# K Nearest Neighbor

In pattern recognition, the k-nearest neighbors’ algorithm (k-NN) is a non-parametric method used for classification. In both cases, the input consists of the k closest training examples in the feature space. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

# Implementation

## Importing Python Machine Learning Libraries

Python 3 together with Scikit-Learn are used to implement the K Nearest Neighbors Classifier.

Together with other libraries to split, normalize the training data and classification report.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

## Data Slicing

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 11].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15)

The above code divides the data into training and testing dataset.

## Data normalization

scaler = StandardScaler()   
scaler.fit(X\_train)  
X\_train = scaler.transform(X\_train)   
X\_test = scaler.transform(X\_test)

StandardScaler transforms data such that its distribution will have a mean value 0 and standard deviation of 1. Given the distribution of the data, each value in the dataset will be normalized to a 0-1 scale using the above code. Ensuring normalised feature values implicitly weights all features equally in their representation.

Template matching & interpolation

KNN does not use training data points to do any generalization unlike any other supervised machine algorithm. In other words, there is no explicit training phase, or it is kept to the minimal. KNN keeps all the training data and use them for the testing phase. It is one of the most time and space consuming classification method.

classifier = KNeighborsClassifier(n\_neighbors=5)   
classifier.fit(X\_train, y\_train)

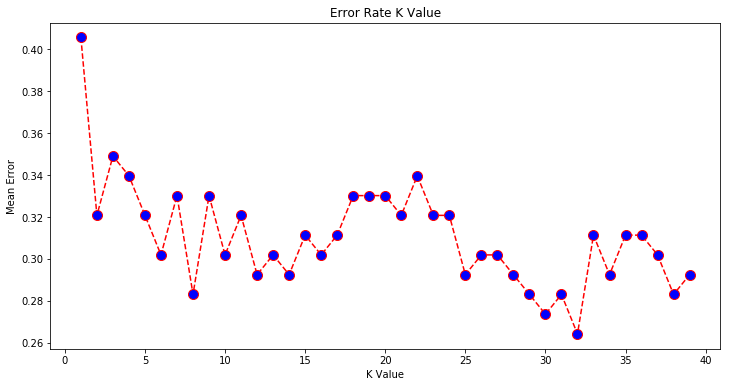
Using the above code snippets, we implemented the KNN classifier and train the classifier with songs data.

# Comparing Error Rate with the K Value

To get the best K value that yields the best result, we must find out error rate of each K value and make a comparison. One way to find the best value of K is to plot the graph of K value and the corresponding error rate for the dataset.

error = []  
# Calculating error for K values between 1 and 40

for i in range(1, 40):   
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train, y\_train)  
 pred\_i = knn.predict(X\_test)  
 error.append(np.mean(pred\_i != y\_test))  
plt.figure(figsize=(12, 6))   
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',   
 markerfacecolor='blue', markersize=10)  
plt.title('Error Rate K Value')   
plt.xlabel('K Value')   
plt.ylabel('Mean Error')



Given the graph above, the mean error rate is the lowest when K = 32. When K = 32 we have the highest accuracy when classifying songs.

# Testing

Testing is an important process in classification is to assign a class label to an unseen record. This process finds a class label for an input instance. It looks for the number of neighbors given a point and the K value. For instance, if K=5, KNN looks for 5 nearest neighbor and classify the test data based on most of the classes it finds.

y\_pred = classifier.predict(X\_test)

Using the above code snippets which makes use of the predict () function of sklearn.svm.libsvm library, we classify and assign a class label for the testing data to predict whether the input song is a top song or a non-top song.

# Results

Results of KNN Classifier using the train\_test\_split data slicing method:

Number of mislabeled points out of a total 600 points: 34, performance 71.66%

[0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1  
 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0  
 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1]

# Analysis

## Confusion Matrix and Measurement

|  |  |  |  |
| --- | --- | --- | --- |
| Predict Class | | | |
| Actual Class |  | Class = 1 | Class = 0 |
| Class = 1 | 101 | 39 |
| Class = 0 | 127 | 333 |
| Accuracy | 72% | | |
| Precision | 0.443 (Top=1); 0.105 (Top =0) | | |
| Recall | 0.721(Top=1); 0.276 (Top=0) | | |
| Cost | a\*0 + b\*50 + c\*50 + d\*0 = 8300 | | |

KNN algorithm identified instances correctly at a rate of 72 % within 2 second. This implies that the selected features of the song might be overlapped. As we have many features, it is very possible that some songs of different class contain similar feature values causing KNN algorithm to misclassify.