Machine Learning and Neural Networks CM3015 Mid-Term Coursework:
Comparing the Performance of Four Supervised Machine Learning Algorithms in Predicting Album Titles of Taylor Swift Songs

Contents

Abstract 2
Introduction 2
Aims 3
Background 3
K-Nearest Neighbour 3
Gaussian Naïve-Bayes Classifier 4
Decision Tree 7
Random Forest 9
Methodology 11
Overview 11
Pre-Processing 12
Feature Selection 12
Data Scaling 12
Preventing Overfitting: N-Fold Cross-Validation, Nested N-Fold Cross Validation and *scikit-learn*'s GridSearch 13
Evaluation Metrics: Confusion Matrices, Accuracy and F1 Scores 14
Results 15
Evaluation 17
Issues related to Non-Normal Feature Distributions 17
Suggestions for Dealing with Imbalanced Datasets 18
Unsupervised Algorithms for Detecting Non-Album Name Related Patterns 19
Feature Refinement 19
Presence of Outliers 19
A Note on Computational Performance Issues 20
Conclusion 20
References 21

Abstract

Taylor Swift is one of the world's most well-known and successful musical artists. This project explores the musical style of Swift's albums by employing machine learning techniques in an attempt to identify whether an album can be predicted using the audio attributes of its songs. The dataset used here was originally collected using the Spotify API. For each song, it lists various audio attributes, such as danceability or energy, as well as the album name. As such, this study aims ***to quantify how distinctive the musical style of each album is****, as well as to compare* ***which machine learning classification algorithm performs best on this dataset****.*

To answer these questions, four supervised learning algorithms were applied to the data. These were k-Nearest Neighbor, Gaussian Naïve-Bayes, Decision Tree and Random Forest. The performance of each algorithm was evaluated using precision and recall, in addition to accuracy. Moreover, techniques such as nested cross-validation and the scikit-learn GridSearchCV feature were used to tune the models' hyperparameters and to reduce the risk of overfitting.

The results show that the weighted k-Nearest Neighbor algorithm performed best on this dataset, achieving an accuracy score of approximately 0.6, as well as recall and precision scores of over 0.7 for certain albums. Furthermore, the results for each algorithm showed that the recall and precision scores for some albums were consistently much higher than for others. For instance, the albums '1989', 'Midnights', 'Red', 'Speak Now', 'evermore' and 'folklore' were classified correctly to a much greater extent than 'reputation' or 'Lover', which received recall scores of 0.2 and 0.13 using k-NN. Moreover, the algorithms frequently misclassified songs belonging to 'evermore' as belonging to 'folklore' and vice versa, indicating that these albums are similar in style. These scores suggest that some albums might have a more distinctive musical style than others, as well as confirming popular opinions that Swift dramatically changed her style during the COVID-19 pandemic when she released 'folklore', closely followed by 'evermore' .

Consequently, this project demonstrates how machine learning can be used to trace the evolution of a singer's musical evolution over the span of their career, as well as explaining why certain albums might have been more successful than others due to shifts in their audio attributes. Some suggestions for future improvements include applying k-Means Clustering to reveal more patterns and structures within the song's audio characteristics. Finally, the project evaluation provides a discussion of how more advanced techniques for dealing with imbalanced datasets, such as this one, could be applied to further refine these models.

Introduction

Data Source: https://www.kaggle.com/datasets/jarredpriester/taylor-swift-spotify-dataset/data?select=taylor_swift_spotify.csv

Taylor Swift is undeniably one of the most successful and influential women in music in 2023, as stated by *Forbes* magazine (Dailey, 2023). Swift's 2023 *Eras* tour was the first ever music tour to gross over 1 billion dollars (Associated Press, 2023). According to *BusinessInsider,* it even had a profound impact on the US economy in the summer of 2023 (Gaines, 2023). The basic aesthetic and conceptual premise behind Swift's *Eras* tour was to visually and narratively promote a different "feel" for each of the singer's albums, to showcase the evolution of her musical style over the years, accompanied by a complete change of set and costume whenever she sang the songs from that particular record.

The main aim of this project is to either confirm or disprove whether, from a *musical* perspective, it is indeed the case that each Taylor Swift album has a distinctive sound and character. The dataset used here (available on Kaggle, but originally compiled through the Spotify API) contains each Taylor Swift song as a row/sample in a table. There are columns representing musical attributes/features such as "acousticness", danceability, energy and loudness. Furthermore, each row/song also has an album attribute to designate which record it was on – these will constitute our "labels". To achieve this aim, several classification algorithms, including K-Nearest Neighbour, Decision Trees, Naïve-Bayes and Random Forest will be implemented in order to predict which album a song belongs to. Then, the performance of these algorithms will be evaluated and compared using confusion matrices as well as accuracy, precision, recall and f-1 scores. Additionally, it will also be assessed whether the algorithms are more likely to classify songs in certain albums correctly than other albums, thus determining which albums have the most 'distinctive' style.

Aims

- *To evaluate and compare the performance of four supervised classification algorithms in predicting the album names of Taylor Swift's songs based on the songs' audio attributes.*
- *To determine which Taylor Swift albums were easiest for the machine learning algorithms to correctly categorize based on the songs' audio attributes, and which posed a greater challenge.*

Background

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)

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).

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 (Siddique, 2023). 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 the particular version of the algorithm used for this 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 (album) 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 distance metrics, 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 (Siddique, 2023). 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. 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 (Maheshwari, 2021). 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 may be preferable for a later, more detailed exploration once a basic foundation has been established. Additionally, while Euclidian distance has been discouraged for very high-dimensional datasets, (Maheshwari, 2021) this dataset consists of 530 entries and less than 10 audio 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 foundation for analysis while Minkowski distance may offer 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 (Jain, 2022). 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 (Hu, Huang, Ke, & Tsai, 2016). Therefore, as one article states, 'k-NN does nothing at **fit**. All the work happens at **predict**' (Takahashi, 2016).

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(*n*m*) time and space complexity, where *n* is the number of training samples and *m* is the number of features. Furthermore, this model requires nested cross-validation to determine which value of *k* is best suited to this particular kind of data and to make the model's results more reliable. 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 (the 'prior').

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):

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

Figure 1: 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 formula, 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. $P(X \cap Y_i)$) for every label, Y_i.

However, with respect to the particular use case of adapting the Bayes Theorem for the purposes of classification, it is not required that this marginal probability is included in the formula. This is because, here, it is irrelevant if the posterior is actually a genuine probability value or not. 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 Neighbor). 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 are better suited to variables represented as discrete counts (often used in language processing) (Sianipar, 2019) .

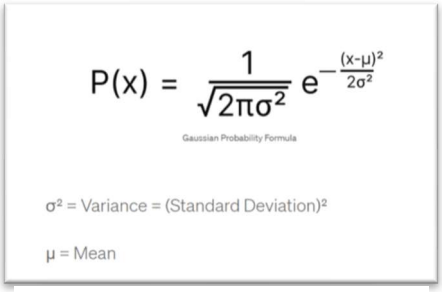

Figure 2: Gaussian Density Function

A limitation of using this formula is that it assumes that the features (X) roughly follow a Gaussian (normal) distribution (Patidar, 2023). 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 $\{x_1, x_2, x_3 \dots x_n\}$, the likelihood in the nominator, $P(X|Y)$ can be broken down into $P(x_1|Y)*P(x_2|Y)*P(x_3|Y)*\dots*P(X_n|Y)$, but with one important caveat. This formula for the likelihood works *assuming* that each feature in the sample, x_i, is unconditionally *independent* from all the other features (MojoCode, 2020). Two variables, X_1 and X_2, are conditionally independent if their joint conditional probability $P(X_1,X_2|Y)$ can be expressed as $P(X_1|Y)*P(X_2|Y)$ (Murphy, 2012). Conditional independence requires 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 to each other. 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, to address this challenge, the logarithm will be taken for both sides of the equation when implementing this classifier. 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 to a greater extent 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 include it not taking in any parameters. Therefore, it can be evaluated more quickly using simple cross-validation instead of nested cross-validation for hyperparameter tuning. Naïve-Bayes classification is also relatively fast, and not prone to overfitting, unlike a decision tree-based algorithm or k-NN

(MojoCode, 2020). However, the fact that it assumes that the feature variables follow a Gaussian distribution is a significant limitation of this classifier which should be taken into account.

Finally, 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.

Decision Tree

Reference: my understanding of decision trees has been supplemented and enriched by this set of videos and quizzes supplied by the excellent Analytics Vidhya resource for data science. Link: Analytics Vidhya: Getting Started with Decision Trees - Course

The next algorithm used for this task Decision Tree classifier. As a Decision Tree is a relatively complex algorithm which can be tuned using *multiple* hyperparameters, the pre-made scikit-learn Decision Tree Classifier will be imported here. Although it would be an interesting challenge to implement a Decision Tree classifier from scratch, unfortunately due to the time constraints for the midterm assignment this is infeasible for the current project. Iterating through every possible value for each of the audio features to determine which one to select for splitting the group of samples is something of a daunting task compared to the implementation of simpler algorithms like K-Nearest Neighbour or the Naive Bayes Classifier, particularly as the Decision Tree takes in so many parameters. As a result, in order to select the optimal hyperparameters, *scikit-learn*'s Grid Search facility will be applied in this here.

A decision tree can be used for both regression and classification tasks. As the objective of this particular project is to try to predict the album a song belongs to, the decision tree classifier rather than the regressor (used to predict continuous target variables) will be applied here. On the most basic level, a decision tree can be conceptualized as a series of yes-or-no questions, or if-else statements, each of which is represented as a decision 'node' in a binary tree. In the context of album-categorization, one of these 'questions' could be something like 'is the value of the danceability feature of the song above 0.5?' This decision 'splits' the original group of samples into two subsets of samples/songs, based on whether they meet this criterion or not. The 'aim' of each split can be conceived of as trying to output two 'pure' or 'homogenous' groups, where each resulting subset contains only samples belonging to one class (album), thus perfectly separating the songs by album. A decision tree can have multiple splits – however, selecting the optimal stopping point at which there will be no more branching depends on various hyperparameters such as maximum tree depth which will be explored in more detail below.

There are many options for which algorithm should be chosen for determining which split should be accepted out of several candidate splits. The most well-known out of these is the 'Gini' split selection algorithm, which is the default criterion that is set in the *scikit-learn* Decision Tree Classifier class. The 'Gini' algorithm is based on the concept of 'Gini impurity', which is calculated with the formula:

- *Gini Impurity = 1 – Gini*, with *Gini* being the summation of *the squares of the probabilities* of each class in each of the sub-groups resulting from that particular 'split'.
- The probability of each class in that sub-group is equal to the number of times a sample with that class appears in the sub-group, divided by the total number of samples in the subgroup.
- Once this *Gini Impurity* measure has been calculated for each of the two subgroups, it is multiplied by the sub-group's weight, which is defined as the number of samples in that group as a proportion of the total number of the samples in the sub-group's parent node.
- Gini Impurity is a measure of how 'pure' or 'homogenous' the samples in a sub-group/node are. As the aim of a decision tree is to output nodes/sub-groups which are as homogenous as possible, a lower Gini impurity is favored over a higher Gini impurity.

- Therefore, Gini Impurity is calculated for all possible splits, and the split with the lowest Gini Impurity is selected.

An alternative algorithm for selecting the optimal split is called *Chi-Squared*.

- The aim of chi-squared is *to quantify the difference in homogeneity of classes* between the parent node (original group of samples) and its child nodes (sub-groups based on the 'split').
- Overall, in this algorithm, the *expected* count of each class for the sub-groups is subtracted from the *actual* count of classes in that sub-group.
- This *expected* frequency of samples belonging to a specific class is calculated as the proportion of those samples in the parent node (the one 'making the decision') multiplied by the total number of samples resulting from the split in the child node.
- The actual frequency is just that: the real count of the number of samples belonging to that specific class in the child node/sub-group.
- For each class existing in the group which is to be split, the difference between this actual count and expected count is calculated, then divided by the 'expected' count and squared.
- Then, the chi-squared value for that sub-group is defined as the sum of these squares for each class.
- As with Gini impurity, each child node's chi-squared score is multiplied by its weight and these results are added together to form the chi-square value for that split.
- A high chi-squared value indicates that the difference in class distribution between the parent node's group and the subgroups in the child nodes is high, and therefore that with this split, the tree has succeeded into taking a step towards creating purer and more homogenous groupings of samples, with more samples belonging to the same class (album in this particular case).
- Therefore, the split resulting in the highest chi-squared value is the one that is selected.

There is also a third algorithm based on the concept of 'information gain' or 'entropy'. 'Entropy' is another measure of how 'pure' a node is, with low entropy representing more homogenous nodes or sub-groups. The entropy of a resultant node is calculated by using the following expression:

$$- p_1 * \log_2 p_1 - p_2 * \log_2 p_2 - p_3 * \log_2 p_3 - \ldots - p_n * \log_2 p_n$$

Figure 3: Entropy calculation

where *n* is the number of possible classes, while *p* represents the proportion of that class in the node.

This entropy value is calculated for both the parent node and as the weighted entropy of the two child nodes. If the entropy of the parent node is lower than the resulting weighted entropy of the two child nodes, this means that the split has not succeeded in making the sub-groups more homogenous, and that this split should be discarded. As such, the split resulting in the greatest reduction in entropy between the parent and weighted child nodes is selected.

Technically, a decision tree could keep splitting and splitting until each 'leaf' or 'terminal node' (defined as a node without any further splits or children) would consist of completely 'pure' subsets of samples. Theoretically, one could imagine how for a set of training data, this splitting would continue until each leaf consisted of only one data point, which would trivially be 'perfectly pure'. It is easy to imagine how this would lead to overfitting, where the model would basically have memorized the labels for every sample in the training data, but failed to grasp generalizable patterns, thus leading to poor performance on the test dataset. This is because the tree would simply have recorded the exact branching path down to the leaf nodes based on the specific, detailed characteristics of the training samples, which might not be applicable to new data. As such, various hyperparameters have to be tuned to stop this overfitting from occurring. Some of these include:

- Only splitting a parent node further if the parent node includes a subset of samples which reaches a certain (integer) threshold.
- Constraining the maximum depth (longest path from root node to a leaf node) of the decision tree, so that no more splits occur once this depth has been attained.
- Setting a minimum number of samples that a sub-group has to have in order to qualify to become a leaf node, and truncating the tree to the parent node if the potential leaf node does not meet this threshold.

Of course, while setting a very high minimum threshold value for the number of samples required for a split would prevent overfitting, much care has to be taken to ensure that this value is not too low, in which case the decision tree will not learn sufficient patterns in the data and therefore underfit. Similarly, with the maximum tree depth constrain, setting this to a number that is too low can result in an insufficiently complex model, and therefore to underfitting. These techniques used to prevent the risk of overfitting can be classed as 'pre-pruning' techniques (Saini A. , 2023) which refers to deleting nodes *while* growing the decision tree.

Additionally, *scikit-learn* also enables input arguments determining the aggressiveness for 'post-pruning' of the tree, with parameters such as *ccp_alpha*, or 'cost-complexity pruning' (Link to scikit-learn docs) which identifies the 'weakest' leaf node (the ones with the greatest impurity) and removes them after the tree has been learnt. As a suggestion for an extension of this project, using PCA dimensionality reduction to decrease the number of features while retaining the most varied characteristics of the sample prior to constructing the Decision Tree could also prevent overfitting by giving the tree 'a better chance of finding features which are discriminative'.

Consequently, we will use the scikit-learn Grid Search facility to search for and evaluate the optimal hyperparameters for these constraints before fitting the Decision Tree Classifier on the dataset and comparing its performance to that of the other supervised classifiers (Saini B. , 2020).

Random Forest

Random Forests are another supervised learning algorithm which constitute a natural extension to Decision Trees. This algorithm has been called an 'ensemble technique', (Saini A. , An Introduction to Random Forest Algorithm for beginners, 2022) because it combines *many* individual Decision Tree models to improve the algorithm's performance. Essentially, the Random Forest classifier basically runs lots of decision trees, called 'estimators', and outputs the final prediction based on a 'majority vote' on the labels returned by the individual decision trees. In this case, the album which the majority of the individual decision trees output for a song or set of features will thus constitute the final prediction. Random Forest algorithms have been renowned in the literature for being excellent at predicting musical genres, with one research paper reporting 90% accuracy (Chaudhury, Karami, & Ghazanfar, 2022). Random Forests are known for having several advantages over simpler decision trees such as greatly reducing the risk of overfitting, and removing the need for cross-validation due to each individual decision tree in the 'forest' training on a slightly different combination of samples. However, the theory behind Random Forest algorithms can initially be daunting and difficult to understand. I will try to give an overview of it here.

The reason that Random Forests reduce the problem of overfitting without relying on cross-validation is by using a technique called *bootstrapping* during which subsets of the original dataset are formed, each of the same size as the original set, but with different combinations of samples (thus some samples might be duplicated, while others are left out completely from a subset). Consequently, every decision tree model inside the Random Forest is trained on a different one of these subsets of data. This reduces the risk of overfitting to a particular set of training data, as the training set is different for each separate decision tree. Secondly, the performance of the models is actually sequentially improved after the completion of training each individual decision tree using a technique called *boosting*. Each tree tries to learn from the errors of the previous tree. Which features should be 'split upon' for the first 'root' node of each tree is decided by comparing the

weighted Gini Impurity (see above in the explanation for decision trees) for splitting the first node on all the features in the dataset. As well as evaluating the Random Forest classifier on a separate test set like the other algorithms, scikit-learn also provides a property of *RandomForestClassifier* called 'oob_score', or 'out-of-bag' score which evaluates the model on the samples that were not included in the bootstrapped subset used for training. Out-of-bag samples basically provide a natural validation-set on which the efficacy of the individual decision trees in the Random Forest can be tested (Bag, 2022). The OOB score counts the number of correctly predicted samples for this out-of-bag test set. Importantly, none of these out-of-bag samples are ever seen by the weaker decision trees in the ensemble during training, thus accounting for the low incidence of overfitting when using Random Forests (Bag, 2022).

Although Random Forests are more robust than decision trees through this maximum utilization of data to reduce overfitting, one disadvantage of them is that they are slower and not as computationally efficient due to the requirement of training all the sub-trees. However, as the data size in this study is limited to only 530 samples in total, this should not pose a significant problem in this context.

As such, the final algorithm applied to the song data in this comparative analysis of the performance of different supervised learning techniques on album classification will be a Random Forest Classifier. This will hopefully result in optimized performance compared to the individual decision tree classifier. In order to select the optimal hyperparameters, the scikit-learn GridSearch algorithm will also be applied here.

Methodology

Overview

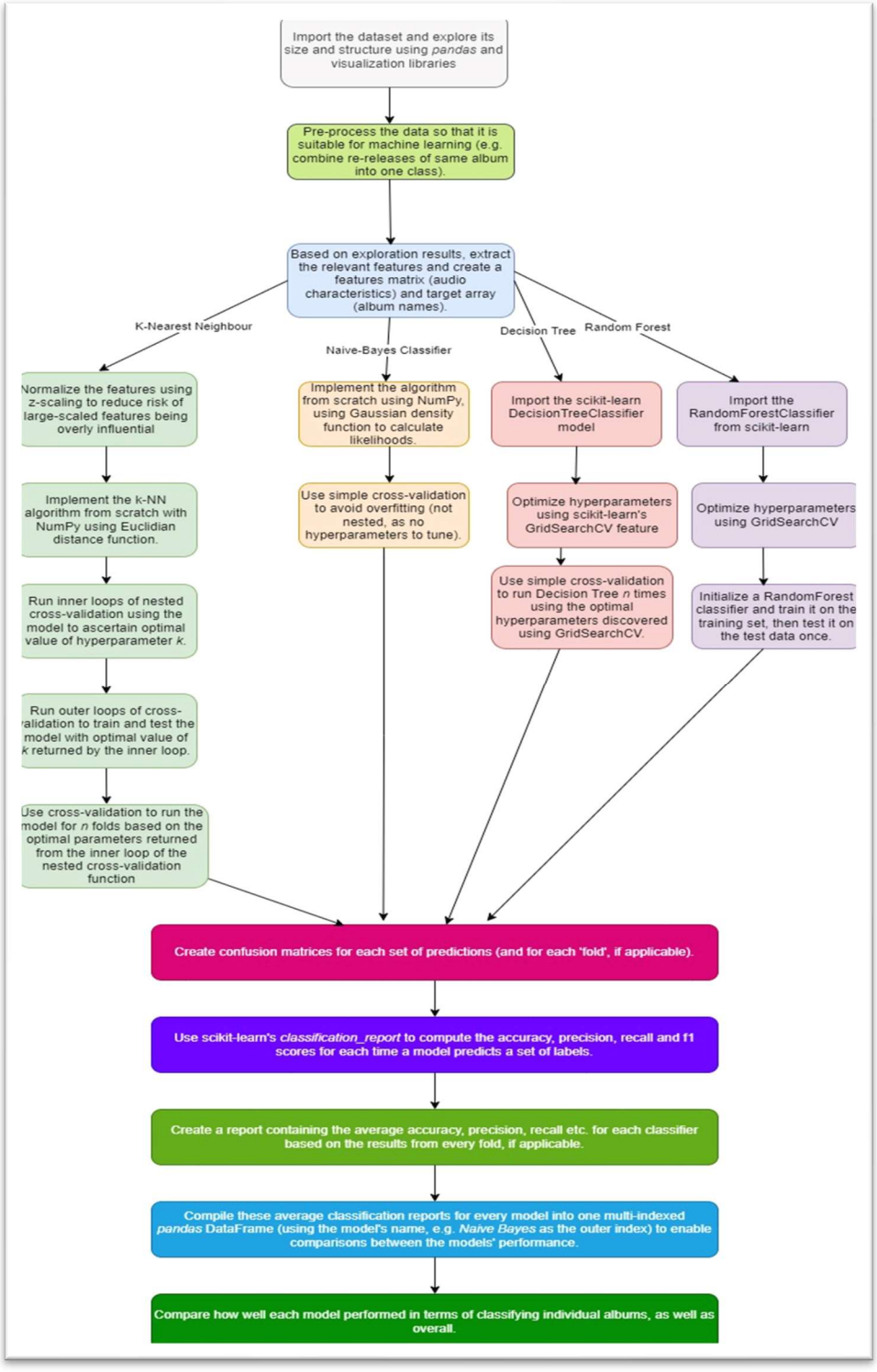

Figure 4: Diagram showing steps in the methodology used for this analysis

Pre-Processing

Overall, the quality of this dataset was clean and well-formatted with no missing values or inconsistent data types. However, one problem with the dataset was that it contained some albums which were re-releases of the same album. As such, it could be surmised that re-releases of the same album will be very similar in terms of their musical attributes, such as energy or 'acoutisticness'. It would therefore be more difficult for the classification algorithms to categorize the songs into the separate re-releases of the same album, as the sound qualities would be so difficult to distinguish. Therefore, the labels for the re-releases were adjusted using *pandas* to ensure that the album names were all the same. The *Live from Clear Channel Stripped* album, which comprises a series of live recordings of songs from other albums, was kept as separate in the final dataset as it would be interesting to explore whether these algorithms can differentiate live from non-live music. While placing re-releases of albums into one group solves the problem of overly-similar albums, it introduces another more substantial challenge. The resulting dataset was imbalanced, meaning that several albums had many more tracks associated with them than others. Attempts to address this issue will be discussed in more detail in the following sections.

Feature Selection

Seaborn's *FacetGrid* feature was used to plot the distributions of each of the audio features using a kernel density plot. This visualization showed that two of the audio features, 'speechiness' and 'instrumentalness' did not vary at all across the different albums. Therefore, these features were not added to the features matrix used to train the models, as they would not help the models distinguish between the various album classes. Next, a Seaborn *PairPlot* was used to visually summarize the relationships between the remaining features using scatterplots, in order to check if any of the variables had strong correlations with one another. This would introduce redundancy into the models, as two closely related variables do not really provide any new information (Khanna, 2020). However, none of the features seemed to be strongly correlated with another, therefore the decision was made to retain seven independent audio variables consisting of *acousticness, danceability, energy, liveness, loudness, tempo* and *valence*.

Data Scaling

As one research article has indicated (Pagan, Zarlis, & Candra, 2023), scaling data using 'z-scores' normalizes data points by subtracting the mean (for that feature) from each data point and dividing by the standard deviation. The advantage of normalizing the values prior to running k-Nearest Neighbor is firstly that it 'reduces the effect of the distribution outliers' (Pagan, Zarlis, & Candra, 2023) and secondly, that due to k-NN being a distance-based classification algorithm, this ensures that features which have a larger scale of values do not become overly important when selecting the *k* most similar neighbors. According to this article, 'Z-score scaling was the most efficient scaling technique' out of a range of scaling methods tested on k-Nearest Neighbor (Pagan, Zarlis, & Candra, 2023). Having different features, some of which have a very small scale, and some of which have a large scale, would mean that the Euclidian distance mechanic for evaluating sample closeness would disproportionately reflect the influence of the larger-scaled features. As a result, using z-scores for k-NN means that each feature's importance is assessed equally. The other algorithms explored in this study, such as Naïve-Bayes and Decision Trees, are not distanced-based algorithms and thus not as affected by inconsistent scaling as k-Nearest Neighbor. Therefore, the original feature values will be retained for these algorithms.

Preventing Overfitting: N-Fold Cross-Validation, Nested N-Fold Cross Validation and *scikit-learn*'s GridSearch

Cross-validation is a technique where the model is iteratively trained and tested on different subsets of the sample. Each subset of the data is called a 'fold'. In simple cross-validation, the dataset is just split into different groups of train-test samples every time, and the model is tested on these different groups, before averaging out the scores for accuracy, precision or other metrics. This method ensures that the results of the algorithm are 'reliable and consistent'. (Pagan, Zarlis, & Candra, 2023) Importantly, k-fold cross-validation is used to prevent the overfitting of data, especially when using smaller datasets such as this one. (Charilaou & Battat, 2022) Overfitting refers to a situation in which the model performs extremely well on the training data, but fails to make accurate predictions for the test data. This can happen when the model learns every detail of the training data too well, memorizing 'its noise and outliers' (Awan, 2023), but does not grasp the most important underlying patterns and structure in the data.

As mentioned previously a problematic characteristic of this dataset is its imbalanced nature, where some albums are overrepresented. Only using one train-test split could easily lead to the situation where certain albums are not represented at all in the training samples and are only included in the test samples. Therefore, it would be expected that the algorithm would not be able to learn anything about the features of these classes and would perform poorly on the test set. As such, cross-validation is essential for ensuring that all classes actually appear in the training data. Moreover, as the total size of the dataset used here is not particularly large (530 songs), using cross-validation ensures that no data is wasted or omitted from the training sets.

For the k-Nearest Neighbour algorithm, the decision was made not only to use simple cross-validation, but to use *nested* cross-validation whereby the training set for each iteration or 'fold' is further divided into a training and validation set, prior to being tested on the original test set. This means that for each outer 'fold' or iteration, within the inner loop, the classifier is run several times using different *hyperparameters*, and evaluated to select the best hyperparameter to use on the test set. For k-NN, there were two possible hyperparameters one could test in this way to find the optimal values: the distance metric used and the value of *k*. Here, we will focus on iterating over different values of *k* to select the one where the model performs best for this dataset. The metric that will be used to decide the optimal value of *k* will not be accuracy, but the f1 score, which is the harmonic mean of both precision and recall. While accuracy is a more intuitive metric (the number of all correct classifications divided by the total number of samples), precision and recall are said to be better evaluative measures when it comes to imbalanced datasets such as this one (Kanstrén, 2020). Accuracy is a notoriously poor measure for evaluating the performance of algorithms on imbalanced data sets (Uddin, 2019). As several of the albums are so heavily overrepresented, the model could potentially attain a very high accuracy score simply by predicting the most common albums most of the time.

With respect to the Naïve Bayes classifier, this algorithm does not take in any hyperparameters. As a result, simple cross-validation without an inner loop will suffice to evaluate the performance of this model.

For the Decision Tree, the different hyperparameter combinations one would need to test with nested cross-validation would lead to so many nested loops to account for all the combinations of values, that the decision was made to use *scikit-learn*'s inbuilt *GridSearchCV* feature, which iterates through thousands to tens-of-thousands of hyperparameter combinations to output the optimal ones. This method will also be used to select the best-performing hyperparameters for the Random Forest algorithm. While single-loop cross-validation *will* be used for the simple Decision Tree model once the optimal hyperparameters will be selected, it will not be used on the Random Forest algorithm. This is because cross-validation can be considered to be 'built into' the Random Forest model, as it is an ensemble of many individual decision trees on which different combinations of the training set are tested. As has been stated in one article, Random Forests are generally 'less likely to overfit than other models' because of being made up of many weaker classifiers trained on different sets of samples. (Ellis, 2021)

Evaluation Metrics: Confusion Matrices, Accuracy and F1 Scores

As touched on above, accuracy is not always the best measure for evaluating a model's performance despite being the most intuitive. As such, in addition to accuracy, the models' performance will also be measured in terms of *precision*, which is defined as the ratio of true positives divided by true positives plus false positives, and *recall*. Recall is calculated by dividing the number of true positives by true positives plus false negatives. More intuitively, precision can be thought of as the likelihood or reliability that a positive classification is actually correct, while recall measures what proportion of a class (i.e. an album) was recognized and classified correctly. The f1-score is known as the 'harmonic mean' of the precision and recall score and basically combines them with the following formula:

*F1 = (2 * Precision * Recall) / (Precision + Recall)*

(Brownlee, 2020)

These scores will be calculated per model for each *n-th* fold in the cross-validation process and summarized using the scikit-learn *classification_report* utility. These reports will then be combined into a table using *pandas* DataFrame and averages of the scores will be computed for each type of algorithm. This table will thus facilitate easy comparison between the performance of the different models.

Additionally, for each model, confusion matrices will be calculated and *visualized* using the scikit-learn ConfusionMatrixDisplay property, which uses a heat map to show the performance of the model on each class. A confusion matrix is a *n x n* matrix where *n* is the number of classes (albums). The diagonal represents the number of correctly-predicted samples (Kundu, 2022), while the off-diagonal values represent incorrectly-identified labels (Chaudhury, Karami, & Ghazanfar, 2022). As can be seen in the Jupyter Notebook, this technique makes it easy to cohesively portray which albums had the greatest recall and which were less likely to be correctly distinguished by the algorithm. Furthermore, the confusion matrix makes it very easy to intuitively grasp which albums were often 'confused' with each other, e.g. whether the algorithm found it difficult to distinguish between the 'folklore' and 'evermore' albums.

Overall, this methodological approach aims to address some of the problems in this dataset such as the imbalanced nature of the data, the inclusion of various album re-releases consisting of similar song samples, and how to reduce the risk of overfitting. Moreover, an explanation of the metrics used to evaluate the classifiers has been touched upon. The following section will look at the results of using these techniques on this dataset and attempt to evaluate how successful the models were in answering the key research aims outlined in the introduction.

Results

The following tables display the average precision, recall and f1 scores for each classifier's performance on each class (album), as well as the model's accuracy score, as the mean for all *n* folds used in the cross-validation:

model	album	precision	recall	f1-score	support
k-NN	1989	0.691312	0.750247	0.711547	14.400000
	Fearless	0.483077	0.513575	0.492092	12.200000
	Live From Clear Channel Stripped 2008	0.100000	0.100000	0.100000	1.600000
	Lover	0.250000	0.201905	0.207792	4.200000
	Midnights	0.713846	0.820513	0.756777	9.200000
	Red	0.598350	0.634839	0.604687	15.200000
	Speak Now	0.761386	0.634957	0.667866	11.200000
	Taylor Swift	0.000000	0.000000	0.000000	3.400000
	accuracy	0.579245	0.579245	0.579245	0.579245
	evermore	0.784740	0.771111	0.758291	7.200000
	folklore	0.718494	0.781962	0.746243	15.200000
	macro avg	0.475872	0.485440	0.470015	106.000000
	reputation	0.133389	0.130736	0.124868	12.200000
	weighted avg	0.571668	0.579245	0.564268	106.000000
Naive Bayes	1989	0.404603	0.226916	0.279269	15.600000
	Fearless	0.173473	0.200641	0.184174	11.400000
	Live From Clear Channel Stripped 2008	0.000000	0.000000	0.000000	1.500000
	Lover	0.200000	0.040000	0.066667	4.000000
	Midnights	0.411111	0.280495	0.318579	11.200000
	Red	0.216249	0.430758	0.276332	13.000000
	Speak Now	0.388974	0.308333	0.329535	12.800000
	Taylor Swift	0.000000	0.000000	0.000000	3.000000
	accuracy	0.300000	0.300000	0.300000	0.300000
	evermore	0.558756	0.639286	0.564132	7.000000
	folklore	0.331824	0.520821	0.390403	14.200000
	macro avg	0.269617	0.257997	0.236379	106.000000
	reputation	0.224762	0.141310	0.145453	12.600000
	weighted avg	0.323557	0.300000	0.280402	106.000000
Decision Tree	1989	0.194632	0.357628	0.239127	15.600000
	Fearless	0.028571	0.033333	0.030769	11.600000
	Live From Clear Channel Stripped 2008	0.000000	0.000000	0.000000	1.800000
	Lover	0.000000	0.000000	0.000000	3.400000
	Midnights	0.359380	0.416468	0.356146	12.600000
	Red	0.155641	0.272456	0.170491	13.800000
	Speak Now	0.289615	0.487548	0.359623	13.200000
	Taylor Swift	0.000000	0.000000	0.000000	3.200000
	accuracy	0.298113	0.298113	0.298113	0.298113
	evermore	0.000000	0.000000	0.000000	6.000000
	folklore	0.387745	0.801129	0.520876	13.200000
	macro avg	0.128689	0.215324	0.152458	106.000000
	reputation	0.000000	0.000000	0.000000	11.600000
	weighted avg	0.182797	0.298113	0.214416	106.000000

Random Forest	1989	0.600000	0.800000	0.685714	15.000000
	Fearless	0.500000	0.285714	0.363636	14.000000
	Live From Clear Channel Stripped 2008	0.000000	0.000000	0.000000	0.000000
	Lover	0.000000	0.000000	0.000000	5.000000
	Midnights	0.555556	0.500000	0.526316	10.000000
	Red	0.454545	0.500000	0.476190	10.000000
	Speak Now	0.444444	0.727273	0.551724	11.000000
	Taylor Swift	0.000000	0.000000	0.000000	4.000000
	evermore	1.000000	0.833333	0.909091	6.000000
	folklore	0.666667	1.000000	0.800000	14.000000
	reputation	0.166667	0.117647	0.137931	17.000000
	micro avg	0.518868	0.518868	0.518868	106.000000
	macro avg	0.398898	0.433088	0.404600	106.000000
	weighted avg	0.463741	0.518868	0.476132	106.000000

Figure 5: Results tables for all four classification algorithms

```
# get accuracy score for random forest
from sklearn.metrics import accuracy_score
rf_acc = accuracy_score(y_test,rf_y_pred)
print(rf_acc)
```

```
0.5188679245283019
```

Figure 6: Accuracy score for Random Forest Classifier algorithm

```
# Print 'out-of-bag' score ('The OOB score is the number of correctly predicted data on OOB samples taken for validation.')
# Basically the oob score is the validation/test-set score the collection of decision trees output after testing their predictions
# on the data points not included in the 'training set' or bootstrapping samples
rf_classifier_oob_score = rf_classifier.oob_score_
print(rf_classifier_oob_score)
```

```
0.5849056603773585
```

Figure 7: Out-of-Bag score for Random Forest classifier: 0.58 score on the natural 'validation set' inside this classifier means that the algorithm performed better than average when predicting the 11 album classes overall.

As can be seen in these tables, the weighted k-NN classifier achieved the best performance in terms of accuracy, macro (or unweighted mean) precision and recall, which were all scored at 0.58 to two decimal places. The decision tree and Naïve Bayes classifier performed very poorly with scores of about 0.3 for accuracy and even lower scores for precision and recall. The unweighted average f-1 score for the Decision Tree was as low as 0.15, implying that the algorithm did not perform any better than randomly guessing each output.

Consequently, these results imply **that the weighted k-NN classifier implemented in the Jupyter Notebook using *NumPy* was the most well-suited algorithm for classifying this type of data**, as it achieved almost 0.6 (to 1 d.p.) accuracy. This was closely followed by the Random Forest model. Additionally, the k-NN algorithm obtained similar **weighted averages** (which take into consideration the number of songs in that album) for precision and recall. This was unexpected, particularly given the more complex techniques used to recognize patterns in a model like Random Forest.

The core aim of this project was to analyze which albums or classes had the highest recall (the measure of how many songs in an album were identified as belonging to that album). In relation to this question, there was a surprising degree of consistency across the different models as to which albums were better recognized than others, even in the weaker models. The albums '1989', 'Midnights', 'Red', 'Speak Now', 'folklore' and 'evermore' received very high recall scores compared to the other albums. For instance, 'folklore' achieved a recall score of 1.0 using Random Forest and even 0.8 using the poorly-performing Decision Tree. In contrast, the albums 'Lover', 'reputation', the debut 'Taylor Swift' album and the album containing live sessions received such low precision and recall scores that this would have had a substantial impact on dampening down the average performance scores of the models. This indicates that these albums do not have as much of a distinct style musically as the previously mentioned ones.

Furthermore, as the confusion matrices displayed in the Jupyter Notebook reveal, there was considerable overlap between the classification of the 'evermore' and 'folklore' albums. As an example, here is the confusion matrix outputted after applying the Random Forest algorithm to the test data:

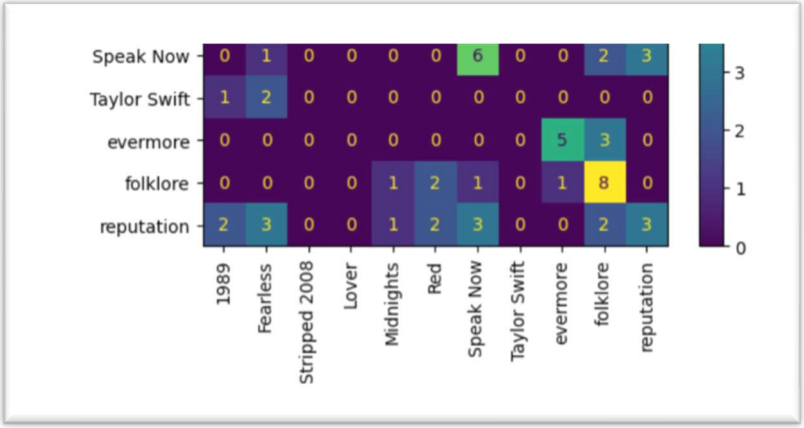

Figure 8: Part of the confusion Matrix for Random Forest Classifier to show overlap between 'folklore' and 'evermore' albums.

This snippet of the confusion matrix outputted for one of the k-NN folds (but this result was also repeated throughout the other folds) clearly indicates that out of 8 songs on 'evermore' in the test sample, 3 were categorized wrongly as 'folklore', while one song out of 9 for 'folklore' was classed as 'evermore'. The Jupyter Notebook contains all of the confusion matrices for each model, most of which display this pattern. This suggests that the musical attributes of the 'folklore' and 'evermore' albums are similar, which is not surprising, as these albums were released shortly one after the other and showed a departure from Swift's previous more poppy style to a more acoustic, indie folk based genre.

Evaluation

In this section, the possible limitations of these algorithms will be addressed, as well as what could be done to improve their performance.

Issues related to Non-Normal Feature Distributions

First of all, in this study, a basic Gaussian Naïve-Bayes classifier was implemented which relied on the Gaussian density function to predict the likelihood of each feature, X, (e.g. energy) given an album class, Y. However, the Gaussian density function assumes that the independent variables or features, X, are normally distributed, while not all of the features used for this assignment followed this Gaussian distribution. Much research has been conducted so far on how Naïve Bayes classification can be improved on features which are *not* normally distributed (Soria, Garibaldi, Ambrogi, & Biganzoli, 2011). Soria, Garibaldi et al. have explored using different estimates for a feature's likelihood to overcome these limitations of the Gaussian density function, primarily a *kernel density estimation*, where the 'density of each continuous variable is estimated averaging over a large set of kernels'. Kernel density estimation is known as an example of 'non-parametric' estimation which means that no assumptions are made about the variables' distributions. (Aptech.com, 2023) It is known to perform better on asymmetric and skewed distributions, and therefore might improve inferring the likelihood for the different features. Another method to address the limitations of the Naïve Bayes classifier may consist of further pre-processing of the features values to remove outliers and therefore to remove noise from data (Singh & Singh, 2020). Furthermore, a well-known alternative technique for making the distribution look more normal is taking the natural logarithm of the values (Brownlee, 2019). A more complex for transforming the dataset include the BoxCox method, which uses a sequence of logarithm-taking and square root operations to force the distribution to appear closer to the Gaussian distribution.

Suggestions for Dealing with Imbalanced Datasets

Despite the merit in exploring these possible adaptations of the Naïve-Bayes classifier, a consideration of the results points to the explanation that the greatest limitation here is the imbalanced nature of this dataset, and the fact that certain albums have very few song instances compared to others. The following plot, constructed using seaborn, shows that there is a strong positive linear correlation between the *support* of an album class and its F1-score:

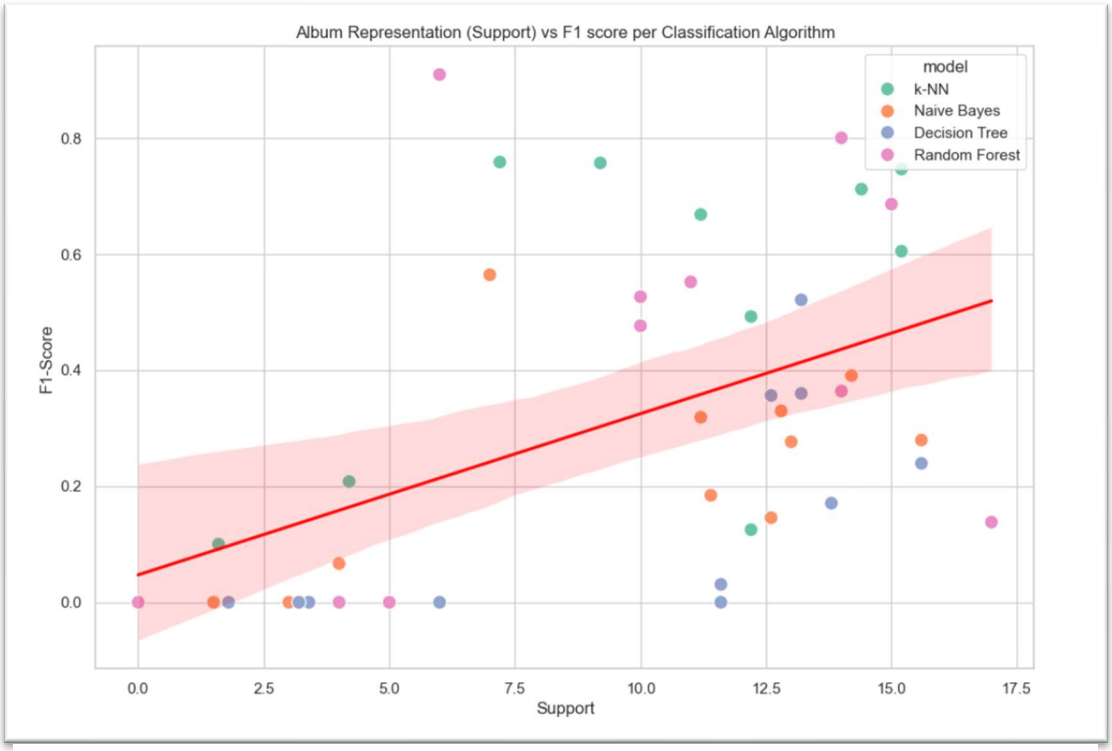

Figure 9: Plot showing the linear relationship between album class representation ('support') and its F1 score over all the algorithms

This plot demonstrates that the greater the support (representation of an album class in a sample), the greater the F1-Score (measuring the album's recall and precision). As such, songs on albums which have fewer samples such as the 'Taylor Swift' album or 'Lover' are less likely to be correctly classified. This leads to the most substantial issue with this data, which is that of not having enough data points for certain categories, as well as the dataset being so *imbalanced*. Much has been written for how to adapt classification models to imbalanced datasets, as these have been recognized as one of the greatest challenges in machine learning (Chou & Yang, 2022). Oversampling and undersampling are two methods. Oversampling refers to duplicating items from the less-represented classes (Pykes, 2020), while undersampling removes samples from the majority classes, and is easy to implement using scikit-learn's *RandomOverSampler* and *RandomUnderSampler* utilities. Consequently, if there were more time to work on this project, these techniques would be one of the first to be applied in order to explore whether the performance of the classifiers can be improved for the less-common albums. Another more advanced oversampling technique mentioned in many machine-learning papers is called *SMOTE*, or Synthetic Minority Over-Sampling Technique (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). Chawla et al. have shown that applying SMOTE to imbalanced classes leads to better performance than simply oversampling the minority classes with duplicates. SMOTE involves artificially generating synthetic instances of the minority (less-represented) classes. The *k* nearest neighbors are selected for each sample within this smaller class, before selecting a set of values for the new synthetic sample which have to lie on the plane between the real instance and its *k*-nearest neighbours inside the feature space. This is an avenue which provides many opportunities for future research, such as how best to select the 'synthetic neighbours' or focusing nearest neighbours on instances which are most likely to be incorrectly-classified (Chawla, Bowyer, Hall, & Kegelmeyer, 2002).

Unsupervised Algorithms for Detecting Non-Album Name Related Patterns

Alternatively, if there were more time to extend this project, the next step towards describing the musical styles of Swift's albums could be by implementing and applying an unsupervised clustering algorithm, such as K-means. As seen in the results, several albums were predicted with greater accuracy than others, while some like 'folklore' and 'evermore' seemed to form one 'cluster' of similar audio features. The advantage of using K-means clustering in this scenario would be to ascertain whether there are any more nuanced structures and patterns which are not easily discerned using supervised learning with album labels: for instance, K-means could be used to test the hypothesis of whether albums released in a similar *time period*, such as 'folklore' and 'evermore', shared more similar features than those released at greater intervals throughout Swift's career.

Feature Refinement

Chaudhury et al. reported excellent results in music genre classification using Random Forest classifiers achieving up to 90% accuracy (Chaudhury, Karami, & Ghazanfar, 2022). However, instead of using the simpler Spotify audio features, they used the GTZAN music dataset which considers more complex audio features such as spectral bandwidth, Chroma STFT and zero-crossing rate which require a greater degree of domain expertise to understand. Investing time into understanding these more refined features would be a valuable addition to understanding of music classification tasks. Moreover, there have been some successful projects using Naïve-Bayes for songs' genre classification using song *lyrics* rather than audio features (Rutter, 2021). There is a possibility that the different albums *do* have their own, distinct styles but that this is based more on the content of the lyrics rather than on the numerical audio features, therefore it cannot be fully concluded that albums such as 'reputation' or 'Lover' do not have their own style, as this style could be related more to textual themes than acousticness or loudness.

Presence of Outliers

Decision Trees performed particularly badly on the classification task in this study, obtaining only 0.3 accuracy across all the classes. This could have been due to the presence of too much noise or outliers in the dataset, which is something that this model is particularly sensitive to (Explorium, 2023). As a result, further pre-processing of the data to visualize (using box-and-whisker plots) and handle outliers using interpolation techniques may improve the performance of this algorithm.

A Note on Computational Performance Issues

Finally, a brief comment should be made on the fact that it was not possible to evaluate a broader range of hyperparameters for the Decision Tree and Random Forest classifiers using grid search due to the available computational resources. The parameters that were optimized in this project, such as *max_depth* and *min_samples_leaf* already took many hours to tune using this technique. As such, if there were more computational resources available, it is possible that these models could be improved further by testing a wider range of values for and types of hyperparameters. For instance, the *max_features* and *min_samples_split* parameters were not optimized for Random Forest for this precise reason.

Conclusion

To summarize, the findings of this project were that the weighted k-Nearest Neighbours Classifier and Random Forest Classifier performed best when trying to predict the label of a Taylor Swift song based on its audio features, achieving accuracy scores of over 0.5. Nevertheless, this score was still lower than expected considering that in some of the literature reviewed above, accuracy scores of 0.9 were achieved while classifying music genres using the Random Forest algorithm (Chaudhury, Karami, & Ghazanfar, 2022). Some reasons for this could have been the imbalanced nature of the dataset and the inability of the models to classify the songs in the smaller album classes, due to the limited number of samples belonging to these albums. Overall, the albums with the fewest songs (samples) were the ones that were indeed more likely to be misclassified.

The Naïve Bayes and Decision Tree algorithms performed rather poorly on this dataset, failing to achieve accuracy scores of over 0.3, with precision and recall scores being even lower. With respect to Naïve Bayes, this poor result could be due to the variables not following a normal distribution, while for Decision Trees, this may likely be due to not removing noise or outliers from the dataset, a property of the dataset which greatly reduces the performance of this algorithm.

Certain Taylor Swift albums received very high precision and recall scores, while others routinely received scores close to 0 regardless of the classification algorithm used. The albums ‘1989’, ‘Midnights’, ‘Red’, ‘Speak Now’, ‘evermore’ and ‘folklore’ were associated with higher recall scores of 0.5 or above, which means that more instances of their songs were positively recognized. With respect to the aim of this project, these scores might suggest that those particular albums have a more distinctive audio quality to them. In contrast, the albums ‘Lover’ and ‘reputation’ received scores close to 0, with almost all of their songs being misclassified by every algorithm, thus suggesting that either their musical style is less distinctive than that of the other albums, or that there were simply too few samples for these albums to make these conclusions: it is important to note that while ‘Red’ or ‘Midnights’ have recently had extended re-releases, Swift fans are still waiting for a deluxe version of ‘reputation’ to come out. Additionally, songs on ‘folklore’ were frequently misclassified as being on ‘evermore’, and vice versa. This adds weight to the argument that Swift drastically changed her style during 2020 when she recorded these two ‘pandemic’ albums, marking a maturation of her musical oeuvre which brought her much success: as one critic writes, ‘When “evermore” and “folklore” were released in 2020, the world witnessed Swift, once again, mastering a genre switch, from pop to indie folk.’ (Hudgins, 2023).

Furthermore, this study could be enhanced by applying the various techniques listed above to address class imbalance, such as oversampling of smaller classes or SMOTE. Additionally, k-Means clustering could yield further insights into which groups of songs are more similar.

In conclusion, machine learning models can corroborate journalists and critics' claims that Swift's musical style evolved significantly during the most recent 'folklore', 'evermore', and 'Midnights' period of her career, as these albums were the ones which achieved very high recall scores compared to the others. This suggests that their musical characteristics are distinct from those of the previous albums.

References

Note: ***the websites referenced here were all functioning correctly and last accessed on the 2nd of January 2024.***

Analytics Vidhya. (2022). *Getting started with Decision Trees*. Retrieved from Analytics Vidhya: https://courses.analyticsvidhya.com/courses/getting-started-with-decision-trees

Aptech.com. (2023, January 17). *The Fundamentals of Kernel Density Estimation*. Retrieved from Aptech: https://www.aptech.com/blog/the-fundamentals-of-kernel-density-estimation/

Associated Press. (2023, December 8). *Taylor Swift's Eras tour becomes first to gross over $1bn – report*. Retrieved from The Guardian: https://www.theguardian.com/music/2023/dec/08/how-much-taylor-swift-eras-tour-money-earnings

Awan, A. A. (2023, August). *What is Overfitting?* Retrieved from Datacamp: https://www.datacamp.com/blog/what-is-overfitting

Bag, S. (2022, October 6). *Out of Bag (OOB) Evaluation in Random Forests*. Retrieved from Python in Plain English: https://python.plainenglish.io/out-of-bag-oob-evaluation-in-random-forests-9da315b9a1d1

Brownlee, J. (2019, August 8). *How to Transform Data to Better Fit The Normal Distribution*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/how-to-transform-data-to-fit-the-normal-distribution/

Brownlee, J. (2020, August 2). *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/#:~:text=We%20can%20calculate%20the%20precision%20as%20follows%3A,Precision%20%3D%200.633

Charilaou, P., & Battat, R. (2022). Machine learning models and over-fitting considerations. *World Journal of W J G Gastroenterology*, 605-607.

Chaudhury, M., Karami, A., & Ghazanfar, M. (2022). Large-Scale Music Genre Analysis and Classification Using Machine Learning with Apache Spark. *electronics*.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, P. W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* , 321-357.

Chou, E. P., & Yang, S.-P. (2022). A virtual multi-label approach to imbalanced data classification. *Communications in Statistics-Simulation and Computation*.

Dailey, H. (2023, December 5). *Taylor Swift Is Forbes' 5th Most Powerful Woman in the World*. Retrieved from Billboard.com: https://www.billboard.com/music/music-news/taylor-swift-most-powerful-woman-in-the-world-2023-list-1235534184/

Ellis, C. (2021, August 24). *Random Forest Overfitting*. Retrieved from Crunching the Data: https://crunchingthedata.com/random-forest-overfitting/#:~:text=In%20general%2C%20random%20forests%20are,subsets%20of%20the%20training%20data.

Explorium. (2023, August 6). *The Complete Guide to Decision Tree Analysis*. Retrieved from Explorium: https://www.explorium.ai/blog/machine-learning/the-complete-guide-to-decision-trees/#:~:text=The%20biggest%20issue%20of%20decision,it%20loses%20its%20generalization%20capabilities.

Gaines, C. (2023, September 17). *Taylor Swift helped lift the US economy this summer with her Eras Tour*. Retrieved from BusinessInsider: https://www.businessinsider.com/taylor-swift-eras-tour-helped-us-economy-2023-9?r=US&IR=T

Harrison, O. (2018, September 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Retrieved from Medium: Towards Data Science: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Hu, L.-Y., Huang, M.-W., Ke, S.-W., & Tsai, C.-F. (2016). The distance function effect on k-nearest neighbor classification for medical datasets. *Springerplus*.

Hudgins, K. (2023). *How Taylor Swift Masterminded Global Success, Explained by SOMD Experts*. Retrieved from University of Oregon School of Music and Dance: https://musicanddance.uoregon.edu/TaylorSwift

Jain, V. (2022, January 31). *Introduction to KNN Algorithms*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2022/01/introduction-to-knn-algorithms/

Kanstrén, T. (2020, September 11). *A Look at Precision, Recall, and F1-Score*. Retrieved from Medium: Towards Data Science: https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

Khanna, C. (2020, November 29). *Multicollinearity — Why is it bad?* Retrieved from Towards Data Science: https://towardsdatascience.com/multicollinearity-why-is-it-bad-5335030651bf

Kundu, R. (2022, September 13). *Confusion Matrix: How To Use It & Interpret Results [Examples]*. Retrieved from V7 Labs: https://www.v7labs.com/blog/confusion-matrix-guide

Maheshwari, H. (2021, October 13). *How to decide the perfect distance metric for your machine learning model*. Retrieved from Medium: Towards Data Science.

MojoCode. (2020, July 30). *Naive Bayes Classifier From Scratch*. Retrieved from MojoCode: https://jayellwolfe.github.io/2020-07-30-Naive-Bayes-From-Scratch/

Mu, X. (2023). Implementation of Music Genre Classifier Using KNN Algorithm. *Highlights in Science, Engineering and Technology*, 149-154.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* Cambridge, Massachusetts: MIT Press.

Ndou, N., Ritesh, A., & Jadhav, A. (2021). Music Genre Classification: A Review of Deep-Learning and Traditional Machine-Learning Approaches. *IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)* (pp. 1-6). Toronto, Canada: IEEE.

Pagan, M., Zarlis, M., & Candra, A. (2023). Investigating the impact of data scaling on the k-nearest neighbor algorithm. *Computer Science and Information Technologies*, 135-142.

Patidar, P. (2023, March 1). *Classification using Gaussian Naive Bayes from scratch*. Retrieved from Medium: Level Up Coding: https://levelup.gitconnected.com/classification-using-gaussian-naive-bayes-from-scratch-6b8ebe830266

Prasath, S. V., Alfeilat, H. A., Hassanat, A. B., Lasassmeh, O., Tarawneh, A. S., Alhasanat, M. B., & Salman, H. S. (2019). Effects of Distance Measure Choice on KNN Classifier Performance - A Review. *Big Data*, 221-248.

Priester, J. (2024, January 4). *Taylor Swift Spotify Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/jarredpriester/taylor-swift-spotify-dataset/data

Pykes, K. (2020, September 10). *Oversampling and Undersampling*. Retrieved from Medium: Towards Data Science: https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf#:~:text=In%20other%20words%2C%20Both%20oversampling,taken%20(Source%3A%20Wikipedia).

Rutter, B. (2021, February 27). *Naive Bayes, Song Lyrics and Genre*. Retrieved from Medium: Towards Data Science: https://towardsdatascience.com/i-built-a-naive-bayes-model-to-predict-genre-from-song-lyrics-and-it-went-ok-ish-639af0b0a078

Saini, A. (2022, August 26). *An Introduction to Random Forest Algorithm for beginners*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/

Saini, A. (2023, September 13). *Decision Tree Algorithm – A Complete Guide*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/#What_is_a_Decision_Tree

Saini, B. (2020, September 29). *Hyperparameter Tuning of Decision Tree Classifier Using GridSearchCV*. Retrieved from PlainEnglish.io: https://plainenglish.io/blog/hyperparameter-tuning-of-decision-tree-classifier-using-gridsearchcv-2a6ebcaffeda#how-does-it-work

Sianipar, A. (2019, July 25). *Predicting a Song's Genre Using Natural Language Processing*. Retrieved from Medium: Better Programming: https://betterprogramming.pub/predicting-a-songs-genre-using-natural-language-processing-7b354ed5bd80

Siddique, A. (2023, May 3). *Exploring KNN with Different Distance Metrics*. Retrieved from Medium: Dev Genius: https://blog.devgenius.io/exploring-knn-with-different-distance-metrics-85aea1e8299#:~:text=In%20conclusion%2C%20the%20best%20KNN,0.982456%20with%20k%3D11%20neighbors

Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification. *Applied Soft Computing Journal*, 1-23.

Soria, D., Garibaldi, J. M., Ambrogi, F., & Biganzoli, E. M. (2011). A ‘non-parametric’ version of the naive Bayes classifier. *Knowledge-Based Systems*, 775-784.

Takahashi, K. (2016, January 6). *K-Nearest Neighbor from Scratch in Python*. Retrieved from Kenzo's Blog: https://kenzotakahashi.github.io/k-nearest-neighbor-from-scratch-in-python.html

Thomas, N. (2020, May 9). *Using k-nearest neighbours to predict the genre of Spotify tracks*. Retrieved from Medium: Towards Data Science: https://towardsdatascience.com/using-k-nearest-neighbours-to-predict-the-genre-of-spotify-tracks-796bbbad619f

Uddin, M. F. (2019). Addressing Accuracy Paradox Using Enhanched. *2019 Sixth HCT Information Technology Trends (ITT)* (pp. 319-324). Ras Al Khaimah, United Arab Emirates: IEEE.