Spotify Modality Predictor

Prathamesh Sai
saisankp@tcd.ie
19314123

Eligijus Skersonas
skersone@tcd.ie
19335661

I. Introduction

As a musician, knowing the modality of a song (whether a song is written in a *major* or *minor* key) is extremely important. The modality of a song tells us a lot about it - for example, songs written in a major key are identified as happy meanwhile songs written in a minor key are identified as sad in western culture. This concept permeates into other areas of life, such as natural language when animals speak. Currently, when a Spotify artist uploads their music to Spotify, they have to manually input the modality of their music. We thought, *what if this process was automatic?* - It could help Spotify, Spotify artists, as well as Spotify users. It would help streamline the process of uploading music for Spotify artists, improve Spotify's song uploading process, and allow Spotify to use these predictions to group songs together to recommend to users based on their taste in music. To implement this, we can take a Spotify song's audio features as the input. Then, we could use a kNN and kernalised SVM classifier to output a predicted modality ($\hat{y}$) as ($\hat{y} = +1$) for a major key and ($\hat{y} = -1$) for a minor key. While songs vary hugely, a song's audio features (such as key or valence) correspond nicely to the modality of a song, therefore the input would very much be able to provide the required output to predict modality.

II. Dataset and Features

A. *Dataset*

To get good predictions, our models must be trained with songs from various genres with varying modality. We used the Spotify API to scrape 5700 songs from Spotify's collection of 80 million songs. We used the Spotify API's "search" functionality to get the first 50 search results (i.e. songs) from searching through 114 different genres (every possible searchable genre on Spotify) which resulted in 5700 songs. To extract the features from each song, we used the Spotify API's "audio features" functionality which returns 18 values about the song such as *acousticness, analysis_url, danceability, duration_ms, energy, id, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time_signature, track_href, type, uri, and valence.* It is evident that there are many values returned from the API which have nothing to do with modality, so before we stored the returned data into a CSV file, we did some pre-processing. Pre-processing consisted of removing values from the Spotify API such as *"type"* (type of the data which always returns "audio_features"), *"analysis_url"* (which is a URL to get an audio analysis of the track), *"uri"* (which is the Spotify URI for the track) and *"track_href"* (which is a URL to the Spotify API endpoint for full details of the track). These values served no purpose for the actual features of the song, so these had to be discarded. Removing these 5 unrelated values from the 18 values originally returned by the API for each song's audio features left us with 13 values to work with (12 values for features, and 1 value for the modality). Also, we used the *"id"* (the Spotify song id) to add the Spotify link to the track for clarity when we are browsing the dataset (not for model training purposes). Also, the mode returned by the API's search results was either a 1 or 0 for major or minor respectively. I changed this to either a 1 (major) or -1 (minor) for clarity purposes as it is more understandable. An example of a datapoint in our dataset is:

Spotify-Song	X1	X2	X3
https://open.spotify.com/track/5vjLSffimiIP26QG5WcN2K	0.618	0.443	2

X4	X5	X6	X7	X8	X9	X10	X11	X12	y
-9.681	0.0526	0.469	0	0.0829	119.949	0.167	198853	4	1

In each datapoint, X1 is *danceability* (a measure from 0 to 1), X2 is *energy* (a measure from 0 to 1), X3, is *key* (using standard pitch class notation from -1 to 11), X4 is *loudness* (using decibels), X5 is *speechiness* (a measure from 0 to 1), X6 is *acousticness* (a measure from 0 to 1), X7 is *instrumentalness* (a measure from 0 to 1), X8 is *liveness* (a measure from 0 to 1), X9 is *tempo* (using beats per minute), X10 is *valence* (a measure from 0 to 1), X11 is *duration_ms* (using milliseconds), and X12 is *time_signature* (a measure from 3 to 7 representing how many beats in each bar). Now, our dataset is filled with 5700 varied songs, and we have removed any unnecessary data with pre-processing. Next, we need to do feature engineering to select the appropriate features to use.

B. *Features*

To select the best features, we need to use as few features as possible to prevent overfitting. We also want to prevent using too few features to prevent underfitting. We can use these two steps:

1) Find features that are actually dependent on the target value (modality).
2) Find out which combination of features gives us the best accuracy.

Step 1: We plotted each of the 12 features against y with the first 20 datapoints with 10 major songs and 10 minor songs instead of 5700 songs as the plot would be too crowded to identify any dependencies otherwise. We can plot this to see if there are any logical changes in the output (target value) for different ranges of values in the features. In short, if there are plots with grouped points for +1 (major) and -1 (minor) in the same range for the feature value, there is no dependence for that feature value and modality.

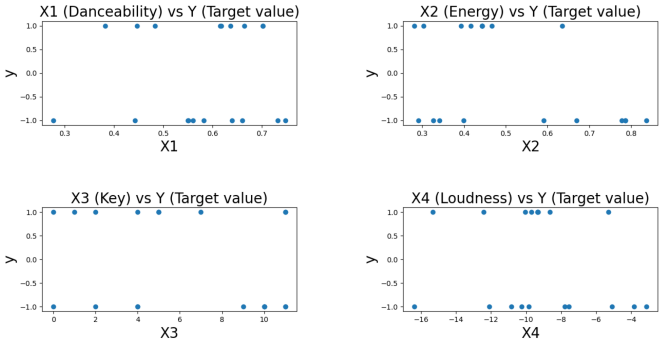

Fig. 1. Analyzing the dependence of features 1-4 against y.

We can see above that the target value on the y-axis (+1 or -1) changes distinctly for different ranges of X1 on the x-axis. For example, when $0.5 \leq X1 \leq 0.6$, it is more likely to result in $y = -1$. There seems to a logical dependence for features X2, X3 and X4 as well. For example when $0.4 \leq X2 \leq 0.5$, it is more likely to result in $y = +1$, when $4 \leq X3 \leq 8$, it is more likely to result in $y = +1$ and when $-8 \leq X4 \leq -4$, it is more likely to result in $y = -1$.

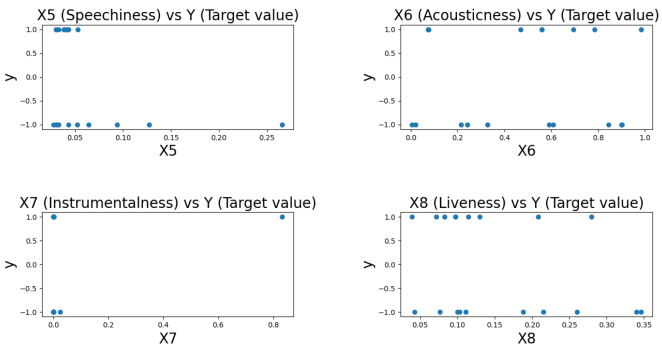

Fig. 2. Analyzing the dependence of features 4-8 against y.

We can see that feature X5 and X7 are not dependent on modality. This can be said because feature X5's y-values are grouped too close to each other when $0 \leq X5 \leq 0.06$. Also, feature X7's y-values are grouped too close to each other when $X7 \approx 0$. Feature X6 and X8 are more dependent on modality as they have ranges that make it more likely to predict +1 or -1. For example, when $0.2 \leq X6 \leq 0.4$ or when $X8 \geq 0.3$, it is more likely to predict $y = -1$.

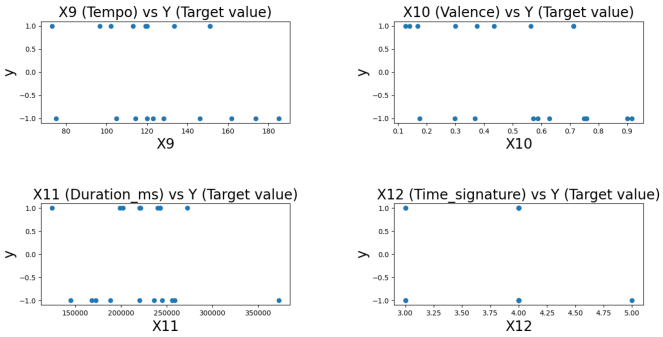

Fig. 3. Analyzing the dependence of features 9-12 against y.

Above it is evident that feature X11 and X12 are not dependent on modality. This is because when $200000 \leq X11 \leq 250000$, the y-values (target values) are almost random. Also when $3 \leq X12 \leq 4$, the y-values are also random and are grouped together showing no logical decision being made for any particular range of X12. This makes sense, as the duration and time signature of a song can vary so much that it doesn't tell much about the modality of a song. However, X9 and X10 are dependent on modality since we can see when $140 \leq X9 \leq 180$ or $0.6 \leq X10 \leq 0.9$, it is more likely to predict $y = -1$. Therefore, by analyzing the dependence of all features, we know that *features X1, X2, X3, X4, X6, X8, X9, X10 are dependent on modality.*

Step 2: We can get a list containing every combination of features X1 to X12 and train a kernalised SVM and kNN classifier with each index to see which combination of features yields the highest accuracy.

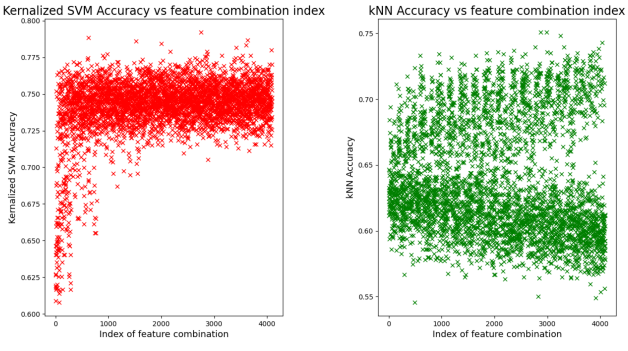

Fig. 4. Index of list of all combinations of features vs accuracy.

There are 4095 different combinations of features X1 to X12, and index 2877 (*X1, X3, X6, X8, X9, X10, X12*) yielded the highest kNN accuracy of 0.75 while index 2751 (*X1, X2, X6, X8, X9, X10, X12*) yielded the highest kernalised SVM accuracy of 0.79. These results make sense, as these features are very similar to the features that we found depend on modality in step 1. The only difference between index 2877 and index 2751 is that one uses X3 and the other uses X2 as the 2nd feature. To figure out which is best, I tried the combination of features from index 2877 and 2751 over 10 repetitions. Using X3 gave an average accuracy of 72.7% for kNN and 76.1% for kernalised SVM, using X2 gave an average accuracy of 67.7% for kNN and 75.4% for kernalised SVM, and using both gave an accuracy of 72.8% for kNN and 75% for kernalised SVM. In conclusion, using index 2877 (X1, X3, X6, X8, X9, X10, X12) has the best accuracy, and combining this with step 1 where we found that X12 is not dependent on modality results in a feature selection of *X1, X3, X6, X8, X9, X10*.

III. Methods

In this project, we use a kNN (k-nearest neighbours) classifier as well as a kernalised SVM (support vector machine) classifier to solve the classification problem of predicting modality.

A. *kNN (k-nearest neighbours)*

The kNN algorithm is an instance based model meaning it makes predictions directly using the training data (represented by $(x^{(i)}, y^{(i)})$ for $i = 1, 2, ..., 5700$ for our dataset). It uses a distance metric $d(x^{(i)}, x)$, which represents the distance between a training data point $x^{(i)}$ and input feature vector x, to to get the k-nearest neighbours (training data points closest to x) by choosing the smallest distances ($d(x^{(i)}, x)$). You can change hyperparameter k to choose the number of neighbours to use. You can also apply a weighting for the neighbouring points, where they can be uniform (equal) weights ($w^{(i)} = 1$) or Gaussian weights where you give less of a weight to training points that are further away from x ($w^{(i)} = e^{-\gamma d(x^{(i)},x)^2}$). Finally, the output for classification problems like ours is calculated by using a majority vote when using uniform weights ($w^{(i)} = 1$) or otherwise aggregating the k-neighbour points N_k using this sign function:

$$sign(\frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}})$$

.

If your model suffers from overfitting/underfitting, you can increase/decrease k respectively to solve this. Decreasing k smooths out the function but could eventually lead to underfitting. Increasing k will make the data fit more precisely but could eventually lead to overfitting. Another potential solution if using Gaussian weights is to decrease γ to smooth out a function to help mitigate overfitting (but may eventually cause underfitting) or increase γ to make the function rougher to mitigate underfitting (but may eventually cause overfitting). Therefore, choosing k and γ with cross-validation is the best option.

B. *Kernalised SVM (support vector machine)*

Kernalised models use a linear model such as SVM, after associating a feature with each training data point. Every feature i is calculated with the function $y^{(i)}K(x^{(i)}, x)$ using a kernel $K(x^{(i)}, x)$ which measures the distance between input x and training data point $x^{(i)}$. This means that when x and the training data point $x^{(i)}$ are far away, K will tend towards 0, and if they are close K will tend towards 1. The kernalised SVM we used uses a Gaussian kernel where $K(x^{(i)}, x) = e^{-\gamma d(x^{(i)},x)^2}$, hence when distance d is big $e^{(x)}$ tends to 0 and when the distance d is 0, $e^{(x)}$ tends to 1 (since $e^{(0)} = 1$). The output will be:

$$\hat{y} = sign(\theta_0 + \theta_1 y^{(1)} K(x^{(1)}, x) + ... + \theta_m y^{(m)} K(x^{(m)}, x))$$

With this, we train θ_0, θ_1,..., θ_m with the SVM cost function. SVM uses a hinge loss function as the cost function, where hyperparameter C is used to apply a penalty. We can use C to manage the tradeoff between overfitting and underfitting. Increasing C decreases the penalty applied to θ, and subsequently makes our model take more features into account to avoid underfitting. Decreasing C will increase the penalty applied to θ, and subsequently makes our model ignore particular features to avoid overfitting. As for γ, it controls how quickly $K(x^{(i)}, x)$ decreases as the distance between $x^{(i)}$ and x increases. As γ decreases, the kernel decreases slowly with distance as it focuses on a large neighbourhood and makes predictions that are smooth, meaning it is used to manage overfitting. As γ increases, the kernel decreases more quickly with distance as it focuses on a small neighbourhood and makes predictions that are less smooth and tend to snap to the nearest training point meaning it can be used to manage underfitting. Therefore, choosing C and gamma with cross validation is the best option. The feature parameters are chosen by performing gradient descent on a hinge loss cost function that uses a penalty.

IV. Experiments

We need to experiment to choose appropriate values for hyperparameters for the models. For the kNN classifier, first we need to choose a value for k, and then find a value for γ (for the weights). For the kernalised SVM classifier, first we need to choose a value for C, and then find a value for γ (for the kernel). We choose all these values using using 5-fold cross-validation.

A. *kNN (k-nearest neighbours)*

Firstly, we chose a range of values for k from 100 to 1500 and plotted it versus F1-Score (which measures the accuracy of the model). We found that a range from 600 to 1000 yields the highest F1-Score with the lowest value of k (to prevent overfitting).

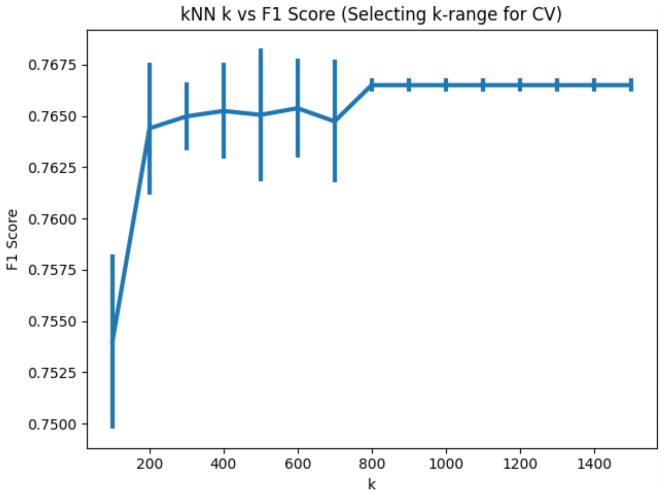

Fig. 5. Plot of F1-Scores vs a wide range of values of k.

We plotted values in this range to perform 5-fold cross-validation. The smallest value of k that yields the highest F1-Score was *k=800*.

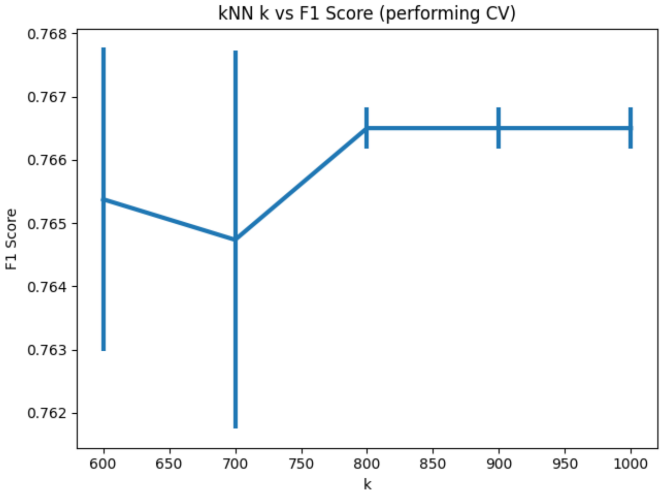

Fig. 6. Plot of F1-Scores vs a smaller range of values of k.

Secondly, we choose a range of values for γ (weights) from 10 to 1000 and plotted it versus F1-Score. For $\gamma \geq 10$, the F1-Score decreases due to overfitting. We found that the range 10 to 50 yields the highest F1-Score with the lowest value of γ (prevent overfitting).

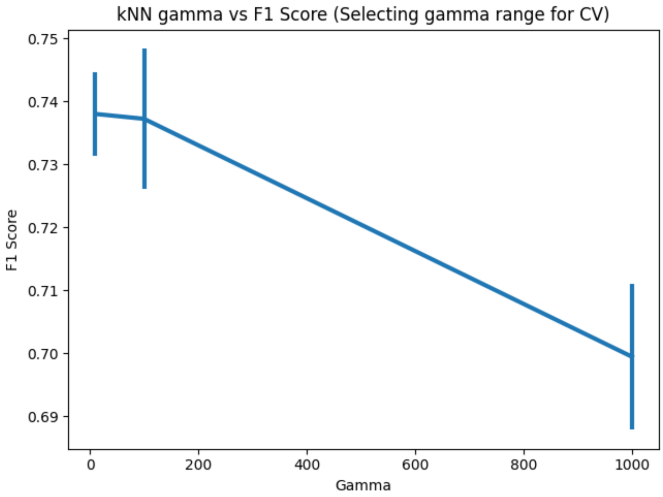

Fig. 7. Plot of F1-Scores vs a wide range of values of γ.

We plotted values in this range to perform 5-fold cross validation. The smallest value of γ that yields the highest F1-Score was *γ=30*.

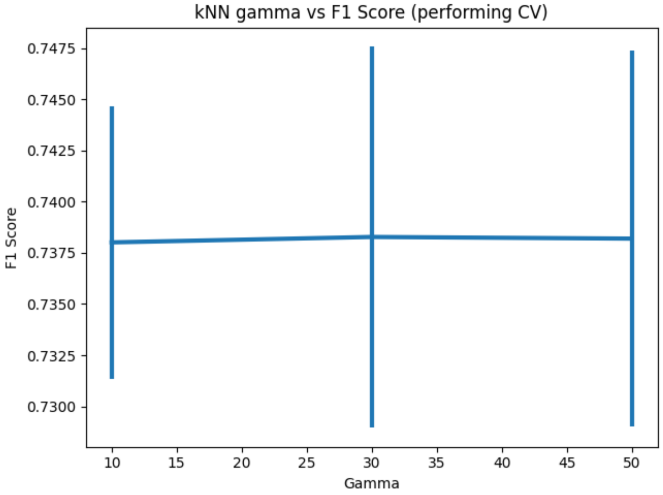

Fig. 8. Plot of F1-Scores vs a smaller range of values of γ.

B. *Kernalised SVM (support vector machine)*

Firstly, we chose a range of values for C from 0.01 to 10 and plotted it versus F1-Score (which measures the accuracy of the model) as shown in Fig. 9. We found that a range from 0.01 to 2 yields the highest F1-Score with the lowest value of C (to avoid overfitting). After plotting values in this smaller range to perform 5-fold cross-validation, we found out that *C=0.6* was the best value for C since it was the lowest value of C that yielded the highest F1-Score as shown in Fig. 10.

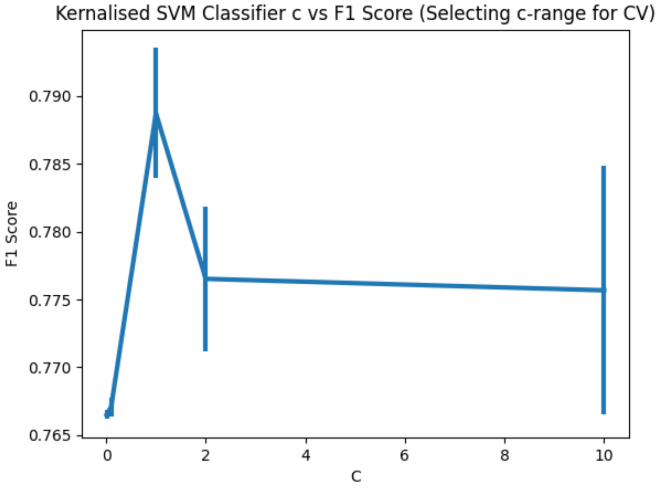

Fig. 9. Plot of F1-Scores vs a wide range of values of C.

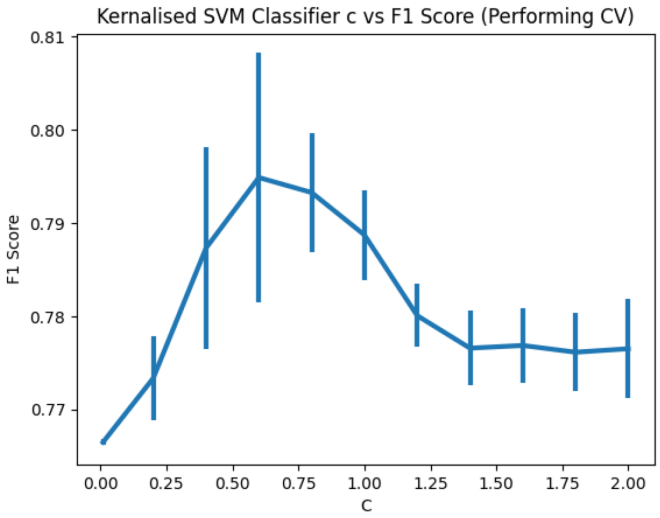

Fig. 10. Plot of F1-Scores vs a smaller range of values of C.

Secondly, we chose a range of values for γ from 1 to 600 and plotted it versus F1-Score. We found that a range from 1 to 100 yields the highest F1-Score with the lowest value for γ (to prevent overfitting).

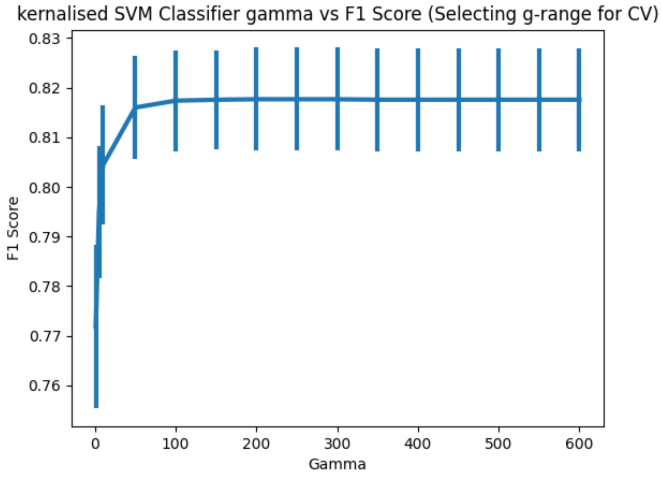

Fig. 11. Plot of F1-Scores vs a wide range of values of γ.

We plotted values in this range to perform 5-fold cross-validation. The smallest value of γ that yields the highest F1-Score was γ=*100*.

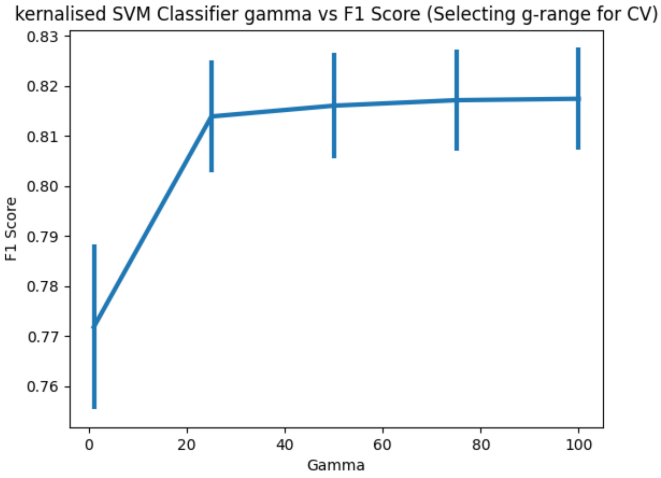

Fig. 12. Plot of F1-Scores vs a smaller range of values of γ.

V. Results & Discussion

Our primary metrics to evaluate the effectiveness of our models include accuracy, precision, recall, ROC curves, and AUC values. The data we use for performance evaluation is from the dataset we used to train our models - we used a 90/10 split. We believe that we did not overfit our models with our training data because we chose hyperparameters with the lowest values that yielded the highest F1-Score using 5-fold cross-validation. The weighted average of accuracy, precision and recall for our models alongside a dummy classifier that predicts randomly can be seen below:

	kNN	Kernalised SVM	Dummy
Accuracy	0.74	0.74	0.49
Precision	0.74	0.84	0.52
Recall	0.74	0.78	0.48

We can see that both models have a fairly equal accuracy ($0.74 = 0.74$) but the kernalised SVM model is the best model since it has a better average precision ($0.84 > 0.74$) and recall ($0.78 > 0.74$). However, both our trained models do much better than the dummy classifier since they have a higher accuracy ($0.74 > 0.49$ & $0.74 > 0.49$), higher precision ($0.74 > 0.52$ & $0.84 > 0.52$) and higher recall ($0.74 > 0.48$ & $0.78 > 0.48$). This reassures us that the models we have trained are worthwhile, as they are able to repeatedly beat someone making a random guess. ROC curves are a plot of true positive rate vs false negative rate. Ideally you want your ROC curve to "stretch" to the top left corner of the plot where the true positive rate is 1 and the false positive rate is 0 (always predicting correctly). You do not want the ROC curve to be

on or under the 45 degree line of the plot as it will mean that your model is no better than predicting randomly or even worse predicting things incorrectly more than it predicts correctly. Below is the ROC curve for the kNN classifier which stretches to the top left corner.

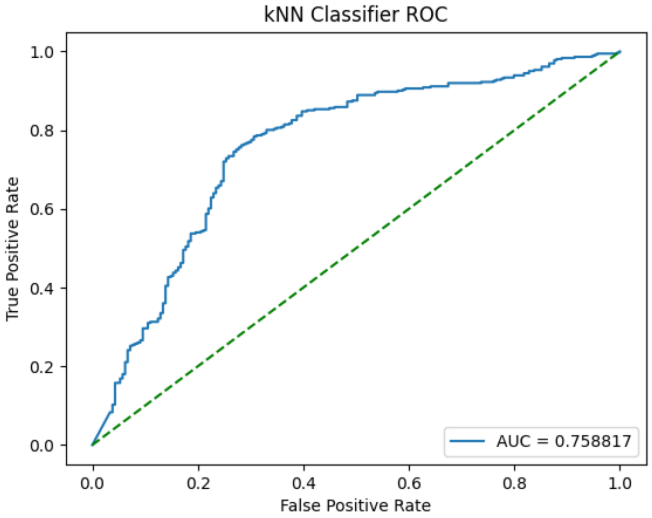

Fig. 13. ROC plot of kNN classifier with its labelled AUC value.

The AUC metric is the area under the ROC curve which value ranges from 0 to 1. Ideally you want the AUC value of your model to be 1 which would mean that you are always predicting correctly. You do not want the AUC value to be at or below 0.5 as it would mean you model is either predicting the same as random predictions or getting more predictions wrong than right. The ROC curve for the kernalised SVM classifier is below which also stretches to the top left corner where the true positive rate is very high and the false positive rate is very low.

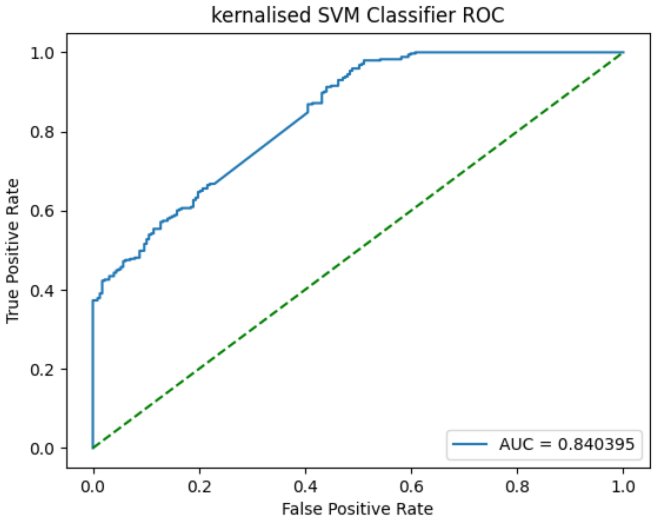

Fig. 14. ROC plot of kernalised SVM classifier with its labelled AUC value.

The dummy classifier's ROC curve can be seen below which is a straight line and does not reach the top left corner.

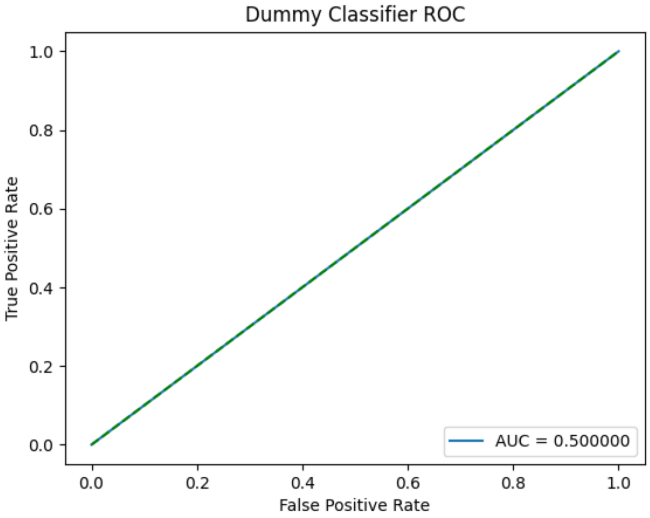

Fig. 15. ROC plot of kernalised SVM classifier with labelled AUC value.

The AUC values of 0.758817 (kNN) and 0.840395 (kernalised SVM) significantly outperform the 0.5 from the dummy classifier. But again, we can conclude that the kernalised SVM is the best model for predicting modality ($0.840395 > 0.758817 > 0.5$). Finally, to retest this conclusion we can look at the confusion matrices for each model for [kNN, kernalised SVM, dummy classifier] respectively:

$$\begin{bmatrix} 137 & 68 \\ 76 & 289 \end{bmatrix}, \begin{bmatrix} 84 & 127 \\ 0 & 359 \end{bmatrix}, \begin{bmatrix} 114 & 97 \\ 202 & 157 \end{bmatrix} = \begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

Using the formula for accuracy $=$

$$(\frac{TN + TP}{TN + TP + FN + FP})$$

The kNN classifier has an accuracy of 74.3%, the kernalised SVM classifier has an accuracy of 77.7%, and the dummy classifier has an accuracy of 47.5%. These results agree with our conclusion, and we are happy with the outcome of the project.

VI. Summary

In this report, we used audio features from 5700 songs across 114 genres to train a kNN and kernalised SVM classifier to predict the modality of a song. We prevented overfitting by choosing hyperparameters with 5-fold cross-validation. The kernalised SVM was the most accurate at prediciting modality, with a weighted average accuracy of 74%, precision of 84%, and recall of 78%. This is because the kNN algorithm chooses the smallest distances ($d(x^{(i)}, x)$) to get the

k-nearest neighbours (training data points closest to input feature vector x). However, since songs are so diverse and varied, this does not predict well as it's limited to the k-nearest neighbours unlike the kernalised SVM which converts each training data point into a feature. With the kernalised SVM, when the distance between the training data point and input x is big, K tends to 0 meaning some features are not used. This results in a dynamic model where the features used for prediction vary from song to song, making it more effective at predicting modality.

VII. Contributions

Prathamesh Sai:

- Scraped songs using the Spotify API and done pre-processing on the data to remove unnecessary data in `data_collection.py`
- Contributed to feature engineering by making the function *select_features_with_best_accuracy()* in `feature_engineering.py`
- Used the kernalised SVM classifier to predict modality in `kernalised_SVM.py`
- Wrote "Introduction", "Dataset and Features", the numerical data from "Results & Discussion", and the kernalised SVM sections of "Methods" and "Experiments" in this report.
- LaTeX formatting, creating `setupEnv.sh` to setup .env for the Spotify API key, and `README.md` for documentation purposes.

Eligijus Skersonas:

- Contributed to feature engineering by making the function *select_features_with_dependency()* in `feature_engineering.py`
- Helped analyze results with `check_performance_utility.py`
- Used the kNN classifier to predict modality in `kNN.py`
- The ROC curves and confusion matrix data from "Results & Discussion", the kNN sections of "Methods" and "Experiments", and the "Summary" in this report.

The file `gaussian_kernel_utility.py` was inspired from lecture notes provided by Professor Douglas Leith.

VIII. Github Link

The GitHub repository for this project can be found at https://github.com/saisankp/Spotify-Modality-Predictor