Music Genre Classification using Spotify API Metrics

Ayhan Okuyan & Emre Dönmez

Department of Electrical and Electronics Engineering, Bilkent University

{ayhan.Okuyan,emre.donmez}[at]ug.bilkent.edu.tr

Overview

This project tries to **classify the genre of a music** according to the metrics that are extracted by **Spotify API**, rather than the conventional MFCC based neural network approaches.

We have preprocessed the data, since the data contained misclassifications and the sample sizes were non-uniform. The data consisted of both **continuous and categorical features** and we have **selected our features** based on the implemented methods.

We have used hybrid **k-Nearest Neighbors (kNN), Random Forest,** and **Multilayered Perceptron (MLP)** algorithms for classification. We have used a three-way split in each of our algorithms to optimize the algorithm parameters.

Dataset

We have used on the **Spotify Tracks DB database** [1]. The dataset has around **232 thousand songs** which contain **26 different genres** and **16 distinct features** to learn with.

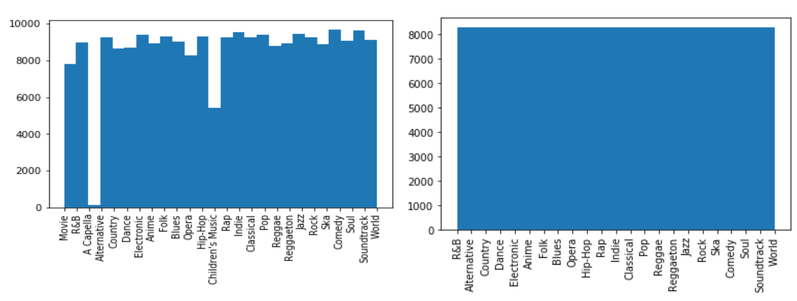
The reasons for choosing this dataset are the following: It has enough samples for training, validation, and testing; it has many features to choose from and train; it is suitable for the algorithms that we are going to use.

Most importantly since the features presented in the dataset such as danceability or instrumentality are very **abstract**, also the motive under the project is using these abstract features in order to decide on the genre of a song would be more **unbiased.**

In the dataset, we see that there are both discrete and continuous features and their scales differing heavily. The most important part of this data are the features, **"danceability", "instrumentalness", "liveness", "energy", "valence" and "acousticness"** since these are features that are extracted by Spotify for their own recommender systems.

Preprocessing Data

The data was distributed highly uneven between classes.



The distribution of data samples before and after preprocessing

We have fixed the misclassifications and removed the underrepresented classes. Then down sampled the data randomly to the same size for each class.

Multilayered Perceptron (MLP)

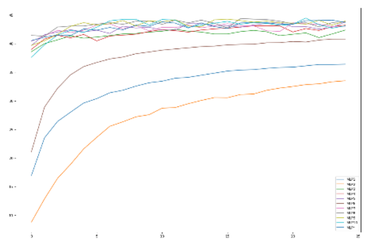
We have implemented a hard-coded generic neural network framework to train multiple different structures all within the same code structure. The framework is used to create networks with

- Various number of hidden layers
- Various hidden units in each layer
- Initialization of weights using selectable normal distribution parameters.
- Four different activation functions for each layer.
  - Sigmoid
  - Tanh
  - ReLU
  - Softmax (with Cross Entropy Loss)
- Two different loss functions.
  - Cross-Entropy (CE)
  - Mean Squared Error (MSE)
- Generic learning rate

Optimized the model using **Stochastic Mini-Batch Gradient Descent** update rule to find the optimal architecture.
Ten various architectures are observed. Most of the architectures saturated around the same range, and we have taken the most successful one. The important patterns found while iterating are given as.

- ReLU is the choice of activation rather than bipolar options.
- Output layer should be Softmax.
- Cross Entropy loss should be used.
- He initialization [2] should used for ReLU networks.

The validation accuracies over training epochs are given below with the chosen architecture.

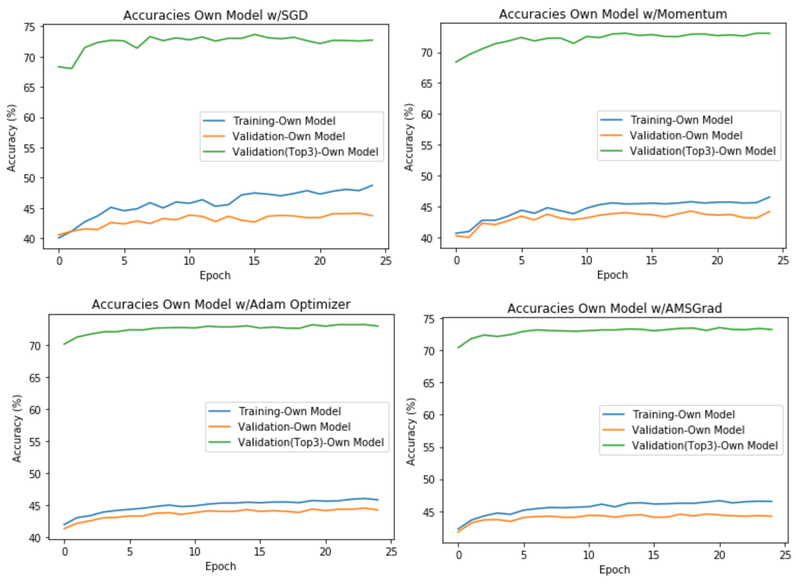


```
Layer 0: Input Dim: 11, Number of Neurons: 500
 Activation: relu
Layer 1: Input Dim: 500, Number of Neurons: 300
 Activation: relu
Layer 2: Input Dim: 300, Number of Neurons: 150
 Activation: relu
Layer 3: Input Dim: 150, Number of Neurons: 65
 Activation: relu
Layer 4: Input Dim: 65, Number of Neurons: 23
 Activation: softmax
```

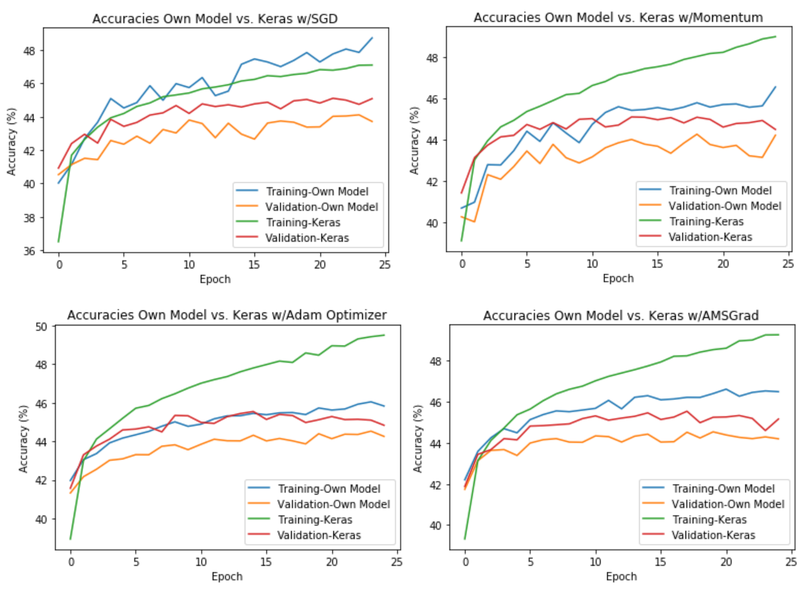
Then we implemented four optimizers:

- SGD (Stochastic Gradient Descent)
- SGDM (Stochastic Gradient Descent with Momentum)
- Adam
- AMSGrad

We have trained the network with all these optimizers to see which performed best.



We compared the algorithms written with their state of the art Keras implementations.



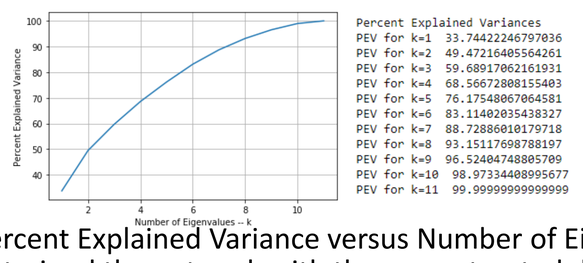
Results

Optimization Algorithm	Learning Rate	Other Parameters	Epochs	Training Period (min)	Test Accuracy
SGD	0.1	-	25	5	43.80%
SGDM	0.1	$\alpha = 0.9$	25	6	44.21%
Adam	0.001	$\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 10^{-8}$	25	7	43.88%
AMSGrad	**0.001**	$\boldsymbol{\beta_1 = 0.9}$ $\boldsymbol{\beta_2 = 0.999}$ $\boldsymbol{\epsilon = 10^{-8}}$	**25**	**6**	**44.44%**

Overall, we observed a test accuracy around 44%. AMSGrad was the best optimizer for the task at hand with 44.44% test accuracy.

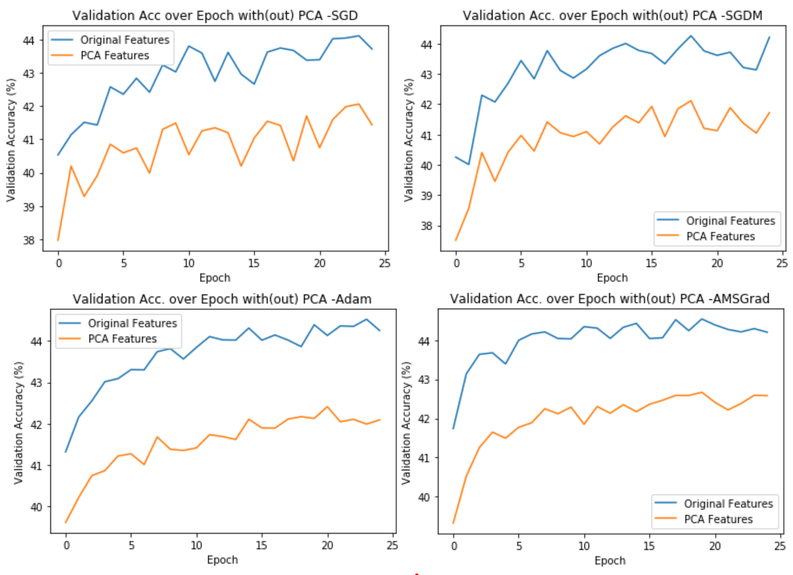
Principle Component Analysis (PCA)

We have also experimented if using PCA on the dataset would improve neural network's performance. Hence, we applied PCA to the data and tested on the selected architecture.
We have decided to take enough principle components to preserve 90% of original data variance. ***(k=8)***



Percent Explained Variance versus Number of Eigenvalues

We trained the network with the reconstructed data. The validation accuracies of the PCA and non-PCA data are compared for each optimizer.



Results

Optimization Algorithm	Test Accuracy (PCA)	Test Accuracy (non-PCA)
SGD	41.50%	43.80%
SGDM	41.65%	44.21%
Adam	41.98%	43.88%
AMSGrad	**42.65%**	**44.44%**

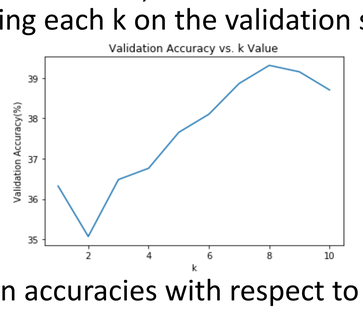
k-Nearest Neighbors (kNN)

We have implemented a kNN algorithm that would let us take advantage of both the categorical and continuous features. Hence, we deployed a hybrid kNN that would use the Mean Squared (Euclidean) distance for continuous features and a parameter-controlled unity distance for the categorical features.

$$d_1(x_{i,n}, x_{j,n}) = \begin{cases} 0 \; if \; x_{i,n} = x_{j,n} \\ \propto^2 \; else \end{cases} \qquad d_2(x_{i,n}, x_{j,n}) = x_{i,n} - x_{j,n}$$

$$d(x_i, x_j) = \sum_{l=0}^{n} ||d_t(x_{i,l}, x_{j,l})||^2 \; , where \; t = \begin{cases} 1 \; if \; lth \; feature \; is \; discrete \\ 2 \; if \; lth \; feature \; is \; continuous \end{cases}$$

To optimize the parameters, we have first found the best k-value by evaluating each k on the validation set.



Validation accuracies with respect to k values

Then we chose k as 8 and then optimized the alpha values according to a specified discrete grid. We used grid-search with values [0.2,0.4,0.6,0.8] to find the best set of alpha values. The best values are,

$$(\propto_1, \propto_2, \propto_3, k) = (0.8, 0.2.0.2.0.8)$$

The total search took 7.4 hours to complete. The results were better than the other algorithms, since we were able to implement the **Artist Name** feature only on this algorithm. The validation accuracy for the optimized values were **68.24%** and the test accuracy was **67.18%.**

Random Forest

Decision Tree is an algorithm that has the advantage of classifying discrete and categorical features as well as continuous features without the need of complicated preprocessing. However, it can be very biased to the training data since it creates splits according to the training data. In order to overcome this, we have used the Random Forest algorithm which consist of many low biased weak decision Trees. Since we are doing a classification task, the split criteria of our trees' in the random forest algorithm is dependent on calculating the Information Gain by the Gini Index and of all the predictions each tree in our Random Forest does, we should get the most occurring class as our final prediction.

$$\boldsymbol{IG = I^{Gini}(D_{parent}) - \frac{N_{left}}{N_{parent}} I^{Gini}(D_{left}) - \frac{N_{right}}{N_{parent}} I^{Gini}(D_{right})}$$

$$\boldsymbol{I^{Gini}(D) = 1 - \sum_{i=1}^{N_{classes}} p_i{}^2}$$

Forest Number	Sample Size	Number of Trees	Number of Features	Maximum Depth	Minimum Leaf Size	Validation Accuracy
1	80	40	5	10	5	18.7%
2	80	80	5	10	5	19.1%
3	120	40	5	10	5	19.2%
4	**120**	**80**	**5**	**10**	**5**	**20.96%**
5	40	40	5	10	5	18.6%
6	40	80	5	10	5	18.3%
7	120	80	3	10	5	20.3%
8	120	80	8	10	5	20.3%
9	120	80	5	20	2	19.3%
10	120	80	5	5	2	13.6%
11	120	80	5	20	10	20.94%
12	120	80	5	5	10	19.5%
13	120	80	5	10	2	15.4%
14	120	80	5	10	10	20.94%

Results

We have choosen the best model which is the 4th model. The algorithm takes about 100 seconds to run including the prediction part for the best model. It gets a test accuracy of 19.2%.

Conclusion

Overall, the algorithm that worked the best was kNN due to the ability to use a highly correlated feature like Artist Name. However, the MLP algorithm also worked significantly good with overall 40%, nearly 10 times the random guess. On the other hand, the random forest did not work as well as we thought since forest algorithms are usually good when dealing with categorical features.

In order to further improve our project, we could have used some form of feature extraction on the data. This could have improved the general correlation between the features and labels.

We have chosen to implement such a project because we believe that the genre classification of music is not only subjective to individuals and it is an issue that needs more attention. Accomplishing such a project perfectly with a higher accuracy would become an important achievement for the music industry.

Adding additional useful features to this kind of a dataset such as the **frequency or vocal tone** could also increase our performance significantly.

References

[1] Spotify DB Tracks, Kaggle, https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db/version/3, [Accessed: 7-Nov-2019]

[2] Kakaraparthi, "Xavier and He Normal (He-et-al) Initialization," Medium, 29-Sep-2018. [Online]. Available: https://medium.com/@prateekvishnu/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528. [Accessed: 09-Nov-2019].