# Background/Explanation of Algorithms Applied to Album Label Classification Task

## K-Nearest Neighbour

**Why use it**

The first classification algorithm which will be tasked with categorizing songs into their corresponding albums here is the K-Nearest Neighbour supervised classification algorithm. The basic premise behind this classifier is calculating the ‘distance’ between every test sample and each training sample. Following the calculation of these distances, the top *k* (where *k* is an integer) training samples with the smallest distance from the test sample are selected for evaluation, along with the target labels describing them. K-Nearest Neighbour algorithms vary in how these *k* neighbours are then interpreted to predict the label for the test sample. One possibility is to return the label which occurs most often amongst the *k* neighbours (the mode). However, a limitation of this method is that it becomes more difficult to select the appropriate label if the set of *k* neighbours’ labels have several modes. Another solution, which will be implemented in this particular project, is to assign a weight to each of the *k* neighbours, with the closest neighbour being multiplied by a factor of 1, the second closest by ½, the third by 1/3, and so on and so forth. The weights for each label – which in this particular case is an album title – are then added up and the title with the highest weighted value is returned as the predicted label.

The ‘closeness’ between each test sample and all of the training samples can be calculated using different measures of distance, such as Manhattan, Euclidian and Minkowski distance. In this implementation, Euclidian distance will be used which is defined as the square root of the squared differences between the features which describe the two samples being compared. For two songs, the difference between audio features, including acousticness, loudness and energy, would be calculated for each one of these features. The differences would then be squared to ensure that negative and positive differences do not cancel each other out, and these squared differences are added together. The Euclidian distance is outputted by taking the square root of this summation, and the *k* training samples with the smallest distance are selected. One thing to take note of when implementing this algorithm is that because K-Nearest Neighbour is a ‘distance-based’ algorithm, the values of the features have to be scaled down prior to applying the algorithm. This is due to the fact that if certain features have a large scale of values, and others have a very small scale, the

This algorithm is based around the concept of measuring the difference, or ‘distance’, between all of the features of two inputs or objects. Often, the Euclidian distance (the square root of the sum of all the differences between each feature squared) is calculated, but an alternative is the Manhattan distance. When a test-object is inputted into the algorithm, the distance between the features of that input-object and all the objects used for training the model is calculated. Then, the classification label for that input is selected based on the label of the object from the training set which is the closest distance away. The *k* in the name of the algorithm defines the number of training samples to be considered when trying to decide which is the closest training sample to the test sample. K-Nearest Neighbour is often called a ‘lazy’ learning algorithm (ref: <https://www.analyticsvidhya.com/blog/2022/01/introduction-to-knn-algorithms/>). This means that during the training phase, all that happens is that the model simply stores the data points in the ‘training’ set. It only calculates the distances, therefore ‘learning’, in order to make the prediction on demand during the inputting of the test data, with the algorithm searching through all the training data objects and calculating the distances to the test object for each one, then sorting the distances and selecting the nearest matches (ref: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4978658/#:~:text=k%2DNearest%20neighbor%20classification,-The%20k%2Dnearest&text=Usually%2C%20the%20Euclidean%20distance%20is%20used%20as%20the%20distance%20metric>.) Therefore, as one article states, ‘k-NN does nothing at **fit**. All the work happens at **predict**’ (ref: <https://kenzotakahashi.github.io/k-nearest-neighbor-from-scratch-in-python.html>)

One problem with K-Nearest Neighbour is that there could be ambiguity and confusion if for the *k* closest neighbours, the neighbours are from different classes. One solution to this could be taking the mode (most frequent value) of these classes, but this could still lead to unresolved situations, such as if *k* were 3, and each of the three closest neighbours belonged to a different class. Therefore, another solution (ref: <https://www.geeksforgeeks.org/weighted-k-nn/>) is to use a *weighted* K-NN classification algorithm, where each of the *k* neighbours is assigned a weight that decreases the further away the neighbour is from the test point. In this way, closer neighbours are given more importance. The function which assigns the weights to the *k* neighbours is called the *kernel* function (ref: <https://vishvaasswaminathan.medium.com/weighted-knn-algorithm-c43b400346bf>) , the simplest of which is the *inverse-distance* function, which is easy to compute by taking the reciprocal of the distance from the test point (1/*distance*). Another issue with K-NN is that while Euclidian distance is ‘the most widely used distance function’ (ref: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4978658/#:~:text=k%2DNearest%20neighbor%20classification,-The%20k%2Dnearest&text=Usually%2C%20the%20Euclidean%20distance%20is%20used%20as%20the%20distance%20metric>), for some datasets, other kinds of distance functions such as cosine similarity measure or Chi-squared distance can lead to drastic differences in the model’s performance. In this study, we will use Euclidian distance as it often provides a basic starting point for the K-NN algorithm, but if there were more time, then it might provide an interesting task to adapt the K-NN class to be able to tweak the distance parameter and compare performance based on this adaptation.

**Literature review**

**How it works step-by-step**

**Problems and limitations with this algorithm**

## Gaussian Naïve-Bayes Classifier

## Decision Tree

## Random Forest

## Evaluation Techniques

**Accuracy and limitations**

**Precision**

**Recall**

**F1 Score**