# Background: Attempting to Classify Taylor Swift songs based on music parameters into album labels

*Here you should explain in your own words how your algorithms work. This is where you can demonstrate that you understand the tools and models that you used. Marks will be gained accordingly.*

Although so far, there are not many studies on classifying Taylor Swift songs (or another artist’s songs) into a specific album, there have been many similar machine learning projects focused on classification of tracks into particular genres based on their audio features. There are many supervised learning algorithms which could be used to try to predict which album a song belongs to, including K-Nearest Neighbour, Decision Trees and Random Forests and Naïve-Bayes classifiers. Additionally, K-Means, K-medians (ref: <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68#:~:text=K%2DMedians%20is%20another%20clustering,median%20vector%20of%20the%20group>.) and Gaussian Mixture Models provide just a few examples of unsupervised clustering algorithms which could potentially separate tracks into groups based on their audio attributes, which could then be compared with the actual groupings of tracks in the albums.

In this project, the K-Nearest Neighbour supervised classification algorithm will be implemented from scratch and then applied to the features matrix of the tracks’ audio attributes, to see if the correct album label can be predicted from those musical characteristics, before evaluating the classifier in terms of its precision, recall and accuracy using a confusion matrix. The K-Nearest Neighbour 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.

Next, a Naïve-Bayes classifier will be implemented and evaluated using those same metrics, as well as a Random Forest classifier. Then the supervised learning models will be compared to see which one had the best performance in terms of songs classification. Additionally, the Random Forest classifier in scikit-learn allows the inspection of which features (i.e. audio attributes) have the greatest impact on the model’s predictions, which will also be analysed.