Music Genre Classification

Thomas Lillis

Student  
9962 Bear Creek Apartments  
Boulder, CO 80303   
+1 303 590 5985

thli2089@colorado.edu

Joseph Marylander

Student  
1025 Adams Circle, E215  
Boulder, CO 80303  
+1 303 475 8838

joma1054@colorado.edu

Connor McGuinness

Student  
805 29th St. #557  
Boulder, CO 80303  
+1 203 521 9512

comc3361@colorado.edu

**ABSTRACT**

Genres within music are culturally agreed upon categorical descriptions used to filter songs and artists. In this work, we use a classification process to classify songs using musical attributes extracted from the audio. Using the Million Song Dataset3 we had access to a million contemporary popular music tracks to use for our classification. The process analyzes certain extracted musical elements like rhythm, time signature, ‘danceability’, key, loudness, etc, in order to fit one of our genre classes such as ‘Pop/Rock’ or ‘Classical’ genres. We used two different methods of classification for out project, a Support Vector Machine (SVM) and K-Nearest Neighbors.

The results of our project showed that our SVM performed significantly better than our KNN algorithm. The SVM showed 68% correct classification while KNN showed about 40% of the genres correctly classified. Our results were somewhat successful but not as great as we had expected when starting the project. Hardware limitations caused much of the time of this project spent waiting rather than making progress so there is much to be expanded on in or this project such as extracting more meaningful attributes for genre classification, better preprocessing data, and better tuning our classification algorithms.

**General Terms**

Algorithms, Performance, Design, Economics, Reliability, Experimentation, Security, Human Factors

**Keywords**

Data Mining, Classification, Machine-Learning, Supervised Learning

# INTRODUCTION

Though how we interpret music genres as humans is a subjective classification, there are still some metrics which specifically fit genres in which most can agree. In this project we use supervised machine learning techniques to attempt to fit songs into 12 general genres that most individuals can agree would be fitting. We attempted to find and use the best attributes extracted from raw audio files to classify the audio files into commonly accepted, broad, and generic genres.

This type of work could be useful for many different types of applications. It may be used for a Pandora type website for finding songs with a similar genre classification that the user might enjoy or could be used by radio stations to find more music they could play for their station.

For this project we used two different types of multi-class classification to see which method is most effective with our dataset, a Support Vector Machine, and K-Nearest Neighbors algorithm.

# LITERATURE SURVEY

In the paper [1] “Exploring different approaches for music genre classification” Antonio Jose Homsli Goulart, et al. describe how they extracted various features from a dataset of songs and how they attempted to classify them. They focused only on three genres of music: Blues, Classical, and Lounge.  In contrast to their research, we plan on classifying our data into as many genres as we can while being accurate. However, their classifications techniques are useful in determining how we should proceed with our own classification.

Similar to our project goal, this team proposed a musical genre classification system using attributes such as rhythmic structure and instrumentation [2]. Their work aims to solve the problem of the large amount of music being created without an automated genre classification system. As well, the team created two graphical interfaces to allow unfamiliar users to interact with their classification system; something our team wants to do. Understanding George Tzanetakis’ and his team’s approach to genre classification will help guide us in creating a similar project but with a different genre classification process.

Thierry Bertin-Mahieux [3] takes a simplified approach to musical genre classification, which is something our team wants to do as well. Instead of allowing tags to take multiple forms like ‘American pop’, ‘indie pop’, or ‘European pop’, they chose to include all of these tags into the tag ‘pop’. Thierry, B. M genre classification process only allows for the following 10 genres: classic pop and rock, folk, dance and electronica, jazz and blues, soul and reggae, punk, metal, classical, pop, and hip-hop. As well as only including these genres, Thierry, B. M excludes other popular genre tags within their dataset. This allows them to focus on artists or songs that are only within ‘pop’ instead of both ‘pop’ and ‘electronic’ for example. Our team’s aim is to follow a similar approach in simplifying the genres we choose.

# DESIGN AND IMPLEMENTATION

## Tasks

# The project had multiple subtasks grouped into main tasks. Initially, main tasks was focused on obtaining and organizing the Million Song Dataset. The first phase of the project was broken down into a collection of subtasks that aim to ensure the data is organized and collected.

# Next, the second phase of the project was focused on parsing the dataset and developing and trying different types of classification methods. Subtasks consisted of developing a parsing script, determining several classification methods, and implementing/testing the classification methods. This phase’s main goal is to create a well thought out classification method and then implement it.

## Obtaining Subset

# We started by downloading an easily obtainable subset of the data that was 1% of the entire dataset from the MSD website. This was about 10,000 total songs to work with and was about 4.7GB of files. The data was split into one file per song and the files were “Hierarchical Data Format” files and were split into directories by a random name that was alphabetized. This required a special python library to be able to read the files which were ‘.h5’ files. This library ended up causing a lot of trouble down the line with the time of execution.

## Aggregation

# We used a BASH script to traverse all of the directories that contained the files and to run the python script that extracts all of the information from each song file and aggregate all of the data with attributes into one text file that we could use for parsing. Each song has about 55 attributes associated with it and these attributes where delimited by tabs because there were commas in some of the attributes. This led to some challenges for a python parsing script down the line which was unable to parse two tabs in a row when an attribute field was empty. The BASH script took a little over an hour to run on a Intel 3rd Generation i7-3520M CPU @ 2.90GHz and 2 cores with 8GB of RAM.

## Obtaining Entire Dataset

# Getting the entire dataset it ended up being a greater challenge than we anticipated. First we attempted to access the entire dataset through a MySQL database that was provided from the million song dataset webpage. Though this database had all the datapoints of the dataset it did not provide every attribute that we needed for our classification but rather mostly just meta data about the songs. Our second option was a slightly more challenging option using Amazon Web Services. Though the dataset was easily accessible through Amazon Web Services it was slightly challenging to work with. We worked with the free tier of AWS which was a single core version of Ubuntu 14.04 with 1 GB of RAM. To get the data we had to attach a 500GB Amazon Public Data Snapshot and mount it to our filesystem on our virtual machine. Then it was then a traversable “Hierarchical Data Format” structure similar to the subset of the dataset.

## Issues with Runtime

# Though we thought it would be as easy as just running our script that we ran on the subset to aggregate our data but unfortunately this was not the case. We started running the script but the runtime of the script completion was an estimated 21 days. We believe this had to do with the lack of power of the amazon machine because it was only a single core with a little amount of RAM. The bottleneck of this script was when the python script was called to read the file. We used a special library to read the “.h5” files which was particularly slow. It took about 1-2 seconds to read a single song out of our million songs. We were unable to transfer the dataset off of the AWS server because we had a max of 15 GB of bandwidth so we were stuck with using the data on the machine. It is possible to upgrade the machine through Amazon but that would have been much more expensive and seemed unnecessary for this scope of project.

# We left this script running for several days but it didn’t seem sustainable so we decided to try and parallelize the script and handle different seconds of the dataset. We made a mistake in implementation and accidently ran over 700 instances of the script in parallel. This used all the memory on the virtual machine and crashed the machine. This stopped our original script running and it had only handled about 50,000 data points. We then tried running 26 instances of the script because there were 26 different head directories each a different letter of the alphabet. This did not speed up the process of the script running unfortunately. This aggregated the songs into about 10,000 songs before we realized the trajectory of the pace of the script was not as fast as even one instance of the script running.

# We settled on running this script with 4 instances working at once. This ran at a rate of about 50,000 songs every 12 hours. We pulled the data off the server after about 250,000 songs but left it running to be able to run on a larger dataset later. Then some simple BASH scripting was done to combine all the files and remove duplicates of all the files the where run from each attempt. This made for about 300,000 songs in our aggregate file for a size of 240 MB. In this file we then used the “|” character for the delimiter because it was easier for our python script to parse than the tabs.

## Appending Genre Tags

# We then needed to merge this data with the genre tags that we acquired from Yajie Hu and Mitsunori Ogihara Million Song Dataset Genre Tags. To do this we wrote another python script that reads in all of the genre tags with their associated ID. It then parses the text file with all the song data in it and if there is a match on IDs then that song with all of the attributes as well as genre tag are put into a new text file. At this stage we then had a text file with all the raw data. We did some basic cleaning of the data like removing whitespace and correcting a few formatting issues.

## Data Pruning

# After we created our merged dataset we needed to prune some features. We created a python script that allows us to remove specific column names from our dataset. Most of the features we pruned were song id’s and md5 values, neither of which are helpful to us. We then needed to convert the genre labels from text into an integer value. This was easily done by having a one to one mapping of the text labels to an integer via a dictionary.

## Data Removal

# The dataset contains a number of “confidence” features that are directly related to another feature. For example, there is feature for the “key” of the song, additionally there is also a “key\_confidence” feature that represents how confident the dataset is that key is correct. We could not figure out a good way to merge these features into a single fully representative feature so we just removed the confidence feature from our set and just used the base feature itself.

## Other Cleaning

# Finally, we had to process some of our data to become usable. There were a number of features that were in the form “shape = (x,y)” where ‘x’ and ‘y’ were some integer value. These “shape values” generally were used to describe a collection of segments of the song. Each of these segments were just a continuous section of the song. A segment could be as short as half a second or as long as 10 seconds. Using the api provided by the Million Song dataset, we were able to resolve most of them to a sort of average value that we then used in our classification.

## Finalize Data

# We did final verification of the data and then we were then ready to run it through both of our classifiers.

# EVALUATION

## Support Vector Machine

We decided to use an SVM as one of our classifiers for it was one of the first classifiers that was really brought to our attention and it appeared to be quite robust in multiclass classification due to its various forms: One vs Rest and One vs One.

### Determining Which SVM to Use

Initially, we did not know much about sklearn nor svms so we attempted to run a classification against just a simple SVM, straight out of the box from the sklearn toolkit. Sklearn has two implementations, SVC and SVR. We chose to use their SVR implementation because it appeared to give us marginally better results as can be seen by comparing figure 1 and figure 2.

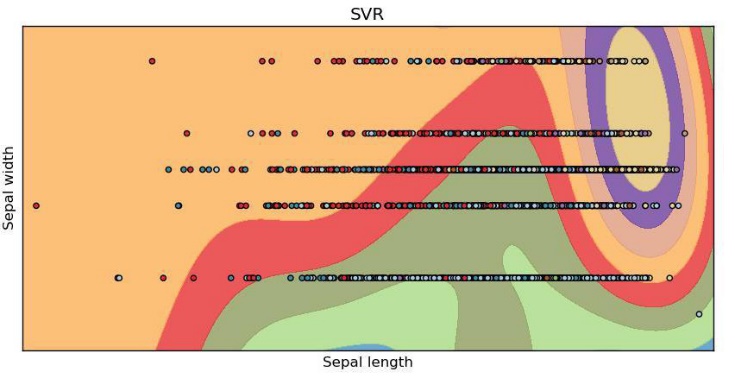


Figure 1.

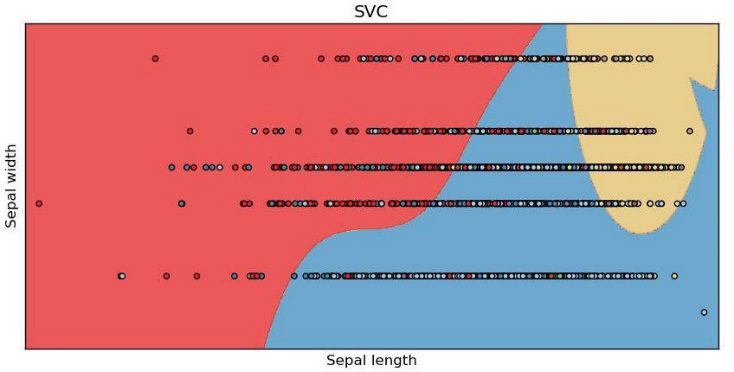


Figure 2

As we investigated more with the sklearn toolkit, we learned that you can wrap these classifiers in either a OneVsRestClassifier or a OneVsOneClassifier. These extended the base functionality of the SVR classifier to handle multiclass classification in either the One Vs Rest or the One vs One technique. Due to the high dimensionality and the fact that it performed better, we chose to use the One vs Rest classifier for our data seat. A comparison between the two techniques can be seen in figure 3 and figure 4.

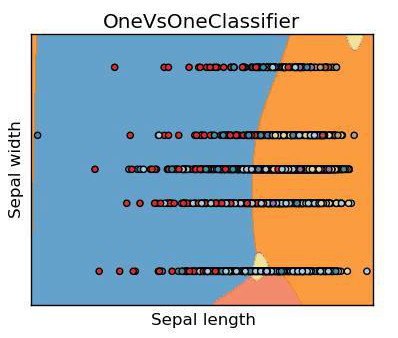


Figure 3

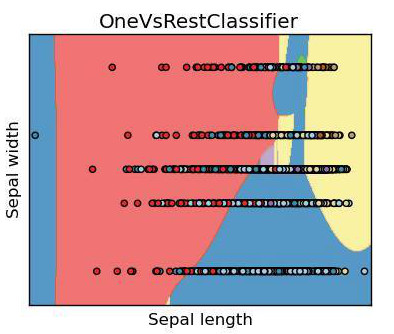


Figure 4

### Optimizing Our Classifier

At this point, with our small testing set, we had about a 50% accuracy. After doing some research on SVMs and general machine learning topics, we learned we need to reduce the dimensionality of our dataset. Sklearn has a package called feature\_selection that we use to figure out which of our 58 remaining features we actually worth keeping. SelectKBest is an algorithm we ran on our dataset that does exactly that: select the k best features to use. We tried a number of various values for ‘k’ but, just like the default value for selectKBest, only using two features gave us the best result. By reducing our dimensionality by 29x we were able to reach an accuracy of 68%.

In order to optimize all of our various variables within both our classifier and our selectKBest we use a combination of a pipeline and a technique called gridsearch. A pipeline, just as the name implies, allows you to take various stages of your classifier and chain them together in a single object. Gridsearch, on the other hand, is a technique that brute forces its way to the ideal parameters. The idea is that you give all possible parameters you want to try and gridsearch will run a combination of the given parameters, and determine the best parameters combination with the best accuracy. We ran our Gridsearch with parameters for selecting the number of features to use, how to test which features are better, and the SVM’s C, epsilon, and gamma values. Our approach for selecting the possible values for these parameters were based on what others suggested in various machine learning guides online.

In order to optimize all of our various variables within both our classifier and our selectKBest we use a combination of a pipeline and a technique called gridsearch. A pipeline, just as the name implies, allows you to take various stages of your classifier and chain them together in a single object. Gridsearch, on the other hand, is a technique that bruteforces its way to the ideal parameters. The idea is that you give all possible parameters you want to try and gridsearch will run a combination of the given parameters, and determine the best parameters combination with the best accuracy. We ran our Gridsearch with varying parameters for selecting the number of features, how to test which features are better, and the SVM’s C, epsilon, and gamma values. Our approach for selecting the possible values for these parameters were based on what others suggested in various machine learning guides online.

These optimizations added approximately 5% accuracy to our final SVM. However, due to issues we will discuss in the next section, this increase in accuracy is not representative of our actual accuracy.

### Issue and Results

Because our dataset has a disproportionately large amount of “Pop/Rock” songs and Pop/Rock songs can be quite varied in their time signature and loudness it caused most songs to be classified as Pop/Rock regardless of if they actually are. This can be illustrated in figure 5.

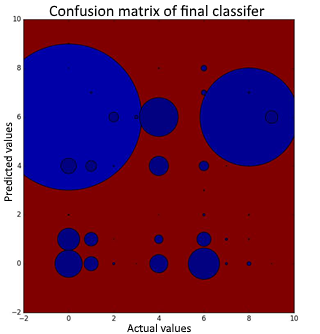


Figure 5

The two large circles in the graph on the left represent the result of having such a large amount and varied set of songs that are all classified as Pop/Rock. A similar result can be seen in figure 6. The red square shows that most things are being classified as Pop/Rock. Ideally, this dataset would have broken Pop and Rock songs into two different classes instead of combining them into one. If they were in two different classes, I imagine that the classification would be more accurate because there would no longer be such great overlap with the other classes.

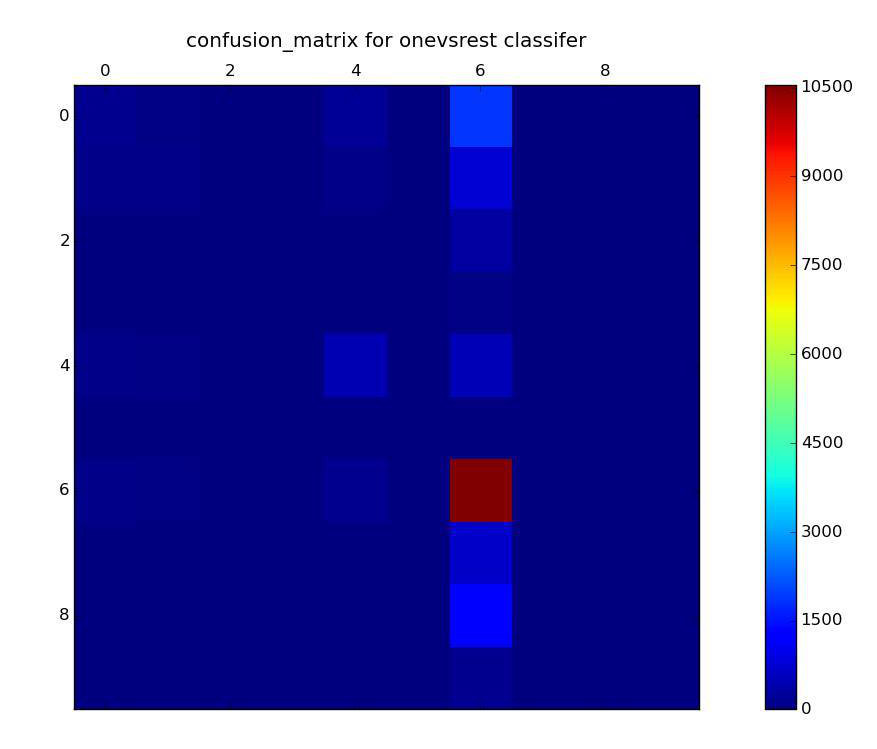


Figure 6

If we remove the Pop/Rock samples from the data set, we get a classification distribution that is more diagonalized, illustrated in figure 7. This means that our classifier has an accuracy on a class by class basis. This is better representation of what our SVM can do when provided with non-skewed data. This further allows us to infer things about songs and genre relationships. For example, we can see from our data that we somewhat often would misclassify country songs as blues songs. This error that even some humans make because of the similarities between the two genres. In a similar manner, we also misclassify R&B songs as blues because their loudness and general time signatures tend to be quite similar thus showing that they have influences on each other.

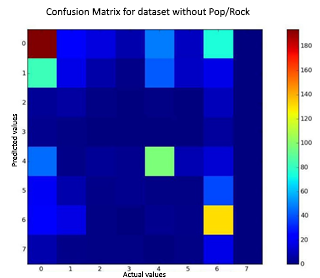


Figure 7

## KNN

It’s well known among data miners that K-Nearest Neighbor is one of the simplest machine learning algorithms. Our implementation follows suite. K-Nearest Neighbor was intuitively simple to understand after looking through multiple documentation about the topic. By choosing a value ‘k’, an object is assigned to an ‘attribute’ (or more commonly a class) that is the class of the majority of the object’s k-nearest neighbors. There are multiple ways to optimize K-Nearest Neighbor that make the algorithm a powerful tool for our program.

### Loading the dataset

Since k-NN is a lazy algorithm that will do all of the computational work on every instance, it was computationally expensive to run on our machines. By cleaning and pruning the dataset, we were able to work with smaller datasets which improved speeds. In order to load the data set into a workable format, we utilized the python data analysis library, “pandas”. Initially, our dataset is loaded in as a DataFrame which we then convert to an array of arrays, with each array being a full row in the dataset. Then we create a list of array objects. Putting the data into this format makes it easier to extract specific values in columns especially since each array has the same number of elements. Essentially each row is indexed in an array making it much easier to work with. Also while loading the data is in this format, we decided to set up the training and test sets. With a simple nested for loop, we traverse the dataset and randomly place a row within the training or test sets based on a constraint set by the user. The standard ratio that we chose to split our data into each set is 67% (training) / 33% (test).

### Evaluating Similarity between Values

Initially, we chose to implement a Euclidean distance to evaluate the relationships between values. Since we did not have any categorical data we did not need to utilize hamming distance for evaluation. The function simply returns the square root of the sum of the powers of the difference between two values within the same attribute. This is where kNN shines as a classification algorithm. Since euclidean distance is calculated without weight or context, it’s returned value is solely a distance measure. kNN just tests how consistent this distance is against an object’s k nearest neighbors.

### Finding Neighbors

Now that we can measure the euclidean distance of two instances it’s easy to apply this to each neighbor. First, for each row in the training set, we find the euclidean distance between two values, one from the test set and one from the training set. The distance is appended to an array. After each row has had its distance appended to an array, the array is sorted. Then, another for loop runs k times (specified by the user), that appends the row and it’s corresponding distance to a new array, ‘neighbors’.

### Accuracy

In order to see how well our k nearest neighbor classification algorithm performed we set up a function that returns the percent correctly predicted. At first, our algorithm ran at a whopping 40% accuracy about every time. ‘K’ was set to 3 so not enough k-nearest neighbors were being tested for which has the majority of votes. By increasing k, we were able to achieve accuracy as high as 60%. A key aspect of our data is that each attribute has different weights. The value for “artist\_term\_freq” has a largely different meaning than the value for “key” and therefore need to be interpreted on separate scales. However, normalizing the values on a scale of 0 to 1 would work just as well. By normalizing the dataset, the euclidean distance algorithm can perform with a much higher accuracy.

# CONCLUSIONS AND FUTURE WORK

## Conclusions

This project has been a first experience for everyone in our group to data mining and machine learning as a whole, so we learned a lot about the process and what it means to learn something from a data set. The most fundamental thing that we learned about data mining as a whole is to not underestimate the size of your data set. We certainly did. We underestimated it twofold: in both aggregating the data, and running our algorithms with the data set. There are a number of ways we can approach future data mining problems to mitigate these issues. First of all, we can use a better approach for aggregating our data from the hierarchical data set than our current python scripts. This likely will not be applicable to all future data sets because they come in many different forms, SQL databases, XML files, et cetera, however, this is still something we need to keep in mind. Second when working with such a large dataset, it is a good idea to run your final classifiers on good hardware. The computers we had at our disposal definitely struggled when running against even a ⅓ of our whole dataset. These things are mainly just applicable to this project and how we could have planned better.

The biggest effect on the accuracy of our classifiers came from preprocessing our dataset. Initially we overlooked this step because we assumed that the SVM we were going to construct would be able to handle all of the raw data: we were wrong. Preprocessing, just as we learned in lecture, is such a vital step in data mining and all machine learning algorithms. Initially, our preprocessing merely included removing of features that we believed negatively impacted our classifiers, removing outliers, and fixing missing values, but this, of course, was not enough. We also tried normalizing some of our data which we found some success with. Also, through our investigation, we learned that binarizing our data would be good if we have categorical data instead of quantitative data. We could have binarized our song titles and artist names and included them in our dataset instead of just removing those features.

We definitely made a number of mistakes during the course of this project. The main mistake we made was not fixing our dataset to have a more even distribution of classes. This was a big problem when we were testing with our SVM classifier. Due to the broadness of the Pop/Rock genre and the sheer number of data points we had for it compared to all of the other classes, it caused everything to be classified as Pop/Rock. Having a better distributed sample testing set certainly would have given us a more accurate idea of our accuracy, but this problem was also found to be a part of the entire dataset thus causing our final classification of the whole dataset to have similar results. Once we figured this out, however, we pruned our dataset of some of the Pop/Rock data points and that provided a much more balanced result. Our classifier was then actually labeling things other than Pop/Rock.

Through our research in machine learning techniques we learned of GridSearchCV and how it attempts to bruteforce the best possible parameters for your classifier. We feel that we trusted the results of this technique too much, especially when it claimed that the best number of features to use was two. This is not to say that GridSearchCV was wrong, but we should have realized when it told us to only use two features, that we need to reevaluate the state of our dataset. Something was wrong. We need to go back to the preprocessing step. But being the eager, inexperienced data miners that we are, we did not go back to preprocessing. But now we know better.

There were a couple of things that we were able to learn from our data that we were able to corroborate with real world knowledge. The first of which is because we can see the similarities between genres due to misclassification, we see the influences certain genres have on each other. For example, we were able to deduce that country songs and blues songs are quite similar and that’s because both genres of music have their origins in old time western music. Furthermore, it is not surprising to see that we has a similar effect with R&B and blues music given that R&B music is directly rooted in blues music, hence the name Rhythm and Blues. This sort of classification, especially if unsupervised clustering methods are used, can possibly show influences that genres have on each other. Furthermore, if you can group songs by year, you can see trends in genres and their influences on each other.

## Future Work

# Something we will plan better for is managing the entire dataset in a better way. This would entail either upgrading the amazon server or starting working with the entire dataset earlier in our project. Not using a slow python library for reading “.h5” files would have been better. We could have rather used a faster C library to do the heavy lifting in that area. With a larger dataset we could get more accurate results and possibly more representative data. For our sample of the entire dataset that we used as our final data, it is unclear if the sample was biased somehow. It could be biased by our script possibly collecting in an ordering that favors a certain genre or a certain attribute.

Another way to expand would to be extract more attributes from the data. The attributes that were given to us as part of the dataset were not the optimal attributes to do classification for genres. Extracting more attributes and being more particular about selecting and finding attributes that would make more sense for classification.

Another interesting expansion of this work would be doing all feature extraction from scratch. This way an individual would be able to input an ‘mp3’ file into the program and it performs the proper feature extraction and uses our model to correctly classify the genre. This way we wouldn’t need to rely on the Million Song Dataset’s features. Some of the features contained in the dataset are unknowns in how they are calculated and what they mean such as ‘Energy’ so we would prefer not to use those types of attributes.

Also, if we used unsupervised learning instead of supervised learning we might have been able to show influences that genres have on each other by looking at what songs are grouped together and referencing their actual genre.

# This project could have also been expanded to work for a specific problem. For example, a music recommendation app could do genre classification on your favorite music and give you recommendations based on your preferences of things selected as similar genres. For this to work properly we would need more resolution in our genre class labels. This is because an individual may not like one subgenre of a certain genre but wants to listen to a different subgenre of that genre. For example Pop/Rock is a very large category. An individual may want to listen to a Rock artist from the 60s and not hear a modern pop artist. Increasing the genre resolution of our research does fix this problem but does introduce the problem of the subjectivity of genre classifications. As the resolution of genre increases so does the subjectivity of which song classified under which genre.

# REFERENCES

1. Goulart Homsi Jose, A., Guido Capobianco, R., and Dias Maciel, C. 2012. Exploring different approaches for music genre classification. *Egyptian Informatics Journal.* 13, 2 (July 2012), 59-63. DOI= <http://dx.doi.org/10.1016/j.eij.2012.03.001>.
2. Tzanetakis, G., Essl, G., and Cook, P. 2001. *Automatic Musical Genre Classification of Audio*. Technical Report. Princeton University.
3. Thierry, B. M., Ellis, D., Whitman, B., and Lamere, P. 2011. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference* (ISMIR 2011).
4. Yajie Hu and Mitsunori Ogihara, “Genre Classification for Million Song Dataset Using Confidence-Based Classifiers Combination”, *in Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, Portland, Oregon, USA, 2012, pages 1083 – 1084

# APENDIX

## Honor Code Pledge

*On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work.*

## Contributions

* Tom Lillis: Worked on getting all of the data, aggregating the data into usable source, managing amazon web services server, appending the needed attributes to data, cleaning data, and basically everything that was not the actual machine learning.
* Joey Marylander: Worked mainly on the SVM classifier and all of the related machine learning aspects thereof. Additionally I worked with Tom on cleaning the final csv of data to remove unneeded features and some preprocessing.
* Connor McGuinness: