# 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)

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 that is very 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 four ‘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.

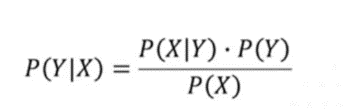
## Gaussian Naïve-Bayes Classifier

The second algorithm which will be applied to the song data is a Gaussian Naïve-Bayes Classifier, which will also be implemented using the *NumPy* library only.

The Gaussian Naïve-Bayes Classifier is another classifier which has been used to categorize song genres, but its *modus operandi* is completely different from that of the k-Nearest Neighbour classifier. It is based on adjusting the probability that an item belongs to a class, given some features, by taking into account *how frequently that class occurs* within a dataset in the first place.

This classification algorithm is based on Bayes Theorem, which predicts the probability that a sample belongs to a certain class *given* its specific set of features. This is known as the ‘posterior’ probability. When adapted to classification purposes, the posterior probability of a sample belonging to all the possible classes given its features is calculated. Then, the class with the greatest posterior probability is selected.

To calculate this posterior probability (probability that a sample belongs to a class given its features) for each potential class (in this case, album name) we can use an adaptation the following theorem (Bayes Theorem):



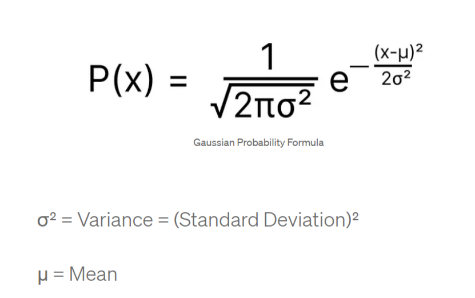
where *Y* is the album label, *X* is the sample (row of feature-values), and *P(Y|X)* is the ‘posterior’ that is described above.

To briefly explain this equation, *P(Y)* is known as the ‘prior’, or here: the overall probability of any song having the class *Y*. This is simply calculated as the ratio between the frequency count of all the songs in that album class and the total number of songs or samples. It is the probability that any randomly-selected song will belong to a specific album *prior* to ‘knowing’ anything about its features, i.e. X.

The denominator, *P(X)* is known as the ‘marginal probability’, as it ‘marginalizes’ the influence of the label Y, thus extracting the pure probability of that sample, X. The reason this is in the equation is because it ensures that the resulting posterior probabilities for each album class will sum to one (as probability must take a value between 0 and 1). It can be expressed as the sum of the joint-probability (i.e. *P(X ∩ Yi*)) for every label, *Y*i.

However, with respect to the particular use case of adapting the Bayes Theorem to classification, it is not required that this marginal probability is included in the formula, as it is irrelevant if the posterior is actually a genuine probability value or not. This is because the objective of the classification algorithm is simply to select the class with the greatest posterior for that sample, which can be done using NumPy’s argmax function again (just like in k-Nearest Neighbour). It therefore makes no different in this regard if the posterior is strictly speaking a probability or not, so long as the largest value is selected.

The most challenging and counter-intuitive task when programming this algorithm is the question of how to calculate *P(X|Y)* in the nominator, which is the conditional probability of the sample X given a label Y and is known as the ‘likelihood’. This can be estimated using a formula known as the *Gaussian density function* given below (and is the reason for why this is called a *Gaussian* Naïve Bayes classifier). The Gaussian model was selected for this classification task because the features used to characterize the samples here are continuous, numerical variables, while other common variations of Naïve Bayes such as Multinomial Naïve Bayes (reference: <https://betterprogramming.pub/predicting-a-songs-genre-using-natural-language-processing-7b354ed5bd80>) are better suited to variables represented as discrete counts (often used in language processing).



A limitation of using this formula is that it assumes that the features (X) roughly follow a Gaussian (normal) distribution (Reference: <https://levelup.gitconnected.com/classification-using-gaussian-naive-bayes-from-scratch-6b8ebe830266> ). As is illustrated on the FacetGrid and pairplots showing the Kernel Density Distribution of the features, many of these audio characteristics do follow a roughly symmetrical, normal distribution, but there are some exceptions such as acousticness, which do not have a symmetric distribution but have two peaks, or a heavier tail (kurtosis) to one side. Therefore, this is a significant limitation of this algorithm that should be taken into account when evaluating its performance.

As every sample, x has multiple features and can be broken down into {x1, x2, x3 … xn}, the likelihood in the nominator, *P(X|Y)* can be broken down into P(x1|Y)\*P(x2|Y)\*P(x3}Y)\*…\*P(Xn|Y), but with one important caveat. This formula for the likelihood works *assuming* that each feature in the sample, xi, is unconditionally *independent* from all the other features (reference: <https://jayellwolfe.github.io/2020-07-30-Naive-Bayes-From-Scratch/>) . Two variables, X1 and X2, are conditionally independent if their joint conditional probability *P(X1,X2|Y*) can be expressed as *P(X1|Y)*\**P(X2|Y)* (Murphy, 2012). Conditional independence means that the influence of the variables on one another ‘is mediated via other variables’, represented as *Y*, and is not ‘direct’. (Murphy, 2012) Naturally, in real life, variables are often directly influenced by one another, as even with the example of song data, it is easy to imagine how features such as tempo and danceability could have some form of relationship or correlation to each other: faster, upbeat songs might be more likely to be easier to dance to. As such, this assumption is why this model is called a *Naïve* classifier. Although features might have complex interrelationships in reality which, to achieve the utmost precision, would have to be modelled using more complex calculations using the chain rule of probability, when in practice the Naïve Bayes Classifier often works well despite abstracting away some of this complexity.

One more extra property of the Naïve Bayes algorithm to remark upon is that due to the computational constraints related to the ability of representing very, very small floating-point numbers, what happens when multiplying many very small likelihood probabilities for each feature is that the value becomes so small that it disappears and the number gets truncated to zero. Therefore, in this implementation of the classifier, the logarithm will be taken for both sides of the equation, to address this issue. Another problem encountered when designing this algorithm was that the Gaussian density function led to unexpected outputs such as *NaN* and *inf* when the standard deviation was 0. Although it is rare for the standard deviation of a feature to be 0, it can happen – therefore, to ensure that the algorithm is more robust, a condition was added to the Gaussian density calculation using *np.maximum* to set the standard deviation to 0.1 in the case of it being 0, in order to avoid the dreaded division by zero error.

A property of Naïve Bayes which makes it an interesting algorithm to compare with k-Nearest Neighbour is this concept of accounting for the prior-probability of each album class in the calculation of the posterior. In contrast to k-NN, the imbalanced nature of this dataset is accounted for in Naïve Bayes through the calculation of this prior. The likelihood of the sample given a class is multiplied by the proportion of the total samples that a specific album represents, thus representing the fact that some albums have many more songs than others. This can be seen as a significant advantage that this algorithm has over k-NN. Other benefits of using this classifier is that it does not take in any parameters, so it can be evaluated more quickly using simple cross-validation instead of nested cross-validation for hyperparameter tuning. It is also relatively fast, and not prone to overfitting, unlike a decision tree-based algorithm or k-NN (reference: <https://jayellwolfe.github.io/2020-07-30-Naive-Bayes-From-Scratch/>) . However, the fact that it assumes that the feature variables follow a Gaussian distribution is a significant limitation of it that should also be considered.

Finally, it should be noted that as the Gaussian Naïve Bayes Classifier is not a distance-based algorithm but based on computing probabilities and statistics for each variable, the features do not have to be normalized using z-scores, so the raw values can be used in the features matrix.

**Implementation and challenges:**

## Decision Tree

## Random Forest

## Evaluation Techniques

**Accuracy and limitations**

**Precision**

**Recall**

**F1 Score**