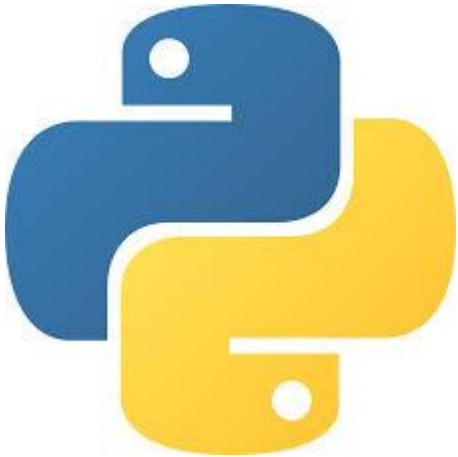
Machine Learning with Python

Cheat Sheet



For more information on the functions, see the documentation for:

Numpy (https://numpy.org/doc/2.3/user/absolute_beginners.html)

Pandas (https://pandas.pydata.org/docs/user_guide/index.html#user-guide, https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf)

Scikit-Learn (https://scikit-learn.org/0.21/user_guide.html)

Keras (https://keras.io/guides/sequential_model)

TPA
Sharing Experience

Descriptive Analysis of Numerical Variables

- **What it is:**
  A first step in data analysis to understand the distribution of variables (mean, min, max, variance, correlations).

- **When to use it:**
  Always at the beginning of a project — to detect anomalies, missing values, and the relationships between variables.

- **Example:** Examining exam scores to see the average, spread, and whether two subjects are correlated.

Descriptive Analysis of Numerical Variables – python code

```
import pandas as pd
df = pd.read_csv("data.csv")    # , index_col='id' , sep='; '
display(df.head())

# Basic statistics
display(df.describe())

# Correlation matrix
display(df.corr())

# Missing values
display(df.isnull().sum())
df = df.fillna(0)

# Separating the features from the value to be predicted (column 'target' for example)
X = df.drop(columns=['target'])
y = df['target']
```

Descriptive Analysis of Qualitative and Temporal Variables

- **What it is:**
  This step summarizes and explores the **categorical (qualitative)** and **temporal (date/time)** variables in a dataset to understand their structure, distribution, and relationships before modeling.
  It helps detect:
  - Data quality issues (missing values, inconsistent categories)
  - Seasonality or time trends
  - Potential predictors for classification or regression
- **When to use it:**

Situation	Why It Matters
Before training any ML model	Helps detect data quality issues
When features are categorical	Essential for proper encoding
When working with time series	Reveals seasonality, trends, and cycles

- **Example:**

Variable Type	Example	Typical Analyses	Typical Graph
Qualitative	Gender, Fuel_Type	Frequency, proportion, mode	Bar chart
Temporal	Date, Timestamp	Trends, seasonality, aggregation	Line chart

TPA
Sharing Experience

Analysis of Qualitative Variables

- **Definition:**
  Variables that represent **categories or labels**, not numerical values.
  Example: Gender, Fuel_Type, Country, Payment_Method.

- **What to look at:**

Objective	Description	Python Example
Unique categories	How many distinct values exist?	`df['Fuel_Type'].unique()`
Frequency distribution	How often each category appears	`df['Fuel_Type'].value_counts()`
Proportion (%)	Relative frequency per category	`df['Fuel_Type'].value_counts(normalize=True)`
Mode	The most common category	`df['Fuel_Type'].mode()[0]`
Missing values	How many NaN per category	`df['Fuel_Type'].isna().sum()`

Analysis of Temporal Variables

- **Definition:**
  Variables that represent **categories or labels**, not numerical values.
  Example: Gender, Fuel_Type, Country, Payment_Method.

- **What to look at:**

Objective	Description	Python Example
Convert to datetime	Ensure correct format	`df['Date'] = pd.to_datetime(df['Date'])`
Extract components	Year, month, day, weekday, hour	`df['Year'] = df['Date'].dt.year`
Find range	Min and max dates	`df['Date'].min(), df['Date'].max()`
Missing dates	Detect temporal gaps	`df['Date'].isna().sum()`
Trends	Mean or count over time	`df.groupby('Year')['Sales'].mean()`
Combine time & category	Analyze category evolution over time	`grouper = pd.Grouper(key='Date', freq='M')` `df.groupby([grouper, df['Item']]).agg({'Sales':'mean'}).unstack()`

Descriptive Analysis of Qualitative and Temporal Variables – python code

```
# Qualitative
df['Category'].value_counts()
df['Category'].value_counts(normalize=True)
df['Category'].mode()[0]
df['Category'] = df['Category'].str.lower() # .title(), .contains(), .replace('old', 'new')
# Temporal
df['Date'] = pd.to_datetime(df['Date']) # to convert a string containing a date to a date
df['Year'] = df['Date'].dt.year

# Visualization
import matplotlib.pyplot as plt
df.groupby('Year')['Sales'].sum().plot(kind='line') # or .mean()
plt.title('Average Sales per Year')
plt.show()
```

Cleaning Variables

- **What it is:**
  Preparing the dataset by fixing or removing incorrect/missing/inconsistent data.

- **When to use it:**
  Always before training a model. A dirty dataset leads to biased or unusable models.

- **Example:** Replacing missing ages with the average age; converting "Male/Female" into numbers.

Cleaning Variables – python code

- **Missing values:** drop or impute

```
df = df.dropna()
df["col"].fillna(df["col"].mean(), inplace=True)
```

- **Categorical encoding**

```
df = pd.get_dummies(df, columns=["category"])
```

- **Outliers** (optional)
  - Interquartile Range = Q3 – Q1 → the width of the box in a boxplot.

```
q1, q3 = df["col"].quantile([0.25, 0.75])
iqr = q3 - q1
df = df[(df["col"] >= q1-1.5*iqr) & (df["col"] <= q3+1.5*iqr)]
```

Standardization

- **What it is:**
  Transforming variables so they all have the same scale (mean=0, std=1).

- **When to use it:**
  Needed when models are sensitive to variable scales (SVM, Logistic Regression, kNN, Neural Networks).
  Not needed for tree-based models (Decision Trees, Random Forest).

- **Example:** If "Income" goes from 0–100,000 and "Age" from 0–100, models might give too much importance to income.

Standardization – python code

```
from sklearn.preprocessing import Standard
Scalerscaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Train-Test Split

- **What it is:**
  Separating data into training (to build the model) and testing (to evaluate performance).

- **When to use it:**
  Always — to avoid overfitting and check generalization.

- **Example:** 80% of patients' data to train a diagnostic model, 20% to test predictions.

Train-Test Split – python code

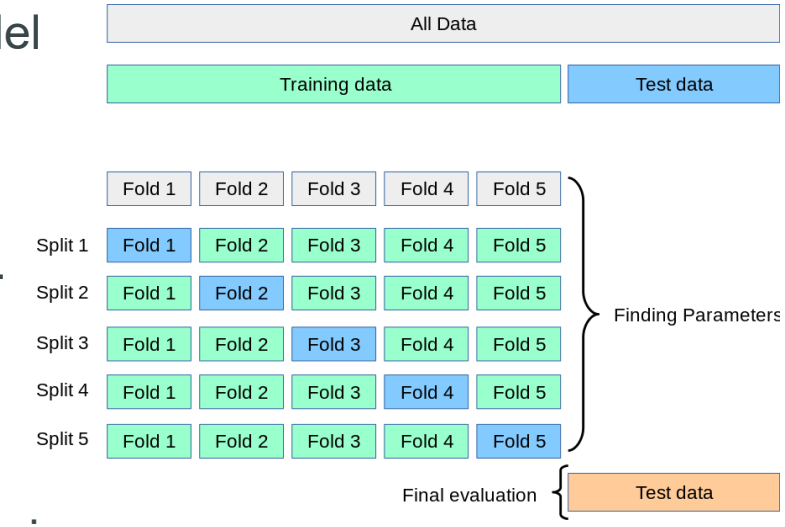
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split( X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Cross-Validation

- **What it is:**
  Cross-Validation (CV) is a technique to **evaluate the performance of a model** by splitting the dataset into several parts (folds) and testing the model on unseen data multiple times.
  Instead of a single train/test split, CV ensures more **reliable and stable evaluation**.
  Helps detect **overfitting** and gives a better estimate of generalization error.



- **Why use it**
  When dataset is **small**, and a single train/test split may be misleading.
  To **compare models** or hyperparameters more fairly.
  To get an estimate of performance that is **less dependent on randomness** in the split.

- **Main Types of Cross-Validation :**

Type	Description	When to Use
k-Fold CV	Split data into k folds. Train on k-1 folds, test on the remaining fold. Repeat k times.	General purpose, balanced approach.
Stratified k-Fold CV	Like k-Fold, but preserves the class distribution in each fold.	Classification problems with imbalanced classes.
Leave-One-Out (LOO)	Each sample is used once as test, others as train.	Very small datasets (but computationally expensive).

TPA
Sharing Experience

Cross-Validation – python code

```
from sklearn.model_selection import cross_val_score, Kfold
from sklearn.linear_model import LinearRegression
import numpy as np
# Define k-Fold (5 folds)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LinearRegression()
# Perform cross-validation
scores = cross_val_score(model, X_train, y_train, cv=kf, scoring="r2")
                                                    # scoring = « accuracy » or « f1 » for classification
print("Scores:", scores)
print("Mean R²:", np.mean(scores))
# Prédictions via cross-validation
y_pred = cross_val_predict(model, X_test, y_test, cv=cv)
```

Documentation:
https://scikit-learn.org/stable/modules/cross_validation.html

TPA
Sharing Experience

Linear Regression

- **What it is:**
  A method to predict a continuous variable (real number) as a straight line relationship between inputs and output.
  - **Polynomial features**
    You can create new features like x^2, x^3, x_1x_2 ,etc. Then the model becomes:
    $$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$
    Still a linear regression because it is linear in the βcoefficients.
    This is called **polynomial regression**, which is just linear regression on **transformed features**.

- **When to use it:**
  When you want to estimate a numerical value, like house price or salary.

- **Example:** Predicting house price from size and number of rooms.

Linear Regression – Explanation (1)

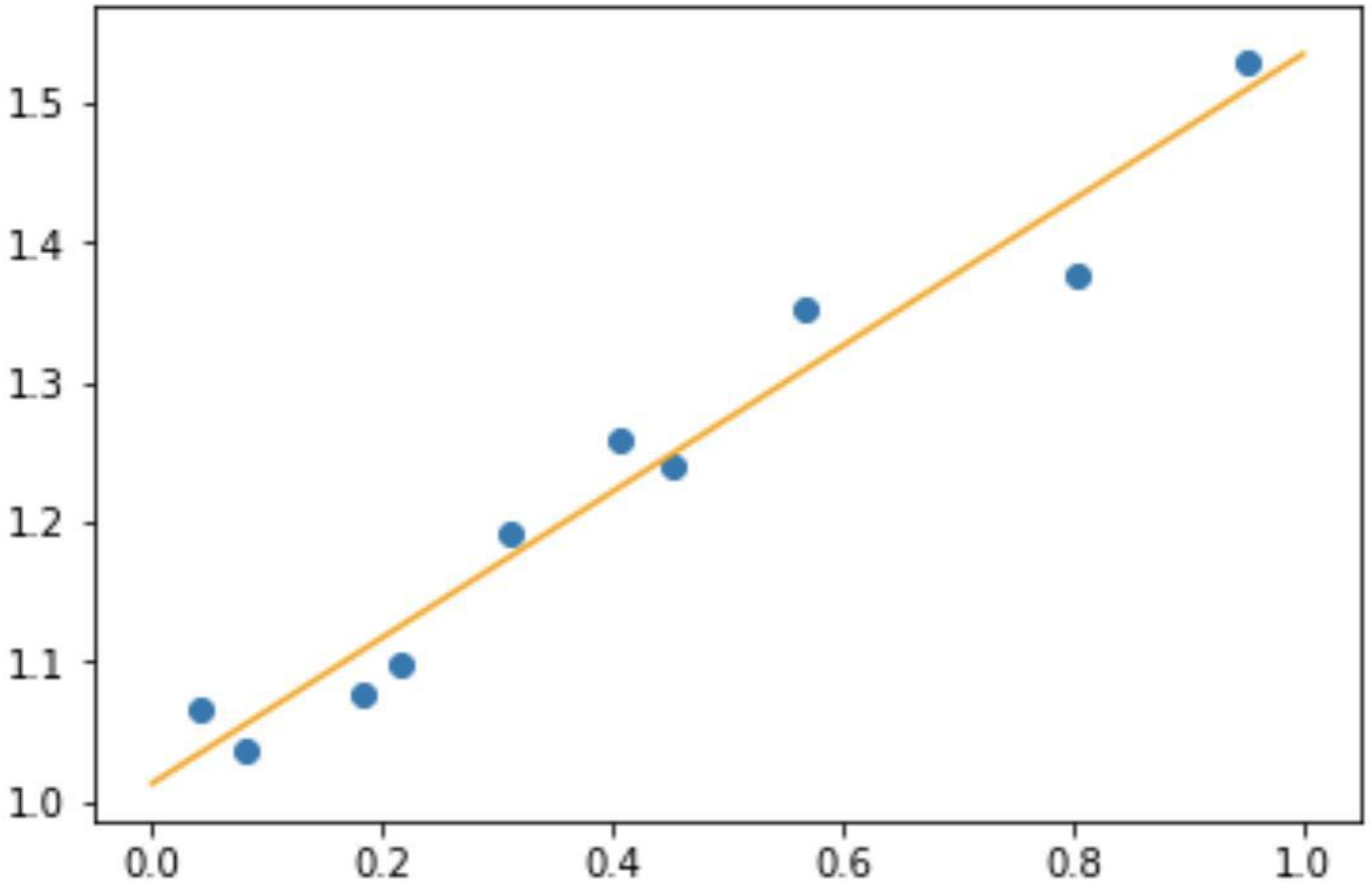
- Regression models seek to find a mathematical model, a function that behaves roughly like data.

- For the model to be found, it is necessary to know a sample of the data that the model must approximate. This fact makes regression models supervised.

- Once the model is defined, the value of the function at any point can be calculated by using new values/features in the model.

Linear Regression – Explanation (2)

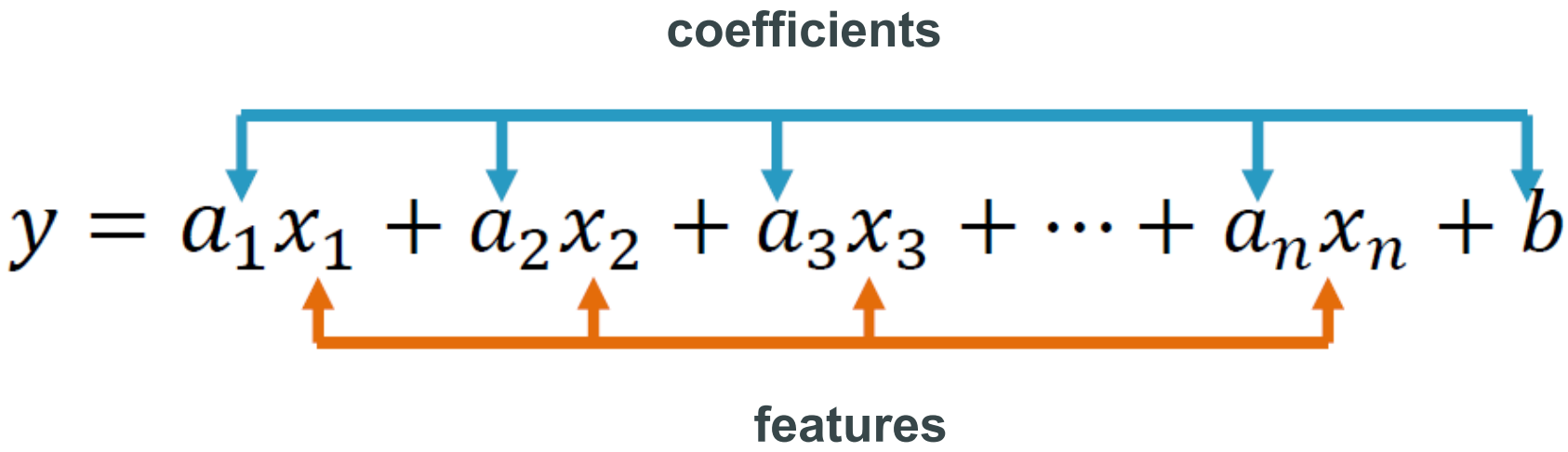
x	y
0.57	1.35
0.80	1.38
0.21	1.10
0.45	1.24
0.95	1.53
0.41	1.26
0.18	1.08
0.08	1.04
0.04	1.07



Regression methods make a hypothesis about the type of mathematical model.
For example, the linear hypothesis:
y = a * x + b

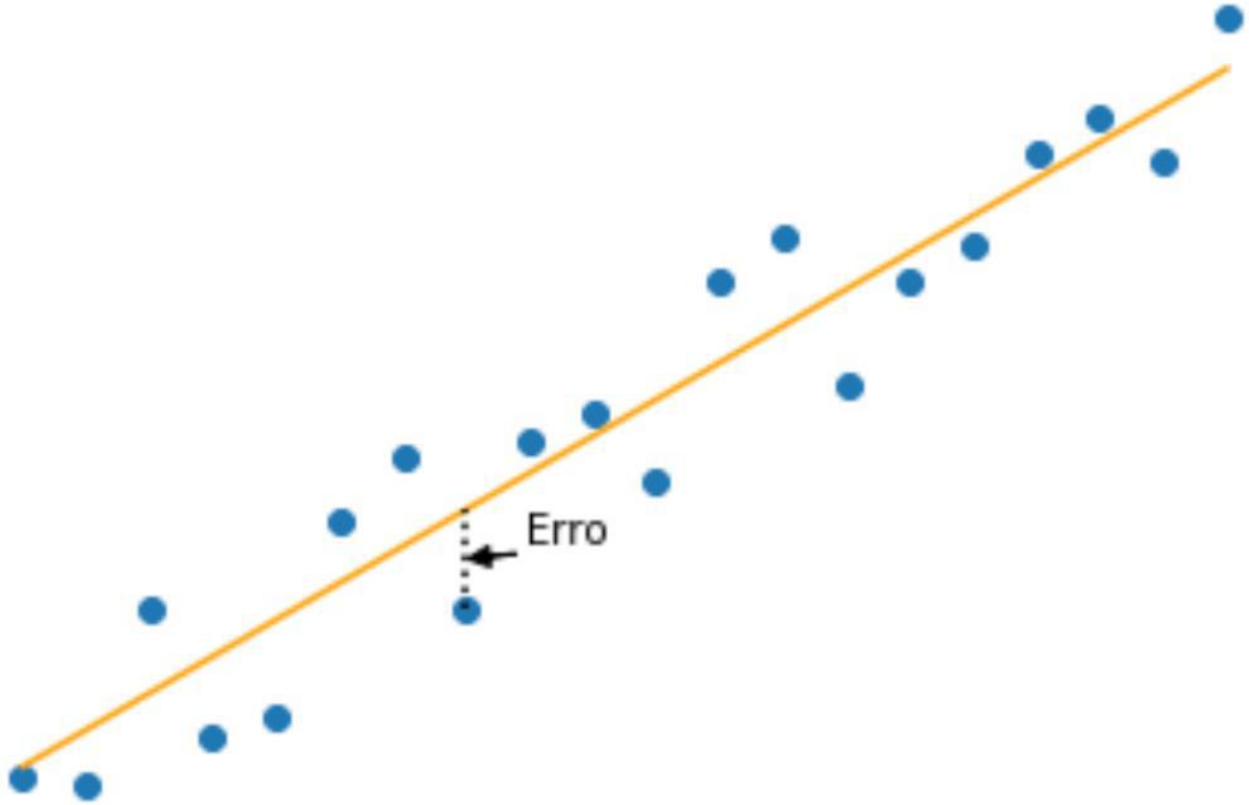
Linear Regression – Explanation (3)

x1	x2	x3	y
0.31	0.86	0.05	3.57
0.05	0.72	0.90	5.96
0.33	0.19	0.72	5.37
0.05	0.88	0.40	4.49
0.81	0.48	0.49	5.03
0.80	0.14	0.74	5.65
0.28	0.57	0.55	4.95
0.55	0.42	0.32	4.37
0.80	0.52	0.12	3.92
0.29	0.55	0.05	3.47



Linear Regression – Explanation (4)

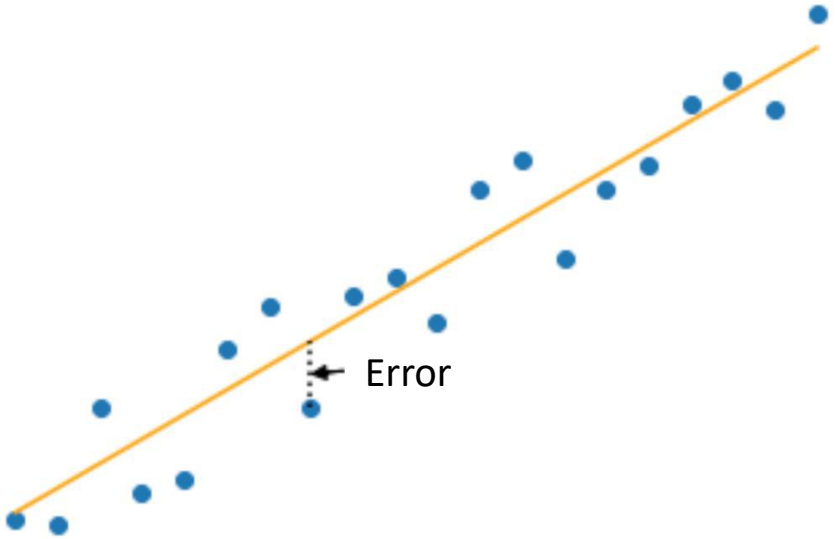
- Once the model type and training data are defined, the coefficients are calculated to **minimize** the error between the model and the data.



Linear Regression – Explanation (5)

Number of samples in the training dataset

$$\text{Total Error} = \sum^{k} \underbrace{(y - a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n + b)^2}_{\text{Error}}$$



- There are several mathematical alternatives for calculating coefficients:
- Optimization methods and
- Linear Algebra: Least Squares

Linear Regression – Explanation (6)

Note: While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as **polynomial or curvilinear regression**.

Precautions that should be taken when calculating regression:

- attributes must be **normalized** to prevent an attribute from being privileged because of its scale;
- **Outliers** significantly affect the calculation of coefficients and should be avoided in training data;
- If the difference between the smallest and highest values of an attribute (or of the prediction variable) is very large, you must apply a **transformation** to the values before using regression and..
- **Highly correlated attributes should not be used simultaneously** in the regression model, as they hinder the analysis of the model and the calculation of the coefficients.

Linear Regression – Explanation (7)

One of the advantages of linear regression is the interpretability of the generated model.

If all attributes are equally important for predicting the y-variable, then the coefficients associated with all attributes must have similar magnitudes.

Coefficients of higher magnitude (farther from zero) indicate a greater importance of the attribute in the prediction of the variable y.

$$y = a1x1 + a2x2 + a3x3 + b$$

$$a1 = 5.04$$
$$a2 = 3.03$$
$$a3 = 0.02$$

Linear Regression – Explanation (8)
Effectiveness of the prediction method

The validation of the prediction model is done by dividing the training data into two parts: **training and testing**.

Regression is done (calculation of the coefficients) using the training data, and the generated model is evaluated in the test data.

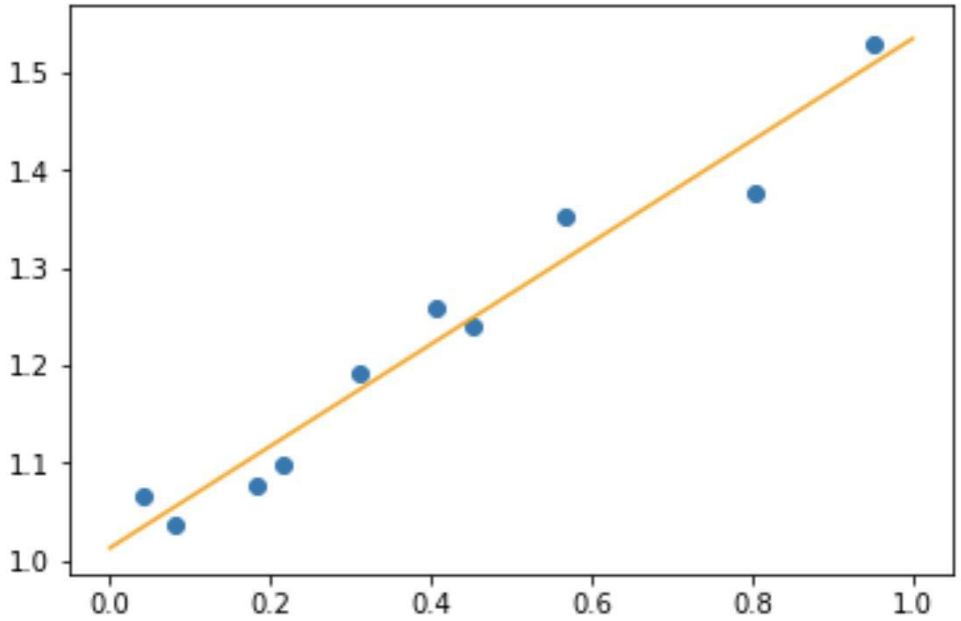
Since the model changes depending on the training data, what is done is to use a mechanism called cross-validation.

Cross-validation: Multiple training and testing samples are randomly generated from the known data. Efficacy and prediction are obtained by calculating the mean error and the mean predicted value from all models produced.

Linear Regression – Explanation (9)

- Regression techniques generate models that are easy to interpret. They should, however, be used with caution, normalizing and transforming attributes when necessary.

- Cross-validation is an important and widely used strategy in the context of regression models.



Linear Regression – python code

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)
model.score(X_train, y_train)

#Equation coefficient and Intercept
print('Coefficient: \n', model.coef_)
print('Intercept: \n', model.intercept_)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Linear Regression with Cross Validation – python code

```
from sklearn.model_selection import cross_val_score, Kfold
from sklearn.linear_model import LinearRegression
import numpy as np
# Define k-Fold (5 folds)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Define model
model = LinearRegression()
# Perform cross-validation
scores = cross_val_score(model, X_train, y_train, cv=kf, scoring="r2")
print("Scores:", scores)
print("Mean R²:", np.mean(scores))
# Prédictions via cross-validation
y_pred = cross_val_predict(model, X_test, y_test, cv=cv)
```

Documentation:
https://scikit-learn.org/stable/modules/cross_validation.html

TPA
Sharing Experience

Classification model

Logistic Regression

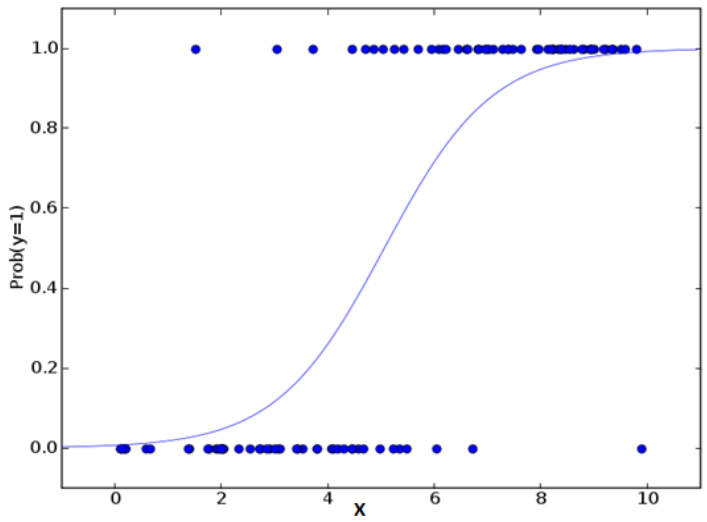
- **What it is:**
  Despite the name, it's for **classification**, not regression. It estimates the probability of belonging to a class.

- **When to use it:**
  When the target is binary (yes/no, 0/1).
  Multinomial logistic regression: generalizes to more than 2 categories:
  - It uses one of two strategies:
  - **One-vs-Rest (OvR)** *(default in scikit-learn)*
    - Train 1 model per class vs. all other classes.
    - Choose the class with the highest probability.
  - **Multinomial (softmax regression)**
    - A single model that directly computes probabilities for all classes at once.
    - Often better when classes are not linearly separable.

- **Example:** Predicting if an email is spam (1) or not spam (0).

Logistic Regression - Explanation

$$y = \frac{1}{1 + e^{-x}}$$



- Logistic regression is a binary classifier that indicates the probability that a given object belongs to class 0 or class 1.The idea is to replace the variable x of the logistic function with a linear equation of the form:

$$y = \frac{1}{1 + e^{-(a_1x_1 + a_2x_2 + \cdots + a_nx_n + b)}}$$

- The linear equation defines a plane (2D line) that divides the space of objects into two parts. Points above the plane are given a positive value by the linear equation. Points on the plane, values equal to zero, and points below the plane receive negative values.

Logistic Regression – python code

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
# If more than 2 categories
# model = LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=200)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Support Vector Machine (SVM)

- **What it is:**
  A classification method that finds the “best boundary” (hyperplane) between classes.
  You can use different types of boundaries

Kernel	Shape of Boundary	When to Use
Linear	Straight line / hyperplane	Data is linearly separable
Polynomial	Curved, polynomial-shaped	Data has feature interactions
RBF (Gaussian)	Flexible, curved	Default choice for most problems
Sigmoid	Neural network-like	Rare, experimental use

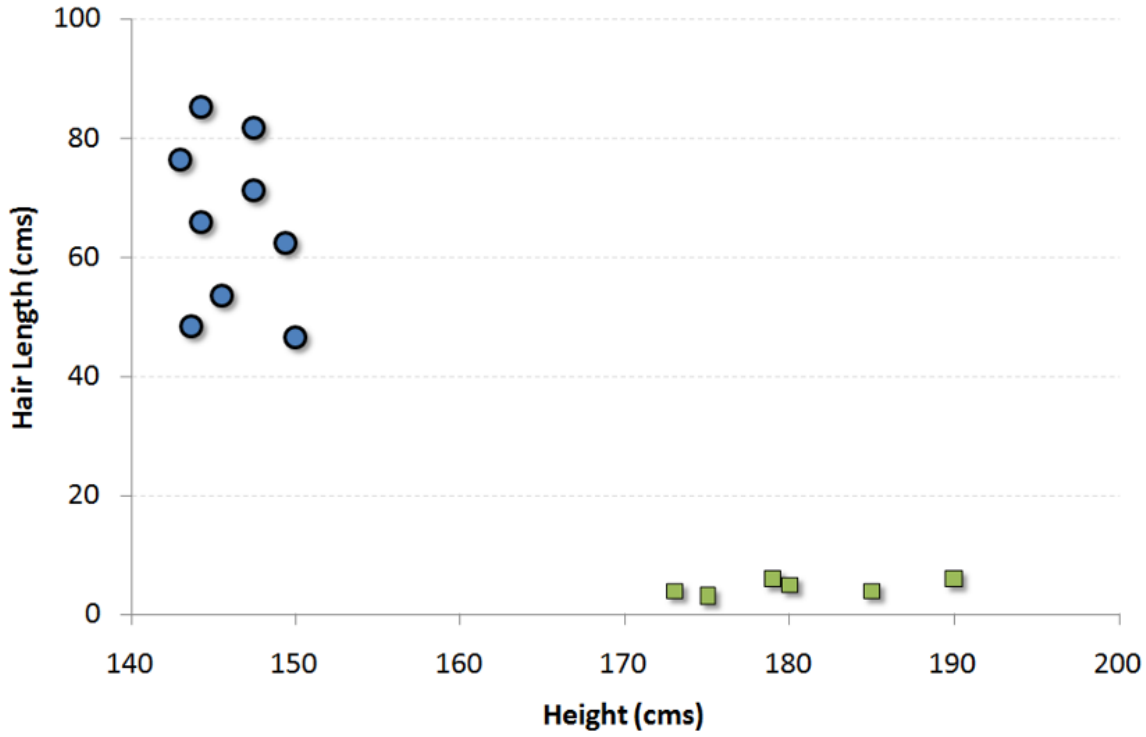
- **When to use it:**
  Effective in small/medium datasets with clear separation.

- **Example:** Classifying images of cats vs. dogs.

SVM (SUPPORT VECTOR MACHINE)

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate.

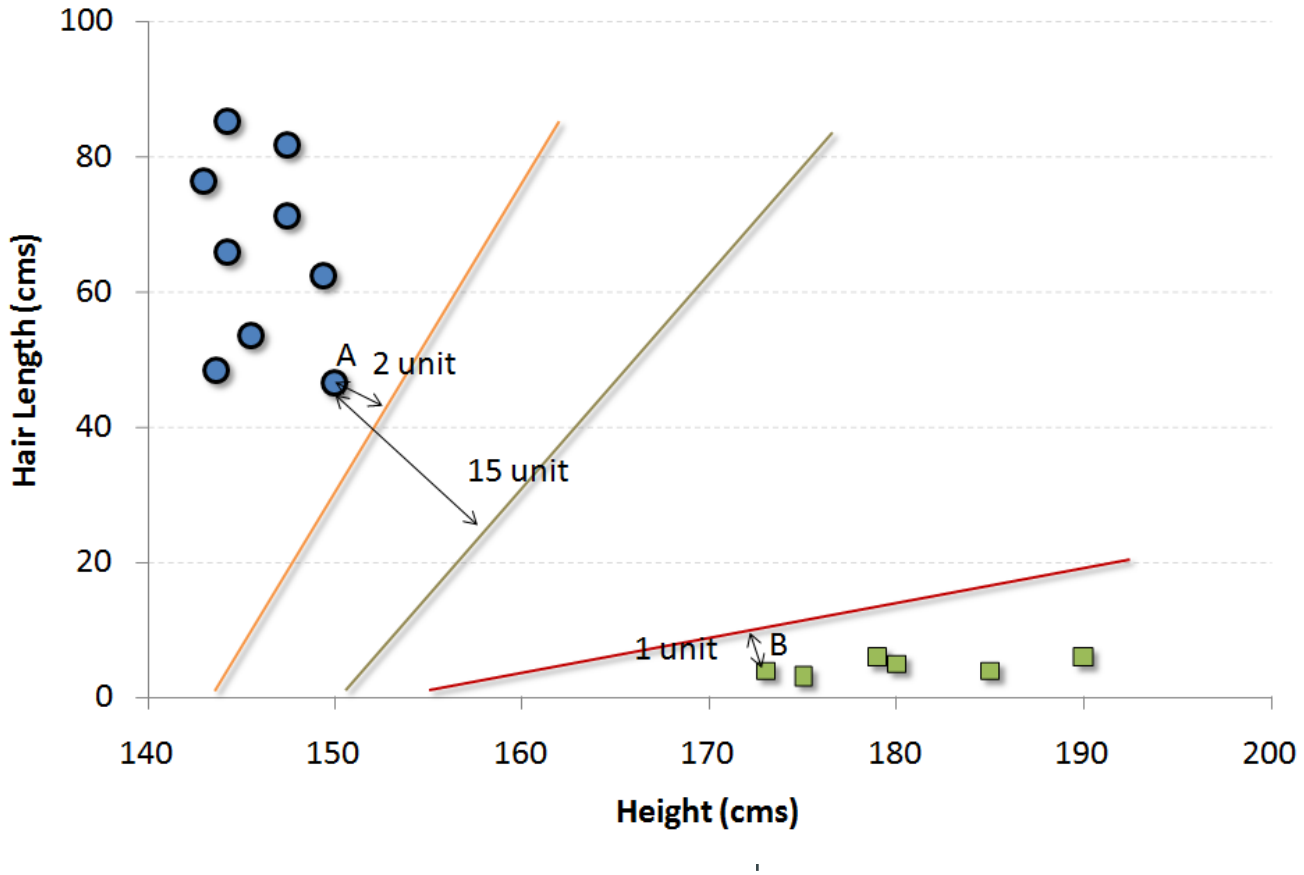
For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)



TPA
Sharing Experience

SVM (SUPPORT VECTOR MACHINE)

Now, we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.



In the example shown above, the line which splits the data into two differently classified groups is the *black* line, since the two closest points are the farthest apart from the line.

This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

TPA
Sharing Experience

SVM (SUPPORT VECTOR MACHINE)

Think of this algorithm as playing JezzBall in n-dimensional space. The tweaks in the game are:

- You can draw lines / planes at any angles (rather than just horizontal or vertical as in classic game)
- The objective of the game is to segregate balls of different colors in different rooms.
- And the balls are not moving.

SVM (Support Vector Machine)

- Python code

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = svm.svc()
# there is various option associated with it, this is simple for classification.
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Support Vector Machine – python code

```
from sklearn.svm import SVC

model = SVC(kernel="rbf")
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

k-Nearest Neighbors (kNN)

- **What it is:**
  Classifies a point by looking at the labels of its nearest neighbors in the dataset.
  You can use different metric:

Metric	Formula	When to Use	Typical Use Cases
Euclidean	$\sqrt{\sum(x_i - y_i)^2}$	Default for continuous numerical data	Image recognition, clustering spatial data
Manhattan	$\sum \lvert x_i - y_i \rvert$	High-dimensional data, grid-like data, robust to outliers	Customer segmentation, pathfinding in grids
Minkowski	$\left(\sum \lvert x_i - y_i \rvert^p \right)^{1/p}$	Generalization (p=1 → Manhattan, p=2 → Euclidean)	Flexible, problem-specific
Cosine	$1 - \frac{x \cdot y}{\lVert x \rVert \lVert y \rVert}$	Focus on direction/shape, ignores magnitude	Text mining (TF-IDF), recommender systems
Hamming	Count of differing positions	Binary or categorical variables	DNA sequence comparison, categorical feature matching

- **When to use it:**
  Good for simple classification problems with small datasets.

- **Example:** Predicting if a fruit is apple or orange based on size and color.

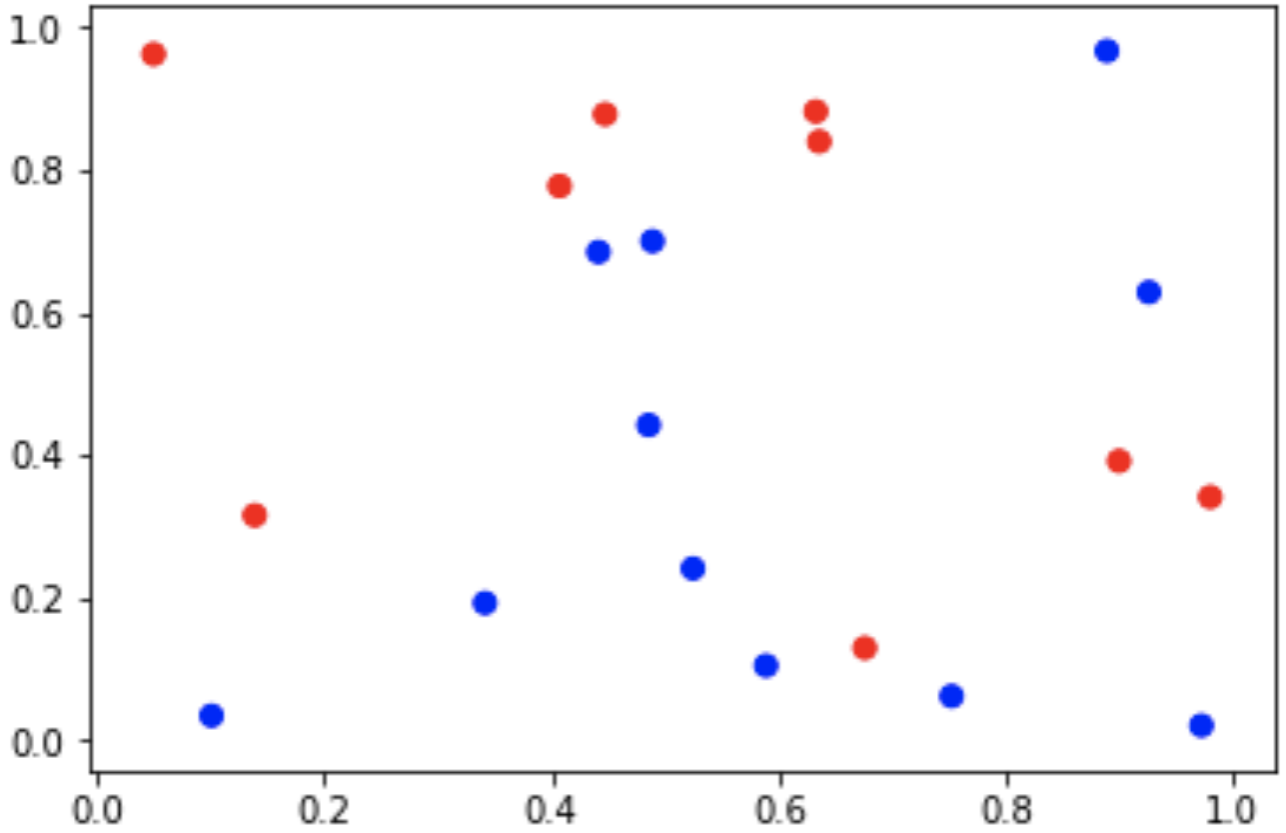
k-Nearest Neighbors – Explanation (1)

- We know that each object in a dataset can be represented as a point in a Cartesian space.
- Object 2 can be represented as a point in a 5-dimensional space.Set of all objects gives rise to a set of points in a space of dimension 5.

	a1	a2	a3	a4	a5
obj1	0.65	0.97	0.44	0.93	0.09
obj2	0.49	0.45	0.25	0.42	0.15
obj3	0.56	0.36	0.26	0.91	0.71
obj4	0.27	0.44	0.26	0.02	0.59
obj5	0.62	0.35	0.93	0.69	0.07

k-Nearest Neighbors – Explanation (2)

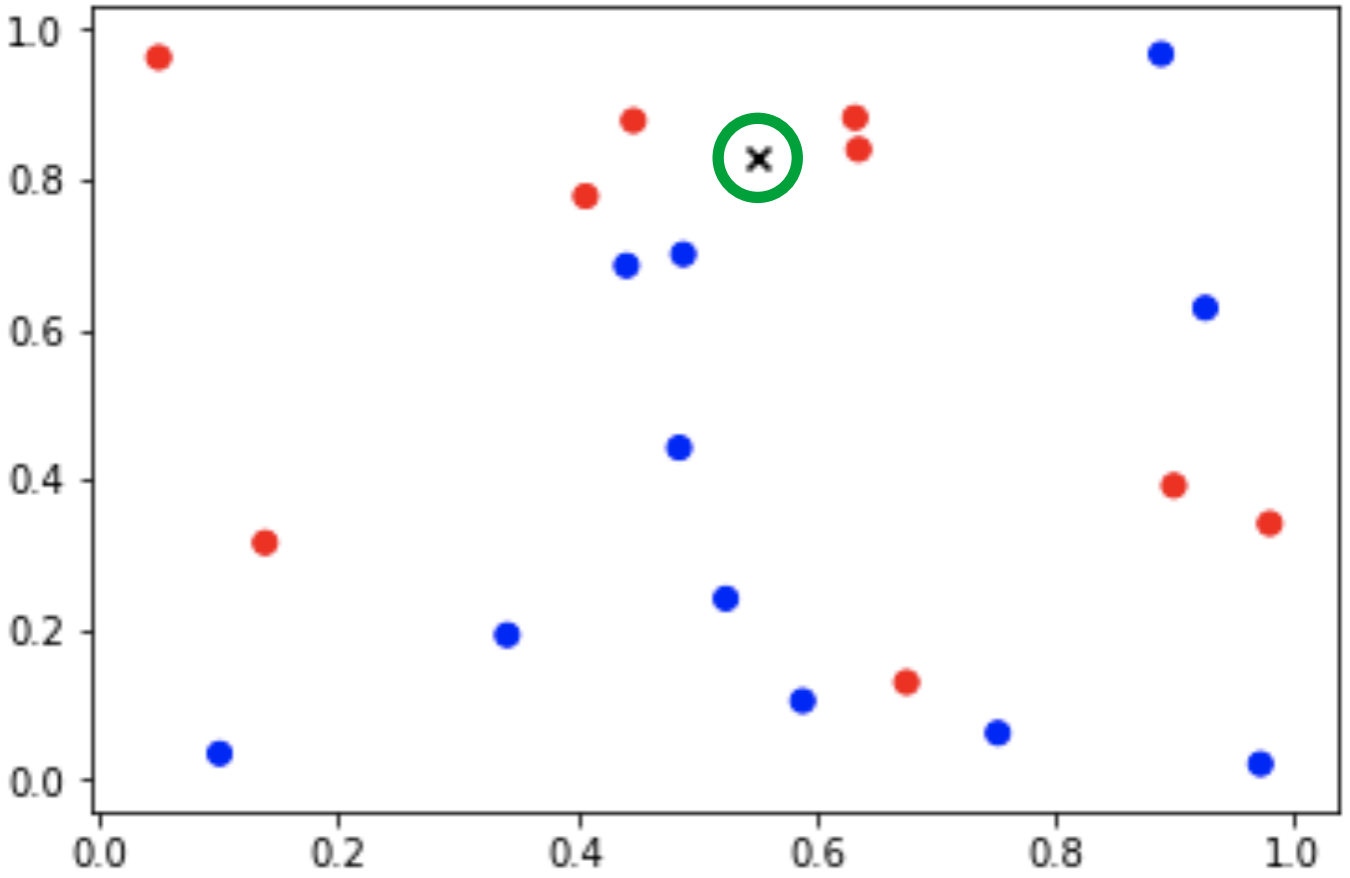
- Suppose, for example, objects with two attributes (2D points) distributed into two classes:



k-Nearest Neighbors – Explanation (3)

- Given a new point whose class is unknown.

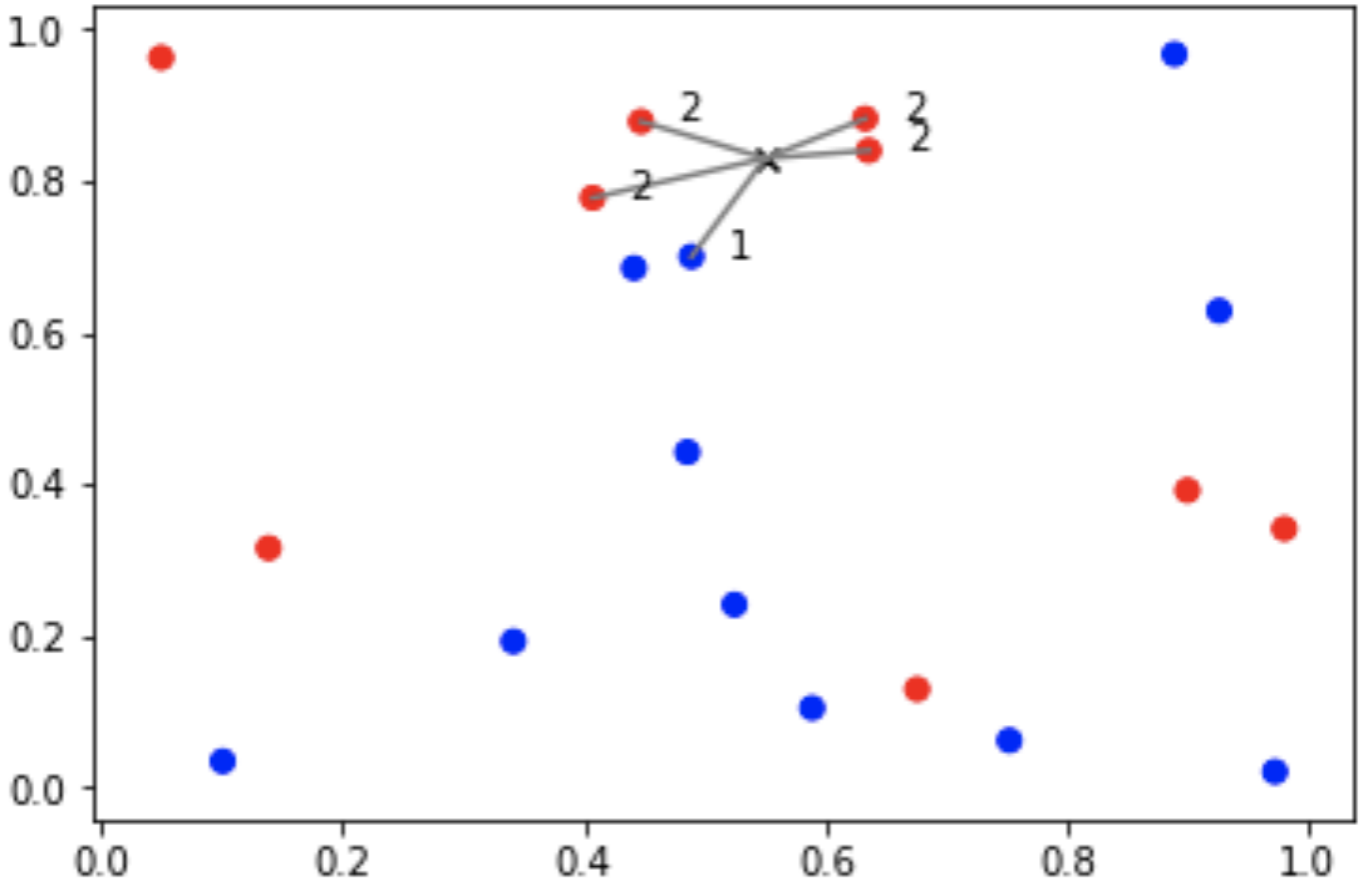
a1	a2	Classe
0.55	0.83	?



k-Nearest Neighbors – Explanation (4)

- One can infer the class of the new point as the same as that of most of its nearest neighbors.

a1	a2	Classe
0.55	0.83	?



k-Nearest Neighbors – Explanation (5)

Advantages:

- fairly simple methodology;
- easy interpretation and
- nonlinearity (the method is more flexible than a linear classifier).

Disadvantages:

- Finding an appropriate value for the number of neighbors to consider may not be a simple task;
- Using outliers as training data tends to get in the way of ranking, and
- Finding the nearest neighbors requires a reasonable amount of computational effort.

k-Nearest Neighbors – python code

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Decision Tree

- **What it is:**
  A model that splits data by asking questions (like a flowchart).

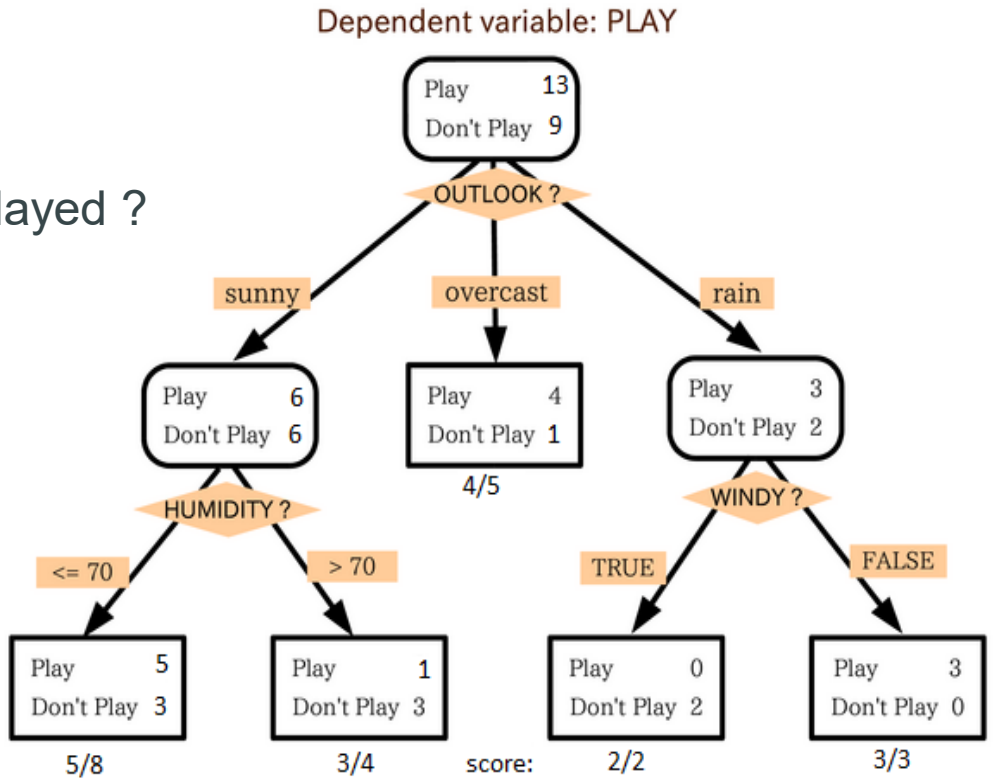
- **When to use it:**
  When interpretability is important.

- **Example:** Deciding if someone should get a loan based on income and credit history.

Decision Tree - Explanation

- It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible

Will the tennis game be played ?



TPA
Sharing Experience

Decision Tree – python code

```
from sklearn.tree import DecisionTree

Classifiermodel = DecisionTreeClassifier(max_depth=5, random_state=42, criterion='gini') )
# here you can change the algorithm as gini or entropy (information gain) by default it is gini

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

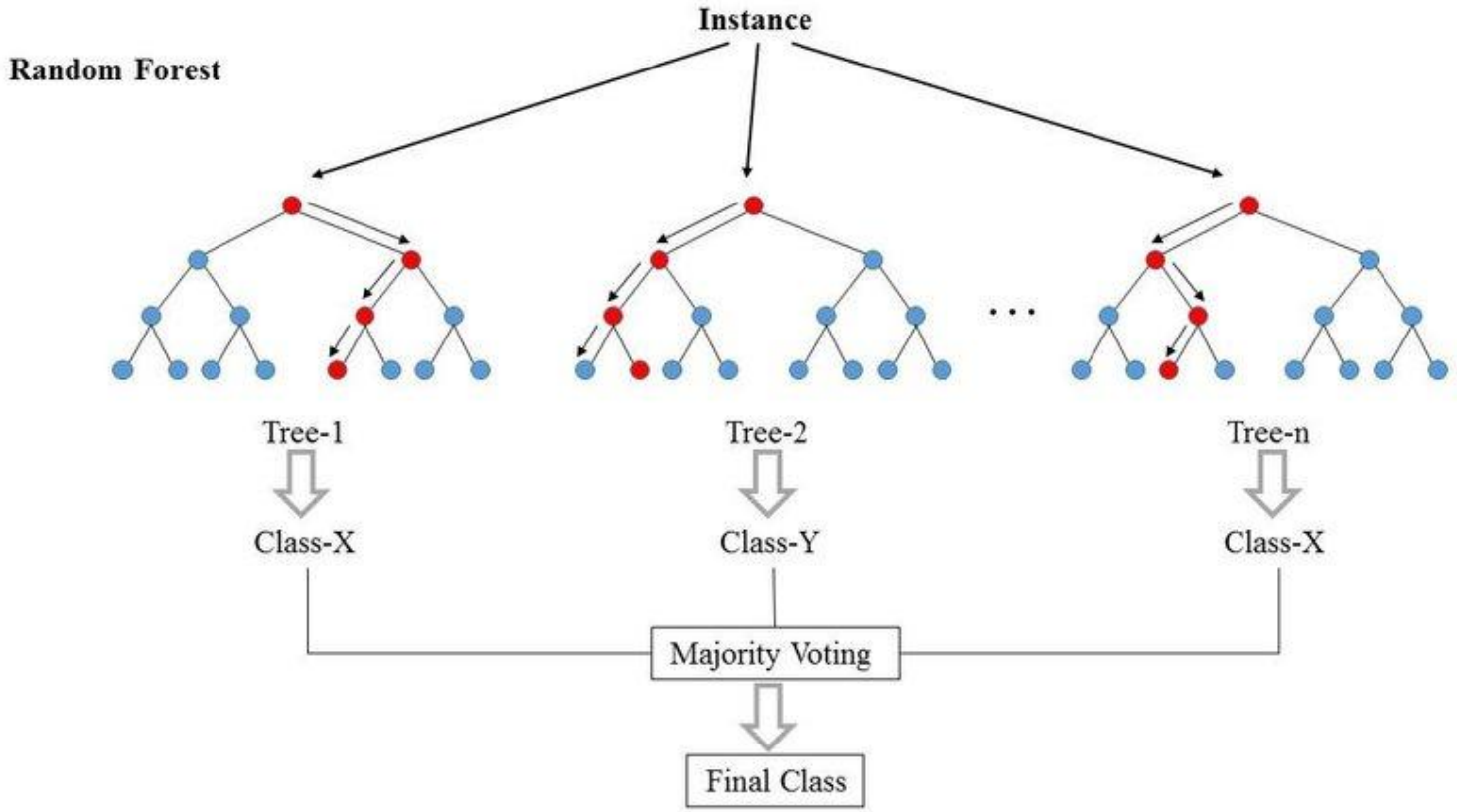
Random Forest

- **What it is:**
  An ensemble of many decision trees, combining their predictions for better accuracy.

- **When to use it:**
  A strong default model for many classification/regression tasks.

- **Example:** Predicting customer churn in telecom data.

Random Forest - Explanation



Random Forest – python code

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

KMeans (Unsupervised Learning)

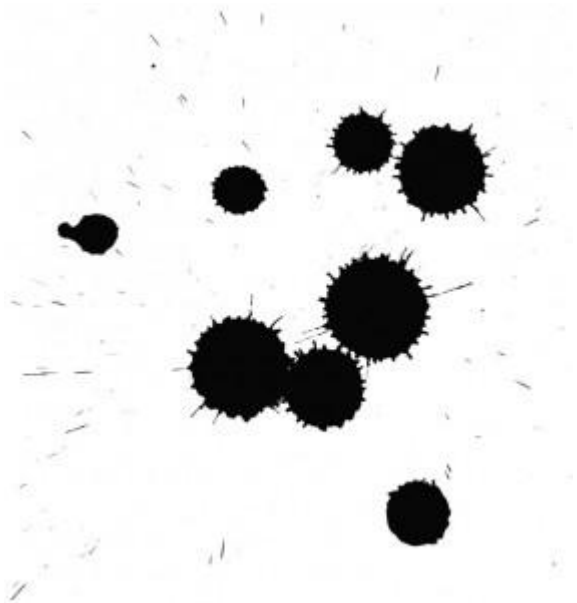
- **What it is:**
  Groups data into clusters without knowing labels.

- **When to use it:**
  When you want to discover natural groupings in data.

- **Example:** Grouping customers into 3 market segments.

Kmeans – Explanation (1)

- It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups
- Remember figuring out shapes from ink blots? **k means** is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!



How K-means forms cluster:

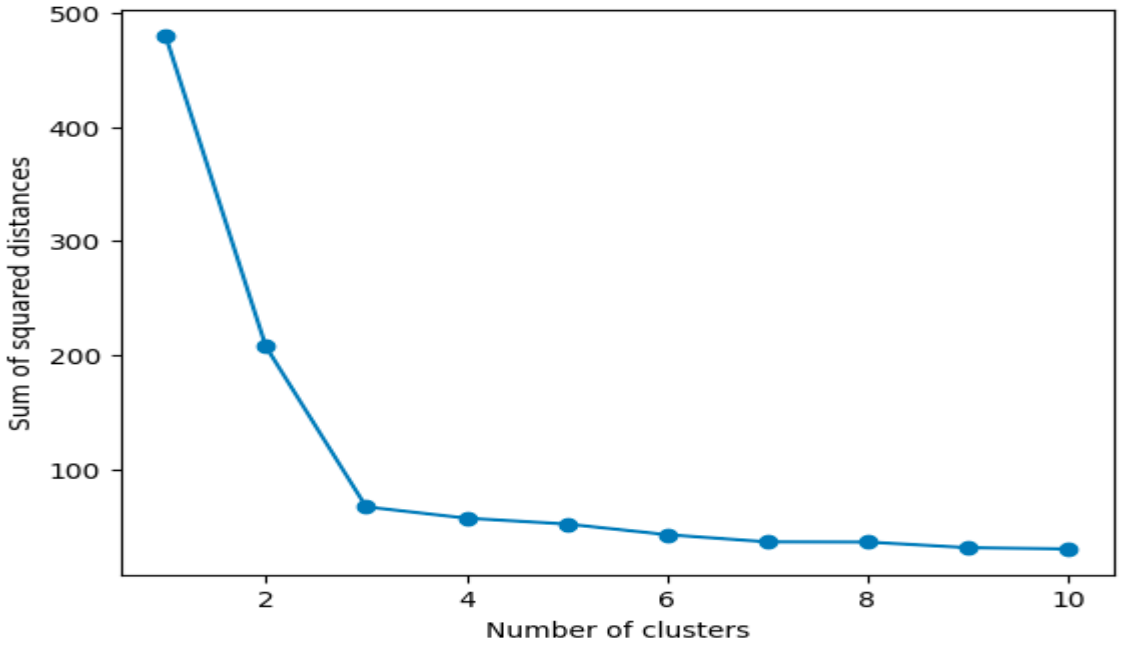
1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.
3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

Kmeans – Explanation (2)

How to determine value of K:

- In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

- We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.



TPA
Sharing Experience

KMeans– python code

```
from sklearn.cluster import Kmeans
import numpy as np
from scipy.stats import mode
kmeans = KMeans(n_clusters=3, random_state=42)

clusters = kmeans.fit_predict(X)

# Optional: map clusters to labels (if ground truth exists)
import numpy as np
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(3):
    labels[clusters == i] = mode(y[clusters == i])[0]
```

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Simple Feedforward Neural Network

- **What it is:**
  A model inspired by the brain, made of layers of neurons.

- **When to use it:**
  When relationships are complex and non-linear, especially with large datasets.

- **Example:** Handwritten digit recognition (MNIST).

Simple Feedforward Neural Network – python code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(32, activation="relu", input_shape=(X_train.shape[1],)),
    Dense(16, activation="relu"),
    Dense(1, activation="sigmoid")])
    # or Dense(len(unique_classes), activation='softmax')) for multi-classes examples

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
# or loss='sparse_categorical_crossentropy' for example
model.fit(X_train, y_train, epochs=20, batch_size=32)
```

Documentation:
https://www.tensorflow.org/guide/keras/sequential_model?hl=en

Quality Measures

- **Regression**
  - **R²**: proportion of variance explained.

$$\text{MSE} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

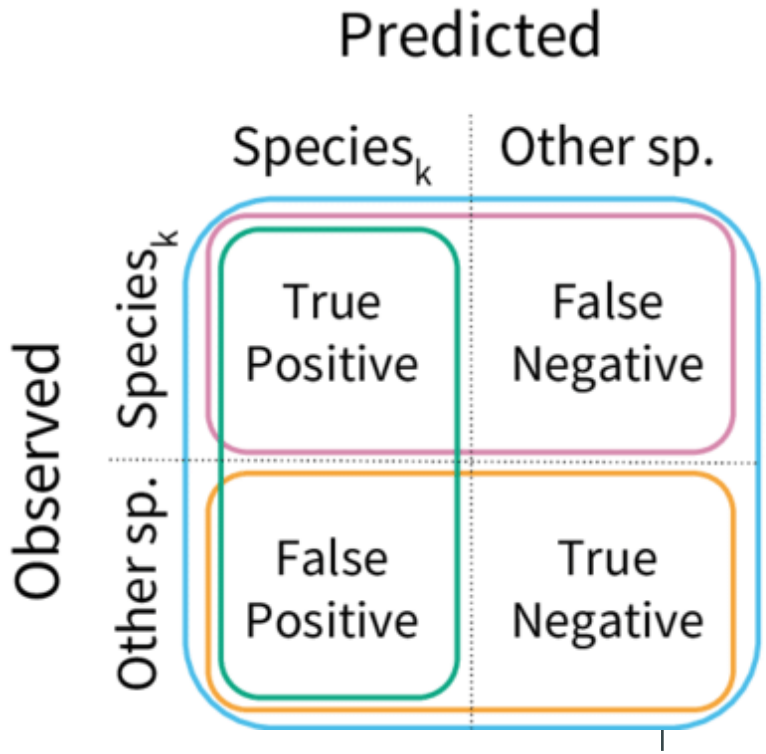
  - **RMSE**: how far predictions are from actual values.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Classification**
  - **Accuracy**: % correctly classified.
  - **Precision/Recall**: how many positive predictions are correct / how many real positives detected.
  - **F1 Score**: balance between precision and recall.
  - **Confusion Matrix**: breakdown of correct vs. incorrect predictions

Quality Measures – Classification models (1)

- The confusion matrix is one of the most widely used mechanisms to evaluate the performance of binary classifiers, i.e., classifiers that must decide between two classes and that can be interpreted as: positive and negative.



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

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

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

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

TPA
Sharing Experience

Quality Measures – Classification models (2)

- Accuracy: The proportion of the total number of predictions that were correct.
- Positive Predictive Value or Precision: The proportion of positive cases that were correctly identified.
- Negative Predictive Value: The proportion of negative cases that were correctly identified.
- Sensitivity/Recall: The proportion of actual positive cases that are correctly identified.
- Specificity: the proportion of actual negative cases that are correctly identified.

Quality Measures – python code

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,
r2_score, mean_squared_error

# Regression
print("R²:", r2_score(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))

# Classification
print("Accuracy:", accuracy_score(y_test, y_pred))
print("F1:", f1_score(y_test, y_pred, average="weighted"))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

Documentation:
https://scikit-learn.org/stable/modules/model_evaluation.html