# **Classification with KNN**

KNN is a classification algorithm for supervised learning, which uses proximity to make classifications or predict the grouping of an individual data point. While KNN can be used for both classification and regression problems, it is typically used as a classification algorithm.

For a classification problem, a class label is assigned based on a majority vote.

Regression, like classification, is a predictive problem setting where we want to use past information to predict future observations. However, in the case of regression, the goal is to predict numerical values instead of categorical values. The variable that you want to predict is often called the response variable. For example, we could try to use the number of hours a person spends on exercise each week to predict their race time in the annual Boston Marathon. As another example, we could try to use the size of a house to predict its sale price. Both of these response variables—race time and sale price—are numerical, and so predicting them given past data is considered a regression problem.

Suppose we need to predict a label of a new observation. KNN classifier generally finds K ‘nearest’ or ‘most similar’ observations in our training set and then uses diagnosis to predict the new observation diagnosis. K is a number that must be chosen in advance. Suppose we have 2 clusters: dogs and cats now when we have a new observation whose diagnosis is unknown. Then, we can look and see, which cluster’s point is closer to our new instance (with the help of a scatter plot). The idea here is that if a point is closer to a point in a scatter plot then the parameters are similar, so we can expect that they would have the same diagnosis.

Suppose we have another new observation that is surrounded by not only one cluster but two clusters. So, in this case, predicting by only using one nearest neighbor will not be sufficient. To improve the prediction we can consider several neighboring points, say K = 3, that are closest to the new observation to predict its diagnosis class. Among those 3 closest points, we use the majority class as our prediction for the new observation. In other words, if we choose our K to be 3 it counts the distances between 3 nearest neighbors, and if 2 of those belong to a dog cluster then the system assigns a new instance to a dog instance.

Distance between points → we decide which points are the K ‘nearest’ to our new observation using the straight-line distance. Suppose we have 2 observations a and b, each having 2 predictor variables ax, ay1 for a and bx, by for b . So the distance between these 2 instances will we:

Euclidian Distance = (ax - bx)2 +(ay - by)2

*Point A - X*: [20,30] [20,30] with class 0

*Point B- X*: [35,20] [35,20] with class 1

Distance*X*new,*A*​=sqrt(30−20)2+(25−30)2​)

Distance*X*new​,*B* =sqrt​(30−35)2+(25−20)2​

Manhattan Distance = ax - bx + ay – by

*Point A – (3,5)*

*Point B- (1,2)*

Distance=|3-1| + |5-2|

Cosine(alpha) Distance = A BAB

For the Vectors A with coordinates (2,3) and B with (5,6) the cosine distance = 0.002

Here are the main differences of Euclidean and cosine distance: Euclidean is useful for coordinate-based measurements. Cosine is better for data such as text where the location of occurrence is less important. Euclidean distance is more sensitive to course of dimensionality

To classify a new observation using the KNN classifier, we have to do the following:

Compute the distance between the new observation and each observation in the training set.

Find the K rows corresponding to the K smallest distances.

Classify the new observation based on a majority vote of the neighbor classes

Implementation

import numpy as np

X = np.random.rand(50, 2) \* 50

y = np.random.randint(0, 2, 50)

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:,1], c=y)

knn = KNeighborsClassifier().fit(X, y)

X\_new = [[30, 25], [10, 22]]

knn.predict(X\_new)

knn = KNeighborsClassifier(n\_neighbors=7).fit(X,y)

y\_new = knn.predict(X\_new)

x = np.concatenate((X, X\_new), axis = 0)

Y = np.concatenate((y, y\_new), axis = 0)

plt.scatter(x[:, 0], x[:,1], c=Y)