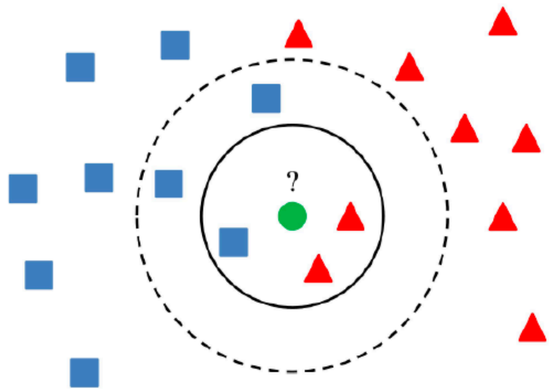
Machine Learning with Python

By Dr. Maryam Rahbaralam

جلسه سوم

K-nearest neighbors (KNN)

- Type of **supervised** machine learning algorithms.
- KNN is a **non-parametric** learning algorithm, which means that it **doesn't** assume anything about the underlying data. This is an extremely useful feature since most of the real world data doesn't really follow any theoretical assumption e.g. uniform distribution, etc.
- One of the **simplest** of all the supervised machine learning algorithms.

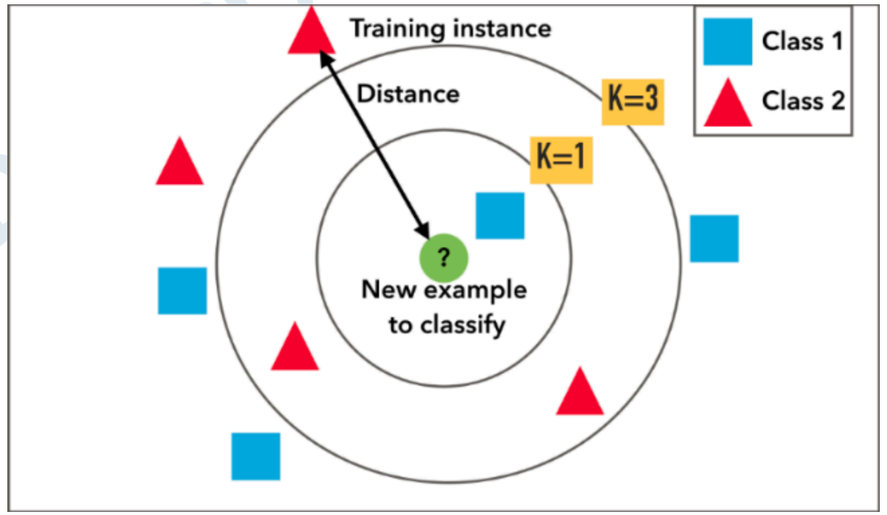


K-nearest neighbors (KNN)

1. It simply calculates the **distance** of a new data point to all other training data points. The distance can be of any type, here we use **Euclidean.**

2. It then selects the K-nearest data points, where K can be any integer.

3. Finally it assigns the data point to the class to which the **majority** of the K data points belong.

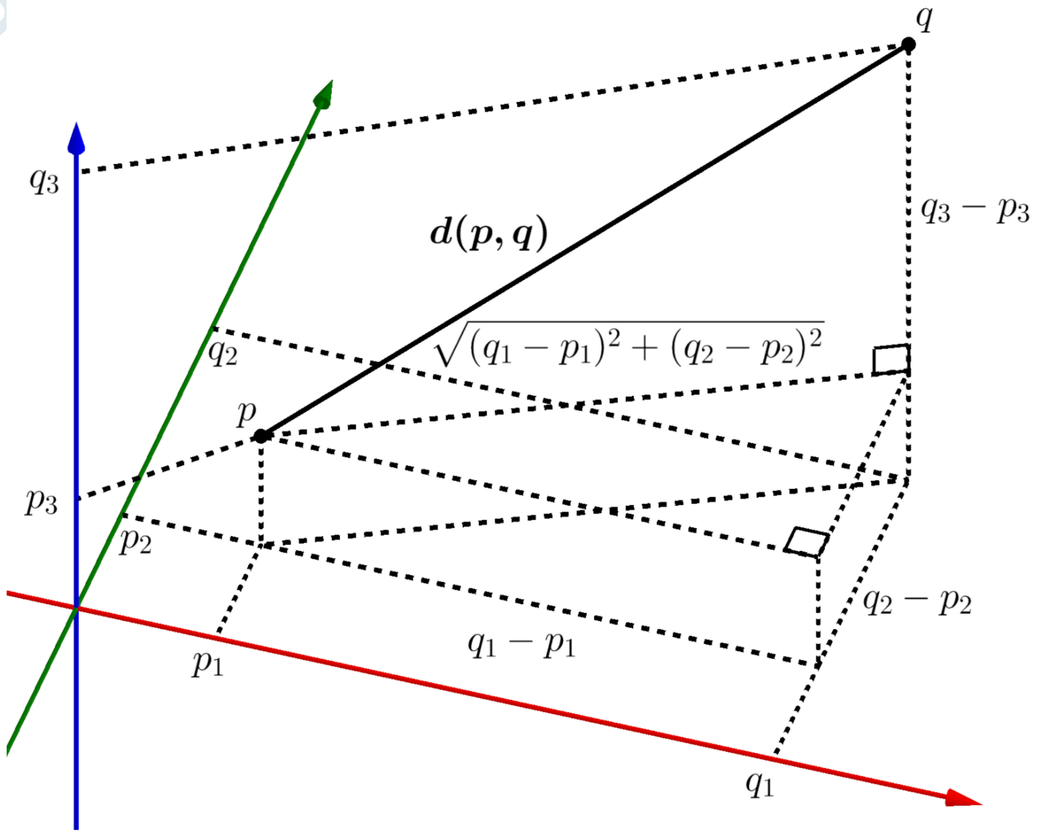


K-nearest neighbors (KNN)

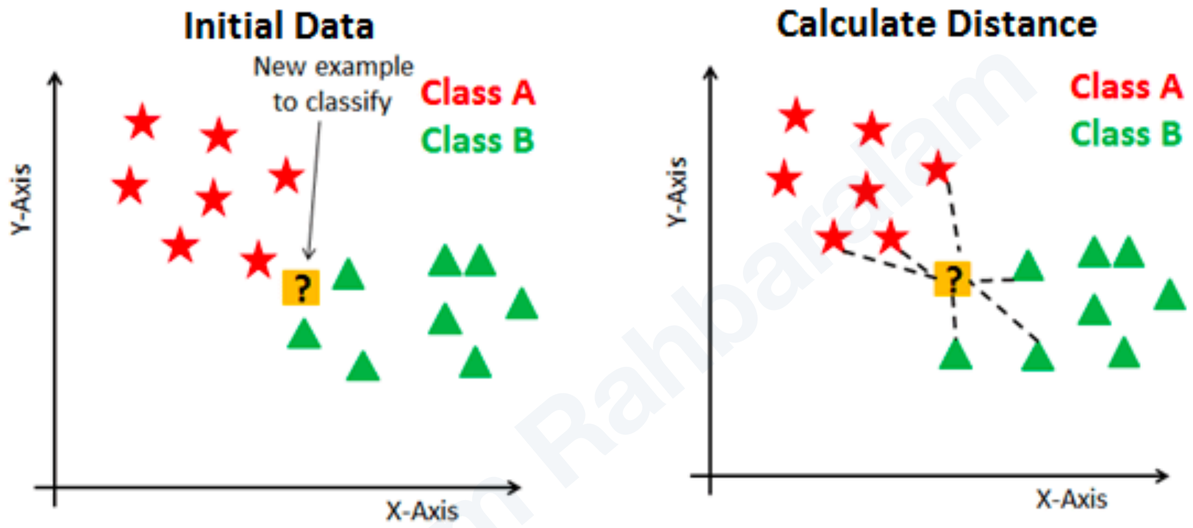
- **Euclidean distance**

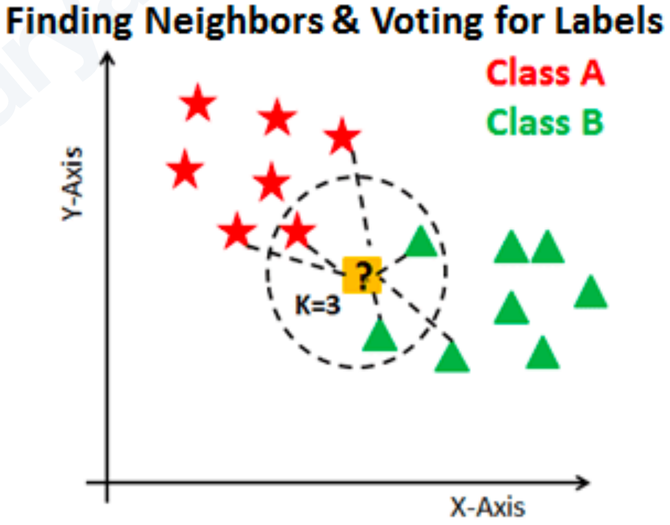
$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$



K-nearest neighbors (KNN)





Feature scaling

- A method used to **normalize** the range of independent variables or features of data.
- In data processing, it is also known as **data normalization** and is generally performed during the data preprocessing step.

Standardization (Z-score Normalization)

- In machine learning, we can handle various types of data, e.g. audio signals and pixel values for image data, and this data can include multiple dimensions.
- Feature standardization makes the values of each feature in the data have **zero-mean** (when subtracting the mean in the numerator) and **unit-variance.**

$$x' = \frac{x - \bar{x}}{\sigma}$$

Confusion matrix

- In the field of machine learning and specifically the problem of statistical classification, a **confusion matrix**, also known as an **error matrix**,[8] is a specific table layout that allows visualization of the performance of an algorithm

- Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).[9]
- The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

Confusion matrix

- Given a sample of 13 labeled animals — 8 cats and 5 dogs where Cats belong to class=1 & Dogs belong to class=0.

actual = [1,1,1,1,1,1,1,1,0,0,0,0,0]

- Now assuming we had previously trained a classifier that distinguish between Cats and Dogs. Now assuming we took the 13 samples and run them through the classifier and the classifier made 8 accurate predictions and missed 5: 3 Cats wrongly predicted as Dogs (first 3 predictions) and 2 Dogs wrongly predicted as Cats (last 2 predictions).

prediction = [0,0,0,1,1,1,1,1,0,0,0,1,1]

Confusion matrix

- With these two label sets (***actual*** and ***predictions***) we can create a confusion matrix that will summarize the results of testing the classifier for further inspection. The resulting confusion matrix looks lik

Cat Dog

actual = [1,1,1,1,1,1,1,1,0,0,0,0,0]

prediction = [0,0,0,1,1,1,1,1,0,0,0,1,1]

		Predicted class	
		Cat	Dog
Actual class	Cat	5	3
	Dog	2	3

- All **correct** predictions are located in the **diagonal** of the table (highlighted in bold), so it is easy to visually inspect the table for prediction errors, as they will be represented by values **outside the diagonal.**

Confusion matrix

- In abstract terms, the confusion matrix is as follows:

		Predicted class	
		P	**N**
Actual class	**P**	**TP**	FN
	N	FP	**TN**

- where: **P** = positive; **N** = Negative;
- **TP** = True Positive; **FP** = False Positive;
- **TN** = True Negative; **FN** = False Negative.

Confusion matrix

- In abstract terms, the confusion matrix is as follows:

		Predicted class	
		Cat	**Non-cat**
Actual class	**Cat**	5 True Positives	3 False Negatives
	Non-cat	2 False Positives	3 True Negatives

Confusion matrix

- Precision

$$\text{Precision} = \frac{tp}{tp + fp}$$

- Recall

$$\text{Recall} = \frac{tp}{tp + fn}$$

- Accuracy (ACC)

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- F1 score

$$\text{F}_1 = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

https://en.wikipedia.org/wiki/Confusion_matrix

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Implementing KNN Algorithm with Scikit-Learn

StandardScaler

1. from **sklearn.preprocessing** import **StandardScaler**
2. scaler = **StandardScaler()**
3. scaler**.fit**(X_train)
4. X_train = scaler**.transform**(X_train)
5. X_test = scaler**.transform**(X_test)

KNeighborsClassifier

1. from **sklearn.neighbors** import **KNeighborsClassifier**
2. classifier = **KNeighborsClassifier**(n_neighbors=5)
3. classifier**.fit**(X_train, y_train)
4. y_pred = classifier**.predict**(X_test)

Confusion matrix

1. from **sklearn.metrics** import **classification_report**, **confusion_matrix**
2. **confusion_matrix**(y_test, y_pred)
3. **classification_report**(y_test, y_pred)