Final Report (5 page max)

Introduction

How good are Spotify's audio analysis features for predicting the genre of a given track? Spotify's API provides some interesting low-level numerical metrics which describe audio features of songs. We were wondering if these metrics (generated only through scanning a song's waveform) could be used to predict the genre of the song.

Genres are a tricky subject. They are used to broadly categorise artists but these categorisations are nowhere near definitive. Genres change over time ("pop" comes from "popular" music which inherently changes with the era) and one person's definition of a genre can easily vary from another's.

An artist's work often crosses genres resulting in a new genre mixture which gets a new name. Spotify accounts for 5,092 genres according to everynoise.com[1], from "pop" to "escape room" to "himene tarava". There are too many genres to accurately predict which is something we had to deal with when cleaning the data.

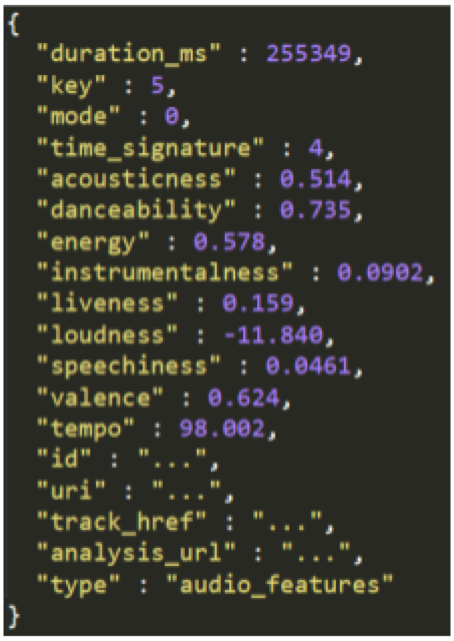
The input to our algorithm is a set of audio features generated by analysing an audio track (done by Spotify) and the output is either one or more likely genres depending on the approach taken, more on this in the methods section.

Dataset and Features

Our dataset was obtained from Spotify's API. A call to the "audio_features" endpoint is shown on the right. The features we used are listed and described below:

- **Acousticness**: How acoustic a track is.

- **Danceability**: How suitable a track is for dancing.
  - Based on tempo, rhythm stability, etc.
- **Energy**: A perceptual measure of intensity and activity.
  - Fast, loud, noisy songs contribute to energy.
- **Instrumentalness**: Whether a track contains no vocals.
  - Closer to 1 = more likely to contain no vocals.
- **Liveness**: Detects presence of audience in the track.
  - > 0.8: most likely a track recorded live.
- **Speechiness**: Detects presence of spoken words.
  - > 0.66: most likely completely spoken word.
- **Valence**: Positiveness of a track.
  - High valence indicates a happy, cheerful track.



```
{
  "duration_ms" : 255349,
  "key" : 5,
  "mode" : 0,
  "time_signature" : 4,
  "acousticness" : 0.514,
  "danceability" : 0.735,
  "energy" : 0.578,
  "instrumentalness" : 0.0902,
  "liveness" : 0.159,
  "loudness" : -11.840,
  "speechiness" : 0.0461,
  "valence" : 0.624,
  "tempo" : 98.002,
  "id" : "...",
  "uri" : "...",
  "track_href" : "...",
  "analysis_url" : "...",
  "type" : "audio_features"
}
```

Spotify's Audio Features

In order to obtain enough track IDs to make a dataset, we browsed spotify and copied some playlist IDs, from which we can call the "playlists/{playlist_id}/tracks" endpoint in order to obtain the track IDs. Using this approach we generated a dataset of over 11,000 songs which proved to be more than enough for training our models.

From these features you can see how certain styles, or genres, of music might make themselves clear through analysis. For example rap would have a high speechiness value; dance music would hopefully have a high danceability score; folk or country music could have a high acousticness rating.

In terms of the features not used, tempo (speed of a track), key (musical framework the song is built on) and mode (major or minor) all vary widely within genres so including them didn't improve our models.

[1] http://everynoise.com/everynoise1d.cgi?scope=all

The associated genre label for a song was included in an artist's API call so for each song the artist's genres were fetched. As mentioned previously Spotify uses an awful lot of genres, most of which are very specific and part of some subtree of a larger, more easily recognisable genre. Here are the genres associated with 'Foals' for example:

[alternative dance, indie rock, modern alternative rock, modern rock, new rave, oxford indie, rock]

We needed a way to map these, sometimes quite niche, genres to a more general dataset. We also needed to somehow distinguish the more relevant genres from the less relevant ones as they are returned in alphabetical order. To do this a list of base genres was drawn up based on the most popular genres from Spotify's database[2]:

[pop, rock, rap, dance, hip, trap, r&b, metal, country, indie, folk, alternative, punk ,electro, psych]

Splitting up the individual words in each artist's genres and also keeping track of the frequency of each base genre allowed us to both map to a more general set of genres and order the genres based on relevance to the artist. So during and after preprocessing Foals' genres were:

= [alternative (x2), dance (x1), indie (x2), rock (x4), modern (x2), new (x1), rave (x1), oxford (x1)]
= [rock (x4), alternative (x2), indie (x2), dance (x1)]

Lastly when training with multiple genres per song the list of genres was converted into a boolean list so a pop + rock song would be *[true, true, false, ..., false]*

Methods

OneVsRest

The OneVsRest approach is quite simple in concept, it will create a unique model per label (in our case a classifier per genre), train them independently and then predict a single genre for the song, in this case the prediction with the highest confidence is used as the output genre of the model. This approach is quite simplistic and enables us to use classifiers we've used in class in order to attempt to predict the most relevant genre of a song.

From this point we found it interesting to experiment with the idea of training independent classifiers for each of the genres, picking the model with the highest accuracy for a single genre and combining the outputs of all the classifiers to enable multiple labels (genres) in our outputs.

Power Set

The powerset approach considers each of the members of the powerset of the labels to be a separate label. It takes potential correlations between labels into account. As the number of labels increases, the combinations of said labels, thus the powerset, grows exponentially. This becomes an issue as it quickly eats up computational power. It also may have many wasted labels as some combinations may come up little to no times. This gives the power-set classifier a high computational complexity.

Multi-Output

Similar to a OneVsRest (OvR) approach, Multi-Output trains a model for each label, however unlike OvR it does not make the assumption that each sample is assigned to one and only one label. So basically the Multi-Output algorithm supports multiple outputs inherently while OneVsRest has to be adapted to do this.

[2] http://everynoise.com/everynoise1d.cgi?scope=all

Models

In the various methods discussed above, we use a set of classifiers that we have seen in lectures, these include the following.

DecisionTreeClassifier

A classifier based on the use of decision trees. Decision trees being a non-parametric supervised learning method. It creates a model which learns simple rules inferred from the data features, and uses them to make predictions.

KNeighborsClassifier

A classifier used to find the label of a given input based on the data-points nearest to it. k represents the number of neighbors that are considered when trying to make a prediction. This value, k, is pre-defined. It is known as a non-generalizing method, as it remembers all of its training data.

LogisticRegression

A model which assigns a probability to each of the possible classifications. The probabilities of the classifications add up to one, meaning that it is ideally used to determine something that may only be classified as one thing.

DummyClassifier

A baseline classifier that chooses the most frequent genre from the training data and predicts that genre for any input.

MLPClassifier (Multi-Layer Perceptron Classifier)

A classifier based on the use of an MLP (Multi-Layer Perceptron). A MLP is a class of artificial neural networks known as feedforward. An MLP consists of multiple layers of perceptrons: an input layer, a hidden layer, and an output layer. The hidden layer may contain multiple sub-layers. Each layer contains a set of nodes. The ones in the input layer nodes take in user-defined values. The ones in the hidden and output layers use a non-linear activation function. In order to train this model, a technique called backpropagation is used. This updates the activation function of each node.

SVC (Support Vector Classification)

A classifier based off of the use of SVMs (Support Vector Machines). SVMs are supervised learning models with associated learning algorithms. The way an SVM works is by taking in training data, and assigning it a point in space. The SVM attempts to create the biggest possible gaps between the categories. When attempting to make a prediction, the predicted category will be based on the side of the gap the input falls on.

RidgeClassifier

A classifier which uses Ridge Regression to determine its results. Ridge regression is a multiple regression model used in order to reduce the standard error for data that suffers from multicollinearity. This is achieved by adding a certain amount of bias. Multicollinearity is when a predictor variable can be linearly determined by the other with significant accuracy.

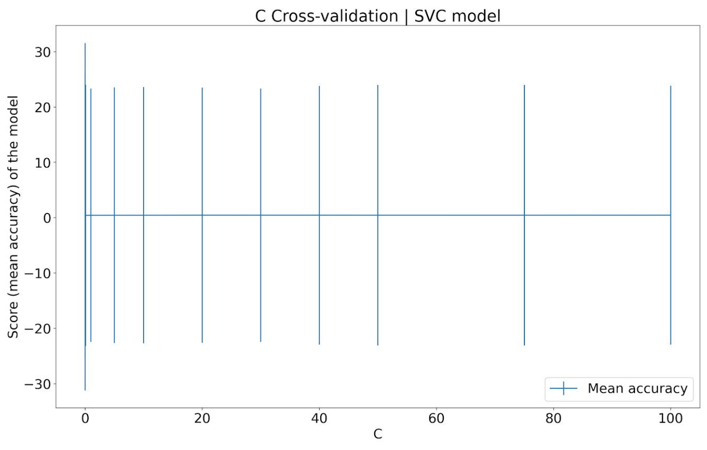
Experiments / Results / Discussion

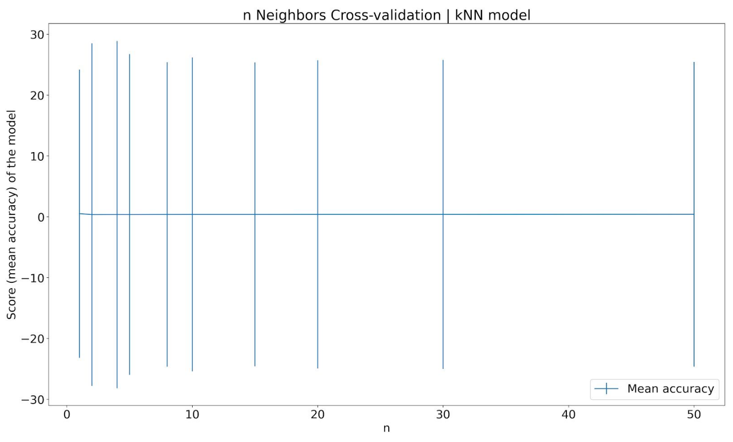
OneVsRest

The approach was to train different classifiers using the OneVsRest approach, the set of classifier models used here include: SVC, Ridge, Logistic, kNN, Decision Tree and a Dummy model. For each of the models, I execute a hyperparameter selection (when applicable), that is:

- SVC, Ridge, Logistic: train using a variety of loss parameters
- kNN: train using a variety of k neighbors values
- Dummy, Decision Tree: train a single model using default sklearn parameters

Some of the cross validation graphs generated for different models include the following:

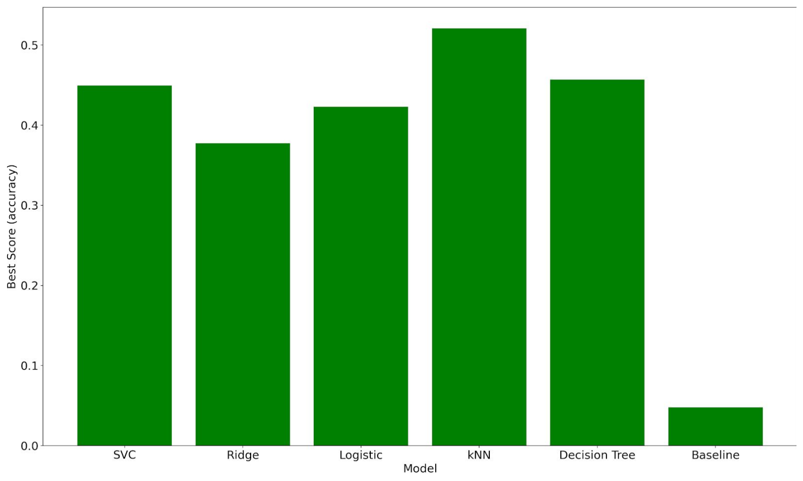




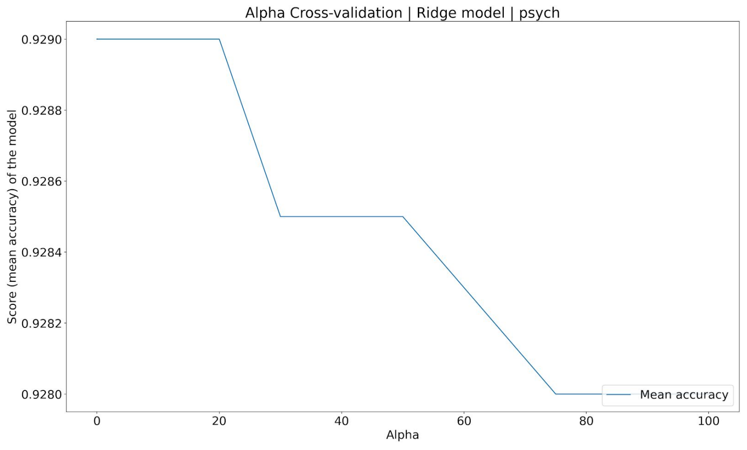
However these graphs were only generated for the report and no manual picking of parameters was done in this process. The system keeps track of the model with the highest accuracy as the final model (out of all of the trained models) thus the parameters for these models are picked automatically. We also keep track of the most performant set of parameters for each trained model, this enables us to to plot a comparison bar chart representing the performance of each model for this specific problem.

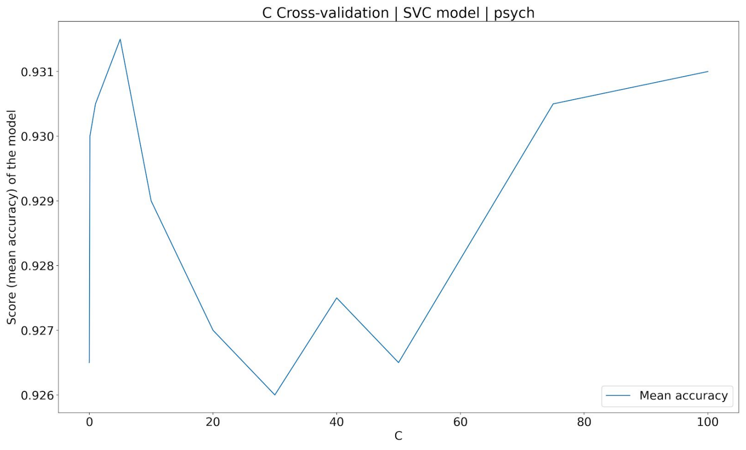
It turns out that running this algorithm using our dataset of 11 thousand songs, we obtain the kNN with n = 1 as the best model, its accuracy is **0.521**.

This value may not seem very high at first, but it is important to remember the context of what we are trying to achieve. The goal is to predict the most important genre attached to a song, we find that a baseline model which predicts the most common genre, in this case “pop” has a very very low confidence level.



While our models' accuracies are much higher in comparison, they still do not surpass the 50% accuracy in prediction which may not seem that high. The potential issue with this approach is keeping only a single genre per song may mislead our models into conflicting behaviors as there are quite often overlaps in characteristics of songs between certain genres, thus predicting the genre with the highest confidence may not always be accurate in terms of the goal of this approach. An approach where all of a song's genres are taken into account during training would be much more appropriate. This left us wondering if it might be easier to instead predict multiple genres per song as the genres predicted here, although possibly not the most important ones, did appear in the songs set of genres most of the time.

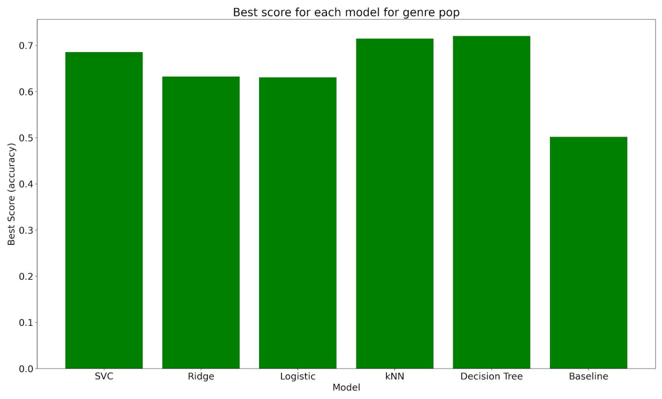


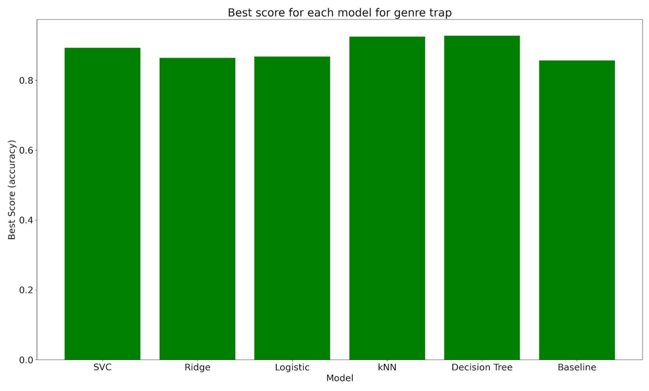


Let’s now transition this approach to the one mentioned above, which is training a set of independent classifiers, using this same approach as seen here. There will be a single classifier per genre and we will consider all of the song's genres here instead of only taking the most relevant one in the previous approach. By picking the most performant model for each genre, and combining the outputs of all of these classifiers for a single song, we should be able to predict all of the genres attached to a song.

Thus, using this approach, we use the same set of models as mentioned above, train them using automatic selection of hyper parameters in the same way and repeat the process for each genre. We obtain the same cross validation plots from the above method, shown above.

And similarly, we also obtain a bar chart model comparison for each of the genres showing all the model types using the hyperparameters which yield the highest accuracy.





From these bar charts we observe multiple things. To start with, the accuracy of individual classifiers for a certain genre is much higher for all of the genres. We also notice that for some genres, the baseline model is quite high and will sometimes beat some models or get close. However, we do find that the set of most performing models for each genre end up not including any baseline models which is good news for us. We obtain the following set of classifiers for each genre, and their associated accuracy:

Genre	Best Performing Model	Accuracy
pop	DecisionTreeClassifier()	0.7205
rock	DecisionTreeClassifier()	0.767
rap	DecisionTreeClassifier()	0.8985
dance	DecisionTreeClassifier()	0.782
hip-hop	KNeighborsClassifier(n_neighbors=1)	0.897
trap	DecisionTreeClassifier()	0.9275
r&b	KNeighborsClassifier(n_neighbors=1)	0.913
metal	KNeighborsClassifier(n_neighbors=30)	0.9195

Genre	Best Performing Model	Accuracy
country	SVC(C=10)	0.8905
indie	SVC(C=10)	0.857
folk	SVC(C=20)	0.9155
alternative	KNeighborsClassifier(n_neighbors=1)	0.7995
punk	KNeighborsClassifier(n_neighbors=2)	0.8905
electro	SVC(C=0.0001)	0.975
psych	KNeighborsClassifier(n_neighbors=2)	0.9345

From all of these models, we combine the outputs of all of the classifiers for every single song, from which we obtain the following confusion matrix:

All Genres, combined out	Ground True	Ground False
Predicted True	3492	1489
Predicted False	2336	22683

Precision	0.701
F1 Metric	0.646

We notice that the accuracy for these models mostly comes from false predictions which makes sense, we do find that true predictions are correct about 70% of the time which is good news for us and an improvement. From these we obtain the above Precision and F1 metrics.

We note an overall improvement in the approach which we can observe in the metrics above. While the dummy classifiers also rank quite high from the fact that niche genres such as electro could easily be predicted as mostly false, we find that training models can pick up on these outputs for the most part which results in quite good results, and a big improvement from the first OneVsRest approach. Finally, it is worth mentioning that kfolds was not used in this approach as training this many models on many songs was very expensive to compute, we opted for an approach were 20% of the dataset is used as test data and the remaining 80% is used as training data.

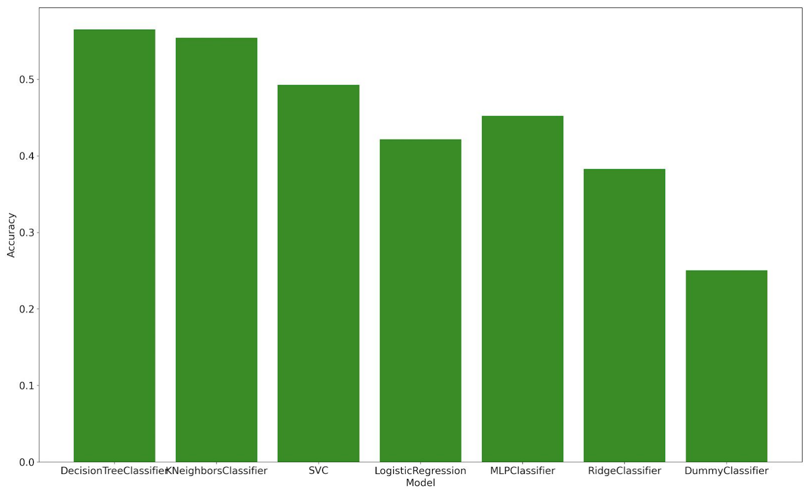
Power Set

Many different models were used in order to test the LabelPowerset approach. These included Decision Tree Classifier, K Neighbors Classifier, SVC, Logistic Regression, MLP Classifier, Ridge Classifier, and Dummy Classifier.

Hyperparameter selection was used when applicable:

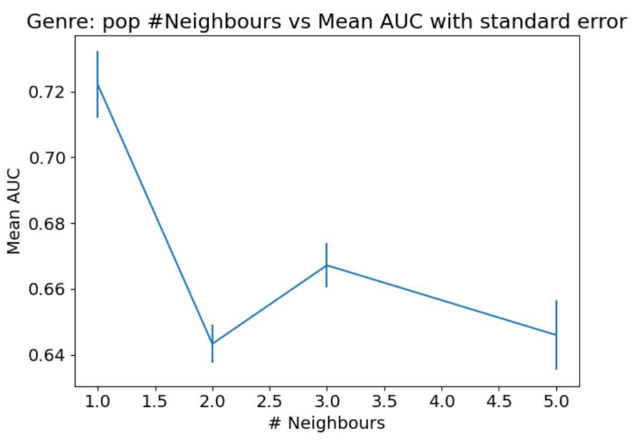
- Number of Neighbors: KNeighborsClassifier
- Loss Parameters: SVC (C), LogisticRegression (C), MLPClassifier (alpha), RidgeRegression (alpha)
- None: DecisionTreeClassifier, DummyClassifier

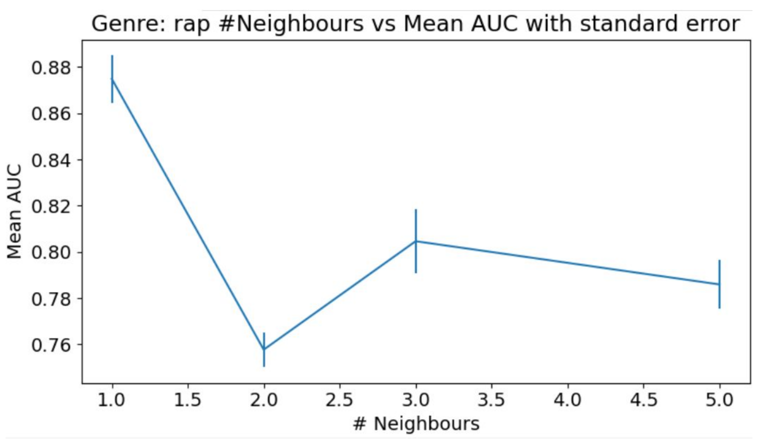
A method used in order to get an accurate estimate is the kFolds method. This method split the data into subsets, all representative of the full dataset. Allowing the model to be trained multiple times, which allows a higher level of precision. The mean of these training rounds being used as the final output. The number of splits chosen is 10. This allows the data to still represent the full dataset with the advantages of training the model multiple times. This was done for each iteration of hyperparameter selection where applicable.



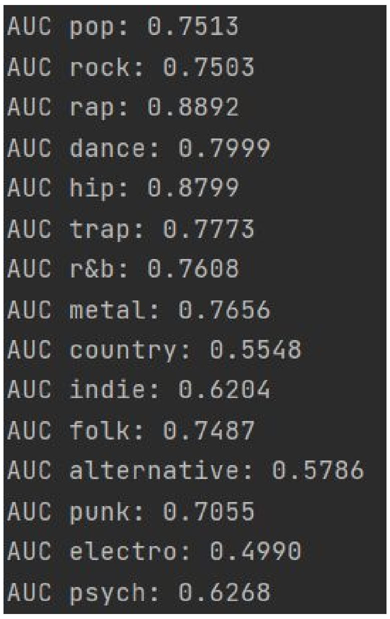
The models that had no hyperparameter selection were simply run once with the kfolds method, and the result from that was taken to be compared to the other models.
The models that had hyperparameter selection ran the same method as the models with no hyperparameter selection, however, it is run multiple times in order to iterate all of the options to be tested. The highest accuracy hyperparameter being used to compare to the other models.
In Appendix A, there are the graphs of each of the hyperparameter selections. The comparison of the models is shown below.

Multi-Output kNN





This approach made use of the multi-output approach which is basically a ready made multi output approach as opposed to the OneVsRest approach which had to be engineered to become a multi-output approach. The above method gives more freedom to the underlying models.



```
AUC pop: 0.7513
AUC rock: 0.7503
AUC rap: 0.8892
AUC dance: 0.7999
AUC hip: 0.8799
AUC trap: 0.7773
AUC r&b: 0.7608
AUC metal: 0.7656
AUC country: 0.5548
AUC indie: 0.6204
AUC folk: 0.7487
AUC alternative: 0.5786
AUC punk: 0.7055
AUC electro: 0.4990
AUC psych: 0.6268
```

The primary metric here was Area under the ROC Curve (AUC). We look to see how well each genre is classified when choosing the hyperparameters; number of neighbours and weighting here. 10 folds were used for this task as our data set was large enough (11,000+ songs) that we could safely use 10 splits while keeping each split large enough to be representative of the entire data set.

The genres were all best classified using a **pure nearest neighbour model** as was also found independently in the OneVsRest classifier case. A single nearest neighbour model can sometimes lead to overfitting though it didn't seem to be the case here. To test this we held out some data from the training and testing process to see how the single nearest neighbour model would fare with completely unseen data and it did just as well as with the train/testing data. Also, with the number of neighbours equal to 1 changing the weighting had no effect on the mean AUC.

A plot of the AUC values with different numbers of neighbours is shown for 'pop' and 'rap' above which both perform quite well. The rest of the results are shown to the right in table form with the number of neighbours = 1 as there were too many graphs to show otherwise. These are really good results for the problem. The results also confirm some suspicions we had at the start. Hip-hop and rap are easier to recognise than more vague genres like indie and alternative.

The confusion matrices for pop and rap are shown below. The average F1 score for all the genres was 0.502 and the average True Positive Rate was 0.503. Not outstanding scores but considering the complexity of the task they seem like good results.

Pop	Ground True	Ground False
Pred True	250	141
Pred False	110	499

Rap	Ground True	Ground False
Pred True	153	56
Pred False	35	756

Summary (100-200 words)

In summary genre prediction is not a trivial task. In a simple test of humans (us) against the machine (our model) we came out second best. For a small set of 30 songs we identified the correct genre(s) 45% of the time with the OvR model achieving 70%. So our best model is quite a bit better than us already. It was also tricky due to the many challenges revolving around defining certain genres, some songs within a certain genre may have metrics which can vary by quite a lot and perhaps we were limited by the data which we were able to find, some more key data points could have made our classifiers more accurate. It is also worth noting that using feature engineering techniques such as polynomial features did not help our models much if at all in terms of making better predictions.

It seems that using the approach of a single classifier per genre worked best in our scenario as limiting songs to a single genre is not realistic in the real world and it also enables our models to be independent and not have conflicting behaviors such as in the OneVsRest single output approach. However we can see how certain genres relate to each other and thus taking correlation into account such as in the PowerSet approach was interesting to pursue.

Contributions

Tom contributed in writing to

- Introduction
- Dataset and Features
- Multi-Output Method
- Multi-Output kNN Results
- Summary

And wrote code

- data_fetcher.py - Genre Parsing
- prediction_model.py - Multi-Output kNN
- util.py - Utility Functions
- genre_plots.py - visualise genre counts

Michael contributed in writing to

- Methods
- Models
- LabelPowerset Method
- LabelPowerset Results

And wrote code

- label_powerset_model.py - Label Powerset
- util.py - Utility Functions

Sam contributed in writing to

- OneVsRest Method
- OneVsRest Results
- Summary

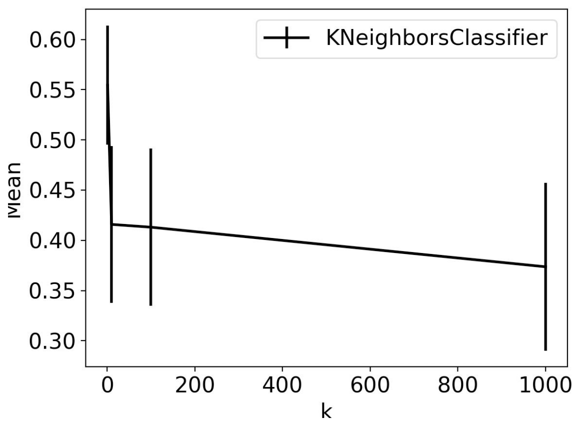
And wrote code

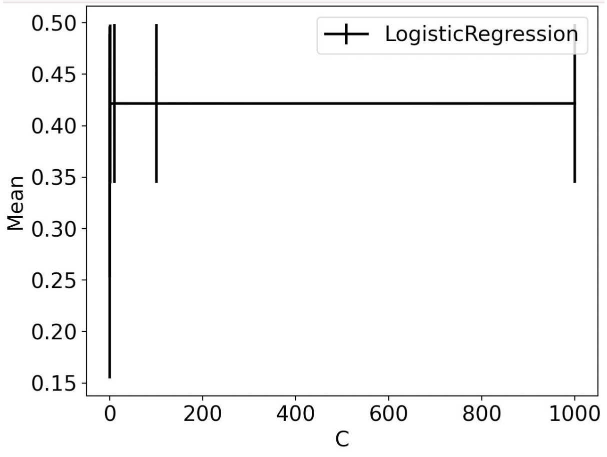
- data_fetcher.py - fetching and dumping data into json
- onevsrest_model_single_output.py - Single output OneVsRest model
- onevsrest_model_multi_output.py - Multi output OneVsRest
- util.py - Utility Functions
- ReadMe - Give an outline of the project
- density_plots.py - visualise the distribution of our data points

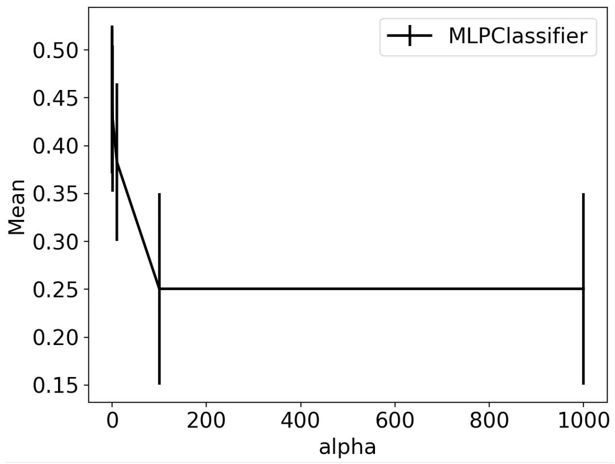
Github Link

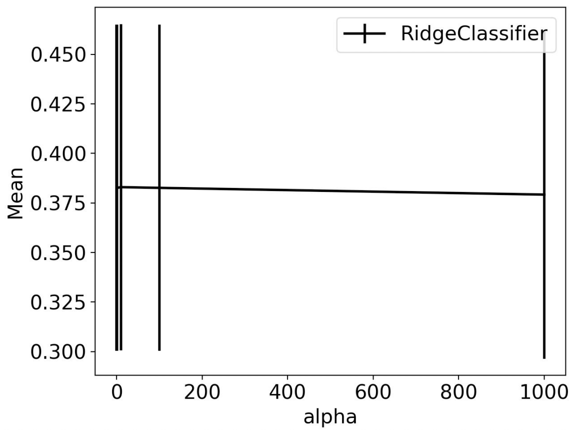
https://github.com/PetitSamuel/genre_predictor
Usernames for each member, as contributions are not representative: (ns: not shown as contributors in most statistics), please find the commit history for individual contributions.
Tom Moran - MySupersuit
Samuel Petit - Samuel Petit and Samuel (ns)
Michael McGuinness - Michael McGuinness (ns) and DaVinciTachyon

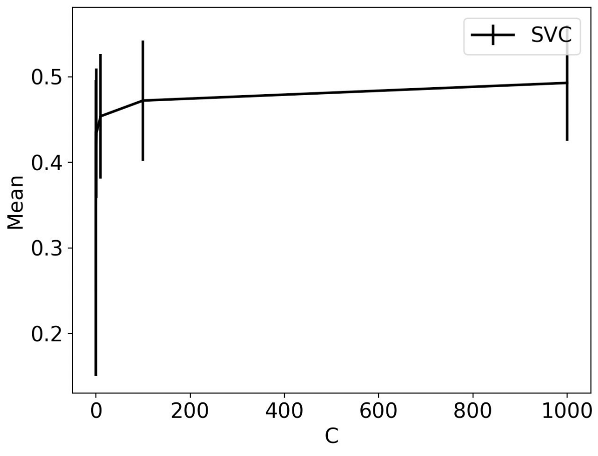
Appendix A - Powerset Hyperparameter Selection


KNeighborsClassifier
Mean
k


LogisticRegression
Mean
C


MLPClassifier
Mean
alpha


RidgeClassifier
Mean
alpha


SVC
Mean
C