K-Nearest Neighbors

Artificial Intelligence

Project Report

Vijeet Nigam

20CS3068 | 3rd Year

Computer Science Engineering

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm that can be used for either regression or classification tasks. It is non-parametric, which means that the algorithm does not make assumptions about the underlying distributions of the data. This is in contrast to a technique like linear regression, which is parametric, and requires us to find a function that describes the relationship between dependent and independent variables.

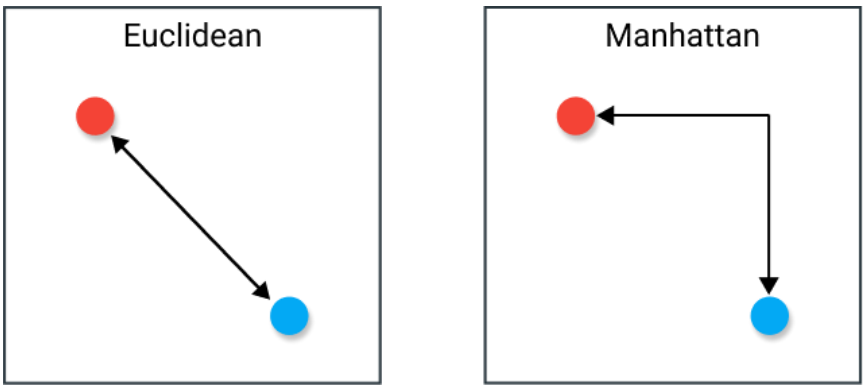
Minkowski Distance

The distance between points is determined by using a generalized formula, known as Minkowski distance, i.e., represented as follows:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}.$$

where X and Y are data points, n is the number of dimensions, and p is the Minkowski power parameter.

When p =1, the distance is known at the Manhattan (or Taxicab) distance, and when p=2 the distance is known as the Euclidean distance.



In two dimensions, the Manhattan and Euclidean distances between two points are easy to visualize, however at higher orders of p, the Minkowski distance becomes more abstract.

Libraries Used

numpy, pandas, collections, sklearn, matplotlib.pyplot

Data Set

To test the KNN classifier, I have uses the iris data set from **sklearn.datasets**.

The data set has measurements (Sepal Length, Sepal Width, Petal Length, Petal Width) for 150 iris plants, split evenly among three species (0 = setosa, 1 = versicolor, and 2 = virginica).

Building KNN Model

Step 1

Defining a function to calculate distance between two points
Here, Taking p = 2

Step 2

Using the distance function to get the distance between a test point and all known data points

Step 3

Sort distance measurements to find the points closest to the test point

Step 4

Using majority class labels of those closest points to predict the label of the test point

Used **collections.Counter** to keep track of the labels that coincide with the nearest neighbor points and then used the **counter.most_common()** to return the most commonly occurring label.

Step 5

Repeat Steps 1 through 4 until All the test data points are classified

Firstly, perform a **train_test_split** on the data (75% train, 25% test), and then scale the data using **StandardScaler().**

Since KNN is distance-based, it is must to make sure that the features are scaled properly before feeding them into the algorithm.

To avoid data leakage, its good to scale the features after the **train_test_split** has been performed. First, scale the data from the training set only **(scaler.fit_transform(X_train)),** and then use that information to scale the test set **(scaler.tranform(X_test)).**

This ensures that no information outside of the training data is used to create the model.

knn_predict()

Function takes in all of the training and test data, k, and p, and returns the predictions my KNN classifier makes for the test set (**y_hat_test**).

The function should return a list of label predictions containing only 0's, 1's and 2's.

Accuracy

My KNN Model's Accuracy: 0.9736842105263158

Sklearn KNN Accuracy: 0.9736842105263158

Effect of Varying 'k'

My KNN classifier Model performed quite well with the selected value of k = 5.

KNN doesn't have as many tune-able parameters as other algorithms like Decision Trees or Random Forests, but k happens to be one of them.

Observing the classification accuracy variation with the varying 'k' value: