What am I listening to?

Automated Genre Detection with LSTM Recurrent Neural Networks

Final Report
INFO290DM: Data Mining & Analytics
UC Berkeley School of Information
Spring 2016

by
Proxima Dasmohapatra , Richa Prajapati, Keshav Potluri, Konstantin Lackner

Introduction

The barrier for creating music has been lowering over the last few decades and as a result we have an increasing number of songs being created each year. With the introduction of cloud technologies and internet growth, music is more freely available than before. Services like Spotify and Pandora, have huge collections of songs available for its users. With this increase in data, it is important for them to classify these songs based on genres to provide better searchability and suggestions to the end user based on the user preference. But as the size of the song collection grows, the process of classification of songs manually becomes tedious. We hence were really interested in using machine learning and classification techniques to classify the songs. The main goal for this project was to implement a long-short term memory (LSTM) Recurrent Neural Network (RNN) that predicts the genre of a song based on features that are extracted from the song. We also aim to identify the features that are useful in genre detection.

A song or an audio file contains many elements in its composition like different instruments, different pitches of the singer's voice, the number of beats per minute etc. These elements are what define the genre of the song and in order to classify the song based on its genre, these elements need to be described in terms of audio features which can be used by the machine learning models. The audio features that are the most useful are Mel-Frequency Cepstral Coefficients (MFCC), Spectral Center, Chroma and Spectral Roll. Majority of songs within one genre share similar features, although this may not be true for every song.

For the classification, we used LSTM RNNs. This was mainly attributed to two reasons:

1. LSTM RNNs have proven to be very powerful in many machine learning tasks, such as speech recognition, image recognition, or playing Atari Games[1].
2. The audio features we are dealing with create timeseries data, where the timesteps are highly interdependent. LSTM RNNs are designed to capture interdependencies, even between long-distant timesteps, which make them ideal for our application.

After researching on different datasets we found the following options:

- Million Song Dataset: A dataset of one million songs with metadata information (e.g. Artist Name, Genre, etc.) and audio features information (timbre, pitch, loudness).
- GTZAN Genre Collection: The dataset consists of ten genres and 100 audio files per genre of approx. 30 seconds length.

Million song dataset did not contain the features we wanted to explore and hence we decided to create the audio features on our own. Hence we opted for the GTZAN Genre Collection.

[1] http://arxiv.org/pdf/1312.5602v1.pdf

The report is structured as follows: First we will explore the GTZAN Genre Collection and explain why this dataset was ideal for the purposes of this project. Then there is an overview of the approach we took towards the project followed by a detailed description of every team member's individual work. Lastly we will present our results, the impact of the project and future steps in the conclusion.

Dataset

The GTZAN dataset consists of 1000 audio files (1.2 GB) with 10 genres of songs. We decided to use the GTZAN dataset for the following reasons:

1. **Easy access**: The dataset has each song 30 seconds long, with its genre is encoded in the file name. It is publicly available and in a file format that is easily supported. This is much more accessible than the Million Song Dataset (~300GB) in hdf5 format.
2. **Flexibility to explore new features**: If we would use the MSD dataset we would be limited to the audio features given in the dataset and would not be able to extract features on our own. But with GTZAN we could extract our own features.
3. **Consistency**: By just using the audio files and analyze the audio features our selves we can ensure consistency in the way the audio files have been analyzed.
4. **Equal distribution across genres**: The GTZAN dataset contains an equal number of audio files for each genre, hence ensuring that the model doesn't get trained on any one majority class.

We looked at research papers that talk about classification of music by genre to identify audio features that were more likely to help distinguish between genres. Since we did not have domain knowledge in the area of audio signal processing, we decided this was the best approach.

Four features that were identified to be most helpful in genre classification were:

a. MFCC (Mel-Frequency Cepstrum Coefficients): In sound processing, the **mel-frequency cepstrum** (**MFC**) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. **Mel-frequency cepstral coefficients** (**MFCCs**) are coefficients that collectively make up an MFC.

b. Chroma: In the music context, the term chroma closely relates to the twelve different pitch classes. Chroma-based features, which are also referred to pitch class profiles, are a powerful tool for analyzing music whose pitches can be meaningfully categorized (often into twelve categories). One main property of chroma features is that they capture harmonic and melodic characteristics of music, while being robust to changes in timbre and instrumentation.

c. Spectral Roll: The spectral "slope" of a natural audio signal is its tendency to have less energy at high frequencies. One way to quantify this is by applying linear regression to the Fourier magnitude spectrum of the signal, which produces a single number indicating the slope of the line-of-best-fit through the spectral data.

d. Spectral Center/Centroid: The **spectral centroid** indicates where the "center of mass" of the spectrum is. Perceptually, it has a robust connection with the impression of "brightness" of a sound.[1] It is calculated as the weighted mean of the frequencies present in the signal, determined using a Fourier transform, with their magnitudes as the weights:[2]

$$Centroid = \frac{\sum_{n=0}^{N-1} f(n)\,x(n)}{\sum_{n=0}^{N-1} x(n)}$$

Introduction to Approaches

The project was planned in the following major steps and the work was distributed uniformly across these steps:

1. Find an appropriate data set
2. Research on Audio Analysis, Feature selection, LSTM and Hyperparameters
3. Extract Audio features from audio files and tag genres
4. Implement LSTM RNN
5. Explore and optimize Hyperparameters
6. Model Optimization : Grid Search, Cross Validation & Ensembling
7. Prediction on Test Set

Richa identified the project research topic and did extensive research and focussed on improving the accuracy of the model. She mainly focussed on the following:

1. **Dataset search :** Searching for publicly available data for the project.
2. **Research** : Searching and studying literature for:
    a. Genre detection methodology
    b. Identification of relevant audio features and their extraction
    c. Identification of learning mechanism best suited for project requirements
    d. Identification of most important hyperparameters
3. **Hyper Parameter Exploration :** Search best hyper parameters values via grid search and cross validation
4. **Analysis of Models:** Creation of visualization of results via confusion matrix and analyzing the results.

5. **Improving Accuracy :** Using ensembling to improve results.

Konstantin primarily worked on implementing the code for:

1. **Audio feature extraction** by generating the identified audio features from the audio files in the GTZAN database using Librosa
2. **Data Preprocessing** by converting the data from 3D array into a format that is accessible for the Neural Network
3. **Implementation of LSTM Recurrent Neural Network (RNN)** by providing a functional model of a LSTM RNN in order to be able to explore the hyperparameter space

Proxima focussed on the following:

1. **Installation** of required packages and tools for the project. This also included setting up the AWS servers.
2. **Research** on literature for relevant features and hyperparameters.
3. **Searching** for a publicly available dataset that can be used for analysis.
4. **Exploring for hyperparameters**: number of epochs and hidden layers. This involved varying the number of hidden layers and the number of nodes in each hidden layer.

Keshav primarily worked on the following:

1. **Dataset search & audio analysis tool search :** Explored different data sets for Audio analysis and explored tools like Yaafe for audio feature extraction.
2. **Research** on Hyper-parameters for Audio analysis
3. **Grid Search for Hyperparameters**: Optimized the Epochs and Learning Rate using grid search. **Cross Validation** to ensure stability in results.
4. **Ensembling :** weighted model and majority vote model.

Individual Work

Konstantin Lackner

My main contributions to the group project are:

1. **Audio feature extraction**: Generate the identified audio features from the audio files in the GTZAN database
2. **Data Preprocessing**: Bring the data into a format that is accessible for the Neural Network
3. **Implementation of LSTM Recurrent Neural Network (RNN)**: Provide a functional model of a LSTM RNN in order to be able to explore the hyperparameter space

Audio feature extraction

The reason to use audio features instead of the raw audio file is to identify the components of the audio signal that help classify genres and to discard all other information (e.g. noise) of the signal. By looking at research papers that are dealing with Genre Classification we identified the following audio features as appropriate input parameters for the LSTM RNN:

- Mel-frequency cepstrum coefficients (MFCC) (Dimensionality: #timesteps x 13)
- Spectral-Center (Dimensionality: #timesteps x 1)
- Chroma (Dimensionality: #timesteps x 12)

- Spectral Roll (Dimensionality: #timesteps x 1)

These features are extracted with “Librosa”, a python package for music and audio analysis which provides easy access to the features above. For each song the four features are extracted as well as their genre tag, which is encoded in the song’s filename. One characteristic about the audio-features is that they return a time-series, i.e. they take a number of samples (specified by “hoplength”, which is usually 512) and calculate the feature for every time slice. The number of samples in a digital audio file is specified by:

$$\#\ of\ samples = \ Length\ in\ seconds \cdot Sampling\ Rate$$

In our case the length of the audio clips is 30 seconds and the sampling rate is 22050 $\frac{1}{s}$ which leads to 661.500 samples per clip. Since the audio features are calculated for time slices of 512 samples, the number of timesteps for each features is:

$$30s \cdot 22050 \frac{1}{s} \div 512 = 1291\ timesteps$$

As a result, for every song we have the audio features and their genre tag accessible, which can be seen in the table below. The entries for the features show their dimensionality.

Song	MFCC	Spectral-Center	Chroma	Spectral Roll	Genre
1	1291 x 13	1291 x 1	1291 x 12	1291 x 1	Classical
2	1291 x 13	1291 x 1	1291 x 12	1291 x 1	Country
…	…	…	…	…	…

Data Preprocessing:

In order to make the data accessible for the LSTM RNN when using the framework “Keras”, the input matrix need to be of size (#samples, #timesteps, #input dimension) and the target matrix of size (#samples, #output dimensions). For the input matrix all audio features have been stacked onto each other so that for every song there is one feature-matrix of size (1291, 27) = (#timesteps, #input dimension) which will feed forward through the network. The genre tag has been transformed into a binary representation as it was discussed in one of the Labs. This is done by creating a matrix with six columns (one for each genre) and denoting a “1” in the column of the belonging genre of the audio clip, and “0” otherwise. As a last step, we split the dataset into 70% Train Set, 20% Development Set and 10% Test Set, which provides us with the necessary data in the right format to train and test the LSTM RNN.

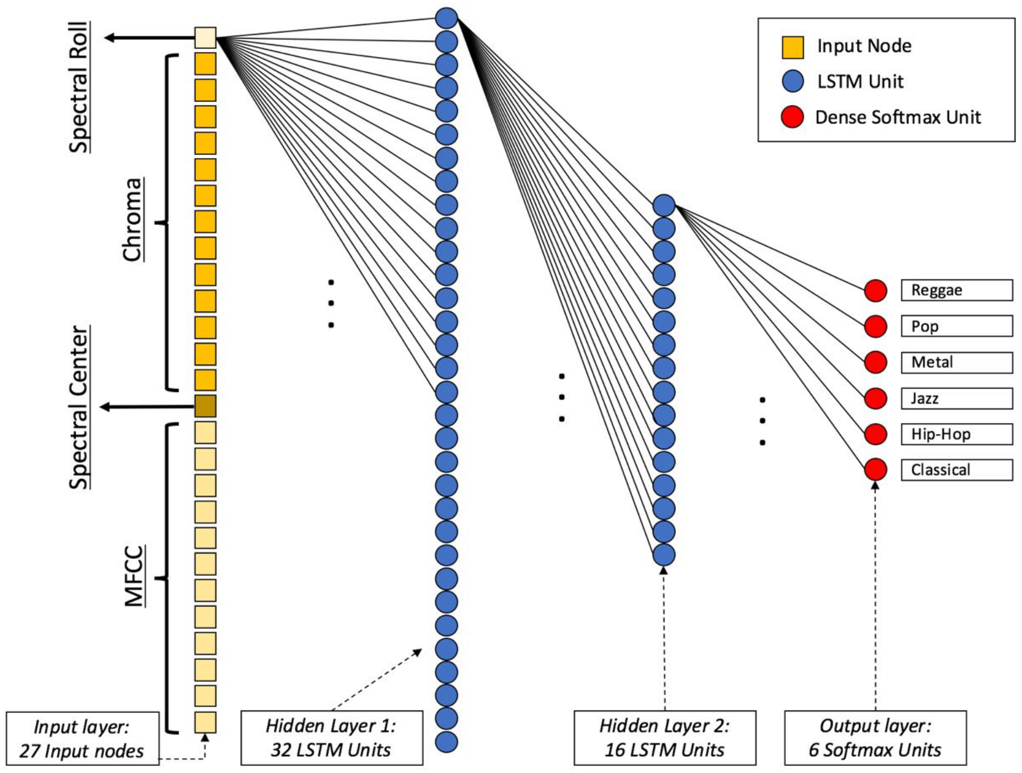
	# of Songs	Input-Matrix	Target-Matrix
Train Set	420	420 x 1291 x 27	420 x 6
Dev Set	120	120 x 1291 x 27	120 x 6
Test Set	60	60 x 1291 x 27	60 x 6

Implementation of LSTM RNN:

We decided to use a LSTM RNN since it can capture interdependencies between long-distant timesteps. The input-dimension of the network is given by the dimensionality of all audio-features combined (27) and the output-dimension by the number of genres (6). The output layer has been designed to be a regular Dense layer with “Softmax” as the activation function, as it was discussed in the Deep Learning Lab. All hidden layers consist of LSTM units. The network has been implemented with the Python framework "Keras" and is built with the following properties:

- Activation Function for LSTM Units: Sigmoid
- Optimizer: Adam
- Loss function at the output layer: Binary Crossentropy

The following shows an example of one possible network architecture with two hidden layers and 32 LSTM units in the first, and 16 LSTM units the second hidden layer.


Spectral Roll
Chroma
Spectral Center
MFCC
Input Node
LSTM Unit
Dense Softmax Unit
Reggae
Pop
Metal
Jazz
Hip-Hop
Classical
Input layer: 27 Input nodes
Hidden Layer 1: 32 LSTM Units
Hidden Layer 2: 16 LSTM Units
Output layer: 6 Softmax Units

Proxima DasMohapatra

My contributions to the project were:

i. Installing required packages and tools for the project. This included finding out dependencies and installing Librosa, the tool required for audio analysis, a backend for Librosa named Theano, scikits.samplerplate for efficient audio resampling.

ii. Researching on literature for relevant features and hyperparameters. *Note: Research was split between team members to ensure that we covered the breadth of research done in the field of music classification.*

iii. Looking for a publicly available dataset that can be used for analysis. We initially planned on scraping 30 second audio previews for songs listed in the Million Songs Dataset, however due to copy right issues, we decided on using a publicly available dataset that was not copyright protected.

iv. Exploring for hyperparameters: number of epochs and hidden layers. This involved varying the number of hidden layers and the number of nodes in each hidden layer. Exploration of number of epochs started once the number of hidden layers and nodes per layer were finalized. We chose the combination that gave the maximum accuracy for the above. *Note: Exploration of these hyperparameters were split between team members owing to large computation time.*

i. Installation of packages and tools required for the project: The installations for this project were done in two rounds. First, the tools and packages were installed in the team's local machines. However, owing to the large computation times, we decided to set up two AWS servers to aid us with processing power. Both rounds of installations had different sets of tools required to be installed, owing to the state and OS of the machines. The major tools and packages required for the project were:
   a. Librosa: An audio analysis tool with Python integration that could be used to extract required data features. Marysas was another tool that was attempted to be used, however Librosa turned out to be better for Python integration.
   b. Theano: The backend used by Librosa to perform audio waveform analysis.
   c. Keras: The library used to implement LSTM

   Since the AWS server uses a hybrid flavor of Unix operating system, installation of a GPU driver posed a challenge. Due to time constraints, the team decided on running models without a GPU, since installation time would have cut into model processing time.

ii. Researching on literature for relevant features and hyperparameters: Due to the lack of domain knowledge in audio analysis, we decided to look at research papers for guidelines on relevant features for music classification and most important hyperparameters in LSTM. According to (McKinney, 2004), audio features are divided into two major categories – static features and psycho-acoustic features. The first category of features constitutes calculated measures such as MFCC, Spectral Roll, Chroma and Spectral Center. These features can be extracted using mathematical operation on the audio waveform. Every audio song is arbitrarily split into two waveforms, which depends on the composer. However, only one of them is perceptible to the human sense of sound. The second category of features, psycho-acoustic features deal with features that humans can perceive, for example roughness, sharpness, etc. Since extraction of such features required knowledge of the domain and algorithms used to derive them, we decided on using more standard features, which were the static features.

iii. Publicly available dataset for analysis: Initially, the team planned on scraping 30 second audio clips on the song list available through the Million Song Dataset, however it relied on using 7digital for scraping music clips. However, 7digital has changed its policy, and no longer allows collecting 30 second audio clips using their API. Hence, we decided on an alternate dataset, the GTZAN dataset. This dataset contains 100 songs for 10 genres, that is available for public use.

iv. Exploration of hyperparameters: Literature on LSTM suggested the probable independence of hyperparameters, which informed our decision of searching through hyperparameters in a sequential manner. Network size of the LSTM, which consisted of varying the number of hidden layers and number of nodes per layer was searched for first to find the best fit. Owing to the large computation time of the LSTM model, this was split within the team and AWS servers in order to make the hyperparameter search faster. I varied the number of hidden layers between 1 and 4, with a constant number of nodes per hidden layer (30 nodes in the first hidden layer, and 10 nodes in each consecutive layers) to observe the impact on accuracy:

Hidden layers	Epochs	Batch size	Learning rate	Accuracy
1 (30)	500	128	0.001	0.45625
2 (30,10)	500	128	0.001	0.49375
3 (30, 10, 10)	500	128	0.001	0.2875
4 (30, 10, 10, 10)	500	128	0.001	0.225

We noticed a drastic drop in accuracy with increase in hidden layers beyond two. This could have been due to the model overfitting on the training data. Hence, we decided to maintain two hidden layers for the LSTM and try varying other hyperparameters to achieve higher accuracy.
Searching for epochs: After some variations to the above model, we chose a LSTM network with two hidden layers with 32 and 16 nodes per layer with a dense output layer to search for the best number of epochs that maximize the accuracy of the model:

Hidden layers	Epochs	Batch size	Learning rate	Accuracy
2 (32,16)	1050	128	0.0065	0.3660714286
2(32,16)	1000	128	0.0065	0.875
2(32,16)	900	128	0.0065	0.3053571429
2(32,16)	910	128	0.0065	0.3553571429
2(32,16)	920	128	0.0065	0.3392857143
2(32,16)	930	128	0.0065	0.2482142857
2(32,16)	940	128	0.0065	0.6125
2(32,16)	950	128	0.0065	0.48125

In few of our earlier runs, we gained the best accuracy by an epoch size of 1000. Hence, we focused one part of the hyperparameter space search for epochs around 1000.

Richa Prajapati

My main contribution to the project were:

1. Finding the project topic and relevant data set
2. Research on:
    a. Genre detection methodology
    b. Identification of relevant audio features and their extraction
    c. Identification of learning mechanism best suited for project requirements
    d. Identification of most important hyperparameters
3. Exploration of best hyper parameters values via grid search and cross validation
4. In-depth analysis and visualization of results via confusion matrix
5. Exploration of relevant ensembling methodologies

Finding the project topic
As part of our Data Mining final project, my team and I wanted to do something unconventional and challenging yet something that fit in the scope and duration of the class project. I scoured through various sites offering machine-learning datasets (including UC Irvine Machine Learning Repository, IAPR, KDNuggests, Kaggle etc.) and finally landed with the Million Song Dataset. Based on the Million Song Dataset, I suggested song genre detection as the topic for our project which was finally accepted by the team.

Research on literature for genre detection methodology, identification of relevant audio features and audio feature extraction
Since no one in my team possessed music analysis domain knowledge, the next step was to figure out how to detect song genres. I took the lead in identifying various research papers that had dealt with this problem. The papers are listed below for reference. Based on these papers, I listed down which audio features we would concentrate on to detect song's genre. We realized that the Million Song Dataset did not contain these features; it just contained metadata and features such as pitch, timbre etc. that didn't meet our requirements. Hence, I recommended extracting the audio features from actual songs using audio extraction software such as "MARSYAS". We later ended up using Librosa because of its Python support. I also suggested using GTZAN dataset because it provided publicly available 30 second clips of songs human-curated with genre tags – perfect for our research.

Research on identifying best machine learning algorithm
Next, I researched to identify which machine-learning algorithm would be best for genre detection. Though we came up with various algorithms to analyze time series audio data, such as Dynamic Time Warping, I suggested going with LSTM because of its high accuracy and reliability with time-series audio generation as well as classification; this was further corroborated by Konstantin (who had earlier worked on music generation) and Zachary, our professor.

Research on literature for LSTM and its hyper parameters relevant for song genre detection
Once, we decided to go with LSTM, we had to find out which hyper parameters to focus on and how to tune them in order to get the best results. The concepts learnt in class and my research on LSTMs told me that the most important hyper parameters for 3264engre detection were the number of hidden dimensions, the size of the input/output layer, and the learning rate – which we decided to explore first. We also decided to include epochs in our parameter search because the various class assignments over this semester had shown that epochs can have a significant effect on the final results. The research suggested that with LSTMs, all the hyper parameters could be tuned independently. Since running LSTM was both memory and time-consuming, my team and I each decided to work on tuning different hyper parameters individually and combine the hyper parameter values giving the best results in the final model.

Exploration of hyper parameters via grid search and cross validation
Though I contributed in identifying the number of hidden layers as well the units within each hidden layer that gave the best result, I was one of the persons responsible for finding the value of learning rate that gave the best results. For this, I found grid-search to be very helpful in identifying the best learning rate in a systematic way. Keshav's exploration of learning rate showed that the range between learning rate of 0.001 and 0.01 gave the best

results. I further refined the grid search by exploring the 0.001 and 0.01range of learning rate further. By setting the learning rate to the range's mid-point of 0.005, the accuracy further increased to 71.875%. Hence, I decided to narrow down the grid search for learning rate to range from 0.001 to 0.01 in steps of 10. The accuracy trend showed that I should further drill down the learning rate range from 0.006 to 0.007 in steps of 2 and finally from 0.0065 to 0.0067 in steps of 1. For consistent results, I did a manual cross-validation by running the code multiple times and seeing that the accuracy did not vary by a larger margin. The cross-validation showed me that all 3 learning rates of 0.0065, 0.0066 and 0.0067 gave consistent and similar results at 500 epochs with other hyper parameters constant.

<u>In-depth analysis and visualization of results</u>

Because all 3 LSTMs were giving similar accuracy, I wrote the code for confusion matrix to quickly and easily analyze classification performance of the 3 different LSTMs on different song genres in a visual format. The analysis of confusion matrix of the 3 LSTMs at 500 epochs showed that while the 3 different LSTMs showed similar accuracy, they identified different songs correctly. For example, LSTM with 0.0065 learning rate seemed to perform well at identifying jazz songs but poorly at identifying hip-hop songs. This deficiency seemed to be made up by LSTM with 0.0066 which seemed great at classifying hip-hop songs correctly. This gave me the idea of ensembling.

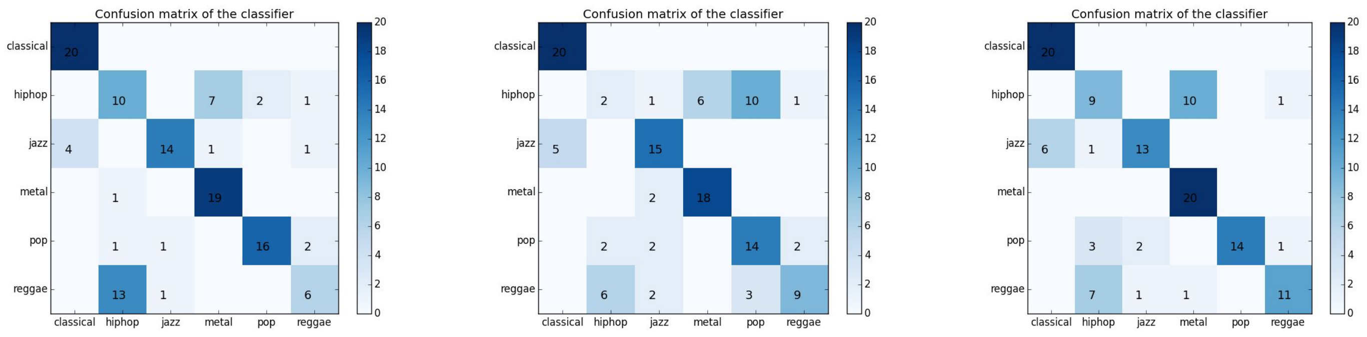


Figure showing the confusion matrix of LSTMs with learning rate as 0.0065, 0.0066, 0.0067 (from left to right) at 400 epochs with accuracy as 0.65, 0.725 and 0.70833 respectively.

<u>Exploration of ensembling methodology</u>

For ensembling, I decided to compare 2 blending ensembling techniques – averaging and maximum of all outputs. I wrote the code to blend outputs from all 3 LSTMs (as average and maximum of the 3 outputs). I then ran the code over different epochs ranging from 200 till 1000 and observed the performance of all 3 LSTMs (with learning rates as 0.0065, 0.0066 and 0.0067) as well as the 2 ensembles. This technique of mine saved the team precious time as it helped us record performance of individual LSTM models and of the 2 ensemble models over different epochs in the same run.

Keshav Potluri

The project was a collaborative effort with every member contributing to every phase of the project, yet each individual had specific tasks assigned to them. My main contributions to the project were as follows :

5. Search for Audio data and tool exploration Yaafe
6. Research on Hyper-parameters for Audio analysis

7. Grid Search for Hyperparameters: Epochs and Learning Rate
8. Cross Validation to ensure stability in results
9. Ensembling

Search for Audio data and tool exploration Yaafe

With the goal of the project clearly defined, our first challenge was to identify a suitable dataset for our analysis, although we had access to the Million Song Database, the data consists of only specific features. Since we wanted to explore features like MFCC, Chroma, Spectral Roll etc. about the songs, we needed audio clips for our analysis. Given the issue of copyright protection, it was difficult identify a dataset suitable for our project. Although we finally shortlisted the GTZAN dataset, I spent considerable amount of time searching for datasets. Some of the data sets that I came across are : USPOP202 : Pop music dataset , Seyerlehner data sets : 19 genres (190 songs), pop (3000+ songs), FreeMusicArchive, FMA API etc. Once the data was available, I was then exploring other tools to extract features. Before we short listed Librosa for audio feature extraction, I also explored the tool Yaafe for feature extraction. Although promising, I spent considerable amount of time integrating it with our python code for automating the feature extraction process. Although we did not proceed with Yaafe, I did spend time exploring and integrating the tool.

Research on Hyper-parameters for Audio analysis

While we were searching for the data and exploring the various tools for audio analysis, we also were searching for literature on audio analysis. I was focussing on searching for literature on Hyper parameter search and optimization for audio analysis using LSTMs. These research papers are listed below for reference. Some of these papers that I explored, provided some wonderful insights like indicating which hyperparameters are more effective in improving accuracy, or how to approach tuning of parameters. Some of the literature also suggested additional methodology that we did not follow in this project, but could be pursued at a later point, like filtering the audio clips and retaining only specific frequencies before extracting features. Searching and studying the literature was very helpful in obtaining good results in our project.

Grid Search for Hyperparameters: Epochs and Learning Rate

Since LSTMs were the only model we were pursuing, Konstantin wrote the base code for the model and then we shared it to focus on the hyper parameter search. I personally had earlier found the process of hyperparameter search very tedious last year in NLP. But during the Kaggle assignment in the Data Mining class, I had implemented Grid Search that was covered in the class and found it really helpful in narrowing down on the optimal hyper parameters. I therefore, suggested following a Grid Search method, albeit a manual one due to the high computational cost of every run. I ran the grid by gradually decreasing the step size to narrow down on the parameter: first by varying the learning rate in the power of 10s (0.1, 0.01, 0.001 etc.) and then varying it in steps of 1 (0.1,0.2,0.3, 0.01,0.02,0.03 etc). A snapshot of some of the runs that I did are as shown.

Layers (Units)	Epochs	Batch Size	Learning Rate	Accuracy
2(32,16)	500	128	0.1	0.125
2(32,16)	500	128	0.01	0.3625
2(32,16)	500	128	0.001	0.5
2(32,16)	500	128	0.0001	0.2125
2(32,16)	500	128	0.00001	0.125

Layers (Units)	Epochs	Batch Size	Learning Rate	Accuracy
2(32,16)	500	128	0.005	0.69375
2(32,16)	500	128	0.006	0.81875
2(32,16)	500	128	0.007	0.8

I followed the same methodology to explore the epochs as well, with runs ranging from 100 epochs to 10000 epochs (The code ran overnight for more than 12 hours for 10000 epochs). The grid search was a valuable methodology that helped in narrowing down the hyperparameters.

Cross Validation to ensure stability in results

Some of the accuracies as observed above were high but was because the model was overfitted for specific runs and the accuracies were not representative of the optimal hyperparameters. In the class I realized for some of my assignments that cross validation was a good measure to identify the accuracy of the model. I hence suggested to perform cross validation on our models. Since the training times are extremely high, I suggested we perform this validation only on models and set of parameters that gave us high accuracy (again manually because Scikit learn cross validation was not optimally suited for our LSTM and its parameter search). I performed the cross validation on few of our potentially final models to obtain a stable accuracy.

Ensembling

Richa implemented the confusion matrix which was very useful in identifying that certain models were better for certain genres and hence we decided to use the concept of ensembling that we learnt in the class to improve the results. I helped Richa with the ensembling and running the models with ensembling. Richa implemented the average and max ensemble, I implemented a weighted ensemble model and a majority vote model. Out of the four models, average performed the best. Ensembling was really useful in improving the accuracy of our models.

Results and Conclusion

Results on the Dev Dataset

As described in our methodology, we began our hyper parameter search by optimizing each hyper parameter individually based on the accuracy results. To do this, we used the grid search methodology.
The results of our exploration are listed below:

Variation of Accuracy with Number of Layers in the LSTM model:

Our LSTM model was using 27 input units to derive 6 output units. The group's previous experience had shown us that in such a case (Where #input units >> #output units) , neural networks (or LSTMs in our case) performed better when the number of units within different hidden layer decrease as the proximity of the hidden layer from input layer decreases. For example, consider a LSTM with 3 hidden layers. This LSTM will perform better if the number of units within the hidden layers varied (30,20,10) as compared to having a constant number of units within each hidden layer (for eg. 20,20,20).

We started with using 30 or 40 as the number of units in the first hidden layer and subsequently decreasing the number of units within each layer. We quickly realized that having 30 units in the first hidden layer and 10 in the last hidden layer gave the best results. The graph below shows the trend in accuracy as the number of hidden layers (with such an arrangement of units within each hidden layer) in an LSTM increases. The exact data can be referenced in the appendix.

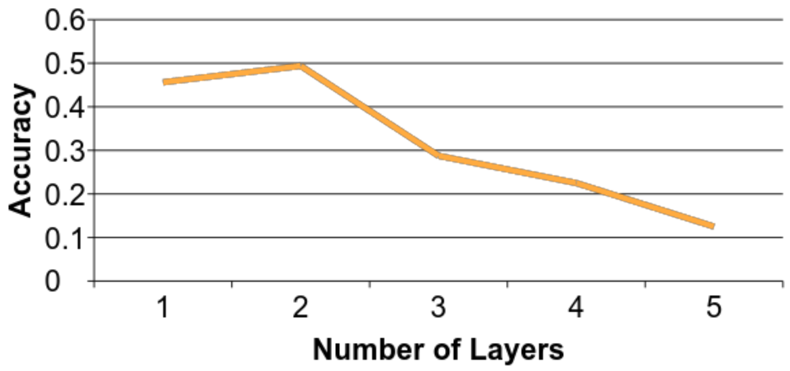


Figure showing variation of accuracy with the number of hidden layers in an LSTM

The graph shows that we got the best accuracy with 2 layers. We then later explored whether changing the number of units within the 2 hidden layers can give us better results. We quickly realized that having 32 units followed by 16 units within the 2 hidden layers gave better results than having 30 units followed by 10 units.

Further changing the output layer of LSTM to Dense layer caused an accuracy jump of 12% (from 36% to 48%).

Variation of Accuracy with Learning Rate:

Having finalized the architecture of the LSTM model, we next moved on to finding out the best learning rate using grid search. For simplicity, we experimented with 500 epochs. Using a coarse grid net, we first started with a small learning rate of 0.00001 and increased it till 0.1 in powers of 10. The variation of accuracy can be seen below.

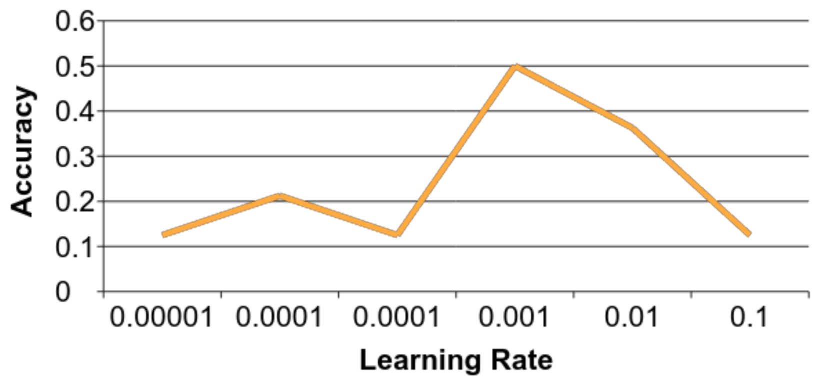


Figure showing variation of accuracy with learning rate using coarse grid search

We then refined our grid search to the range between 0.001 and 0.01 because they both showed best performance in terms of accuracy. We increased the learning rate in steps of 0.001 and realized that the accuracy gave best results in the range between 0.005 and 0.008 - especially around 0.0065. We further narrowed down our grid-search to the range between 0.005 and 0.008 and concentrated on capturing accuracy around 0.0065. The results below shows that learning rate of 0.0065 gave best results followed by 0.0066 and 0.0067:

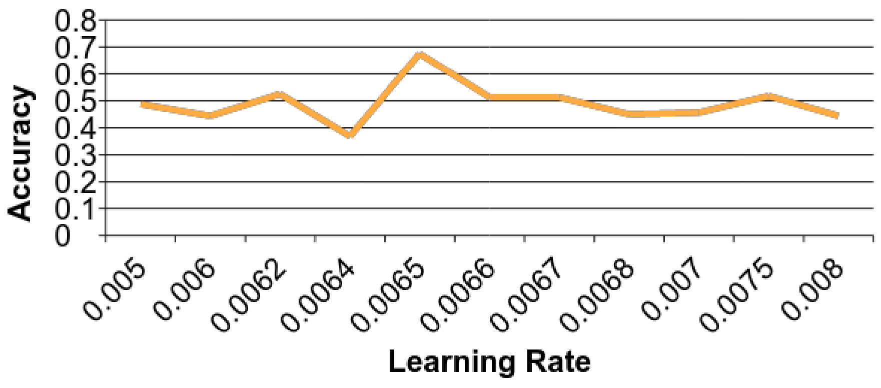


Figure showing variation of accuracy with learning rate using fine grid search

<u>Variation of Accuracy with Epochs:</u>

Now that we had found the best LSTM architecture and learning rate, we concentrated on finding the number of epochs after which the LSTM gave the best results. As seen in the graph below, 600 epochs gave the best accuracy with learning rate as 0.0065.

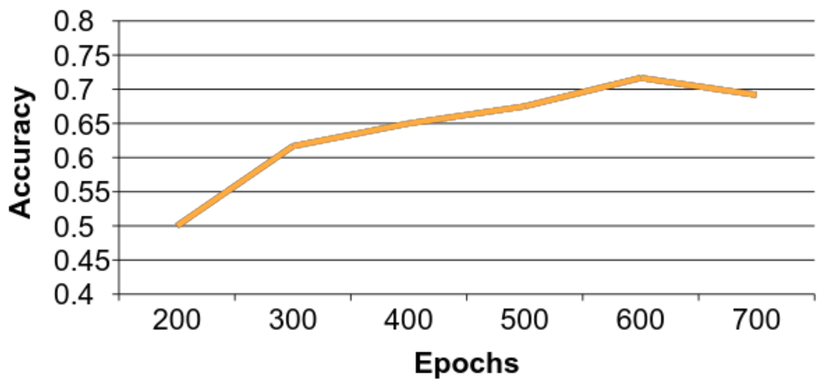


Figure showing variation of accuracy with epochs

Variation of Accuracy with Learning Rate and Epochs:

Since training the LSTM took a lot of time, we tried to find out whether combination of learning rate and epoch apart from 0.0065 and 600 respectively, gave a better accuracy.

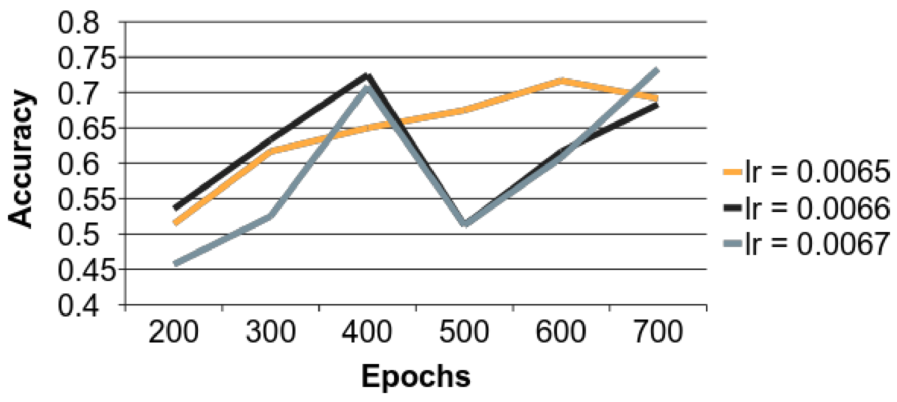


Figure showing variation of accuracy with epochs and learning rate

As seen in the graph above, learning rate of 0.0066 and 0.0067 gave a better accuracy than 0.0065 at a much lower epoch of 400.

Ensembling:

Given the similar accuracy of learning rates of 0.0065, 0.0066 and 0.0067 at 400 epochs, we decided to explore analyze which genres they classified correctly using confusion matrix. The result of confusion matrix of learning rates of 0.0065, 0.0066 and 0.0067 at 400 epochs is shown below.

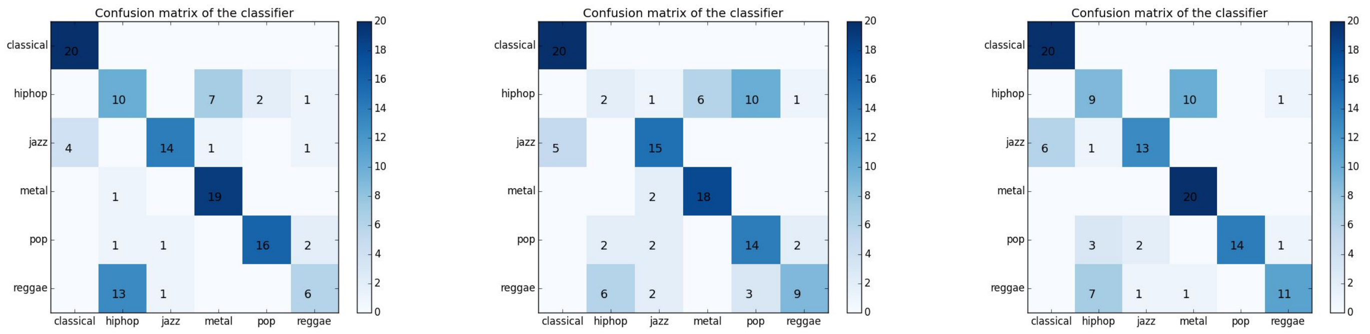


Figure showing the confusion matrix of LSTMs with learning rate as 0.0065, 0.0066, 0.0067 (from left to right) at 400 epochs with accuracy as 0.65, 0.725 and 0.70833 respectively.

Once can see that though the different learning rates depicted similar accuracies at 400 epochs, they actually identified different genres correctly. For example, LSTM model with learning rate of 0.0065 had a higher accuracy at identifying jazz but poor accuracy in identifying hip-hop. Similarly, LSTM model with learning rate as 0.0066 seemed to be good at good at identifying hip-hop but not reggae. Finally, LSTM model with learning rate as 0.0067 depicted a higher accuracy at identifying reggae.

Because of their complementary results, we decided to combine the 3 models into a single ensemble using blending methods like average or maximum.

The results of running the ensembled model using average and maximum blending at 400 epochs is shown below:

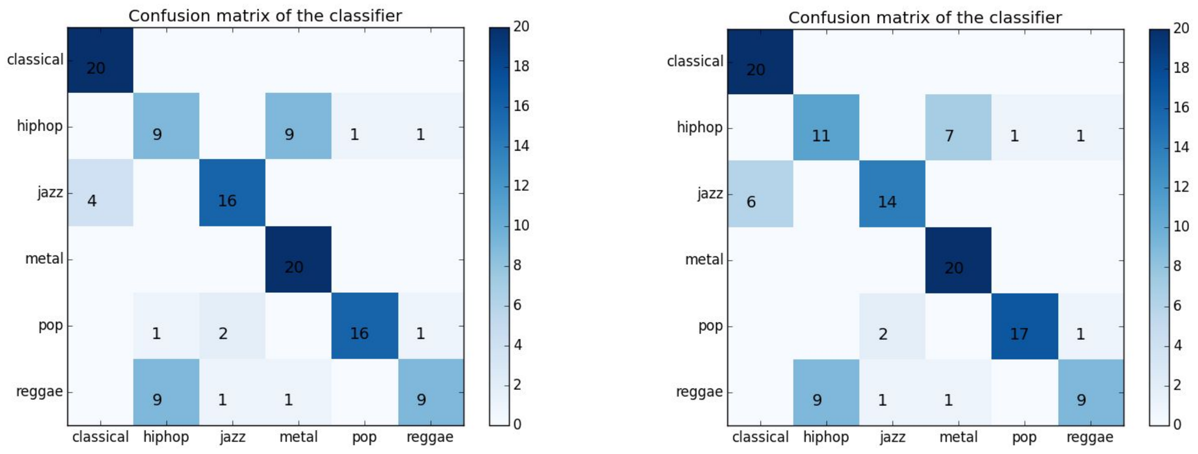


Figure showing the confusion matrix of ensembled LSTMs using averaging and maximum blending methods (from left to right) at 400 epochs with accuracy as 0.75 and 0.75833 respectively. Both the ensembles used the same 3 LSTM models with 2 hidden layers (having 32 and 16 units in each hidden layer) and learning rate as 0.0065, 0.0066 and 0.0067 respectively.

The precision and recall of the average ensembled method for different sing genres is shown below:

Genre	Precision	Recall
Classical	0.83	1
HipHop	0.47	0.45
Jazz	0.84	0.8
Metal	0.67	1
Pop	0.94	0.8
Reggae	0.82	0.45

The precision and recall of the maximum ensembled method for different sing genres is shown below:

Genre	Precision	Recall
Classical	0.77	1
HipHop	0.55	0.55
Jazz	0.82	0.7
Metal	0.71	1
Pop	0.94	0.85
Reggae	0.82	0.45

Looking at the increased accuracy of the ensembled model, we again tried to see if they gave a better accuracy at another epoch rate. Below are the results:

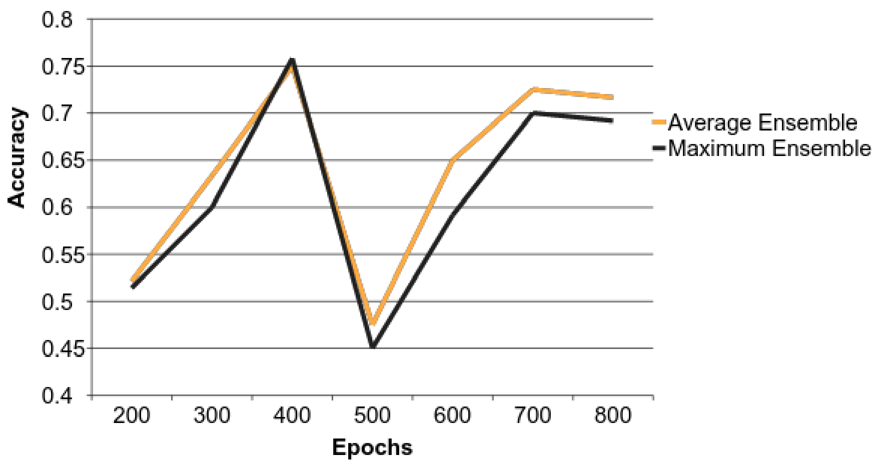


Figure showing the variation of accuracy of different ensembling methods with epochs

As can be seen in the graph above, though ensembling method using maximum blending method gave a higher accuracy at 400 epochs, the average ensembling method had a better accuracy over the different epochs.

Results on the Test Dataset

Satisfied with our exploration, we decided to run both ensemble models on the test data set at 400 epochs. Again note that both the ensembles used the same 3 LSTM models with 2 hidden layers (having 32 and 16 units in each hidden layer) and learning rate as 0.0065, 0.0066 and 0.0067 respectively. The confusion matrix of both ensemble models is shown below:

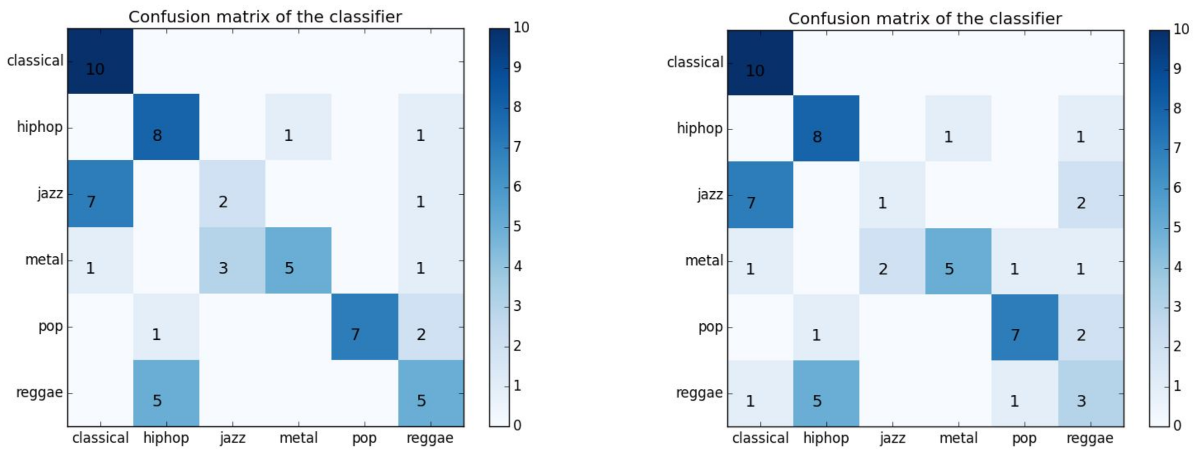


Figure showing the confusion matrix of ensembled LSTMs on test data set using averaging and maximum blending methods (from left to right) at 400 epochs with accuracy as 0.6167 and 0.5666 respectively. Both the ensembles used the same 3 LSTM models with 2 hidden layers (having 32 and 16 units in each hidden layer) and learning rate as 0.0065, 0.0066 and 0.0067 respectively.

As seen in the results above, despite having a lower accuracy on the dev data set, the average ensembling methods actually had a better accuracy on the test data set. This can be because the way our code is modelled, the average ensemble is equivalent to voting by the 3 different LSTM models.

Conclusion

Despite having a lower accuracy on the test data set, our findings correlate with a lot of research findings:

1. Accuracy varied with different genres
2. Classical was easily identified by almost all models
3. Jazz is usually frequently misclassified as metal or classical
4. Pop is difficult to classify correctly by different models - however our model did a great job at identifying it correctly

Despite these similarities, our approach was significantly different than all other researches conducted on genre detection based on:

1. We used a more varied but equally sampled dataset (predicting 6 genres as compared to 4 genres by other research)
2. We used many more audio features to detect genres as compared to other researches which mostly used 1-2 audio features
3. We used ensembling method - something that has never been explored before in song genre detection

We also believe that our project research is very relevant and applicable in the real world. As explained before, our project can help cloud-based song services like Spotify and Pandora, in automated classification of the ever-increasing collection of these songs based on genres to provide better searchability and suggestions to the end user based on the user preference.

Future Work

Given how useful song genre detection can be in the real world, some of the future work that can be done in complement to our research is to:

- Identify if audio signals can be transformed before feature extraction to improve accuracy. For example by using filters to analyze only those specific frequencies which are more distinct per genre and remove other frequencies.
- Explore more features/hyperparameter using GPUs.
- Include more genres in the audio
- Explore whether multi-genre songs can be detected

References

1. Diab, O., Mainero, A., & Watson, R. (2012). Musical Genre Tag Classification With Curated and Crowdsourced Datasets
2. Nayebi, A., & Vitelli,M. (2013). GRUV: Algorithmic Music Generation using Recurrent Neural Networks

3. Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2015). LSTM: A Search Space Odyssey
4. Skymind. (2016). A Beginner's Guide to Recurrent Networks and LSTMs. Retrieved May 07, 2016, from http://deeplearning4j.org/lstm.html#long

Appendix

Exploration Findings

<u>For LSTM Architecture:</u>

Hidden layers	Epochs	Batch size	lr	Accuracy		Training Data	Prediction made based on	Comment
1(30)	500	128	0.001	0.45625	proxima			
2(30,10)	500	128	0.001	0.49375	proxima			[RP]: For this combination, I am getting 0.45625 sometimes and 0.475 sometimes
3(30,10,10)	500	128	0.001	0.2875	proxima			
4(30,10,10,10)	500	128	0.001	0.225	proxima			
5 (30, 25, 20, 15, 10)	500	128	0.001	0.125	konsti	training set	dev set	all predictions were 'pop'
4(30, 25, 15, 10)	500	128	0.001	0.34375	Richa		dev set	
2(30,10)	500	128	0.001	0.45625	Richa		dev set	
3(30,20,10)	500	128	0.001	0.41875	Richa		dev set	
2(30,10)	500	128	0.01	0.3375	Richa		dev set	
2(30,10)	441	128	0.01	0.3625	Richa		dev set	
0 (no hidden layers, only input and output)	500	128	0.001	0.1321428571	konsti	dev set	train set	mostly all predictions were 'hip-hop'
2 (16,4)	500	128	0.001	0.1589285714	konsti	dev set	train set	
2 (16,4)	500	128	0.001	0.34375	konsti	train set	dev set	
2 (16,4) The output layer has	500	128	0.001	0.45	konsti	train set	dev set	

been changed to Dense								
2(20,4)	500	128	0.001	0.28125	Richa	train set	dev set	
2 (32, 16) The output layer has been changed to Dense	500	128	0.001	0.48125	konsti	train set	dev set	
3 (32, 16, 8) The output layer has been changed to Dense	500	128	0.001	0.43125	konsti	train set	dev set	
2(20,10)	500	128	0.1	0.125	Keshav	train set	dev set	
2(20,10) Output to Dense	500	128	0.1	0.125	Keshav	train set	dev set	

For Learning Rate

Hidden layers	Epochs	Batch size	lr	Accuracy		Training Data	Prediction made based on	Comment
2(30,10) Output to Dense	500	128	0.00001	0.125	Keshav	train set	dev set	
2(30,10) Output to Dense	500	128	0.0001	0.2125	Keshav	train set	dev set	
2(32,16) Output to Dense	500	128	0.0001	0.125	Richa	train set	dev set	
2 (32, 16) Output to Dense	500	128	0.001	0.5	Richa	train set	dev set	as cross-verification
2 (32, 16) Output to Dense	500	128	0.005	0.71875	Richa	train set	dev set	
2 (32, 16) Output to Dense	500	128	0.0075	0.8	Richa	train set	dev set	
2 (32, 16) Output to Dense	500	128	0.005	0.69375	Keshav	train set	dev set	as Verification for Richa's Find
2 (32, 16) Output to Dense	500	128	0.006	0.81875	Keshav	train set	dev set	
2 (32, 16) Output to Dense	500	128	0.0065	0.85625	Keshav	train set	dev set	

2 (32, 16) Output to Dense	1000	128	0.0065	0.93125	Keshav	train set	dev set	
2 (32, 16) Output to Dense	10000	128	0.0065	0.73125	Keshav	train set	dev set	
2 (32, 16) Output to Dense	1000	128	0.0065	0.875	Richa	train set	dev set	as verification for Keshav's find
2 (32,16) Output to Dense	500	128	0.005	0.4875	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.006	0.44375	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0062	0.525	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0064	0.36875	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0065	0.675	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0066	0.5125	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0067	0.5125	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0068	0.45	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.007	0.45625	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.0075	0.5175	Richa	train set	dev set	
2 (32,16) Output to Dense	500	128	0.008	0.44375	Richa	train set	dev set	

For Epochs:

Hidden layers	Epochs	Batch size	lr	Accuracy		Training Data	Prediction made based on
2 (32,16) Output to Dense	200	128	0.0065	0.514285714	Proxima	train set	dev set
2 (32,16) Output to Dense	300	128	0.0065	0.616666667	Richa	train set	dev set
2 (32,16) Output	400	128	0.0065	0.65	Richa	train set	dev set

to Dense							
2 (32,16) Output to Dense	500	128	0.0065	0.675	Richa	train set	dev set
2 (32,16) Output to Dense	600	128	0.0065	0.716666667	Keshav	train set	dev set
2 (32,16) Output to Dense	700	128	0.0065	0.691666667	Richa	train set	dev set
2 (32,16) Output to Dense	800	128	0.0065	0.691666667	Keshav		
2 (32,16) Output to Dense	1000	128	0.0065	0.691666667	Keshav	train set	dev set
2 (32,16) Output to Dense	200	128	0.0066	0.535714286	Proxima	train set	dev set
2 (32,16) Output to Dense	300	128	0.0066	0.633333333	Richa	train set	dev set
2 (32,16) Output to Dense	400	128	0.0066	0.725	Richa	train set	dev set
2 (32,16) Output to Dense	500	128	0.0066	0.5125	Richa	train set	dev set
2 (32,16) Output to Dense	600	128	0.0066	0.616666667	Keshav	train set	dev set
2 (32,16) Output to Dense	700	128	0.0066	0.683333333	Richa	train set	dev set
2 (32,16) Output to Dense	800	128	0.0066	0.675	Keshav		
2 (32,16) Output to Dense	1000	128	0.0066	0.675	Keshav	train set	dev set
2 (32,16) Output to Dense	200	128	0.0067	0.457142857	Proxima	train set	dev set
2 (32,16) Output to Dense	300	128	0.0067	0.525	Richa	train set	dev set
2 (32,16) Output to Dense	400	128	0.0067	0.708333333	Richa	train set	dev set
2 (32,16) Output to Dense	500	128	0.0067	0.5125	Richa	train set	dev set
2 (32,16) Output to Dense	600	128	0.0067	0.608333333	Keshav	train set	dev set
2 (32,16) Output to Dense	700	128	0.0067	0.733333333	Richa	train set	dev set

2 (32,16) Output to Dense	800	128	0.0067	0.55	Keshav		
2 (32,16) Output to Dense	1000	128	0.0067	0.55	Keshav	train set	dev set

For ensembling:

Hidden layers	Epochs	Batch size	lr	Accuracy by Average Ensemble	Accuracy by Average Ensemble	Done by	Training Data	Prediction made based on
2 (32,16) Output to Dense	200	128	0.0065	0.5214285714	0.5142857143	Proxima	train set	dev set
2 (32,16) Output to Dense	300	128	0.0065	0.6333333333	0.6	Richa	train set	dev set
2 (32,16) Output to Dense	400	128	0.0065	0.75	0.75833	Richa	train set	dev set
2 (32,16) Output to Dense	500	128	0.0065	0.475	0.45	Richa	train set	dev set
2 (32,16) Output to Dense	600	128	0.0065	0.65	0.5916666667	Keshav	train set	dev set
2 (32,16) Output to Dense	700	128	0.0065	0.725	0.7	Richa	train set	dev set
2 (32,16) Output to Dense	800	128	0.0065	0.7166666667	0.6916666667	Keshav		
2 (32,16) Output to Dense	1000	128	0.0065	0.6166666667	0.5666666667	Keshav	train set	dev set