Predicting Genre based on Song Characteristics using Machine Learning Models

Boskaljon, F. (2621685) Brunet de Rochebrune, F.D. (2619225) Eisen, H.E. (2608943)
Lei, van der, J.E. (2620342) Dijkstra, R.K. (2598272)

Sunday 29 March 2020

Abstract

This report focuses on finding out which method works best for determining the genre of a song, based on the given features of that song. Several machine learning techniques were used, of which Neural Network, Support Vector Machine, and Random Forest are compared more in-depth. Based on the analysis (on training data) of the three classifiers, the hypothesis is formed that the neural network will outperform the others. The results based on the test set confirmed the hypothesis and showed that the Neural Network works best for predicting the genre of a song based on its features.

1 Introduction

Spotify is the world's most-used freemium music streaming platform. It provides access to over 50 million tracks and users can browse this enormous library by artist, album or genre. Spotify categorises their songs in over 1000 sub-genres, which is obviously not done by hand. Curious about how Spotify classifies new songs, we decided to attempt to construct an algorithm which assigns a genre to a song, based on their musical elements. These musical features are generated by Spotify's music-intelligence division and freely available online. For this project we confine ourselves to the 25 main genres to keep our classification task achievable. We define our research question as follows: How can we best predict the genre of Spotify's songs based on their musical features?

2 The Data

Our dataset[1] originates from Kaggle, an online community of machine learning fanatics and data scientists. It contains 232,725 Spotify tracks which all have 18 different features. Some features like `artist_name`, `key`, `track_lang` and `mode` are strings. Other features like `duration, loudness` and `tempo` are numeric and some like `acousticness, dancebility` and `liveness` are already normalised between 0 and 1.

To get more insight in how our features are correlated to each other, we created a heatmap of the correlations, which is displayed in Figure 6 in Section 6. As some of our features are categorical, we had to make sure to use the correct measures of correlation. When both of the features are numeric we use the Pearson correlation coefficient. This has values between $+1$ and -1, where $+1$ is perfect positive linear correlation, 0 is no linear correlation and -1 is perfect negative linear correlation. When one feature is numeric and one feature is categorical we use the correlation ratio, which ranges from 0 to 1. When both features are categorical we use the uncertainty coefficient, also known as the Theil's U. It is the uncertainty of feature 1 given feature 2, which also ranges from 0 to 1. To clarify, 0 means feature 2 provides no information about feature 1, while 1 means feature 2 provides full information about feature 1.

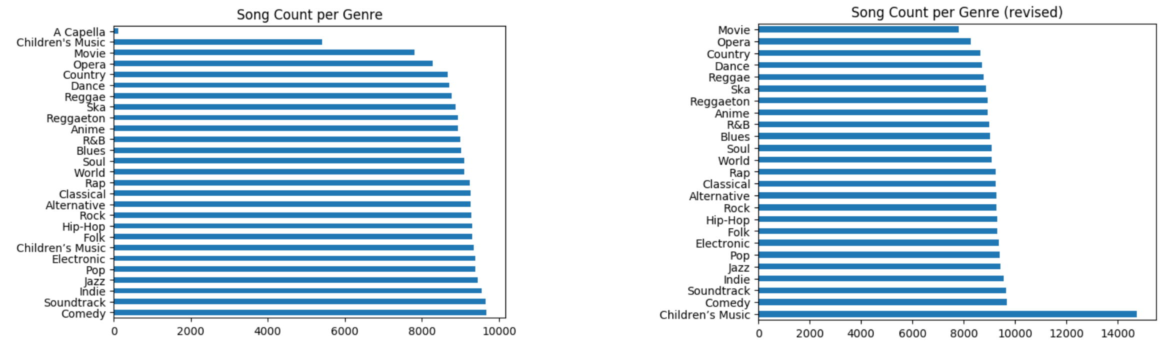
From our heatmap we can see the following. The feature which we want to predict, `genre`, is highly correlated with `speechiness`, `artist_name` and `popularity`.

2.1 Preparation

We start by looking at the different genres. As can be seen in Figure 1, and more specifically in Figure 1a, the genre 'Children's Music' appears twice in the chart. This can be attributed to the fact that these two equivalent

[1] `https //www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db#SpotifyFeatures.csv`

genres have different apostrophes in their name. As these two genres are in fact the same one, we concluded we should merge them together. It can also be seen that the genre 'A Capella' contains very few songs compared to all other genres. In order to create a more balanced dataset we decided to omit all songs from the genre 'A Capella' from the dataset. The new count of songs per genre is displayed in Figure 1b, where we can now see that the genre counts are more balanced.



(a) Song Count per Genre (b) Adjusted Song Count per Genre

Figure 1: Song counts for raw and adjusted data

Looking closer at the data, it becomes clear that some (90,000) songs appear twice or more times in the dataset. These songs have the same features, but fall under different genres and sometimes also have a different popularity within those genres. In order to deal with this, we split the training and test set (using a random state of `18`) in such a way that no songs that appear in the test set also appear in the training set (possibly with another genre). This prevents the problem of classifying the song incorrectly, even though they are correctly classified. The problem therefore becomes a multiclass classification rather than a multilabel classification problem.

As some of the features are non-numerical, they have to be one-hot encoded in order to use them. Features that are one-hot encoded are `mode, time_signature,` and `key`. Furthermore, the title of the song can be used to extract information such as the length of the title, if the song contains a featuring or remix, the language of the song, and if there is any explicit language used. These features are also one-hot encoded and added to the dataframe. It would also be possible to one-hot encode the artists, but we chose not to do this. We found that using the artists in these features significantly increases the accuracy of the model (as artist is highly correlated with genre), yet it also enlarges the dimensionality of the data (and hence increasing the runtime). Choosing to include the artist would furthermore make our model not sustainable as the model then mainly classifies based on artist. If new data (including new artists) were introduced, the model without the inclusion of artist names would be more generalisable.

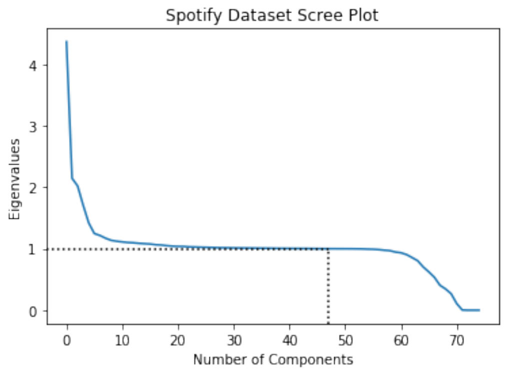
2.2 Approach

Firstly, a number of algorithms were tried, namely KNN, Multinomial Logistic Regression, Support Vector Machine, Neural Network, and Random Forest. Of these algorithms, the Neural Network, SVM, and Random Forest will be further elaborated on. These algorithms have proven to work best for the problem faced [14].

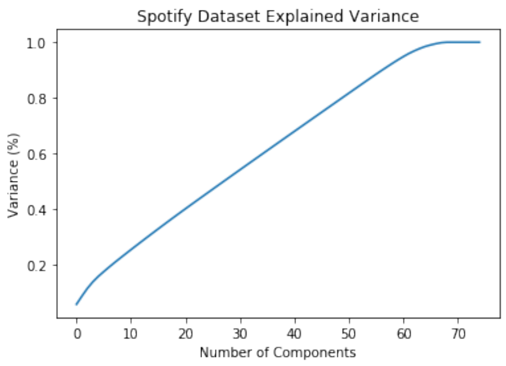
If one would make a random guess of genre for a song, the probability that this guess would be the correct guess is around $\frac{1}{25}$ or 0.04, as the dataset is quite balanced. This is used as baseline to compare the models with, by using the accuracy score as performance metric. This metric works good for balanced data [17], which we are dealing with.

It is then investigated if using PCA would be beneficial in this case. In general, using PCA could lead to dimension reduction, which ensures the run times remain reasonable and prevents overfitting. To choose the number of dimensions, the plots as displayed in Figure 2 are used.

Taking a look at the scree plot as displayed in Figure 2a, Kaiser's stopping rule can be used (which states that we should only use the factors that have an eigenvalue of larger than 1.00 [1]) to find all factors with an eigenvalue larger than 1. Based on this, it can be concluded that using 47 components would be enough to construct a model. If we then look at Figure 2b, which shows the explained variance per number of components, it can be seen that for 47 components, there is a rather small amount of variance explained and only 74% of the data would remain. It would be desirable to use more components in order to remain a high performance of the algorithms, yet with keeping a larger amount (take for example 90%) of the data, the number of features is almost the same as the original number of features, leading to no real benefit of using PCA.



(a) Scree Plot



(b) Explained variance

Figure 2: PCA Plots

For each of the models, optimisation of hyperparameters was investigated. For Support Vector Machines and Random Forests this hyperparameter selection took place according to a gridsearch scheme with a threefold cross-validation. For the Neural Network this selection of hyperparameters took place with the help of a validation set. This validation set was constructed in a likewise manner as previously described for the test set. When the models are finalised, we will approximate performance of the three models, and finally hypothesise on the most accurate classifier. Lastly, we will run all models once on the test data and report those results to compare the performance of each model. More on the results can be found in Section 4.

2.3 Test Setup

All modelling is run on a Dell PowerEdge R720 2U Rack Server with the following specifications:

Processors: Dual Intel Xeon E5-2697 v2 (12 cores, 24 threads, 30 M Cache, 2.70 GHz Base, 3.50 GHz Turbo)
RAM: 384 GB 1600 MHz Registered ECC Low Voltage DDR3 (24x Kingston KTD-PE316LV/16G)
Graphics Card: Nvidia Quadro M4000 (1664 CUDA cores, 8 GB GDDR5)
Storage: Samsung 850 EVO 250 GB Solid State Drive
Network: Dual 10 Gigabit and Dual 1 Gigabit Ethernet (Dell Broadcom 57800S Network Daughter Card)
Operating System: Ubuntu Server 16.04.6 Xenial Xerus LTS (Bare Metal)

The server is hosted at F. Brunet de Rochebrune's home and shared on the internet for remote access from anywhere. To keep performance overhead minimised, Jupyter runs via a mixture of bare metal and Docker containers.

3 Methods

3.1 Neural Network

Neural Networks or Artificial Neural Networks (ANN) are a subset of learning algorithms within the machine learning field that are loosely based on the concept of biological neural networks. An ANN comprises of the following components:

Input Layer: The first layer that receives N-dimensional data and passes it on to the hidden layer(s).
Hidden Layer(s): The learning process of the neural network, where information is extracted from the input layer and transformed to new features.
Output Layer: The last step of the neural network, where the hidden layer will be translated into a preferred output vector.

The input and the output layer are dependent on the data. We have 75 total features, so an input layer of $m \times 75$, with m the number of instances in our dataset. The output layer consists of all the different genres we want to predict. We have 25 genres we want to predict, so our output layer will also be 25 columns wide.

3.1.1 Output Layer

Our neural network will be classifying 25 genres from input data. If a neural network is a regressor, the output layer has a single node. If it is a classifier it can also have a single node, unless you have multiple classes – as in our case. For binary classification the logistic function (a sigmoid) and softmax will perform equally well, but to reduce mathematical complexity (and computation time) the sigmoid is the natural choice. When we have multiple classes, however, we can't use a scalar function like the logistic function, as we need more than one output to know the probabilities for all the classes. For our output layer we will thus use the softmax activation.

Each song will only have a single genre assigned to it, this is multi-class classification. The default loss function for this is cross-entropy, also called softmax loss. Mathematically, it is the preferred loss function under the inference framework of maximum likelihood, so we will also use it.

3.1.2 Number of Hidden Layers

There is no single magic formula you can fill in for obtaining the optimum number of hidden neurons, however, there are general rules and estimations that were picked up over time and followed by researchers and engineers. J. Heaton [7] gave an indication on the number of layers to use in a neural network, see the following table:

Hidden Layers	Result
0	Only capable of representing linear separable functions or decisions
1	Approximate a function with continuous mapping from one finite space to another.
2	An arbitrary decision boundary with rational activation functions and can approximate any smooth mapping to any accuracy.
> 2	Complex representations for layer layers.

For most neural networks one or two layers should suffice. Any more layers would not see direct advantages, unless dealing with complex input and output. Looking at our data, we could say that one layer should suffice. If later this poses any real problem or seems too scarce, we could add another.

3.1.3 Number of Neurons

In order to keep the neural network general and prevent overfitting, the number of nodes or neurons has to be kept as low as possible, while not obliterating accuracy. We need to find a balance. If we have a large excess of neurons, the network becomes a memory bank that can recall the complete training set, but would not preform well on a validation set.

T. Masters proposed a rough approximation of the required number of neurons obtained by the geometric pyramid rule [13]. As determined above, we will be using a network with only a single layer, this would result in the following formula for calculating the maximum number of neurons in our network to prevent overfitting: $\sqrt{n \cdot m}$. With n the number of input layers and m the number of output layers. It's a rather simple formula for one layer, but for more layers it gets more complex. For our model this would result in a value of around 43.3, so a maximum of 43 neurons.

Researchers at the Oklahoma State University also had a say on how many neurons you should choose and came with the following formula: $N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$. With N_h maximum number of neurons, N_i number of input neurons, N_o number of output neurons, N_s number of samples in training data set, α an arbitrary scaling factor usually 2-10 [5]. We chose an alpha of 5, which resulted in an N_h value of around 323. This number seemed rather high, compared to Masters' result. Due to this, we decided to benchmark ourselves and experiment how many neurons would lead to what accuracy. For the benchmark network we used a ReLU activation layer, softmax on the output layer and the Adam optimiser.

With 10 neurons the network seems to have some trouble classifying some inputs, the accuracy is rather low. This it partly logical, as the number of neurons is less than the number of output layers. Upping the number of neurons to 50, we immediately saw the accuracy increase by 5%. Train and validation accuracy and loss are close to each other, everything seems great. If we keep bumping this up, to 100 neurons, the accuracy still increases. However, the more epochs the model runs, we can already see a difference starting to appear between the train and validation accuracy and loss curves. Of course, increasing the number of neurons will only widen this difference. For this, in the end, we decided to limit the number of neurons to 50.

3.1.4 Activation Functions

The activation function is an important hyperparameter. We decided to try all the common activation functions and see which one performs best on the model. In Figure 1 we can see that the ReLU activation function yields the highest accuracy. ReLU is the most popular activation and has often been used to achieve state-of-the-art results, for example a model on the ImageNet photo classification dataset by A. Krizhevsky, et al. [10]. We will be using ReLU for the rest of the model.

`elu`	`softmax`	`selu`	`relu`	`tanh`	`sigmoid`	`hard_sigmoid`	`exponential`	`linear`
41.76 %	28.00 %	41.49 %	44.67 %	42.63 %	41.13 %	41.17 %	42.06 %	38.95 %

Table 1: Training Accuracy for Different Activation Functions

3.1.5 Optimisers

When the Adam optimiser first got introduced, researchers got very excited about its power and speed. The paper showed some very optimistic charts, boasting big gains in terms of performance [9]. However, after the introduction people started noticing that despite the superior training time, the Adam optimiser does not converge to an optimal solution in some areas. For some tasks, for example image classification, real good performance is still only achieved by applying stochastic gradient descent with momentum. Wilson et al. also showed that Adam did not generalise as well as SGD with momentum did, when testing on a diverse set of deep learning tasks. Trying to close the gap with SGD, a lot of research had to be done first.

Adam optimiser can be viewed as a combination of RMSprop and momentum. Nesterov accelerated gradient (NAG) is superior to vanilla momentum. Nadam (Nesterov-accelerated Adam) thus combines Adam and NAG [3]. Nadam boasted a 3% accuracy increase in the current model.

3.1.6 Regularisation and Dropout

Regularisers allow for applying penalties during model optimisation to reduce overfitting and increase generalisation. These penalties are incorporated in the loss function. A linear regression model that implements L1 norm for regularisation is called lasso regression. One that implements (squared) L2 norm for regularisation is called ridge regression. The key difference is that Lasso shrinks the less important features' coefficients to zero, thus removing them altogether. Lasso will work well in case we have a big number of features, which we do.

Applying the regulariser showed surprising results. Using the parameter (called alpha or lambda) of 0.001, chosen arbitrarily, we saw that there was an immediate decrease of performance. There is no set formula for finding an optimal regularisation factor, but we decide to investigate by merely trying many different models. Looking at Figure 8 we can see that the more intense our regularisation gets, the less accuracy we get on our validation set. Even with just low amount of regularisation we get a stark drop in accuracy.

Dropout was also thoroughly tested, and showed similar results. There were no real improvements on performance, only a negative impact.

3.2 Random Forest

Random forests can be considered as one of the best performing off the shelf methods for classification problems in machine learning. It does not need a lot of extra tuning and makes use of the decision tree model.

3.2.1 Decision Tree

A Decision tree is often the most interpretable machine learning model in multi-classification. The learner is easy to understand, able to handle both numerical as categorical data and is very robust [4][8][12] . The tree categorises a data set by setting splitting rules for each node in the tree. The data set moves down from the top of the tree along the nodes to a leaf which holds a class. There are many algorithms for creating a decision tree. Most of them work top-down, by searching for the best attribute upon which to split [15]. The best attribute to split on is the one that creates the least uniform distribution. There are different metrics to measure the uniformity of a distribution. For further analysis, two node impurity measures are considered: Gini index and Entropy.

3.2.2 Splitting Rules

The Entropy is the expected code length of an element sampled from a distribution. The information gain of a split is the parent's Entropy (before the split) minus the weighted average of the children's Entropy after the split: $I_o(G) = H(O) - H(O \mid G)$ where $H(p) = -\sum_i p_i \log_2 p_i$ and $H(O \mid G) = \sum_i \frac{|S_i|}{|S|} H(S_i)$.

The Gini index measures the chance of incorrectly classifying a random item from a distribution, if it was randomly classified according to the distribution's class labels. It can be computed by: $I_G(p) = 1 - \sum_i p_i^2$. The Gini gain is used for choosing the best split, it is given by the parent's Gini (before the split) minus the weighted average of the children's Gini after the split.

Comparing the impurity measures we can see from Figure 10 that both metrics are lower bounded by zero and the value of pure nodes is zero. The Gini is upper bounded by 0.5, whereas Entropy is upper bounded by 1. A drawback of using Entropy is that it favors attributes with a large number of distinct values, which can result in adapting irrelevant attributes [20]. Another disadvantage is that it is computationally more intense then Gini because of the log.

3.2.3 Overfitting

For single trees, there is always a danger of overfitting due to instability in the tree. The learner can fit too closely to the training set and adapt random variation present in the set itself. Such a model's performance will be poor on the test set because it does not generalise well [16]. A solution for overfitting without reducing the accuracy is an ensemble method. In this report, the random forest ensemble method is used to reduce the variance.

3.2.4 Ensembling

A random forest consists of a set of trees constructed from bootstrap samples from the training set. The bootstrapping is used to equalise the influence of particular observations [16]. The construction of the trees is done without any stopping or pruning criteria. The attribute set for splitting is randomly limited by a parameter to allow less stronger variables to be included in the model and reveal potential insight [16]. The trees are merged to a forest by averaging the predicted classes to one response. The ensemble method utilises the fact that classification trees are instable but on average produce the right prediction [16]. The hyperparameters for the Random Forest building algorithm are selected using a gridsearch analysis with a threefold cross validation.

3.3 Support Vector Machine

The Support Vector Machine classification algorithm was introduced by V. Vapnik in 1995. The algorithm is essentially focused on a binary classification problem. For a binary classification problem, the algorithm defines a hyperplane which separates the two classes. This separation is made while enforcing a maximum distance of all points to this hyperplane. The points closest to the hyperplane are called the support vectors, and the distance to the hyperplane is called the margin. When the hyperplane perfectly separates two classes it is called a hard-margin SVM. When points are allowed to violate the margin, a soft-margin SVM is constructed. For data not linearly separable, defining a separating hyperplane is difficult. In order to combat this problem, input vectors must be non-linearly mapped to higher dimensional feature spaces, in which a linear hyperplane may be more easily constructed. This non-linearly mapping ensures high generalisation of the SVM-classifier and in turn results in a more powerful model [2].

3.3.1 Kernel Functions

Though this expanding of the feature space can be costly, we can apply the kernel trick which implicitly performs this mapping to higher feature spaces. Defining the hyperplane therefore is computationally only dependent on the dimensionality of the input, and not on the dimensionality of the feature space. Several kernel functions exist and in general there is no best kernel; a choice of kernel must be decided upon by means of experimentation. For now, we look at the polynomial and RBF-kernel:

Polynomial Kernel: $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$. The polynomial kernel is used to create a polynomial classifier of degree d. Radial Basis Function (RBF) Kernel: $K(\mathbf{u},\mathbf{v}) = \exp(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2})$. Another implementation can also be used, such as the following: $K(\mathbf{u},\mathbf{v}) = \exp(-\gamma \|\mathbf{u}-\mathbf{v}\|^2)$ with γ as a hyperparameter; $\gamma = \frac{1}{2\sigma^2}$. The RBF-kernel is mostly used due to its power in forming non-parametric decision boundaries; as a result it might also be prone to overfitting since possibly any decision boundary may be formed [19].

3.3.2 Multiclass SVM

The SVM until now has been depicted as a binary classifier, yet this classifier can also be extended to the realm of multiclass classification. Two well known approaches for this are the construction of One-Versus-Rest (OVR) and One-Versus-One (OVO) Classifiers. The OVR-classification method was firstly mentioned by V. Vapnik in 'Statistical Learning Theory' (2000) [18]. For the OVR-classifier, a binary classifier for each class is constructed. Each of these classifiers is trained on instances of its own class and other classes in order to distinguish a class from the other classes. In the end, a new instance is classified according to these κ classifiers. It does so based on the highest probability an instance belongs to according to the κ-classifiers.

The OVR-classifier is relatively simple and generalises well; however M.Arun Kumar and M.Gopal (2011) note that each of the κ-classifiers within the OVR-classification problem do not 'communicate information' since all classification problems are seen as independent problems [11].

Another well known multiclass classification approach is posed by Hastie and Tibshirani (1998), namely by applying pairwise coupling, which goes under the name One-Versus-One classifiers . This pairwise coupling is applicable to SVMs. $\frac{\kappa(\kappa-1)}{2}$ classifiers are constructed to construct a SVM for each pair of classes. To classify a new instance, the new instance is classified by the $\frac{\kappa(\kappa-1)}{2}$ classifiers and a class is chosen based on the majority rule of outcomes of these classifiers [6].

Given the dimensionality, the number of instances and the number of classes of our data, computing a OVO-classifier is computationally too costly. We will therefore focus our efforts for the rest of this report on the use of the OVR-classifier and the choice of kernels. For our analysis, we will investigate three types of classifiers, namely the linear SVM and the SVM with polynomial and RBF kernel. Furthermore, we will investigate the hardness of our SVM-classifier and its impact, based on adapting the Hyperparameter C. If $C \to \infty$ we obtain a Hard-Margin SVM. For the investigation of hardness of our classifier we have proposed the following grid of choices for C: [0.5,1, 10, 50]. We compare the effects of different kernels and their choices for C, based on a GridSearch scheme with a three fold cross validation. We do this by the methodology as described in Section 2.2, consequently normalising data during implementation.

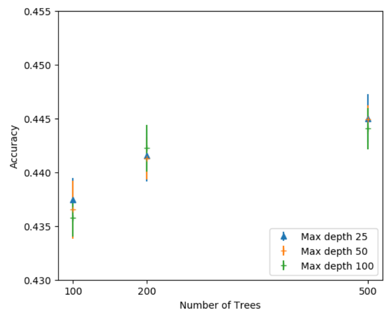
4 Results

4.1 Neural Network

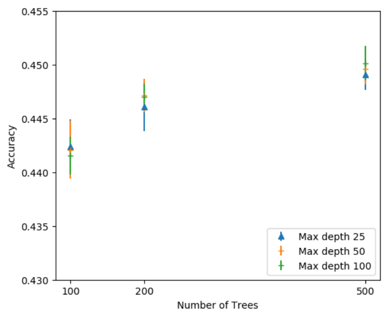
For our neural network we manually checked which hyperparameters gave the best results, as grid search seemed hard to implement for a neural network. Manual comparison was done with the help of a validation set. We noticed that increasing the complexity of our neural network did not increase the accuracy. Our optimal network has one hidden layer with 50 neurons and can be seen in Figure 7. The activation function ReLU yielded the best results, as seen in Table 1. Regularisation and dropout against overfitting were tested but did not boast any improvements. If they were applied, even in low amounts, the accuracy already dropped 25%. The Adam optimiser worked well for the network, however, applying the Nadam optimiser yielded some 2-3% performance boost over Adam.

4.2 Random Forest

The hyperparameter selection for the random forest building algorithm is done using a gridsearch analysis with a threefold cross validation on the training set. The selection of hyperparameters given to the grid search are shown in Table 2.



(a) GridSearch with Entropy as splitting rule **zoomed-in**



(b) GridSearch with Gini as splitting rule **zoomed-in**

Figure 3: Random Forest Grid Search Results with Errorbars

No. trees	Max depth	Impurity
100	10	Entropy
200	50	Gini
500	100	

Table 2: Grid search parameters

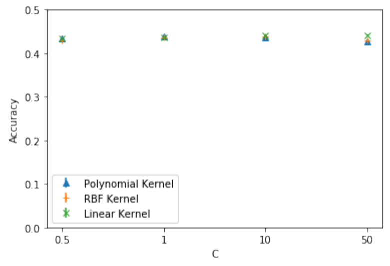
Figure 3 below shows the gridsearch training scores. In general, the accuracy of the model rises if there are more trees in the ensemble (Please note the scaled y-axis). Per plot, it is possible to see that there is not much difference between the parameters for the tree's maximum depth. However, the Gini (3b) performs better compared to the Entropy (3a) as splitting metric. The hyperparameters which resulted in the highest accuracy score on the training set were: `'criterion': 'gini', 'max_depth': 100, 'n_estimators': 500`. This selection of hyperparameters is also used in our further analysis to predict the test set. Results for forest with more than 500 trees is not investigated in this report. This is because of the relation between a high increase in computation time and a potentially small benefit in accuracy.

4.3 Support Vector Machine

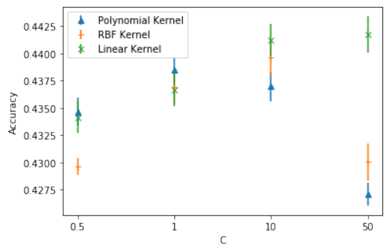
In Figure 4, the results for our gridsearch on different kernels and choices for C is depicted. Figure 4b shows a clearer insight in the different choices for Kernel functions and choices of C (Please note the scaled y-axis). From these two plots it is clear that different kernels perform approximately in a likewise manner.

Although these results can be statistically tested for significant differences, it is evident from the figures that these results are statistically irrelevant. This can be seen in the minimal differences between the classifiers, as displayed in Figure 4a. In order to test significance differences, Overlapping 95% Confidence Intervals indicate no such difference exists. Non-overlapping 95% Confidence Intervals indicate significant differences while maintaining $\alpha = 0.05$.

Due to the irrelevance and insignificance of some of these differences, any classifier can be chosen. For the latter we focus on the two best performing classifiers (according to the displayed plots), namely Linear SVMs with C=10 and C=50. In order to test the following hypothesis:
H_0:$\mu_{accuracy;C=10} = \mu_{accuracy;C=50}$ and H_1:$\mu_{accuracy;C=10} \neq \mu_{accuracy;C=50}$ at a significance level of $\alpha = 0.05$. We construct the following confidence intervals for both classifiers. $CI_{95\%,Linear_{C=10}} = [0.4383,\ 0.4441]$ and $CI_{95\%,Linear_{C=50}} = [0.4384,\ 0.4450]$. These Confidence Intervals overlap, thus indicate no significant difference exists between these two classifiers. For the remainder of our analysis we restrict ourselves to the use of the Linear (C=10) SVM. Since it performs reasonably well (compared to other SVM classifiers), no significant difference exists between Linear (C=10) and Linear(C=50) SVM and, lastly, Linear (C=10) has lower runtime when compared to Linear (C=50) SVM.



(a) SVM GridSearch for C and Kernels



(b) SVM GridSearch for C and Kernels, **zoomed-in**

Figure 4: SVM Gridsearch Results with Errorbars

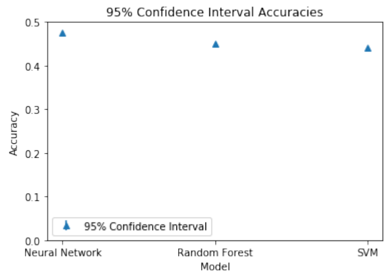
4.4 Comparison of Methods and Final Results

For all three models the hyperparameters are selected that work best for our data. These three models will be used to form our hypothesis on performance of the models on the test data. Using threefold cross validation we run each model one last time to calculate the mean accuracy with corresponding standard deviation on the training data. The results can be found in the column Training Score in Table 3. These scores are also plotted with standard deviation as error bars in Figure 5. From Figure 5b it is evident that the training score of 47.65% of the Neural Network is significantly higher than the other two models. As a hypothesis, we therefore expect that our Neural Network will also perform best on our test set.

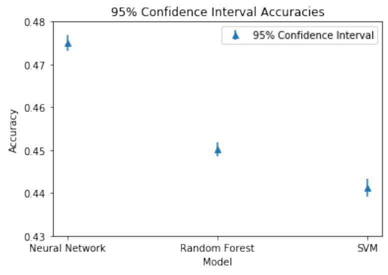
Running the three models on the test set, we obtain accuracy scores for each of the models. The accuracy of the models on the test set can be found it the column Test Score in Table 3. When comparing the accuracy of each of the models with our previously stated baseline ($p = 0.04$), each of the models outperforms the baseline. Furthermore, it is clear that a difference in accuracy between the three classifiers exists. Moreover, for all three classifiers, the test score dropped w.r.t. our estimations based on the training data. Even though these test scores have dropped, no sign of severe overfitting is present. One remarkable insight is the fact that Neural Network performs best as classifier on the test data, which confirms our previously stated hypothesis and also (indirectly) answers the research question.

Classifier	Training score (σ)	Test score
Neural Network	47.65 % (0.0018)	47.34 %
Random Forest	45.01 % (0.0016)	43.82 %
Support Vector Machine	44.12 % (0.0020)	41.62 %

Table 3: Accuracy of Neural Network, Random Forest, and SVM



(a) Training Data Results



(b) Training Data Results, **zoomed-in**

Figure 5: Performance Results on Training Data, with Errorbars

In Figures 11, 12, and 13, several confusion matrices for the three classifiers have been constructed based on the test set. In all three confusion matrices it stands out that the three models struggle most with classifying the following genres: Rap, R&B, Indie, Dance, Soul and Alternative.

Furthermore, it stands out that the SVM model incorrectly classifies a considerable number of songs as Rock. Between the Random Forest and Neural Network only marginal differences exist. Also, the most common classification mistake made is classifying Rap as Hip-Hop. Comedy is classified most accurately by all three models.

Based on the results of our research, we can answer the research question. Our implementation of the Neural Network is the best means to predict which genre Spotify's songs belong to, as compared to other classifiers we have constructed.

5 Future Research

Given our current implementation of the different models and hyperparameter selections, we propose the following options to perform further research on this genre classification problem. First, we advise to gather more extensive data from which more features can be derived, such as the lyrics. These lyrics can then be used to create a term frequency matrix (and calculate tf-idf scores). Furthermore, a wider variety of classification methods may be implemented, accompanied by a more extensive gridsearch, or even more sophisticated methods may be dealt with. Additionally, to make our model implementable by Spotify, songs should be classified into sub-genres as well. Lastly, as this problem was originally sketched as a multilabel classification, we propose that genre classification may be improved upon if viewed as a multilabel classification task.

References

[1] J. D. Brown. "Choosing the Right Number of Components or Factors in PCA and EFA." In: *Shiken: JALT Testing & Evaluation SIG Newsletter* 3.2 (2009), pp. 19–23. URL: http://hosted.jalt.org/test/PDF/Brown30.pdf.

[2] C. Cortes and V. Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995), pp. 273–297. DOI: https://doi.org/10.1007/bf00994018.

[3] T. Dozat. "Incorporating nesterov momentum into adam". In: (2016). URL: http://cs229.stanford.edu/proj2015/054_report.pdf.

[4] A. Ghosh, N. Manwani, and P.S. Sastry. "On the Robustness of Decision Tree Learning Under Label Noise". In: *Advances in Knowledge Discovery and Data Mining*. Springer International Publishing, 2017, pp. 685–697. DOI: 10.1007/978-3-319-57454-7_53. URL: https://doi.org/10.1007/978-3-319-57454-7_53.

[5] M.T. Hagan et al. *Neural Network Design*. Oklahoma State University, p. 469. ISBN: 0-9717321-1-6.

[6] T. Hastie and R. Tibshirani. "Classification by pairwise coupling". In: *The Annals of Statistics* 26.2 (Apr. 1998), pp. 451–471. DOI: 10.1214/aos/1028144844. URL: https://projecteuclid.org/euclid.aos/1028144844.

[7] J. Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008. ISBN: 1-60439-008-5.

[8] G. James et al. *An Introduction to Statistical Learning*. Springer New York, 2013. DOI: 10.1007/978-1-4614-7138-7. URL: https://doi.org/10.1007/978-1-4614-7138-7.

[9] D.P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/abs/1412.6980.

[10] A. Krizhevsky, I. Sutskever, and G.E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] M. Arun Kumar and M. Gopal. "Reduced one-against-all method for multiclass SVM classification". In: *Expert Systems with Applications* (May 2011). DOI: 10.1016/j.eswa.2011.04.237. URL: https://doi.org/10.1016/j.eswa.2011.04.237.

[12] M.W. Lee et al. "Decision Tree Applications for Data Modelling". In: *Encyclopedia of Artificial Intelligence*. IGI Global, pp. 437–442. DOI: 10.4018/978-1-59904-849-9.ch067. URL: https://doi.org/10.4018/978-1-59904-849-9.ch067.

[13] T. Masters. *Practical Neural Network Recipies in C*. Elsevier, 1993. DOI: 10.1016/c2009-0-22399-3. URL: https://doi.org/10.1016/c2009-0-22399-3.

[14] Y. Peng et al. "FAMCDM: A fusion approach of MCDM methods to rank multiclass classification algorithms". In: *Omega* 39.6 (Dec. 2011), pp. 677–689. DOI: 10.1016/j.omega.2011.01.009. URL: https://doi.org/10.1016/j.omega.2011.01.009.

[15] L. Rokach and O. Maimon. "Top-Down Induction of Decision Trees Classifiers—A Survey". In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 35.4 (Nov. 2005), pp. 476–487. DOI: 10.1109/tsmcc.2004.843247. URL: https://doi.org/10.1109/tsmcc.2004.843247.

[16] C. Strobl, J. Malley, and G. Tutz. "An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests." In: *Psychological Methods* 14.4 (Dec. 2009), pp. 323–348. DOI: 10.1037/a0016973. URL: https://doi.org/10.1037/a0016973.

[17] A. Tharwat. "Classification assessment methods". In: *Applied Computing and Informatics* (Aug. 2018). DOI: 10.1016/j.aci.2018.08.003. URL: https://doi.org/10.1016/j.aci.2018.08.003.

[18] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, 2000. DOI: 10.1007/978-1-4757-3264-1. URL: https://doi.org/10.1007/978-1-4757-3264-1.

[19] J. Vert, K. Tsuda, and B. Schölkopf. "A Primer on Kernel Methods". In: *Kernel Methods in Computational Biology*. The MIT Press, 2004. DOI: 10.7551/mitpress/4057.003.0004. URL: https://doi.org/10.7551/mitpress/4057.003.0004.

[20] A.P. White and W.Z. Liu. "Bias in information-based measures in decision tree induction". In: *Machine Learning* 15.3 (June 1994), pp. 321–329. DOI: 10.1007/bf00993349. URL: https://doi.org/10.1007/bf00993349.

6 Appendix

Correlation Matrix

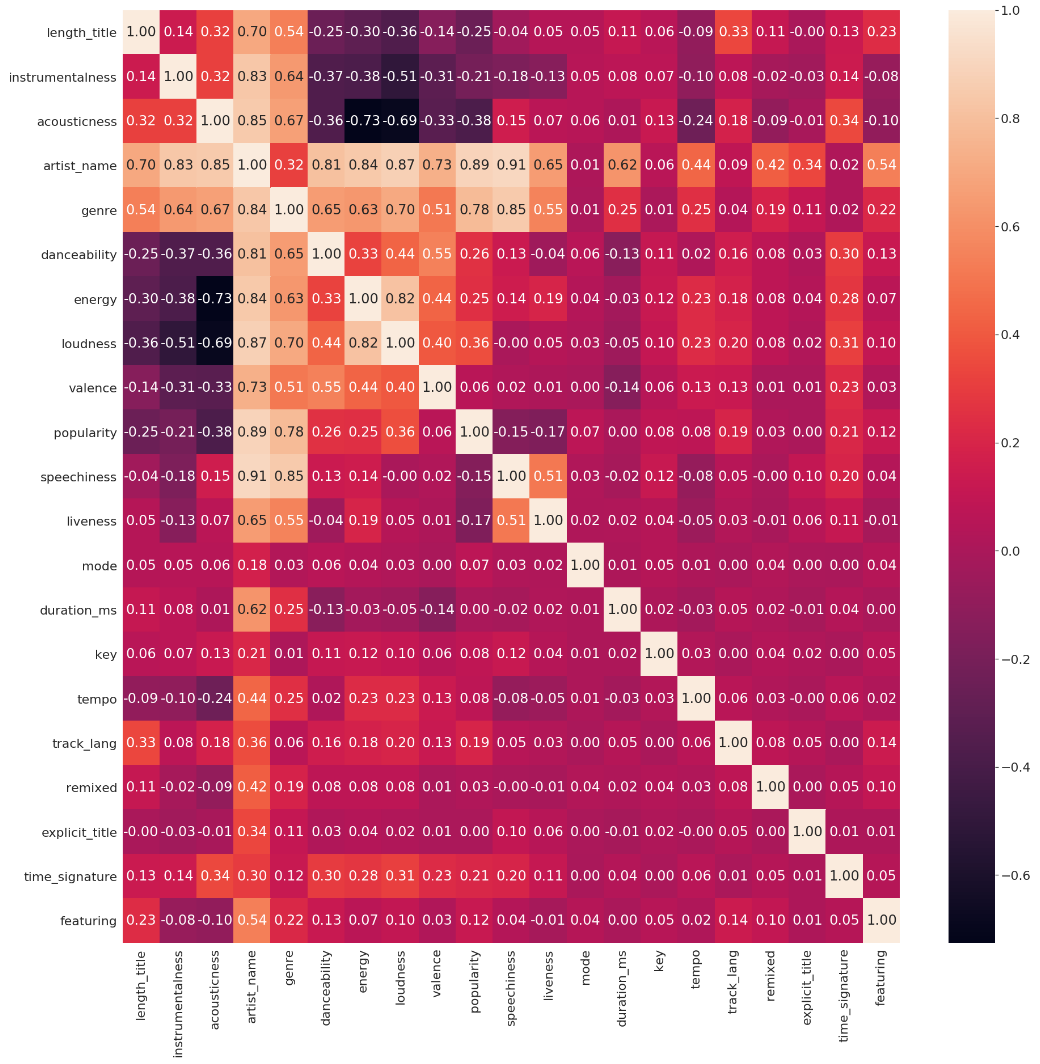


Figure 6: Heatmap for correlation between features

Neural Network

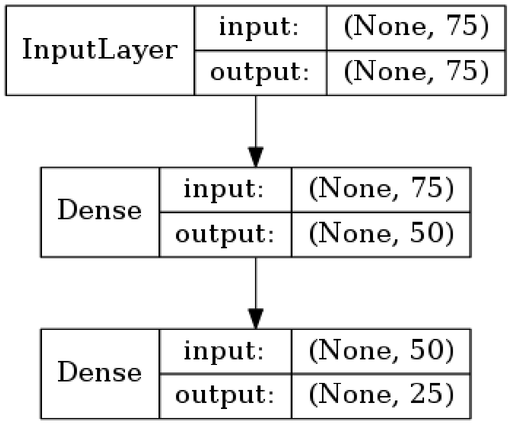
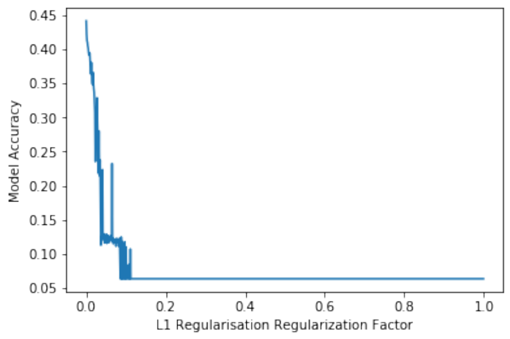
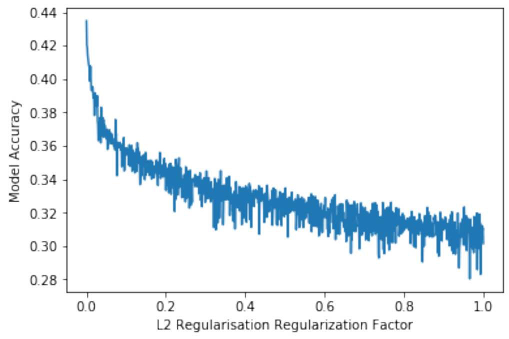


Figure 7: Neural Network Summary

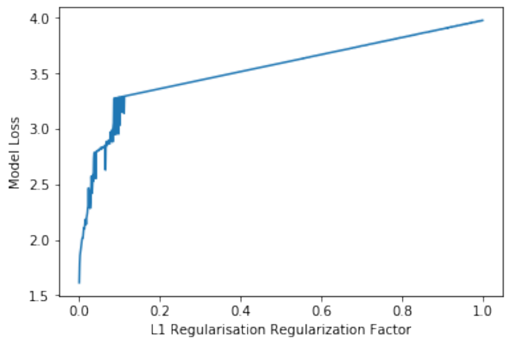


(a) Model Accuracy with L1 Regularisation with Different Regularisation Factors

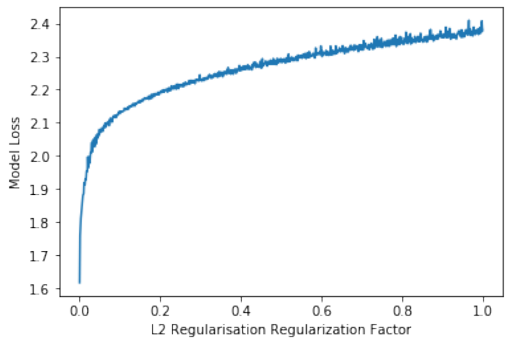


(b) Model Accuracy with L2 Regularisation with Different Regularisation Factors

Figure 8: Regularisation Plots, the Effect on Accuracy



(a) Model Loss with L1 Regularisation with Different Regularisation Factors



(b) Model Loss with L2 Regularisation with Different Regularisation Factors

Figure 9: Regularisation Plots, the Effect on Model Loss

Random Forest

Entropy and Gini

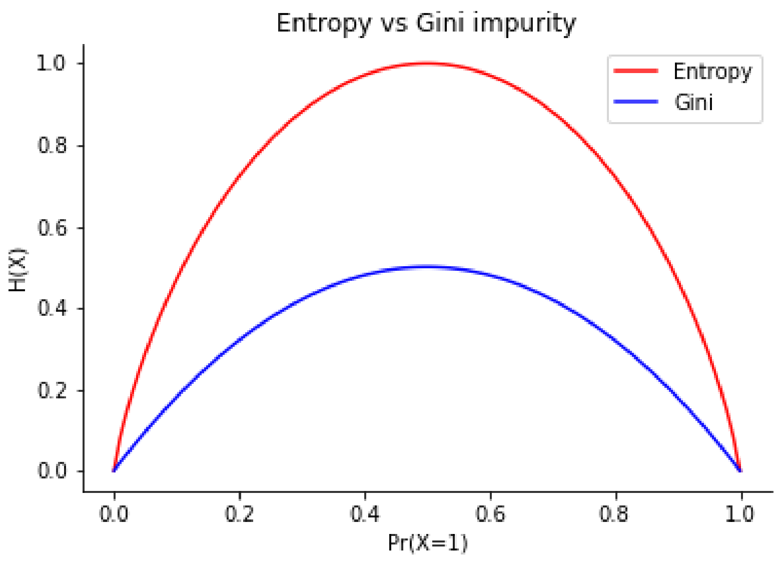


Figure 10: Gini versus Entropy

Results

Neural Network

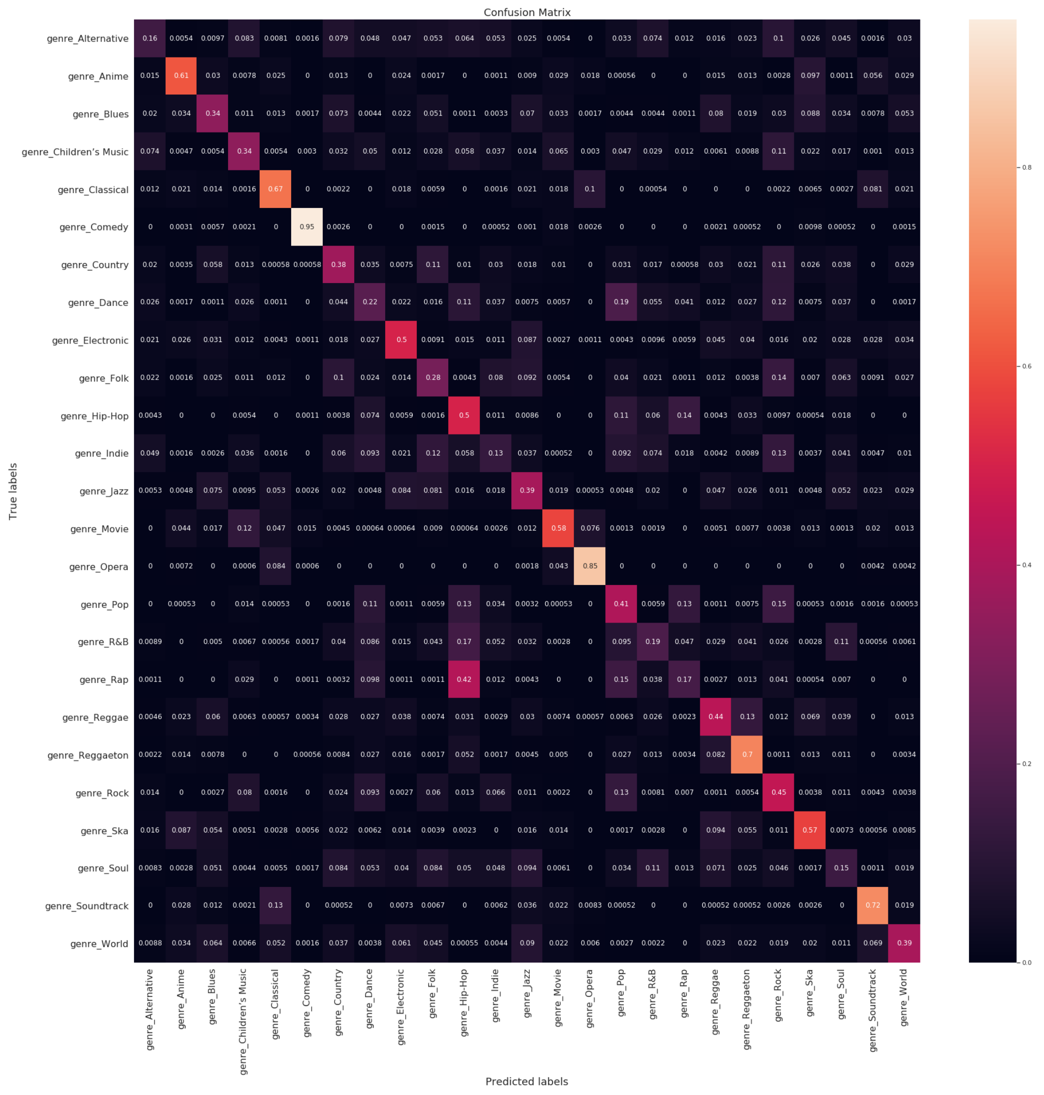


Figure 11: Confusion Matrix Neural Network

Random Forest

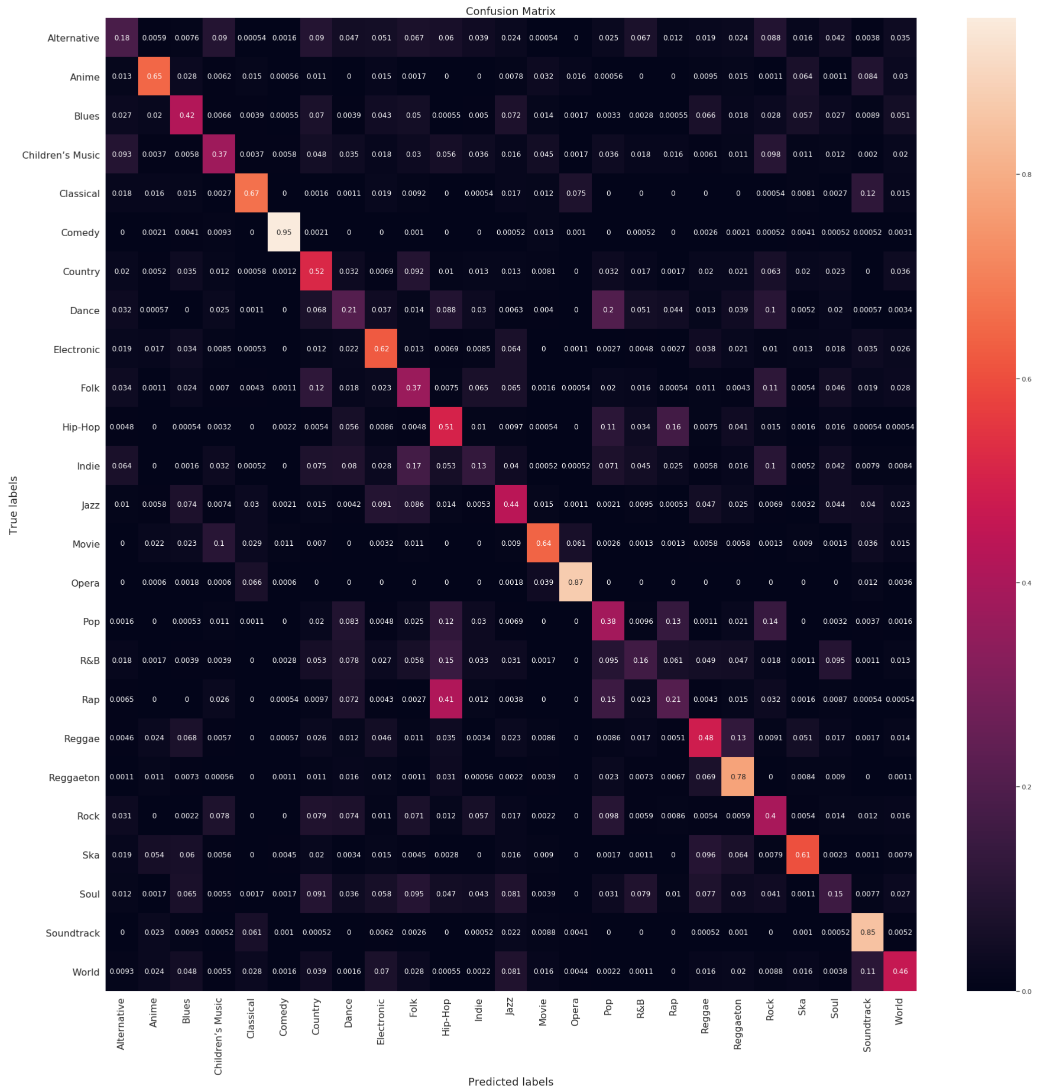
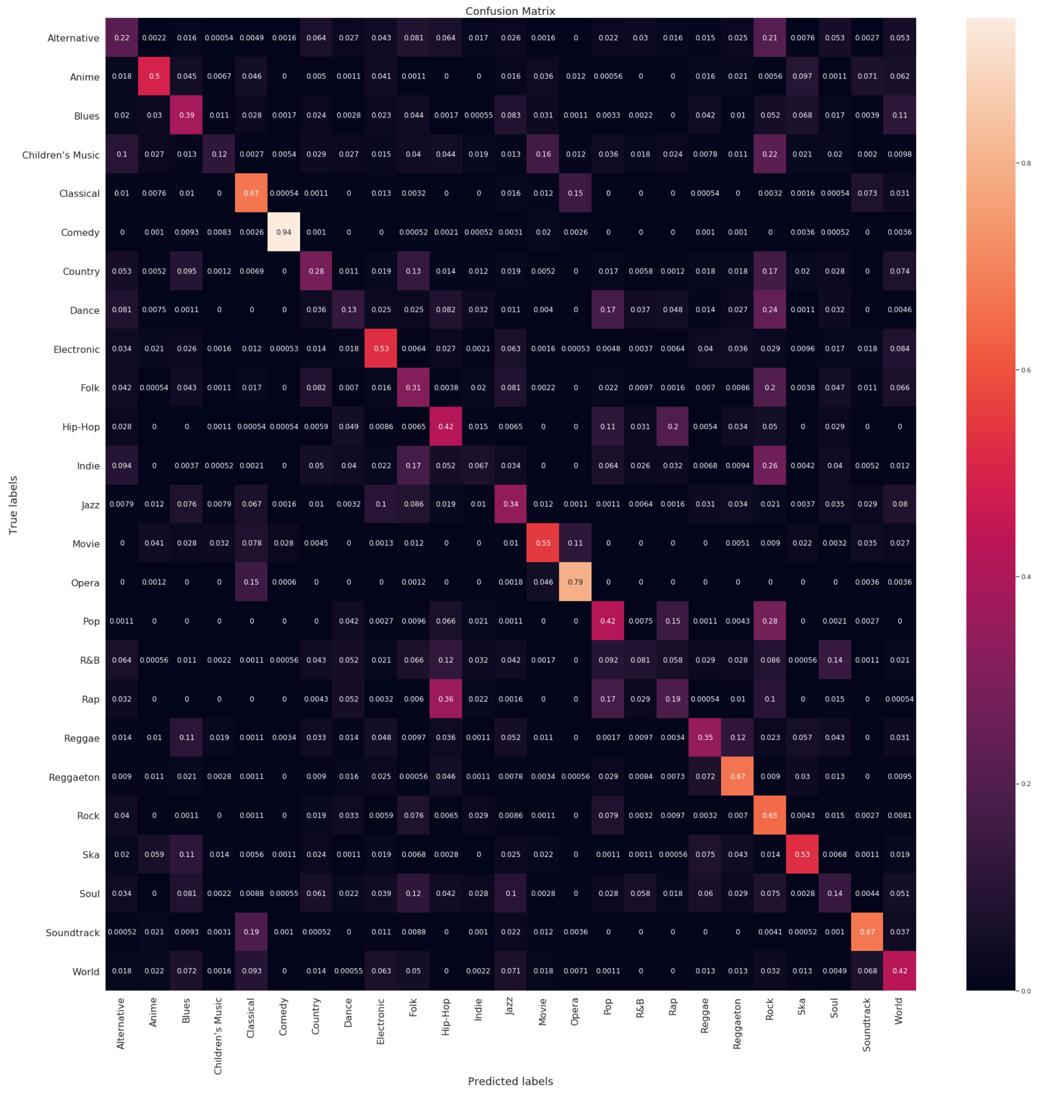


Figure 12: Confusion Matrix Random Forest

Support Vector Machine



	Alternative	Anime	Blues	Children's Music	Classical	Comedy	Country	Dance	Electronic	Folk	Hip-Hop	Indie	Jazz	Movie	Opera	Pop	R&B	Rap	Reggae	Reggaeton	Rock	Ska	Soul	Soundtrack	World
Alternative	0.22	0.0022	0.016	0.00054	0.0049	0.0016	0.064	0.027	0.043	0.081	0.064	0.017	0.026	0.0016	0	0.022	0.03	0.016	0.015	0.025	0.21	0.0076	0.053	0.0027	0.053
Anime	0.018	0.5	0.045	0.0067	0.046	0	0.005	0.0011	0.041	0.0011	0	0	0.016	0.036	0.012	0.00056	0	0	0.016	0.021	0.0056	0.097	0.0011	0.071	0.062
Blues	0.02	0.03	0.39	0.011	0.028	0.0017	0.024	0.0028	0.023	0.044	0.0017	0.00055	0.083	0.031	0.0011	0.0033	0.0022	0	0.042	0.01	0.052	0.068	0.017	0.0039	0.11
Children's Music	0.1	0.027	0.013	0.12	0.0027	0.0054	0.029	0.027	0.015	0.04	0.044	0.019	0.013	0.16	0.012	0.036	0.018	0.024	0.0078	0.011	0.22	0.021	0.02	0.002	0.0098
Classical	0.01	0.0076	0.01	0	0.67	0.00054	0.0011	0	0.013	0.0032	0	0	0.016	0.012	0.15	0	0	0	0.00054	0	0.0032	0.0016	0.00054	0.073	0.031
Comedy	0	0.001	0.0093	0.0083	0.0026	0.94	0.001	0	0	0.00052	0.0021	0.00052	0.0031	0.02	0.0026	0	0	0	0.001	0.001	0	0.0036	0.00052	0	0.0036
Country	0.053	0.0052	0.095	0.0012	0.0069	0	0.28	0.011	0.019	0.13	0.014	0.012	0.019	0.0052	0	0.017	0.0058	0.0012	0.018	0.018	0.17	0.02	0.028	0	0.074
Dance	0.081	0.0075	0.0011	0	0	0	0.036	0.13	0.025	0.025	0.082	0.032	0.011	0.004	0	0.17	0.037	0.048	0.014	0.027	0.24	0.0011	0.032	0	0.0046
Electronic	0.034	0.021	0.026	0.0016	0.012	0.00053	0.014	0.018	0.53	0.0064	0.027	0.0021	0.063	0.0016	0.00053	0.0048	0.0037	0.0064	0.04	0.036	0.029	0.0096	0.017	0.018	0.084
Folk	0.042	0.00054	0.043	0.0011	0.017	0	0.082	0.007	0.016	0.31	0.0038	0.02	0.081	0.0022	0	0.022	0.0097	0.0016	0.007	0.0086	0.2	0.0038	0.047	0.011	0.066
Hip-Hop	0.028	0	0	0.0011	0.00054	0.00054	0.0059	0.049	0.0086	0.0065	0.42	0.015	0.0065	0	0	0.11	0.031	0.2	0.0054	0.034	0.05	0	0.029	0	0
Indie	0.094	0	0.0037	0.00052	0.0021	0	0.05	0.04	0.022	0.17	0.052	0.067	0.034	0	0	0.064	0.026	0.032	0.0068	0.0094	0.26	0.0042	0.04	0.0052	0.012
Jazz	0.0079	0.012	0.076	0.0079	0.067	0.0016	0.01	0.0032	0.1	0.086	0.019	0.01	0.34	0.012	0.0011	0.0011	0.0064	0.0016	0.031	0.034	0.021	0.0037	0.035	0.029	0.08
Movie	0	0.041	0.028	0.032	0.078	0.028	0.0045	0	0.0013	0.012	0	0	0.01	0.55	0.11	0	0	0	0	0.0051	0.009	0.022	0.0032	0.035	0.027
Opera	0	0.0012	0	0	0.15	0.0006	0	0	0	0.0012	0	0	0.0018	0.046	0.79	0	0	0	0	0	0	0	0	0.0036	0.0036
Pop	0.0011	0	0	0	0	0	0	0.042	0.0027	0.0096	0.066	0.021	0.0011	0	0	0.42	0.0075	0.15	0.0011	0.0043	0.28	0	0.0021	0.0027	0
R&B	0.064	0.00056	0.011	0.0022	0.0011	0.00056	0.043	0.052	0.021	0.066	0.12	0.032	0.042	0.0017	0	0.092	0.081	0.058	0.029	0.028	0.086	0.00056	0.14	0.0011	0.021
Rap	0.032	0	0	0	0	0	0.0043	0.052	0.0032	0.006	0.36	0.022	0.0016	0	0	0.17	0.029	0.19	0.00054	0.01	0.1	0	0.015	0	0.00054
Reggae	0.014	0.01	0.11	0.019	0.0011	0.0034	0.033	0.014	0.048	0.0097	0.036	0.0011	0.052	0.011	0	0.0017	0.0097	0.0034	0.35	0.12	0.023	0.057	0.043	0	0.031
Reggaeton	0.009	0.011	0.021	0.0028	0.0011	0	0.009	0.016	0.025	0.00056	0.046	0.0011	0.0078	0.0034	0.00056	0.029	0.0084	0.0073	0.072	0.67	0.009	0.03	0.013	0	0.0095
Rock	0.04	0	0.0011	0	0.0011	0	0.019	0.033	0.0059	0.076	0.0065	0.029	0.0086	0.0011	0	0.079	0.0032	0.0097	0.0032	0.007	0.65	0.0043	0.015	0.0027	0.0081
Ska	0.02	0.059	0.11	0.014	0.0056	0.0011	0.024	0.0011	0.019	0.0068	0.0028	0	0.025	0.022	0	0.0011	0.0011	0.00056	0.075	0.043	0.014	0.53	0.0068	0.0011	0.019
Soul	0.034	0	0.081	0.0022	0.0088	0.00055	0.061	0.022	0.039	0.12	0.042	0.028	0.1	0.0028	0	0.028	0.058	0.018	0.06	0.029	0.075	0.0028	0.14	0.0044	0.051
Soundtrack	0.00052	0.021	0.0093	0.0031	0.19	0.001	0.00052	0	0.011	0.0088	0	0.001	0.022	0.012	0.0036	0	0	0	0	0	0.0041	0.00052	0.001	0.67	0.037
World	0.018	0.022	0.072	0.0016	0.093	0	0.014	0.00055	0.063	0.05	0	0.0022	0.071	0.018	0.0071	0.0011	0	0	0.013	0.013	0.032	0.013	0.0049	0.068	0.42

Figure 13: Confusion Matrix SVM