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

## K-Nearest Neighbour

***“The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other:* *Birds of a feather flock together.”*** (Harrison, 2018)

**Why use it**

The first classification algorithm which will be tasked with categorizing songs will be the K-Nearest Neighbour supervised classification algorithm. K-NN is a classifier which is intuitive and straightforward in its implementation, while according to one research article, its ‘performance competes with the most complex classifiers in the literature’. (Prasath, et al., 2019) Additionally, an article published this year which ran the K-NN classifier on the GTZAN music dataset found that the average accuracy for genre-prediction for songs using this method was 70%, which was a promising result. (Mu, 2023) Other studies over 90% average accuracy for classifying music data using k-NN data (Thomas, 2020) (Ndou, Ritesh, & Jadhav, 2021). As such, this algorithm was selected on the basis of its relative simplicity of this algorithm and past successful results for studies which share the objective of music classification.

In simple terms, this classifier works by calculating the ‘distance’ between every test sample and each training sample. As one article states: ‘the K-NN algorithm *assumes* that similar things exist in close proximity’. (Harrison, 2018) Therefore, in the context of music, the algorithm looks at songs with similar values for their audio features, such as acousticness or loudness, and groups together songs based on these similarities. The main hypothesis in this project is that the songs which have similar audio features belong to the same Taylor Swift album, and therefore will be grouped together, meaning that the algorithm should be able to predict the album name of a test sample (song) just by looking at its audio characteristics.

Following the calculation of the distances between the test sample and all of the training samples, the top *k* (where *k* is an integer selected by the user) training samples with the smallest distance from the test sample are selected for evaluation, along with their target labels (ref: <https://blog.devgenius.io/exploring-knn-with-different-distance-metrics-85aea1e8299#:~:text=In%20conclusion%2C%20the%20best%20KNN,0.982456%20with%20k%3D11%20neighbors>.). 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, as it is the most widely used metric in the literature (Prasath, et al., 2019). It is basically the measure of the magnitude of a line drawn in multidimensional space between the two points (ref: <https://blog.devgenius.io/exploring-knn-with-different-distance-metrics-85aea1e8299#:~:text=In%20conclusion%2C%20the%20best%20KNN,0.982456%20with%20k%3D11%20neighbors>). This is calculated by taking 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.

Familiarity is not the sole reason for selecting Euclidian distance as a starting point (ref: <https://blog.devgenius.io/exploring-knn-with-different-distance-metrics-85aea1e8299#:~:text=In%20conclusion%2C%20the%20best%20KNN,0.982456%20with%20k%3D11%20neighbors>.) The Manhattan distance, which calculates the absolute value of the distances between data points before summing them and finding the square root, is said to be better suited to binary values rather than floating-point values of features which is what we have here (ref: <https://towardsdatascience.com/how-to-decide-the-perfect-distance-metric-for-your-machine-learning-model-2fa6e5810f11>) The Minkowski distance, which is like a parent class or abstract generalization of the Euclidian and Manhattan distance, calculates the differences between two points and then raises these differences to a power *p*, which is 1 in the case of Manhattan distance and 2 in the case of Euclidian distance. This *p* value can be tweaked to determine the influence of certain features, thus imparting a greater level of granularity and control, which is preferable for a later, more detailed exploration once a basic foundation has been established. Additionally, while Euclidian distance has been (ref: <https://towardsdatascience.com/how-to-decide-the-perfect-distance-metric-for-your-machine-learning-model-2fa6e5810f11>) discouraged for very high-dimensional datasets, this dataset consists of 530 entries and audio 9 features, which is not outstandingly high-dimensional. There are also some other more obscure distance measures such as Chebyshev or Jaccard distance, but these are aimed towards more specific use cases and kinds of data. Consequently, Euclidian distance presents a reasonable starting point for analysis but Minkowski distance offers an opportunity for future refinement and extension of this project.

One important consideration 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. Otherwise, if certain features have a large scale of values, and others have a very small scale, the features with the large scale will contribute disproportionately to the overall Euclidian distance, and their importance will thus be overrepresented when selecting the neighbours, while that of the other features will be diminished. As such, prior to training the K-NN model, the audio features will be scaled to their ‘z-values’ by subtracting the mean and dividing by the standard deviation for that feature.

K-Nearest Neighbour is sometimes referred to as a ‘lazy’ learning algorithm (ref: <https://www.analyticsvidhya.com/blog/2022/01/introduction-to-knn-algorithms/>). This means that during the training phase, the model merely stores the features matrix and labels belonging to the ‘training’ set, without doing any actual learning. It is only when the ‘predict’ method is called on the model that it actually does anything more complex, i.e. calculating the distances, sorting training samples according to these distances, and calculating the weights for the label for the *k* nearest neighbours (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>)

Although K-Nearest Neighbour is easy to understand and often performs as well as more complex algorithms, it has some disadvantages, such as high computational cost in terms of time and space complexity, because of needing to calculate the distance between one test sample and all of the training samples *n* times, where *n* is the number of training samples (Prasath, et al., 2019). This results in an O(nm) time and space complexity, where *n* is the number of training samples and *m* is the number of features. Furthermore, this model requires cross-validation to determine which value of *k* is best suited to this particular kind of data. Even more so than for other classification algorithms, it is imperative that the K-NN model is tested on different data than the training set. As the ‘fit’ phase of the modelling process does nothing else but to store the training samples and their labels, using the same data to ‘predict’ the labels would simply result in the model looking up the inserted sample in the stored training dataset, and outputting the correct label, thus resulting in a perfect accuracy score of 100%. However, it would probably perform very badly on any new, unseen data. Therefore, to prevent this *overfitting* (where the model’s performance on the training data is almost perfect, but where it fails to *generalize* or make accurate predictions for new data), and to find the optimal value for *k*, nested cross-validation with five ‘folds’ (which seems reasonable for a dataset of 530 values) will be used rather than simply running the model on a training set and a holdout set.

## Gaussian Naïve-Bayes Classifier

## Decision Tree

## Random Forest

## Evaluation Techniques

**Accuracy and limitations**

**Precision**

**Recall**

**F1 Score**