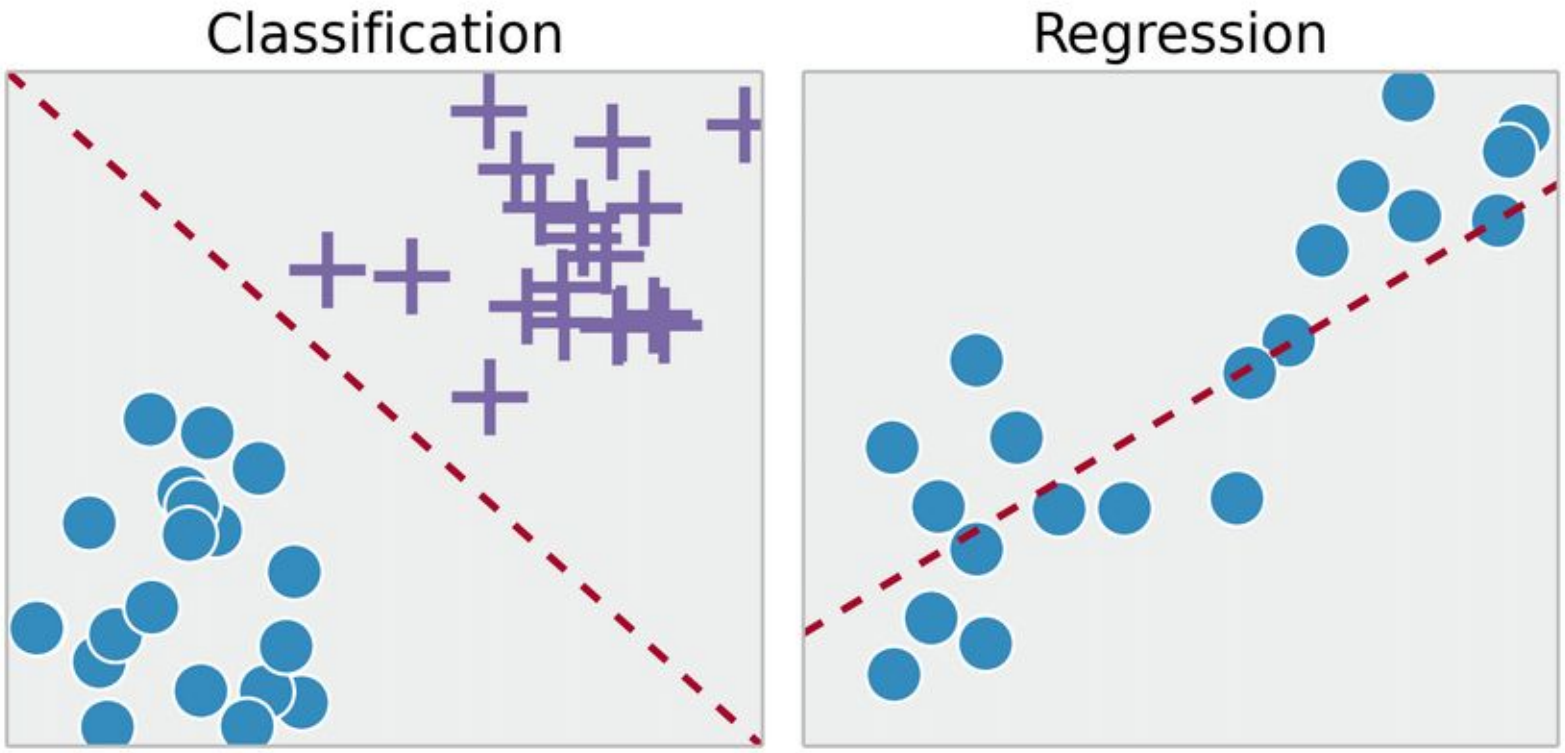
```
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train,y_train)

# predict probabilities
y_probs = clf.predict_proba(X_test)
y_probs = y_probs[:, 1]

auc_score = roc_auc_score(y_test, y_probs)
print("AUC = ", auc_score)
fpr, tpr, threshold = roc_curve(y_test,y_probs)
```



Topic 3
Regression

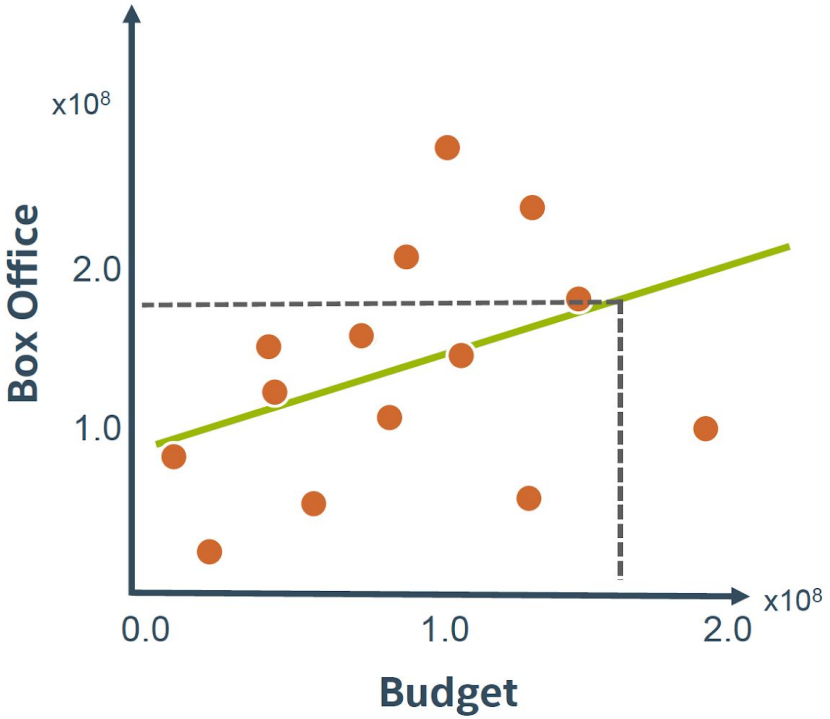
What is Regression?

- Regression it to predicting a continuous-valued attribute associated with an object.
- Regression is a supervised learning method. During training, actual values are provided.



Regression Applications

- Linear regression is the most common regression model. Many predictive models use linear regression models
- You can use a linear regression model to predict the box office from the budget.



$$y_\beta(x) = \beta_0 + \beta_1 x$$

β_0= 80 million, β_1= 0.6

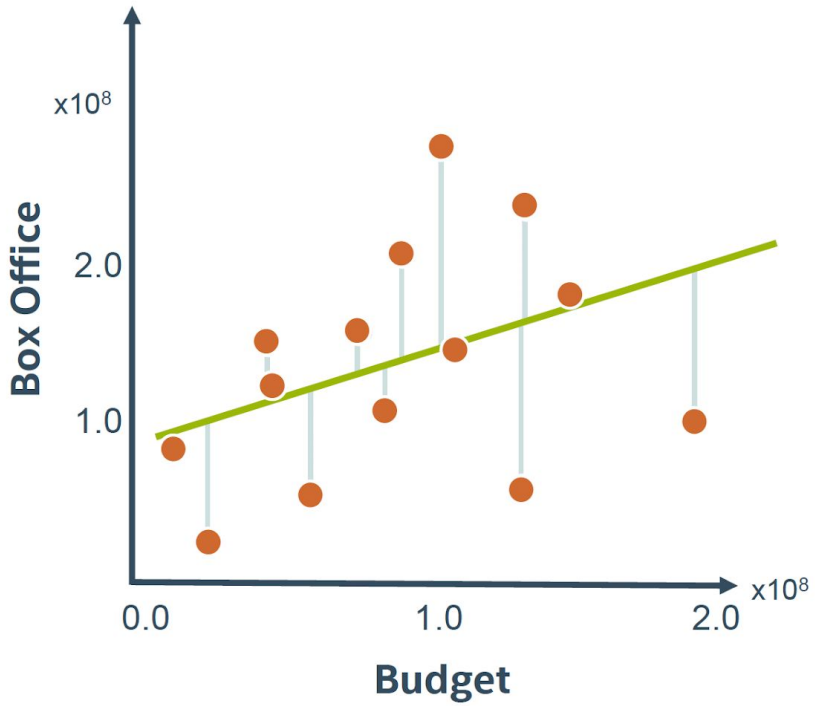
Predict 175 Million Gross for 160 Million Budget

Regression Algorithms

- Linear Regression (Most Common)
- Ridge Regression
- Lasso Regression
- Elastic Net Regression

Residues

- Residue is the difference between the predicted value and actual value



$$y_\beta\left(x_{obs}^{(i)}\right) - y_{obs}^{(i)}$$

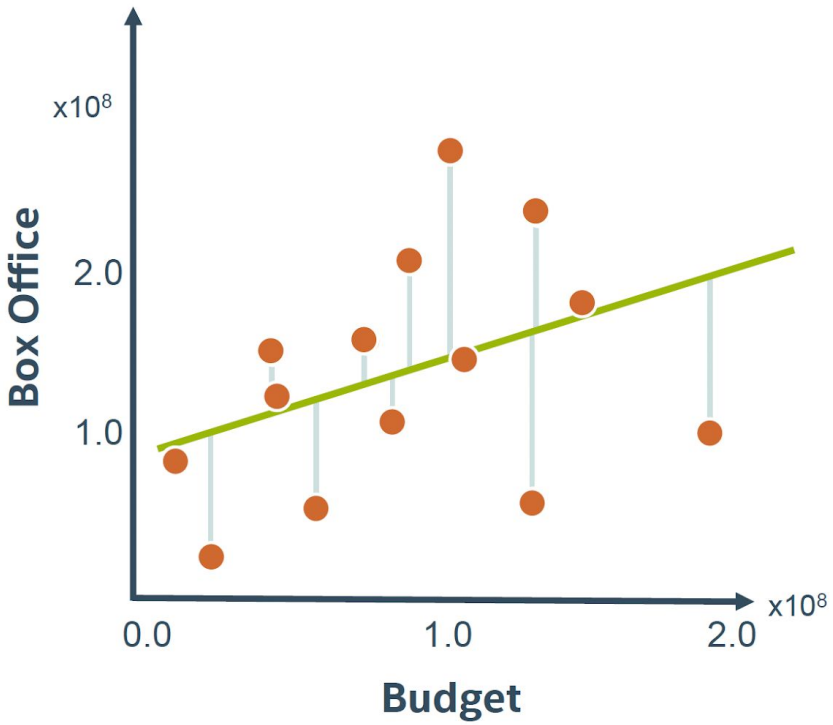
Predicted value

Observed value

$$\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}$$

Mean Square Error

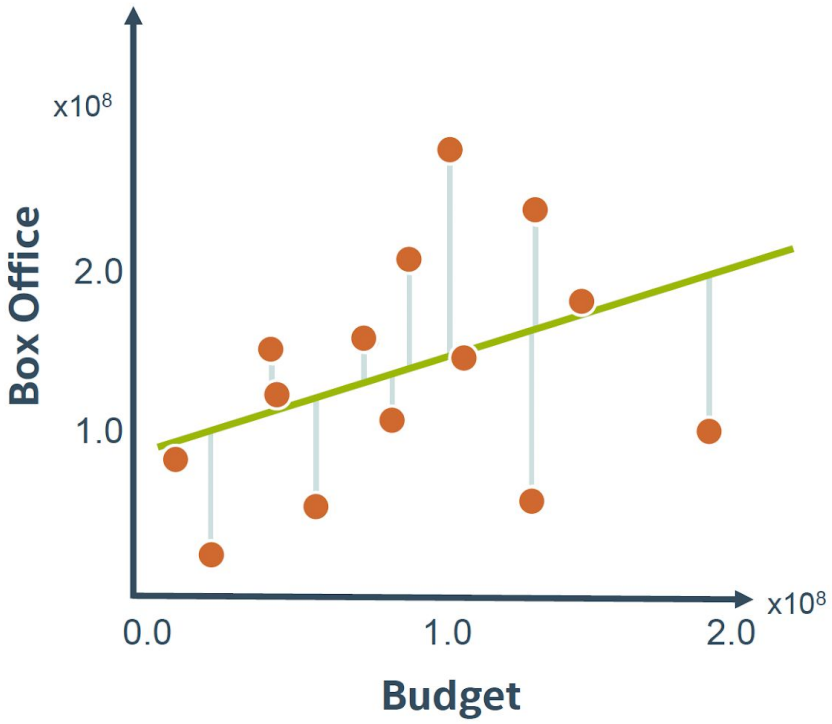
- Mean Square Error (MSE) is the common loss function to measure how good is the linear regression model.



$$\frac{1}{m}\sum_{i=1}^{m}\left(\left(\beta_0 + \beta_1 x_{obs}^{(i)}\right) - y_{obs}^{(i)}\right)^2$$

Minimum Mean Square Error

- Machine Learning aims to minimize the MSE to find the best linear regression model.



$$\min_{\beta_0,\beta_1} \frac{1}{m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

Demo: Build Predictive Regression Model

This is a demo of how to build a predictive regression model with linear. The steps are as follows:

- **Prepare the data**
  - Import raw data
  - Clean data eg remove missing data
  - Normalize the data
  - Split raw data to train and test dataset
  - Create input and output
- **Build the model**
  - Create a linear regression model
- **Train the model**
- **Evaluate the model**
- **Make prediction**

Boston Housing Price Dataset

There are 14 features for this dataset.

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in $1000's

Import the Data

We will get some raw data from github Eg boston hosting price

import pandas as pd

dataset_path = "https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"

Prepare the Data for Training

- **Remove missing data**

  X = X.dropna()

- **Create input and output**

y = X.pop('medv')

Split train and test

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.3,random_state=100)
```

Normalize/Scale the Input Data

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Define the Linear Regression Model

```
from sklearn import linear_model

lm = linear_model.LinearRegression()
```

Train the Model

```
lm.fit(X_train,y_train)
```

Model Metrics

Sum of Squared Error (SSE):

$$\sum_{i=1}^{m} \left(y_\beta(x^{(i)}) - y_{obs}^{(i)} \right)^2$$

Total Sum of Squares (TSS):

$$\sum_{i=1}^{m} \left(\overline{y_{obs}} - y_{obs}^{(i)} \right)^2$$

Correlation Coefficient (R2):

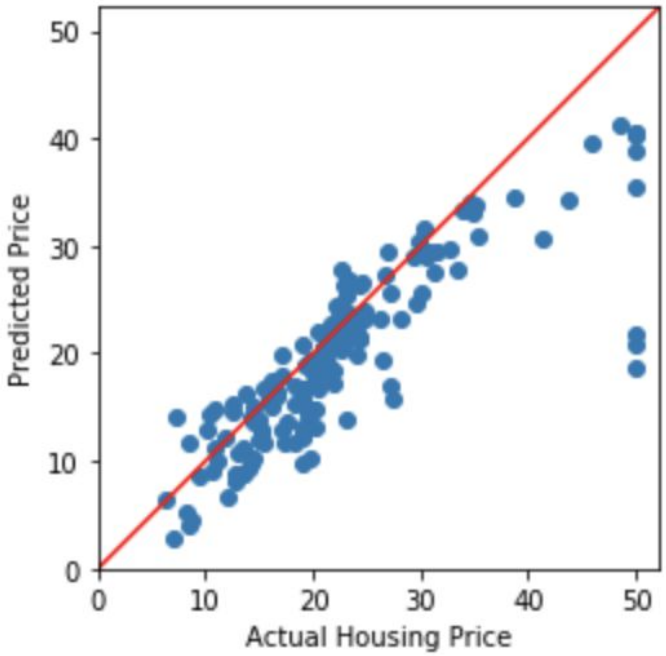
$$1 - \frac{SSE}{TSS}$$

Model Metrics

```
from sklearn.metrics import mean_squared_error, r2_score

yhat = lm.predict(X_train)
mse = mean_squared_error(y_train,yhat)
print('Mean Squared Error, Training: ',mse)
rsq = r2_score(y_train,yhat)
print('R-square, Training: ',rsq)

yhat = lm.predict(X_test)
mse = mean_squared_error(y_test,yhat)
print('Mean Squared Error, Testing: ',mse)
rsq = r2_score(y_test,yhat)
print('R-square, Testing: ',rsq)
```

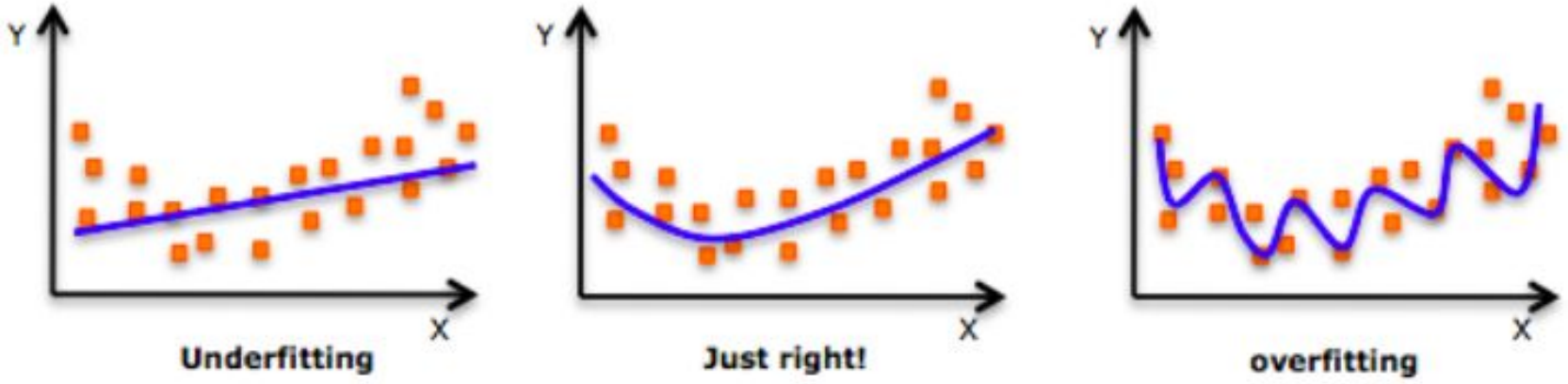
Evaluate the Model

```
%matplotlib inline
import matplotlib.pyplot as plt

yhat = lm.predict(X_test)
plt.scatter(y_test,yhat)
plt.xlabel('Actual Housing Price')
plt.ylabel('Predicted Price')
plt.plot([0, 50], [0, 50],'r')
```



Underfitting and Overfitting

- Too many parameters might result in overfitting.
- Too few parameters might result in underfitting.
- One way to reduce overfitting is to apply regularizations to the regression model.



Regularisations

Regularisations can be used to penalize large coefficients and suppress overfitting issue.

Ridge Regularization

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} {\beta_j}^2$$

Lasso Regularization

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda \sum_{j=1}^{k} |\beta_j|$$

Elastic Net Regularization

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2 + \lambda_1 \sum_{j=1}^{k} |\beta_j| + \lambda_2 \sum_{j=1}^{k} \beta_j^2$$

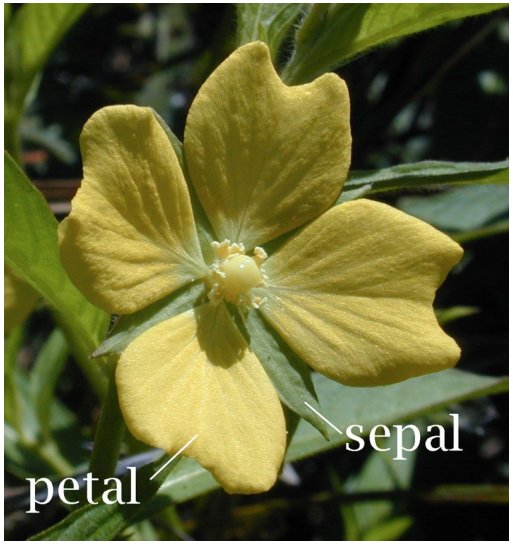
Regularizations

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
rr = Ridge(alpha=0.01)
lr = Lasso(alpha=0.01)
er = ElasticNet(alpha=0.01,l1_ratio=0.5)

rr.fit(X_train,y_train)
lr.fit(X_train,y_train)
er.fit(X_train,y_train)
```

Ex: Predictive Regression Model

- Import the iris.dataset from github
  https://raw.githubusercontent.com/pandas-dev/pandas/master/pandas/tests/data/iris.csv"
- Create a linear regression model to predict the sepal width from sepal length, petal length and petal width
- Perform all the steps as shown in the demo
  - Prepare the data
  - Build the model
  - Train the model
  - Make prediction
  - Save the model

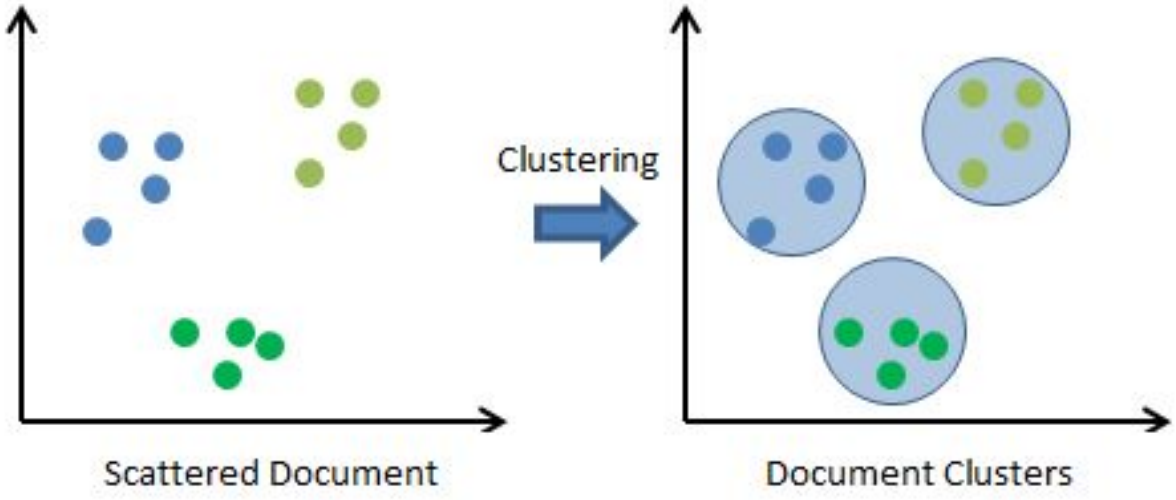
Time: 30 mins



Topic 4
Clustering

What is Clustering?

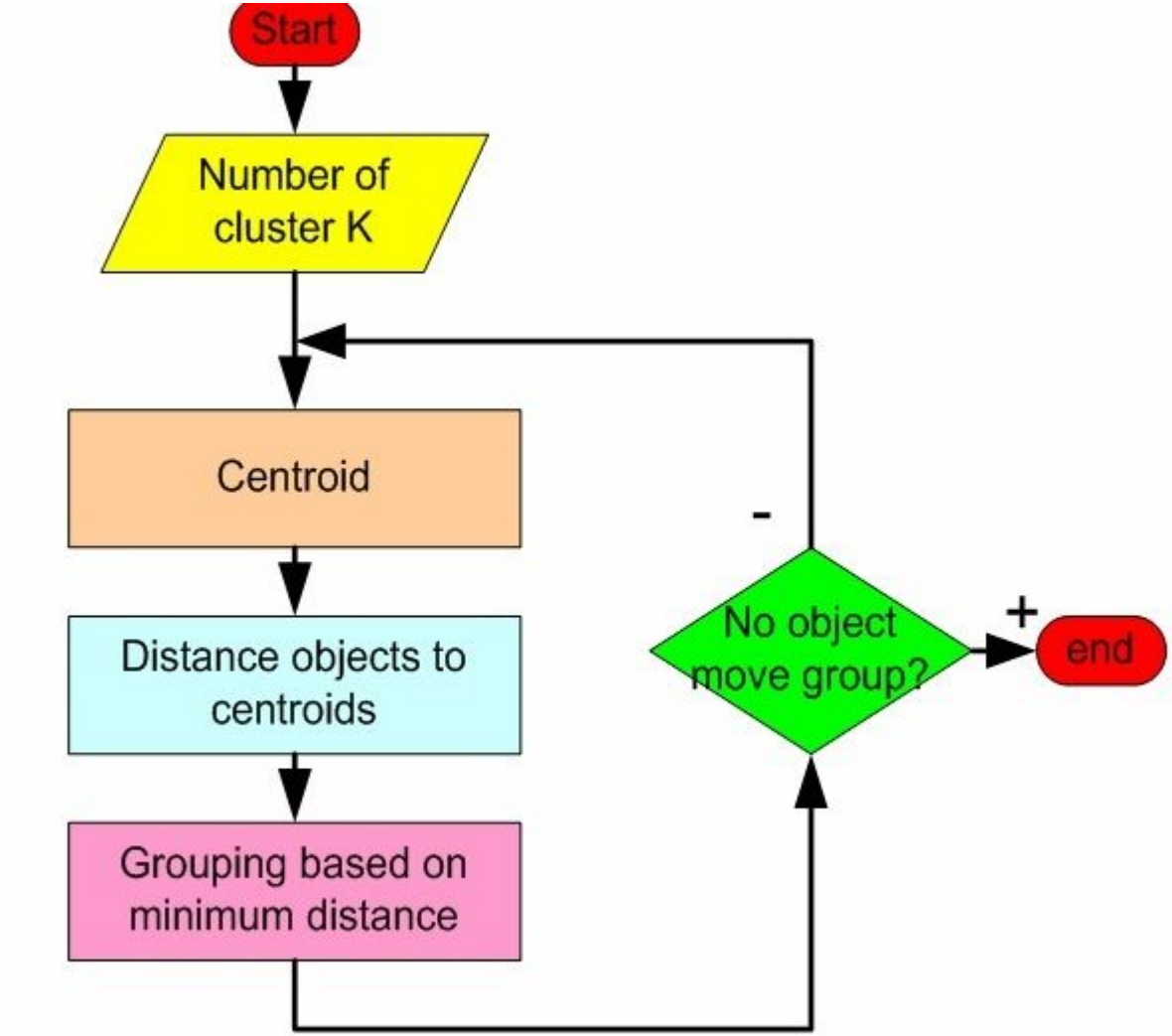
- Clustering is the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar to each other than to those in other groups (clusters).
- Clustering is a unsupervised learning since no labels (targets) are needed for training.



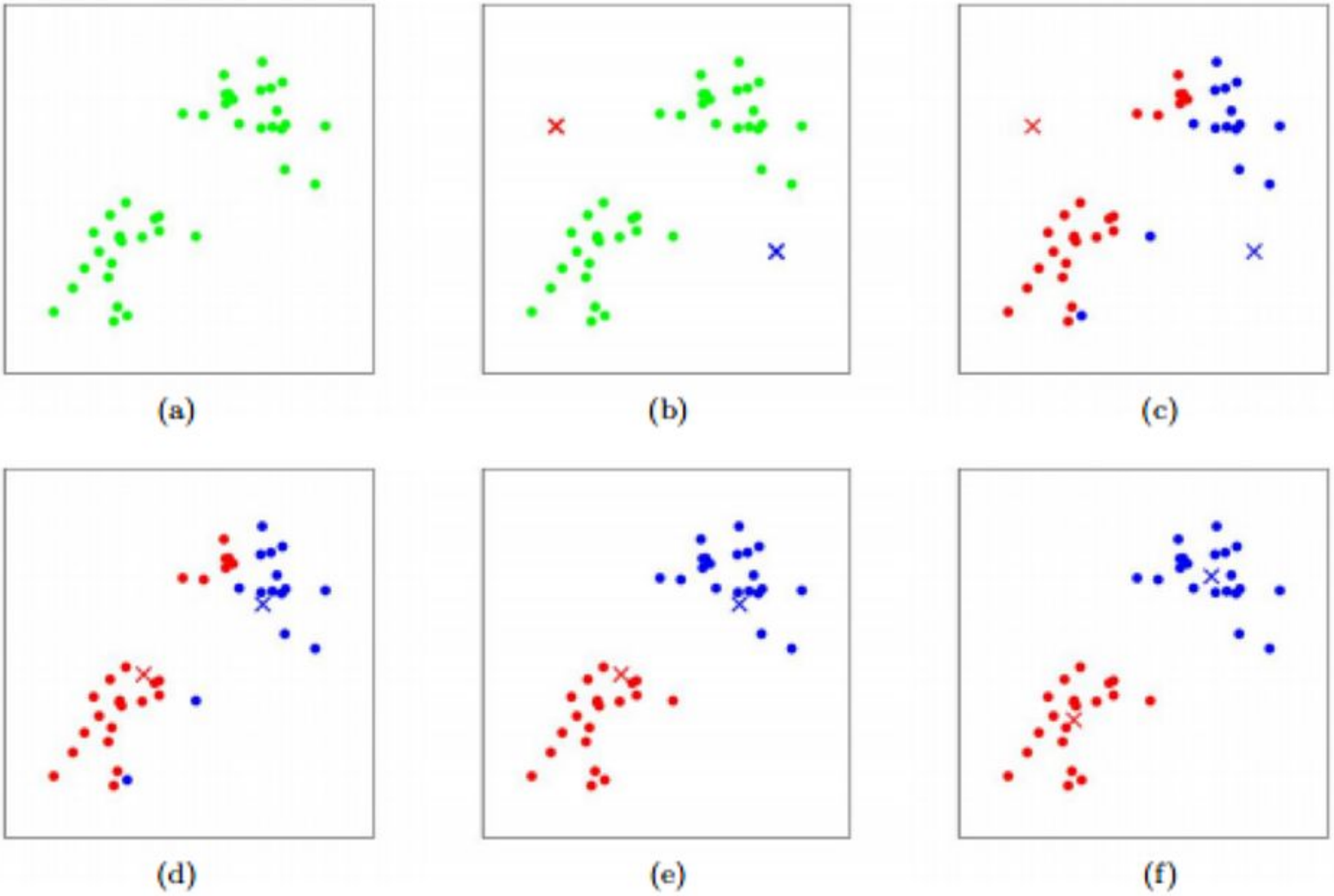
Key Clustering Algorithms

- K-Means Clustering
- Hierarchical Agglomerative Clustering

K-Means Algorithm



K-Means Algorithm



K Mean Algorithm Simulation

Click on the images below to start k-mean clustering simulation (k=4)

Seeds starts on left

Seeds randomly assigned

K-Means Pros & Cons

Pros:

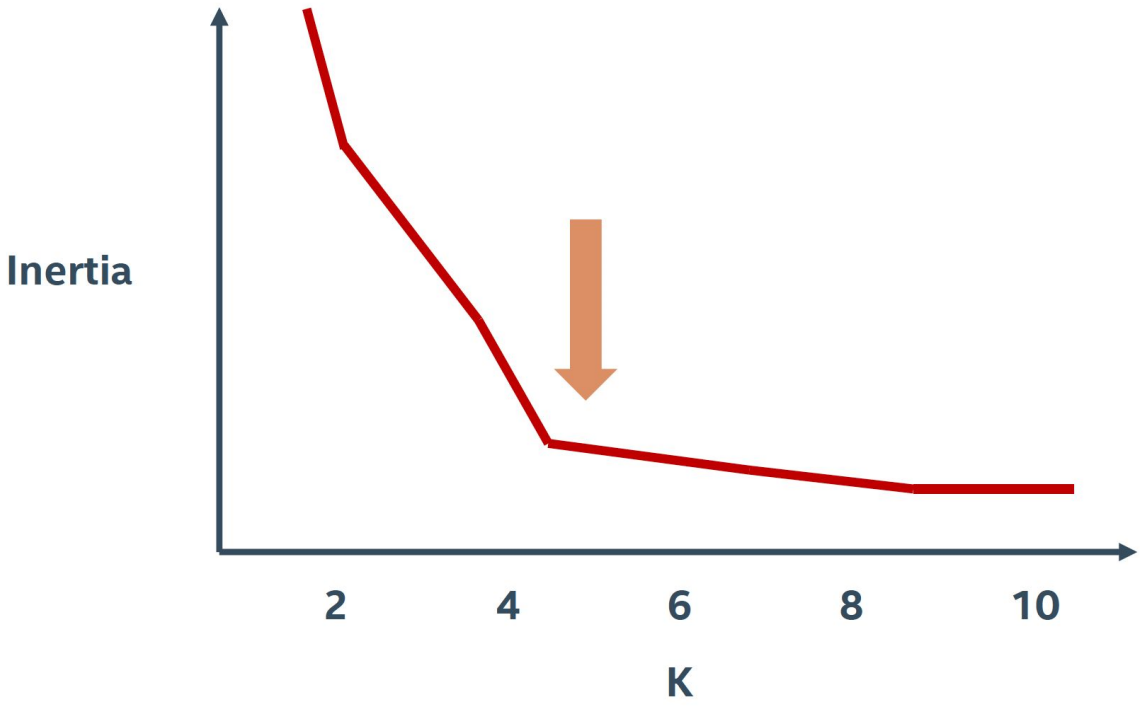
- Simple and fast, Easy to implement

Cons:

- Need to choose K
- Sensitive to outliers
- Prone to local minima.
- Extremely sensitive to initialization.
- Bad initialization can lead to:
  - poor convergence speed
  - bad overall clustering

How to Choose K

- Inertia is sum of squared distance from each point xi) to its cluster Ck
- Smaller value corresponds to tighter clusters
- Value decreases with increasing K as long as cluster density increases

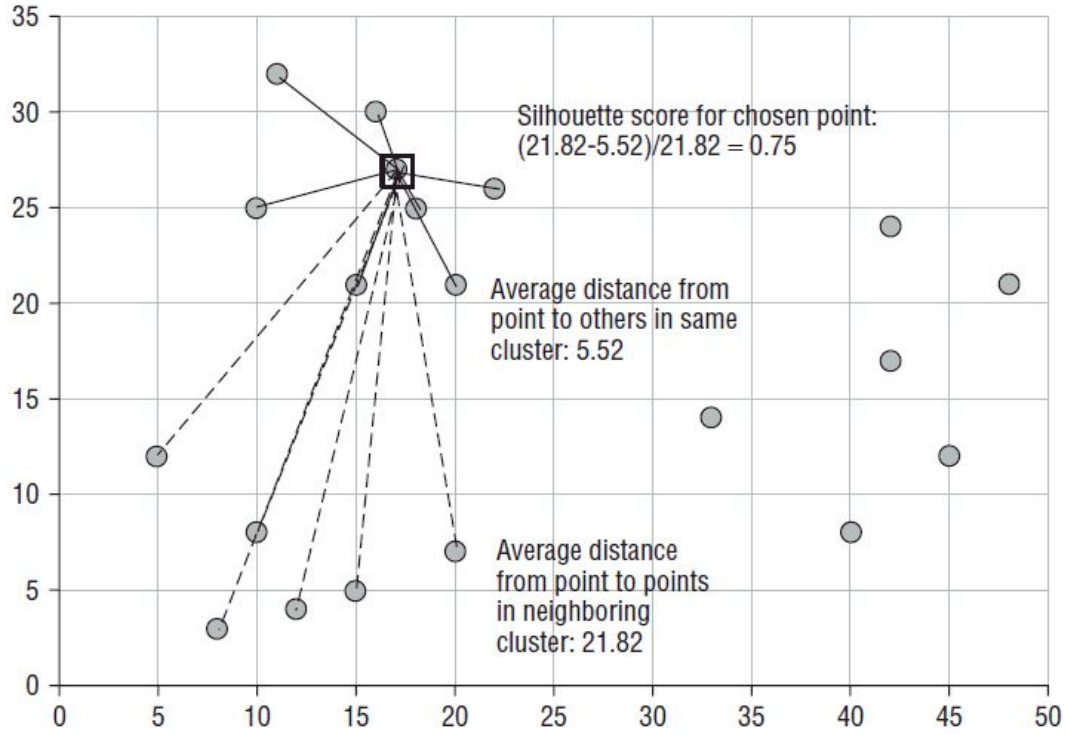


Silhouette Analysis

- Silhouette analysis refers to a method of interpretation and validation of consistency within clusters of data.
- The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).
- It can be used to study the separation distance between the resulting clusters
- If the Silhouette index value is high, the object is well-matched to its own cluster and poorly matched to neighbouring clusters.

Silhouette Score

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max(a, b)



Silhouette Analysis

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max(a, b)

```
from sklearn.metrics import silhouette_score
for i in range(2,5):
    cluster = KMeans(n_clusters=i,random_state=10)
    cluster.fit(X)
    s = silhouette_score(X, cluster.labels_)
    print('Cluster: ',i, 'Silhouette Score :',s )
```

K-Means Clustering

sklearn.cluster.KMeans(n_clusters = ...)

n_clusters: number of clusters

Clustering Attributes

cluster_centers_: Coordinates of cluster centers

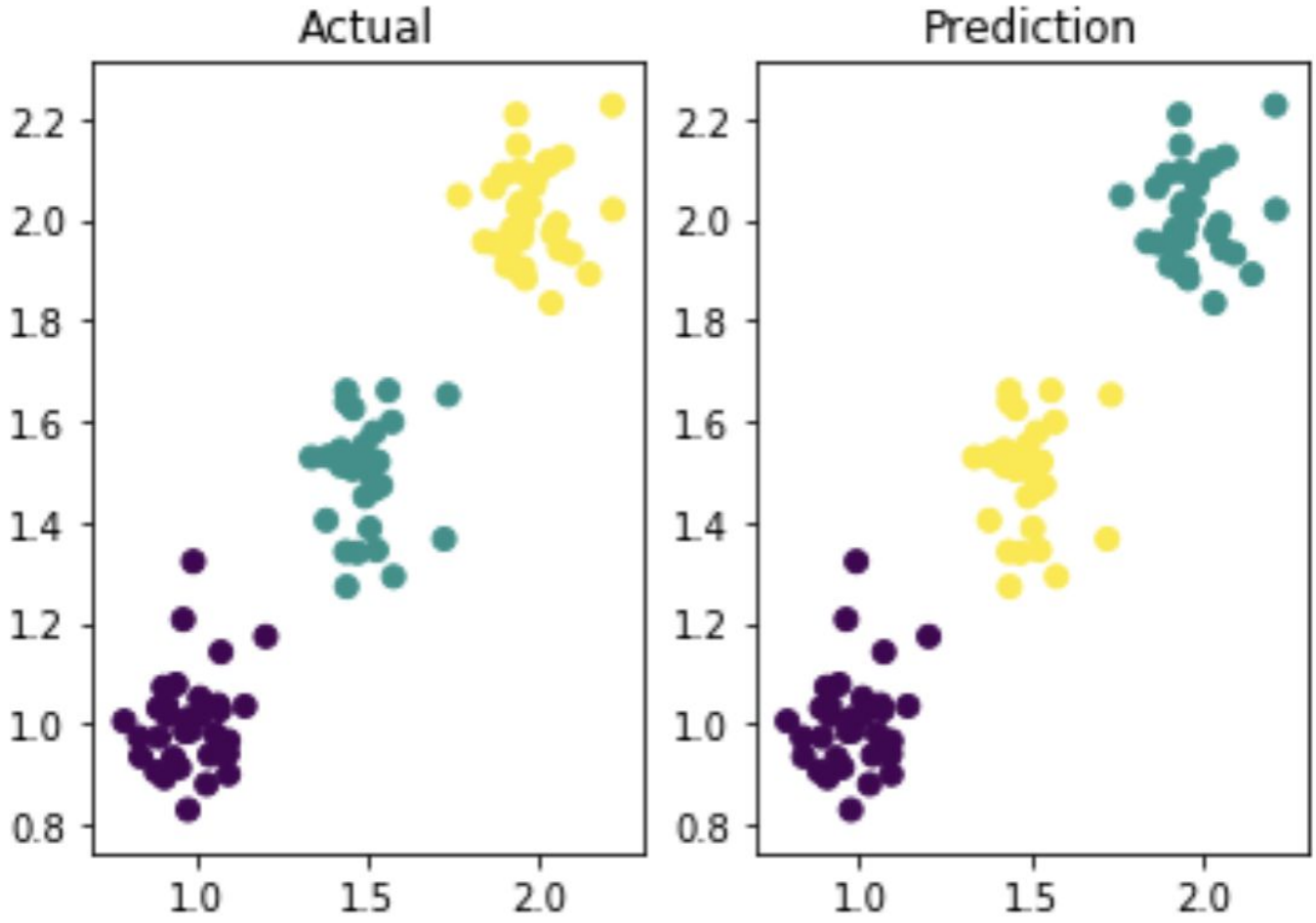
labels_ : Labels of each point

Clustering Steps

```
# Step 1 Model
from sklearn import cluster
cluster = cluster.KMeans(n_clusters=2)

# Step 2 Training
cluster .fit(X)

# Step 3 Evaluation
plt.scatter(X[:,0],X[:,1],c=cluster.labels_)
plt.show()
```

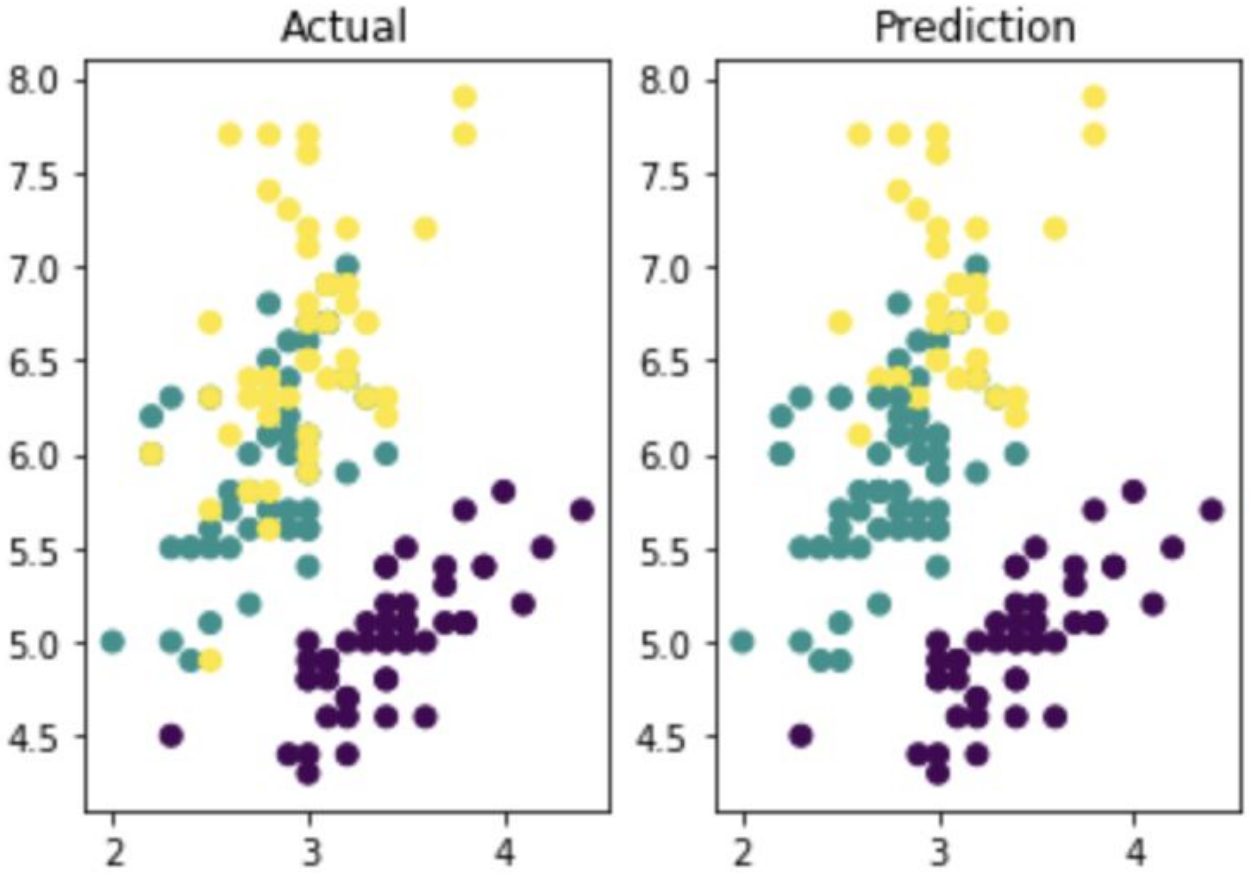
K- Means Clustering Demo

- Blob generator dataset



Ex: K-Means Clustering

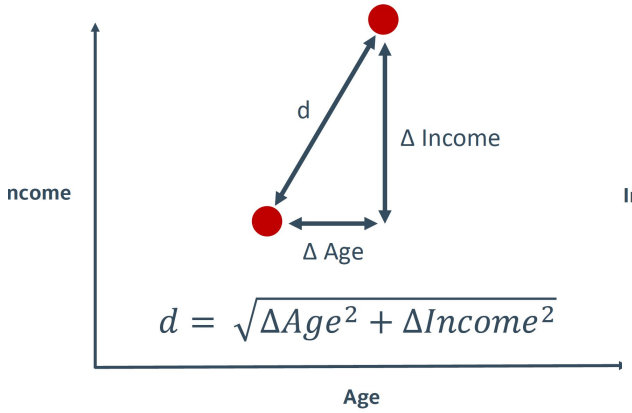
Apply K-Means Clustering to iris dataset



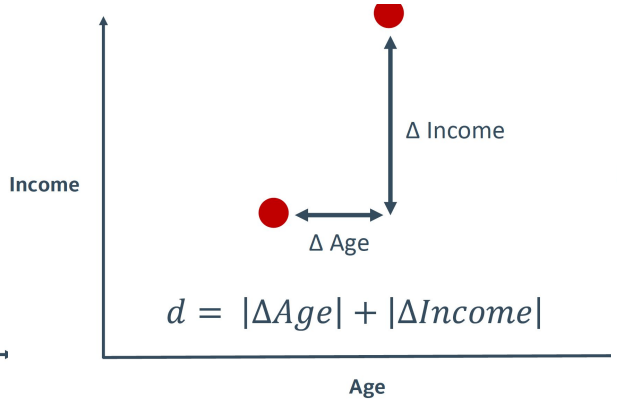
Distance Metric Choice

Choice of distance will significantly impact the clustering algorithms

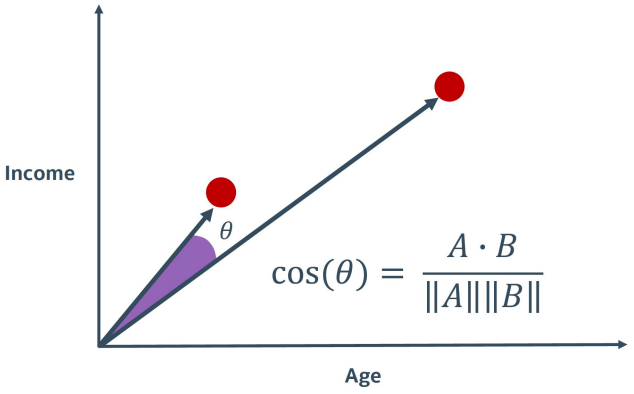
Euclidean (L2)



Manhattan (L1)

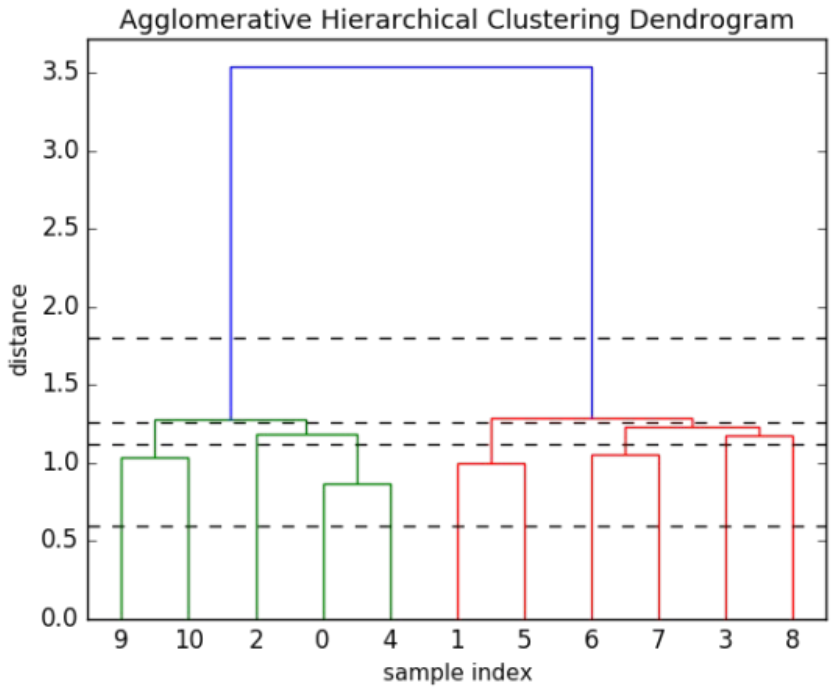


Cosine



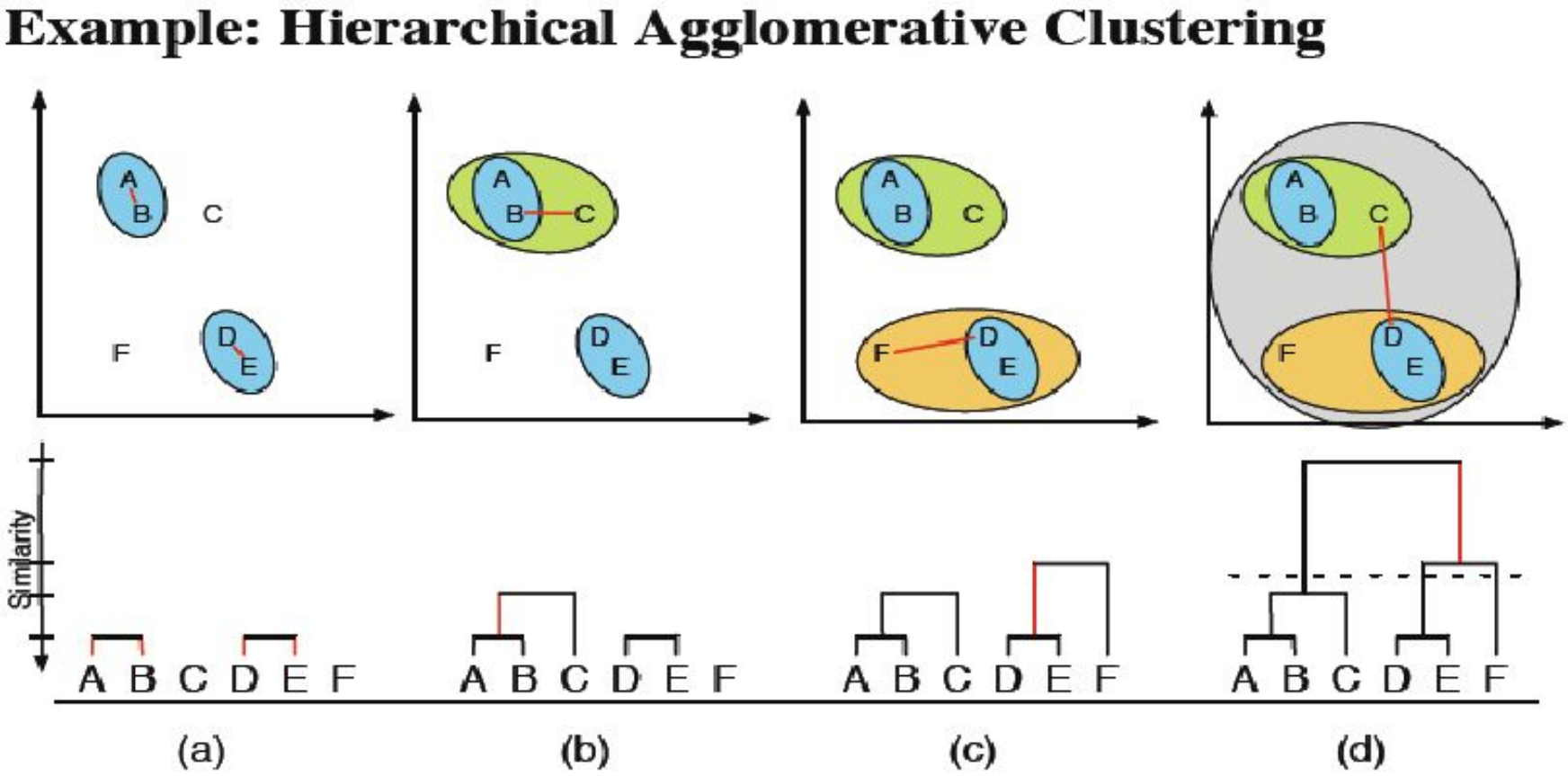
Hierarchical Clustering

- Hierarchical clustering is where you build a cluster tree (a dendrogram) to represent data, where each group (or “node”) links to two or more successor groups.
- The groups are nested and organized as a tree, which ideally ends up as a meaningful classification scheme.



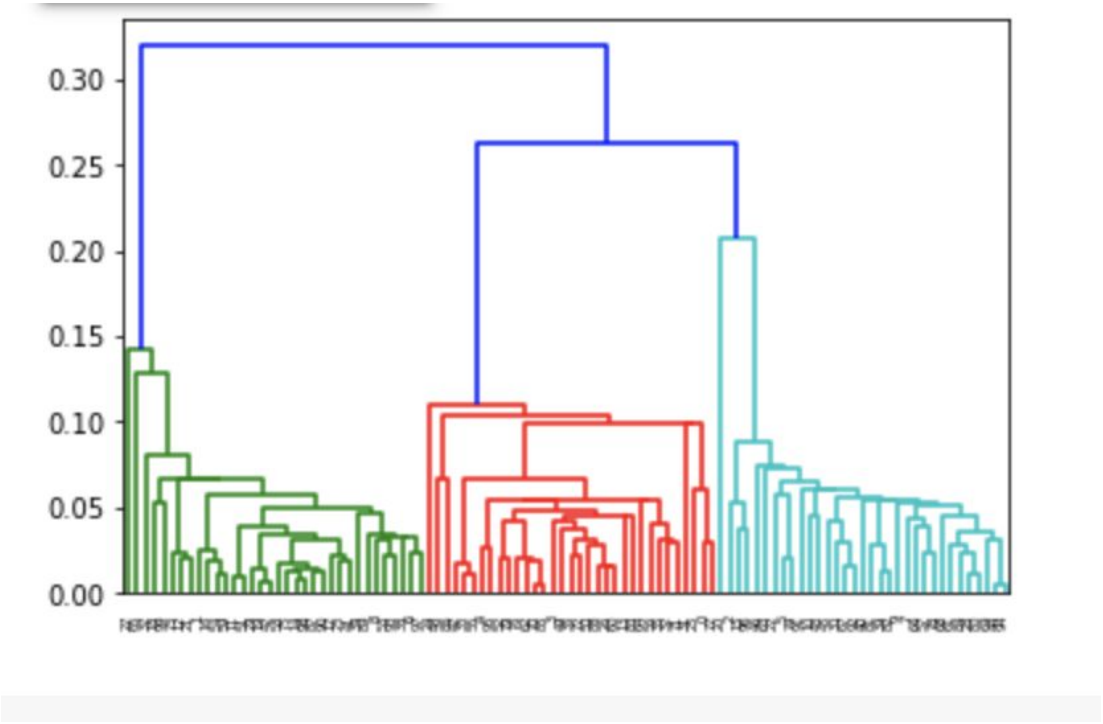
Hierarchical Clustering

Clusters are consecutively merged with the most nearby clusters. The length of the vertical dendogram lines (linkage) reflect the nearness



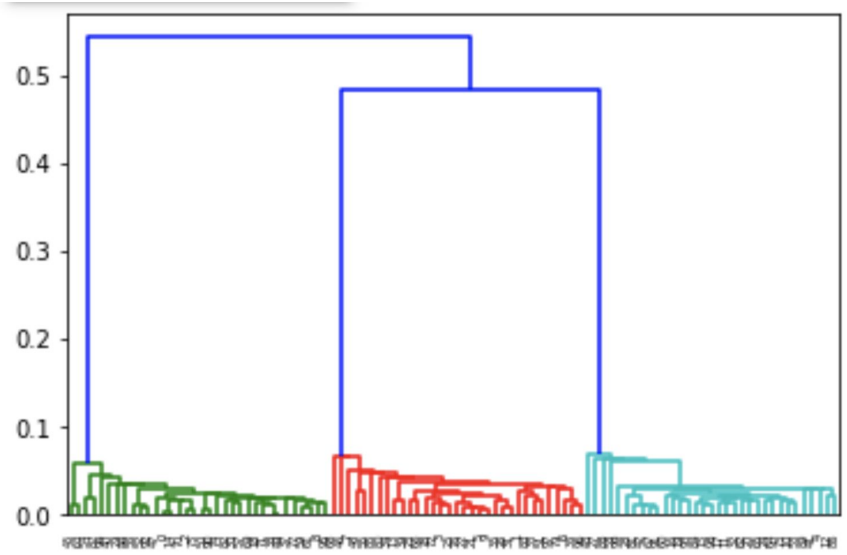
Dendrogram

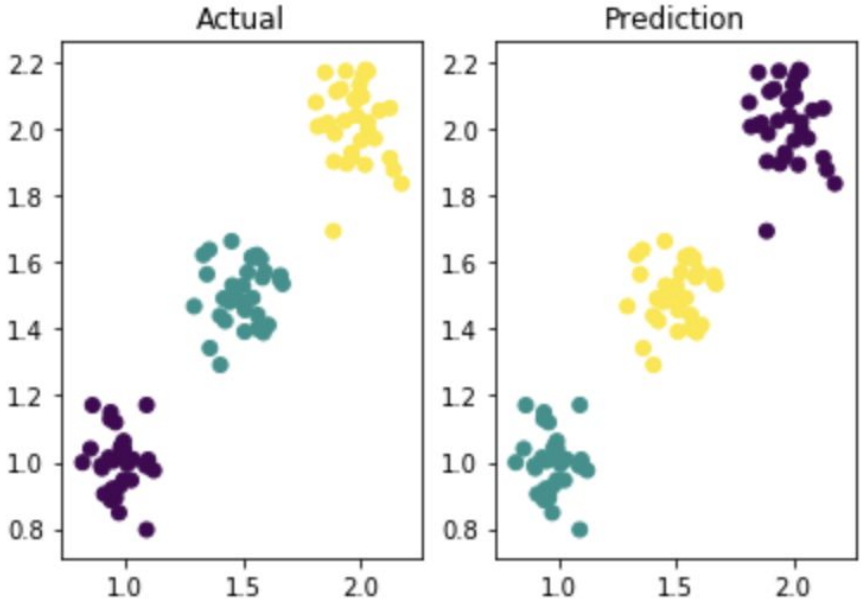
```
from scipy.cluster.hierarchy import dendrogram, linkage

Z = linkage(X)
d = dendrogram(Z)
plt.plot(d)
```



Hierarchical Clustering

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters = 3)
cluster.fit(X)
```

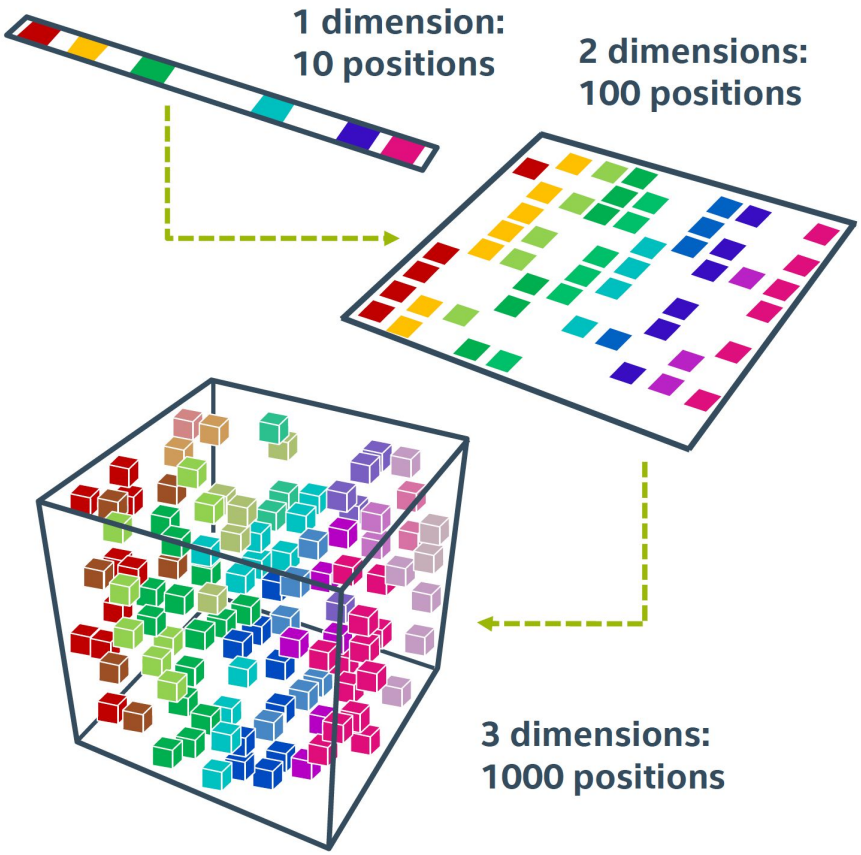




Topic 5
Dimension Reduction

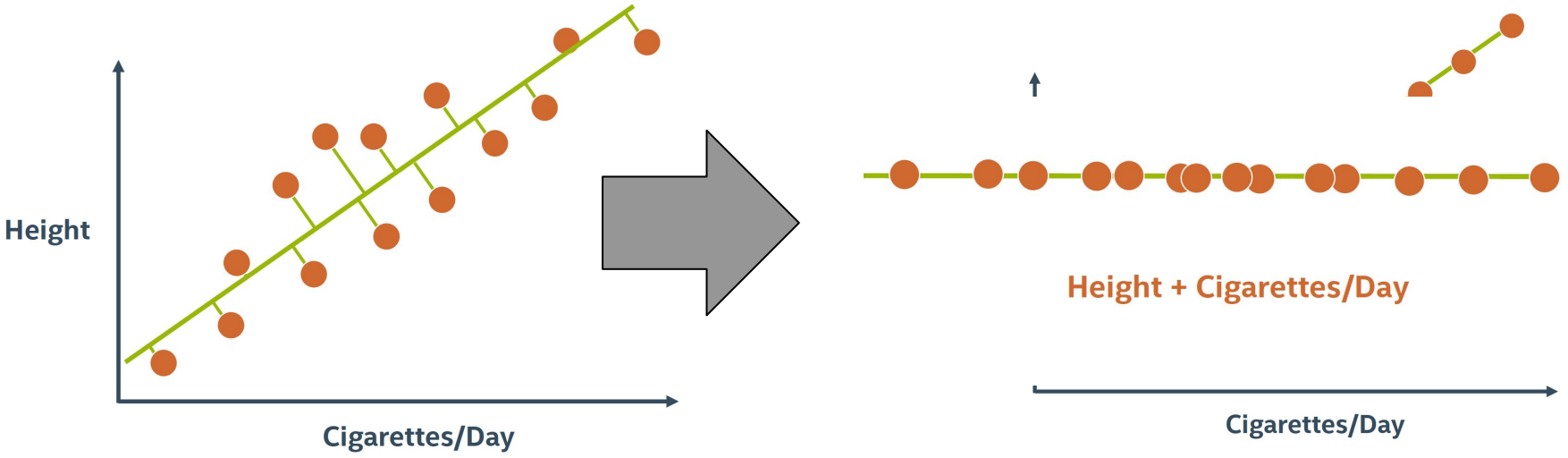
Curse of Dimensionality

- Theoretically, increasing features should improve performance
- In practice, more features leads to worse performance
- Number of training examples required increases exponentially with dimensionality



Dimension Reduction

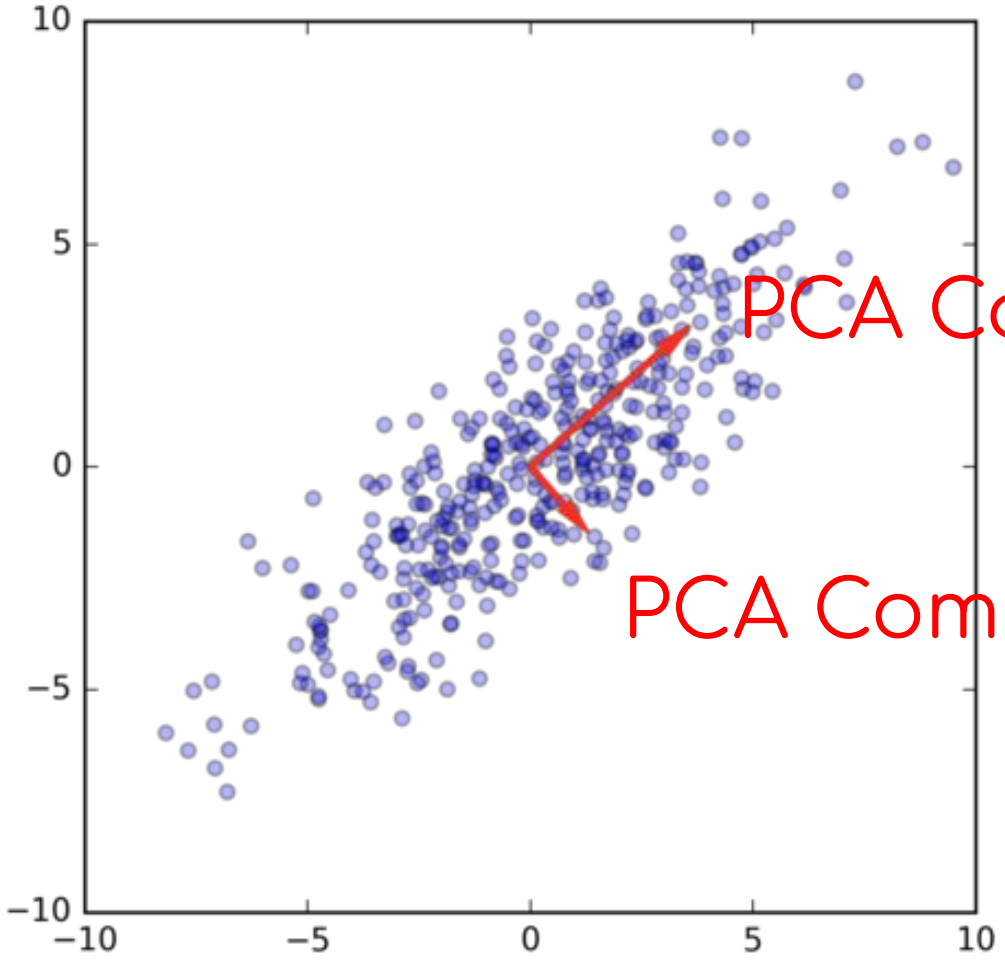
- Data can be represented by fewer dimensions (features)
- Reduce dimensionality by selecting subset (feature elimination)
- Combine with linear and non linear transformations
- Both features increase together. Can we reduce number of features to one?



Principal Component Analysis

- Create single feature that is combination of height and cigarettes. This is Principal Component Analysis (PCA)
- PCA is the most common dimensionality reduction technique.
- PCA was invented in 1901 by Karl Pearson

Principal Component Analysis



The principle components have the largest variations

PCA for Compression

PCA is used extensively for image compression and data compression.

144 Dimensions

4 Dimensions

PCA Math... (Optional)

Suppose we have $X_1, X_2, \ldots X_n$ features (dimensions), and we want to look at the correlation among the features, we can form a covariance matrix

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1 - \mu_1)(X_1 - \mu_1)] & \mathrm{E}[(X_1 - \mu_1)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_1 - \mu_1)(X_n - \mu_n)] \\ \mathrm{E}[(X_2 - \mu_2)(X_1 - \mu_1)] & \mathrm{E}[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_2 - \mu_2)(X_n - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n - \mu_n)(X_1 - \mu_1)] & \mathrm{E}[(X_n - \mu_n)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}.$$

The covariance matrix by definition is symmetric

Principal Component Analysis

PCA uses Spectral Decomposition Theorem that claim that any symmetric matrix can be decompose to a diagonal matrix

$$A = Q\Lambda Q^T$$

$$= \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_n v_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$= \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_n v_n v_n^T$$

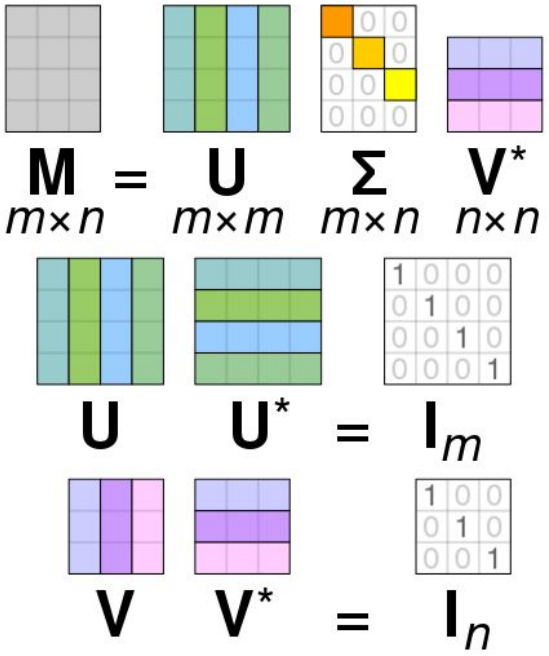
v_1, v_2, ..v_n are the eigenvectors (principal components) and λ_1,λ_2..are the eigenvalues (variations)

Singular Value Decomposition (SVD)

Scikit Learn use Singular Value Decomposition (SVD) to compute the eigenvalues or covariance of PVD.

sklearn.decomposition.PCA(n_components)

n_components : Number of components to keep.



PCA Attributes

- **explained_variance_:** The amount of variance explained by each of the selected components.
- **explained_variance_ratio_:** Percentage of variance explained by each of the selected components.
- **components_:** Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained_variance_.
- **n_components_:** The estimated number of components

PCA Simple Demo on Iris Dataset

X_t= pca.fit_transform(X)

pca.explained_variance_

pca.explained_variance_ratio_

Summary Q&A

Feedback

https://goo.gl/R2eumq

Thank You!

Tu Anqi
tuanqi96@gmail.com