Predictive Analysis of Music using Historical Data
Final Project Report for EE-660 [USC Fall 2015]
12-08-2015 [Dec 08, 2015]

Nishant Nath
MS-EE '16 (USC)
nnath@usc.edu

Arnav Mendiratta
MS-EE '16 (USC)
amendira@usc.edu

Ahmet Can Ozbek
MS-EE '16 (USC)
ozbek@usc.edu

PROJECT HOMEPAGE

BITBUCKET URL (development git):
https://bitbucket.org/arnavmendi/project660

GITHUB URL (cleaned up dump):
https://github.com/nishantnath/MusicPredictiveAnalysis_EE660_USCFall2015

ABSTRACT

Predicting a genre of a music is not a trivial problem. Genre cannot be solely reliably predicted on the basis of the tonal and frequency content of an audio signal. Given that there are only 12 distinguishing notes in music, predicting genre becomes a hard problem, not to mention, is also very subjective. This project deals with using machine learning techniques to predict the genre by training various models and fine tuning the best model to match the benchmark results. Along with the qualitative and quantitative features of an audio waveform, we made use of metadata available within the dataset, like year of release, artist popularity scores etc. and we also ran crawlers to parse the missing data from Wikipedia to make the existing dataset more rich and complete in hope to giving back to the community with an improved million song database for research purposes.

We used 80 % of a million song as a set for training our models and validating to find the best model that works for classifying genre. The remaining 20% was left unused and used only for testing purposes in order to prevent the bias of our model.

Multiple models were used for training, but one final combination general model was created based on the best performing Random Forest and Support Vector Machines and was tested upon using the unseen test sample with missing information which gave us the error rate of 57%.

DATASET

Million Song Database (referred as MSD here-on).

Compiled by: Lab ROSA (Columbia University – NSF grant)

Compiled using: The Echo Nest API (+ several independent datasets)

Source (10k samples ~1% of MSD): http://labrosa.ee.columbia.edu/millionsong/

Availability: InfoChimps , AWS Public Data Set , OpenScienceDataCloud

Size: 274GB compressed (hdf5 format) [130kB hdf5 file ~ 2.2 MB csv file]

Files: 1,000,000 (covering 1 million songs, 1 song = 1 file)

Other Statistics: About 515,576 dated music tracks, tracks from 44,745 unique artists

Data range: ~100 years [1922-2011] (~ 10 Music decades)

MOTIVATION

We would like to start off with a case-study in point. Soundcloud – Inarguably the world's largest music collection on cloud claims to have 40 million registered users with 175 million unique monthly listeners. An independent survey puts the percentage of amateur music producers at nearly 90%. With the music industry growing in exponential rate, the motivation for the topic comes from two key areas – MIR (Music Information Retrieval) and Personalized Music Recommendation Systems. There is a constant need for automatic tagging of music mainly for its metadata but also for user-predictions and success factors. As our topic suggests, we would be performing predictive analysis on music encompassing both regression as well as classification problems.

PROBLEM STATEMENT & GOALS

As with any good machine learning application, the question to be answered determines the veracity of the application. For our problem, we have come up with 5 basic questions which give us deep insights regarding music. They are:

(*C = classification problem, *R = Regression Problem)

1. ***Automatic Genre Tagging (*C)***
   Given a song & its features – predict which Genre it belongs to?
2. ***Predict the popularity of the artist among listeners (*R)***
   Given a song & its features – predict how hot/viral the music artist turns out to be?
   This is being done as a proof of concept for song hotness prediction.

TASK 1:

Although terms like jazz, metal, rock, or pop are widely used, the boundaries between them remain fuzzy, so much so a piece of music can be considered of different genres by two musicologists or even by same musicologist over a period of time, so the problem of automatic genre classification becomes a nontrivial task.

Physical model:

Difficult to model

Non-Linear behavior:

YES

Dimensionality of Feature Space:

11 broad features (expanded as 1+1+1+1+1+1+2+12+24+24+6 i.e. d = 74)

Dimensionality of Reduced Feature Space:

4 broad features (expanded as 1+1+24+24 i.e. d = 50)

Sparsity:

Few features are toeplitz by nature and hence sparse, reduced feature set is not sparse.

Missing Data:

High Proportion

Significant Amount of Pre-processing required:

YES

TASK 2:

Estimating the success or popularity of the song before it actually reaches the market (listener) is an important piece of information from business perspective. Music Industry is one of the top gross revenue generating industries in the media segment. Such metric are usually factored in by experts and domain-experts. If such a task is delegated to machine and its prediction is equally better or more accurate than its human counterparts, it makes it valuable and hence this task is non-trivial.

Physical model:

Difficult to model

Non-Linear behavior:

YES

Dimensionality of Reduced Feature Space:

6 song-level features i.e. d = 4

Sparsity:

Few features are toeplitz by nature and hence sparse, reduced feature set is not sparse.

Missing Data:

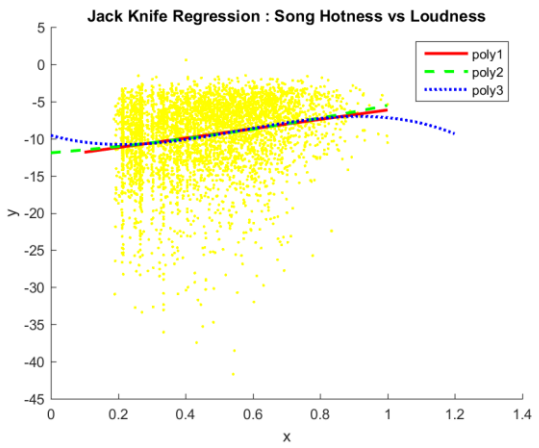
High Proportion

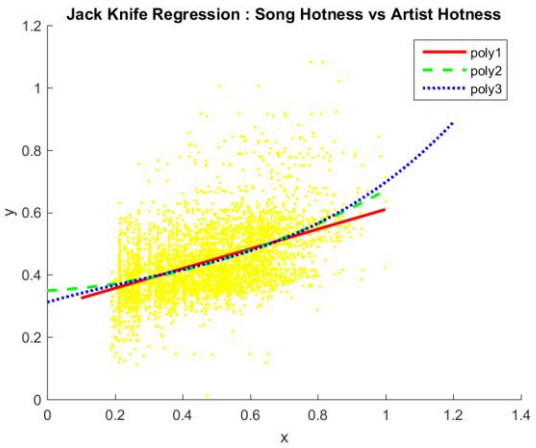
Significant Amount of Pre-processing required:

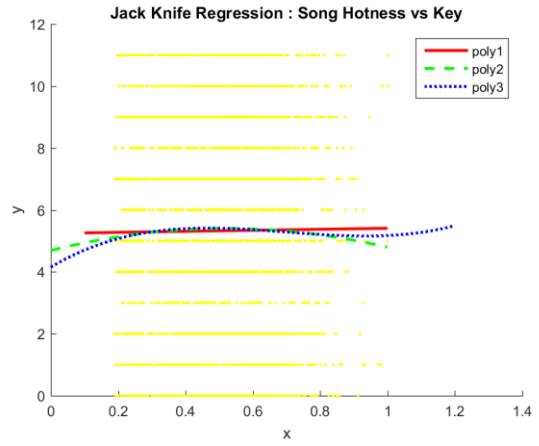
Already done

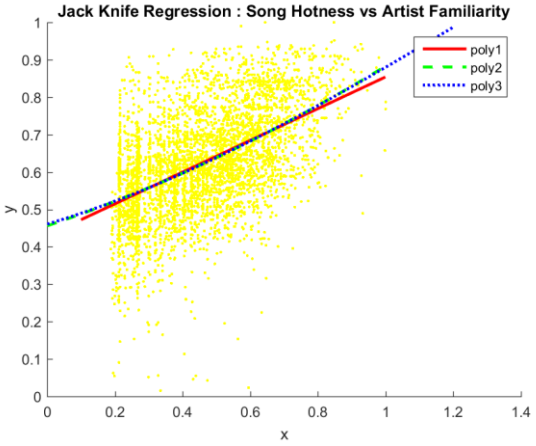
Initial Tests:

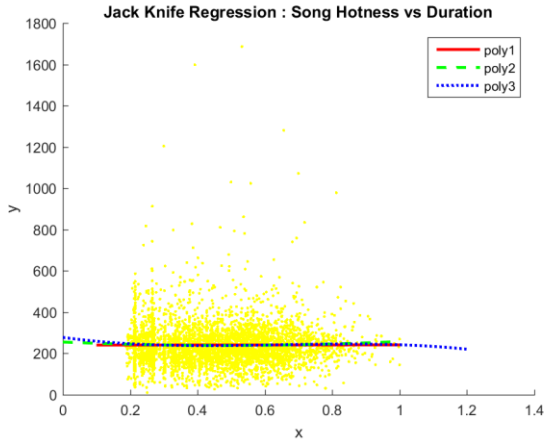
To show how the tasks are non-trivial & physical model cannot be easily modeled + non-linear pattern of distribution.

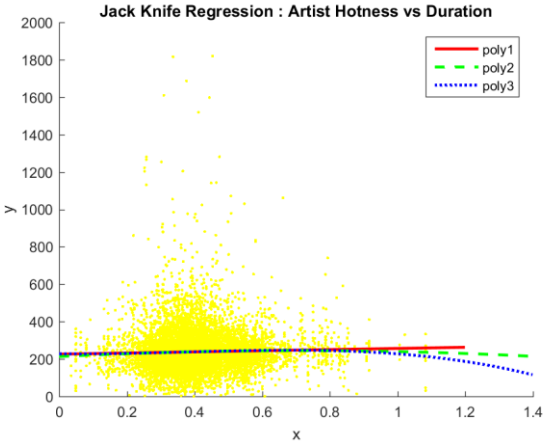

Jack Knife Regression : Song Hotness vs Loudness
poly1
poly2
poly3
x
y

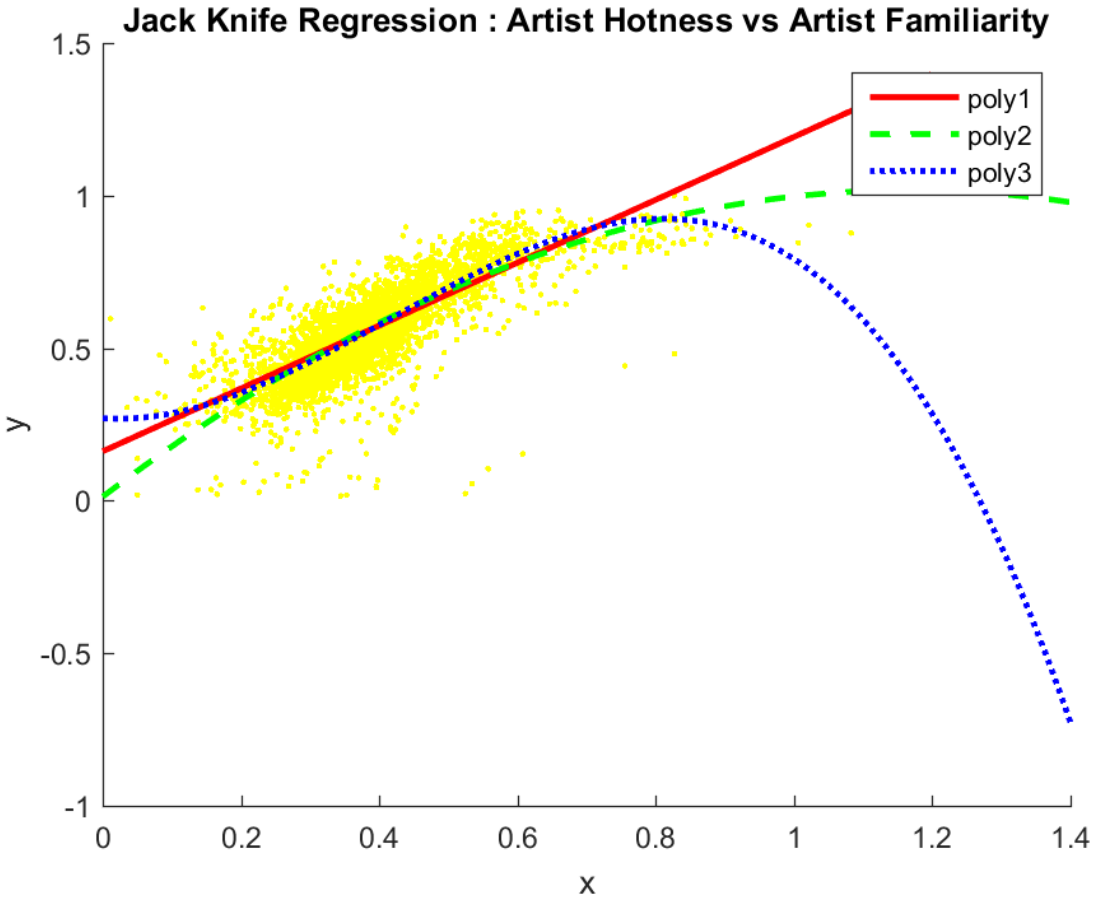

Jack Knife Regression : Song Hotness vs Artist Hotness
poly1
poly2
poly3
x
y


Jack Knife Regression : Song Hotness vs Key
poly1
poly2
poly3
x
y


Jack Knife Regression : Song Hotness vs Artist Familiarity
poly1
poly2
poly3
x
y


Jack Knife Regression : Song Hotness vs Duration
poly1
poly2
poly3
x
y


Jack Knife Regression : Artist Hotness vs Duration
poly1
poly2
poly3
x
y



The only linear trend we find is the high correlation of artist hotness & artist familiarity, which intuitively makes sense: *"A popular artist has higher rating i.e. hotness"*

LITERATURE REVIEW

To obtain a better understanding of the problem at hand, we started looking at the very beginning. In the year 2005, Music Information Retrieval started getting its due attentions with the first annual MIREX conference. Sorting through the articles/papers relevant to us, one of the earliest publications in this field is from 2006. This article[1] attempted to solve the problem of genre classification automatically for use in the coming electronic & digital media industry. Nicolas et al. looked at three broad domains – expert systems, unsupervised classification & supervised classification. A basic question they pointed out was – how should the genre be tagged: By artist, By Album or By song? They remarked (and I quote) *"it is not that simple anymore for an album, which may contain heterogeneous material. The same applies to artists; some of them have covered such a wide range of genres during their career that it does not make much sense to try to associate them with a specific class"*. Coming to features, they remarked in a very broad sense: timbre, loudness, melody, harmony and rhythm to be acceptable & important features. Humans with low or no musical training were able to recognize the genre with 53% & 72% accuracy by listening to 250ms and 3s of audio respectively. The authors also discuss the pros and cons of using singing part vs music part for genre classification. Some commonly used algorithms are: LDA, HMM, GMM, k-means, k-nearest neighbor, SVMs and ANN. MIREX 2005 best results for mean accuracy ranged from 62%-72% for different dataset which is good by the standards of human accuracy. But beyond the accuracy, the confusion matrix tells a different story. Most genres had mean truth prediction ranging from 32%-50%. Due to unbalanced datasets, the accuracy results did not provide a proper insight into the algorithms' efficacy.

[TABLE3] CONFUSION MATRIX FOR THE DATASET I AND FOR THE ALGORITHM SUBMITTED BY THE AUTHORS TO MIREX 2005.

TRUTH / PREDICTION	AMBIENT	BLUES	CLASSIC	ELECTRONIC	ETHNIC	FOLK	JAZZ	NEW-AGE	PUNK	ROCK
AMBIENT	**52.94%**	0.00%	0.00%	7.32%	4.82%	0.00%	0.00%	26.47%	0.00%	5.95%
BLUES	0.00%	**76.47%**	0.00%	0.00%	0.00%	4.17%	0.00%	0.00%	0.00%	3.57%
CLASSIC	2.94%	0.00%	**100.00%**	0.00%	8.43%	0.00%	0.00%	0.00%	0.00%	0.00%
ELECTRONIC	5.88%	0.00%	0.00%	**53.66%**	6.02%	4.17%	4.55%	5.88%	0.00%	19.05%
ETHNIC	2.94%	0.00%	0.00%	7.32%	**59.04%**	12.50%	4.55%	20.59%	0.00%	0.00%
FOLK	0.00%	5.88%	0.00%	1.22%	3.61%	**62.50%**	0.00%	2.94%	0.00%	2.38%
JAZZ	0.00%	2.94%	0.00%	3.66%	6.02%	4.17%	**81.82%**	8.82%	0.00%	5.95%
NEW AGE	29.41%	0.00%	0.00%	4.88%	4.82%	8.33%	4.55%	**32.35%**	0.00%	5.95%
PUNK	0.00%	0.00%	0.00%	0.00%	0.00%	4.17%	0.00%	0.00%	**100.00%**	4.76%
ROCK	5.88%	14.71%	0.00%	21.95%	7.23%	0.00%	4.55%	2.94%	0.00%	**52.38%**

[TABLE2] CLASSIFICATON ACCURACIES OF THE ALGORITHMS SUBMITTED AT MIREX 2005.

	DATASET 1 NORMALIZED	DATASET 2 NORMALIZED	DATASET 1	DATASET 2
MAX ACCURACY	73.04%	82.91%	77.75%	86.92%
MIN ACCURACY	53.47%	49,89%	55.29%	47.68%
MEAN ACCURACY	67.28%	72.61%	68.38%	75.88%

Results from Reference [1]

A similar project done towards the coursework was from Carnegie Mellon University students from music & machine learning departments in this report[2]. It proposed a cross-modal retrieval framework of model-blending combining lyrics and audio-features. The broadly assumed the genres to be: *"classical pop and rock, classical, dance and electronica, folk, hip-*

hop, jazz, metal, pop, rock and indie, soul and reggae" using a training set of about 143,589 songs (~ 14.35% of MSD) and 12,700 songs in test/tuning set (~1.27% of MSD). They used a very strong assumption based on most frequently occurring tags by people and assumed them to be the genres because of which their genre labels have overlaps. They remarked (and I quote) "Given the scale of the data we have, any super-linear algorithm such as K-NN or kernelized support vector machines (e.g. with a radial basis kernel) is computationally prohibitive". They used features like timbre, loudness and tempo as audio features as multi-layer HMM and Lyrics bag-of-words and emotional valence as subsequent layer. They used Canonical Correlation Analysis (CCA) to combine the features and used a multi-class logistic regression for the classification algorithm. Their accuracy in classification ranged from 16%-78% for different genres and due to unbalanced dataset, the results are not comparable while confusion matrix gives a very poor picture of truth prediction accuracy. Some interesting remarks they concluded were: *"BW-HMM performs best on classical music"*, *"pop and rock genres are typified less by musical features"*, pop rock folk and metal have higher degree of confusion, bag-of-word lyrics data is useful and often more accurate than audio features, bag-of-words are best for metal and hip-hop while extremely poor for classical or jazz, "*Combining the bag-of-words submodel with the timbre HMM achieves 35.2% accuracy, higher than either of the individual models"*, *"adding loudness and tempo can always bring about 2% to 3% increase in accuracy"*, lyrics emotional valence (sentiment features) are not useful.

Model	Acc. %	Model	Acc. %
(†)BW	31.4	Sp	28.3
(†‡)Lyrics	22.1	Sp+BW	32.6
(†)BW+Lyrics	35.2	Sp+Lyrics	33.3
LT+BW+Lyrics	37.5	LT+Sp+Lyrics	36.0
(†)LT+Sp+BW+Lyrics	**38.6**	LT+Sp+BW	34.1
Emot+LT+Sp+BW+Lyrics	38.5	*Most common class*	*10.5*

BW = Baum-Welch HMM, Sp = Spectral HMM, LT = loudness and tempo,
Lyrics = all words from lyrics, Emot = lyric emotion scores

Table 4: Final results from different model combinations. Best result is in bold. Note that 95% confidence intervals are approximately $\pm 1\%$ $(= 1/\sqrt{n})$. (†): These models are shown in Table 5 and Figure 3. (‡): The lyrics-only model has 40.0% accuracy on songs that have lyric data.

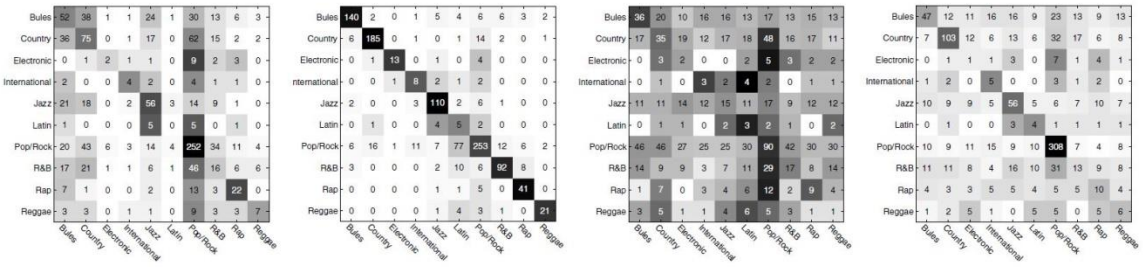
Results from Reference [2]

Genre	Lyrics	BW	BW+Lyrics	Final Model
classical	*0.0*	75.0	77.7	78.1
metal	*71.3*	65.8	57.8	63.6
hiphop	*67.7*	40.4	45.1	52.2
dance	*6.7*	34.2	45.1	45.7
jazz	*0.0*	18.2	30.1	36.9
folk	*35.9*	41.3	35.5	32.5
soul	*15.6*	19.7	19.1	24.6
rock/indie	*41.2*	6.9	17.5	21.7
pop	*37.4*	7.6	15.5	16.2
classic rock/pop	*21.0*	5.9	10.7	16.1
Totals (Table 4)	*40.0* (22.1)	31.4	35.2	38.6

Table 5: Accuracy (%) results per class. The "Lyrics" column only shows accuracy rates for songs that have lyrics data. These are the same models in Figure 3. The "Final Model" is LT+Sp+BW+Lyrics. Note there are about 1000 songs per class, therefore 95% confidence intervals are approximately ±3%.

Results from Reference [2]

A more concentrated effort was done by authors in this paper[3] from SIGIR '12. They treated classification as a inherent binary classification and cascaded such classifications to achieve a better accuracy using insights and proper weight assigning of data. With a combined classifier using Neural Networks, Logistic Regression, Naïve Bayes and SVM on several combinations of data, they claim to achieve 83.66% accuracy which is highly acceptable compared to human accuracy. They do use a unbalanced dataset but proper weight assigning negates the effect of imbalance.



(a) Neural Net by audio features (b) Neural Net by artist terms (c) Logistic Regression by lyrics (d) Naïve Bayes by social tags

Figure 2: Confusion matrixes of four sub-classifiers

Table 1: Data sources

Name	Extracted information	Number of records
MSD	Audio features, artist terms	1,000,000
MuisXmatch	Lyrics features	237,662
Last.fm tags	Social tags	505,216

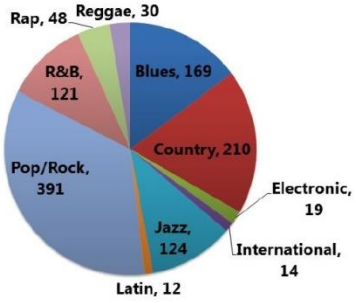
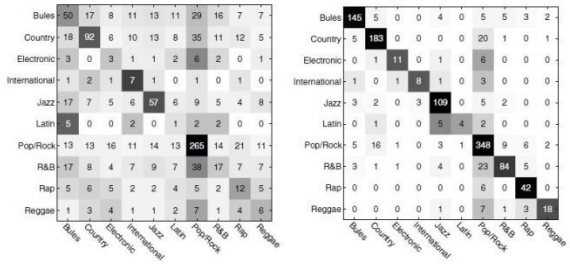


Figure 1: Genre Samples in AllMusic.com



(a) SVM by long vector (b) Combined classifier

Figure 3: Confusion matrixes by all data

Table 2: Experiment result comparison

Data	Method	Accuracy
Audio	Neural Net	42.70%
Artist terms	Neural Net	76.27%
Lyrics	Logistic Regression	18.54%
Social Tags	Naïve Bayes	48.59%
All data	SVM	44.82%
All data	Combined classifiers	**83.66%**

Results from Reference [3]

<u>Conclusions derived from literature review</u>

- Considering 10 genres, random chance (Bernoulli trials) gives us a probability of correct classification as 10%.
- Humans with low or no musical training can classify with accuracy of 53%-72%. Being able to match those accuracy levels or perform better than it can be set as a goal or acceptable accuracy for a machine based automatic genre classification.
- Instead of considering part of music, making use of complete music can lead to better global features and avoid local shortcomings.
- For unbalanced dataset, prediction accuracy is not a proper metric for comparison of algorithms. Confusion matrix helps but to an extent.
- Model-blending and/or cascaded models performs better
- Pop, Rock, Folk and Metal have higher degree of confusion among themselves – need to design a better segmentation concept or model-blending.
- Loudness and Tempo can bring about slight increase in accuracy
- Weight assigning of data is a solution for imbalanced dataset

PRIOR & RELATED WORK - NONE

Although the current project is new and no previous-work has been done by either of the members directly relating to the topic, our background provides motivation towards the work on the said topic.

Nishant has worked on Music Information Retrieval earlier and was involved in an early-stage start-up in the field and brings on board knowledge about the domain as well as the industry.

Arnav has worked extensively on music in production & post-production side and brings on board great insights about music industry.

Ahmet has a deep passion for music and brings on board domain knowledge of music & multimedia.

PROJECT FORMULATION & SETUP

The entire project was completed with a SCRUM approach –

- Develop Idea / Hypothesis
- Implement Idea / Hypothesis / Algorithm
- Test Algorithm on 10k set (~1% of MSD)
    - Compute error & other metrics
    - Cross Validate & Tune Parameters
    - Select best params / model
- Scale up the algorithm to test on MSD set (1 Million set)
    - Compute error & other metrics
    - Cross Validate & Tune Parameters
    - Select best params / model

As mentioned in the topic, we would be working with a real-life dataset which by its inherent nature is highly noisy & has missing information. Noise comes into picture given the dataset's dependency on user-ratings & tags on websites such as musicbrainz, 7digital, lastfm and many other websites. This unreliable source also means missing data or unintelligible data which needs pre-processing before it is usable.

Major areas where we spent significant amount of time & effort in pre-processing:

- Accessing the h5 files data (million songs – compressed h5 files totaling ~274 GB)
    - Initially we decided to use AWS Public Dataset for accessing the dataset which meant going over and beyond the free tier limits so we opted for a 1-time download of the entire repository from OpenScienceDataCloud
    - We ran initial tests on 10k set (~1% of MSD) and found the features and metadata of importance to us. After exhaustive testing on different algorithms we were able to formulate a reduced data set, thereby reducing 274GB of data to effectively ~1GB of data (~99.6% reduction)
    - For parallelizing the work without having to monitor resources, we manually partitioned the data into 4 classes – A to F, G to M, N to T, U to Z.
- Imputation of Data
    - We had missing information for almost all the fields of which Year was a major categorical feature. To fill this data (to improve training effectiveness), we assumed that the genre can be labeled through songs' album name and using album name and Wikipedia API the url to DBPEDIA (semantic Web Database) then we can parse the RDF ontology for a branch labeled as 'releaseDate' and extract the year from it using regular expressions.
    - We followed the same procedure to impute missing genre labels (for ground truth values) using Wikipedia/DBPedia Tags from RDF ontology with a branch labeled as 'genre'
    - The above tasks were run on 4 parallel t2.micro instance in AWS. Each instance had a 1vCPU with High Frequency Xeon Processors burstable performance upto 3.3 GHz.

ALGORITHMS FORMULATION & SETUP

For all purposes of our algorithms, we use the Python Scikit Learn library. For the visualizations, we mostly used Python Matplotlib. Some initial test for visualizing data presented here were also done in MATLAB and Microsoft Excel. Various parameters were tried as provided by the Scikit learn library but only the results for the best performing classifiers are reported.

Initially tests were made using a small subset of million songs (about 10,000 songs). Running of these algorithms on the 300 GB million song were memory and compute exhaustive tasks. So all the algorithms for million song were performed on the Amazon Web Services (AWS) servers. Although setting it up and installing the python platform on AWS took us significant amount of time but, the time that the algorithms took to run drastically improved. Instead of using the entire data set, we extracted the features as we needed, performed feature combination as described in later sections, crawled on Wikipedia to get missing data and true labels, compiled our own dataset as saved it as a .bin file to be used by our model.

By this method, we were able to reduce the ~300 GB million song dataset to just 1 GB binary file for genre classification. We trained multiple algorithms on this data set and used separate validation and test sets. The dataset was initially divided into model selection and testing data. Since loading them to memory using python takes time we used Pickle objects and read these python object from file. The model selection dataset was further divided into training and validation sets. The training set was used to fit the classifier and tune parameters. The validation was done on individual machine learning algorithms and the testing set was used on final model which was a combination of various models.

All the algorithms we used and how we set up our model to combine different algorithms is presented in next section.

METHODOLOGY

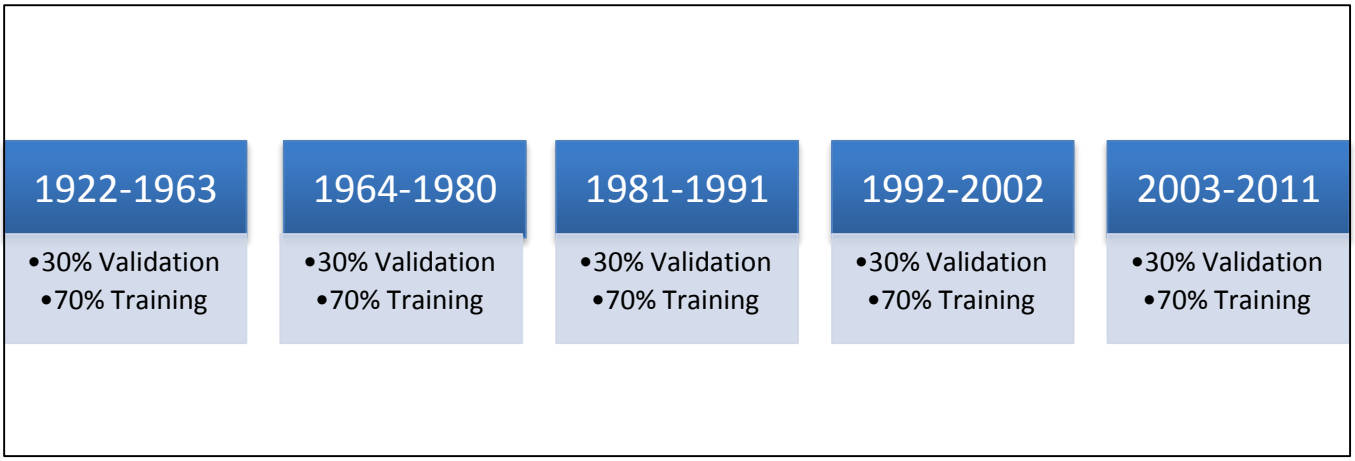
In our dataset we have one million songs. First of all, we should state that we are going to use 20% percent of that data for testing.

The remaining data, which is 80% percent is divided into 5 groups according to the specific time periods. In million song dataset, the year of the songs range between 1922 – 2011. For our model, we divide them into time periods as

- 1922– 1963
- 1964 – 1980
- 1981 – 1991
- 1992 – 2002
- 2003 – 2011

Each of the time period group is divided in a way that 70% will be training set and 30% will be validation set. Each time period group will be trained separately.

To get the testing result, we will classify each test sample by its corresponding time period model. For example, suppose our test sample is a song from 1970, then we will use the model trained with the samples between the years 1964 - 1980 to predict its output label (genre). With this approach we are expecting a higher accuracy for correct classification.



IMPLEMENTATION

FEATURE SELECTION

During the phase when working with the 10K set (1% MSD), out of 74 potential features we attempted to reduce the dimensionality of the features by using the following methods:

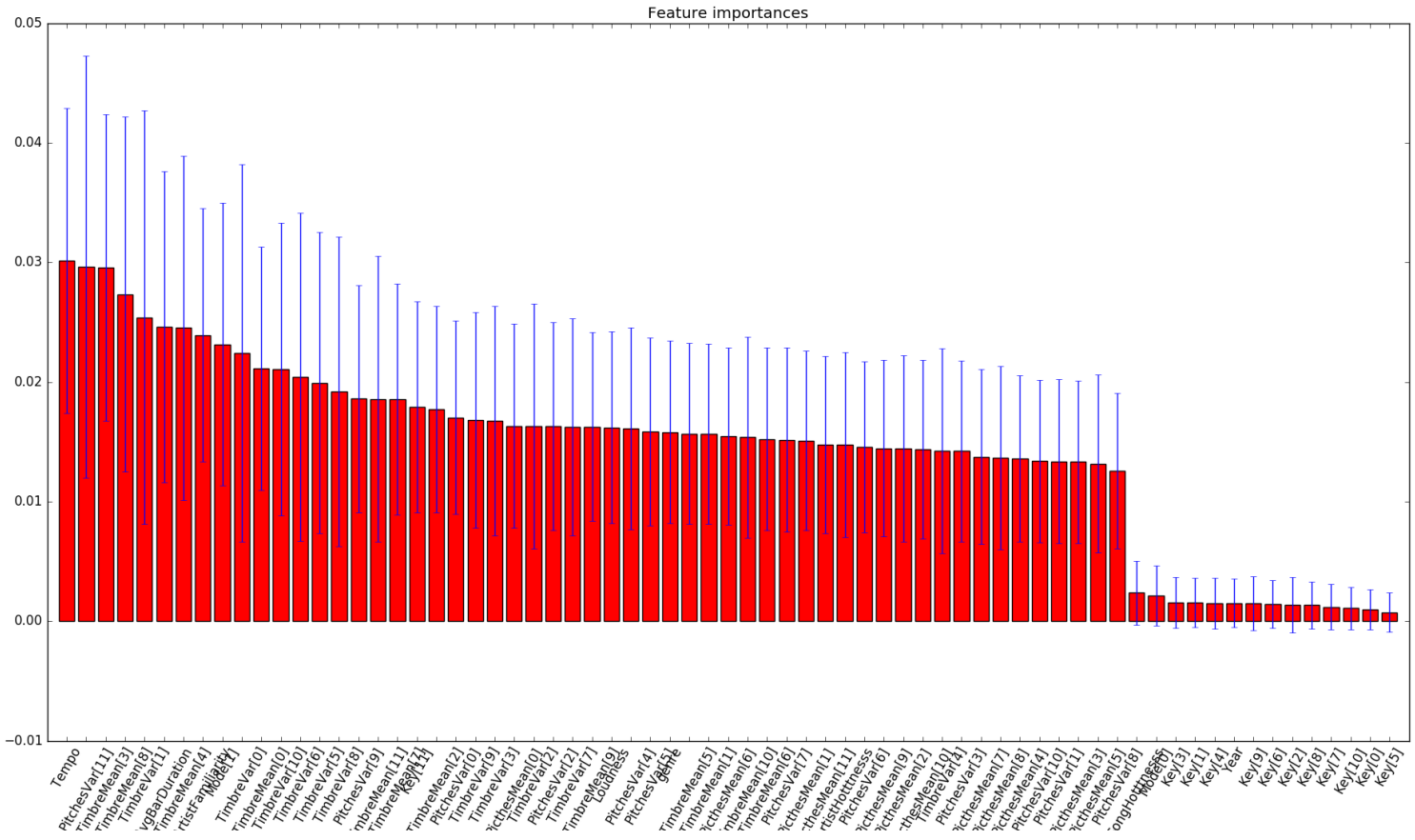
Principal Component Analysis (PCA) a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Randomized PCA is similar to PCA except that the python implementation is based on randomization + approximation rather than the strongly mathematical linear algebra SVD method.

Manifold PCA is a hybrid PCA algorithm that uses concept of manifold learning to reduce the features while not falling victim to non-linearity and imbalance of feature space.

We found that the manifold PCA was most effective in reducing the features in a proper fashion on our dataset. Most python machine learning algorithms have an built-in PCA technique to reduce features. Implementing manifold PCA followed by the algorithm is the best approach but owing to time constraints we skipped doing it.

The figure below shows the features in order of that importance. This figure was obtained while training the Random Forest Classifier. The bars represent the standard error between the features.



MODEL SELECTION

Parametric Models

Parametric Models in Machine Learning are algorithms which are guided by providing well-tuned parameters for generating the model from data. These models usually require less data compared to non-parametric models and also perform better than them on real datasets.

Linear discriminant analysis (LDA) is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements. LDA approaches the problem by assuming that the conditional probability density functions $p(\underline{x}|y=0)$ and $p(\underline{x}|y=1)$ are both normally distributed with mean and covariance parameters $(\underline{\mu}_0, \sum_0)$ and $(\underline{\mu}_1, \sum_1)$, respectively. Under this assumption, the Bayes optimal solution is to predict points as being from the second class if the log of the likelihood ratios is below some threshold T, so that;

$$(\vec{x}-\vec{\mu}_0)^T\Sigma_0^{-1}(\vec{x}-\vec{\mu}_0)+\ln|\Sigma_0|-(\vec{x}-\vec{\mu}_1)^T\Sigma_1^{-1}(\vec{x}-\vec{\mu}_1)-\ln|\Sigma_1| \;>\; T$$

Assuming the class covariances to be equal, i.e. $\sum_0 = \sum_1 = \sum_0$ and that the class covariances have full rank, above decision criterion becomes a threshold on the dot product:

$$\vec{w} \cdot \vec{x} > c$$

for some threshold c, where

$$\vec{w} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$
$$c = \frac{1}{2}(T - \vec{\mu_0}^T \Sigma_0^{-1} \vec{\mu_0} + \vec{\mu_1}^T \Sigma_1^{-1} \vec{\mu_1})$$

Quadratic discriminant analysis (QDA) is similar to LDA except that there is no assumption that the covariance of each of the classes is identical. When the normality assumption is true, the best possible test for the hypothesis that a given measurement is from a given class is the likelihood ratio test. Suppose there are only two groups, y ∈ {0, 1} , and the means of each class are defined to be $\mu_{y=0}$, $\mu_{y=1}$ and the covariances are defined as $\sum_{y=0}$, $\sum_{y=1}$. Then the likelihood ratio will be given by:

$$\frac{\sqrt{2\pi|\Sigma_{y=1}|}^{-1} \exp\left(-\frac{1}{2}(x-\mu_{y=1})^T \Sigma_{y=1}^{-1}(x-\mu_{y=1})\right)}{\sqrt{2\pi|\Sigma_{y=0}|}^{-1} \exp\left(-\frac{1}{2}(x-\mu_{y=0})^T \Sigma_{y=0}^{-1}(x-\mu_{y=0})\right)} < t$$

for some threshold t. After some rearrangement, it can be shown that the resulting separating surface between the classes is a quadratic. The sample estimates of the mean vector and variance-covariance matrices will substitute the population quantities in this formula.

Support vector machines (SVMs) are based on two properties, margin maximization (which allows for a good generalizationof the classifier) and nonlinear transformation of the feature space with kernels (as a data set is more easily separable in a high dimensional feature space).

We tried using different Kernel Types ('linear', 'poly', 'rbf', 'precomputed', 'sigmoid') and for the purpose of conciseness of the report only the cases that work best for our dataset.

Non Parametric Models

Non Parametric Models in Machine Learning are algorithms which do not require specific parameters for generating the model from data. Compared to parametric models, non-parametric models require more number of observations points (training samples) for the model to work optimally. For this reason, non-parametric models behave not so optimally on real datasets and are usually treated as baseline classifiers to quickly test the data and achieve a baseline result for comparing other algorithms.

k-means clustering is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. The data space is partitioned into Voronoi cells. The problem is

computationally difficult (NP-hard). On 10K set (1% MSD), k-means performed not optimally and hence we decided to not pursue it as an algorithm of our choice.

k-nearest neighbors is an extension of the family of voting classifiers and is referred to as lazy-learner as the function is only approximated locally and all computation is deferred until classification. On 10K set (1% MSD), k-nearest neighbors performed optimally and hence we decided to pursue it as an algorithm of our choice as a baseline classifier. k-nearest neighbors can be implemented using 'auto', 'kd-tree', 'ball-tree' and 'brute-force' algorithms where 'auto' defaults to a 'kd-tree' for k <= N/2 and 'brute-force' for k > N/2. When the distance metric is pre-specified 'auto' deafults to 'ball-tree'. All these algorithms change the way the near-ness is calculated but does not affect the overall model and hence k-nn is considered non-parametric

k-nearest centroid is similar to k-nearest and k-means algorithm other than instead of relying on distance between points and mean points, it takes distance of data from centroids of clusters in model. On 10K set (1% MSD), k-nearest neighbors performed optimally and hence we decided to pursue it as an algorithm of our choice as a baseline classifier.

Naïve Bayes (Gaussian) is the Naïve Bayes model with a strong Gaussian data distribution assumption. Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. On 10K set (1% MSD), k-nearest neighbors performed optimally and hence we decided to pursue it as an algorithm of our choice as a baseline classifier. Compared to other non-parametric models, Naïve Bayes performs better and using in conjunction with dimension reduction algorithms like PCA provides even better results. We also tried variants of naïve bayes like multinomial naïve bayes and Bernoulli naïve bayes but considering the random nature of our data, gaussian naïve bayes gave better results.

Machine Learning Ensemble Meta-Algorithms

Bagging

Bagging, also referred as bootstrap aggregation is a machine learning ensemble meta-algorithms designed to improve the stability and accuracy of the base algorithm. It reduces variance and helps avoid over fitting. Given a standard training set D of size n, bagging generates m new training sets D_i, each of size n', by sampling from D uniformly and with replacement. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification). On 10K set (1% MSD), bagging using k-nearest neighbor classifier performed optimally and hence we decided to pursue it as an algorithm of our choice for improving our base classifiers like random forests, etc.

Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. A weak learner is defined to be a classifier which is only slightly

correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.

AdaBoost can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems, however, it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (e.g., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

Random Forest operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. Decision trees are a popular method for various machine learning tasks. In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, because they have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a decision or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x')$$

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is

deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

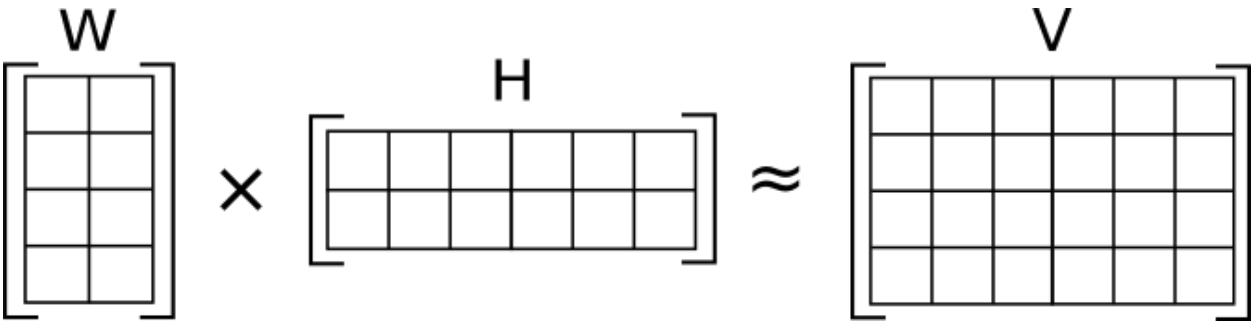
Random Forests use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called feature bagging. Typically, for a classification problem with p features, $\sqrt{p}$ features are used in each split. The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

Too look for how our data is sampled we used the unsupervised **Mean Shift Clustering** approach that locates the discrete density functions for data samples. We intended to use this for classification, but the clusters do not represent the actual 10 genres in our case. The visualization of these clusters have been shown in later section where we discuss the various models on our dataset. It is generally useful for detecting the modes of this density. It uses a kernel function to determine the weight of nearby points and then re-estimates the mean.

Regression Algorithms

Non-Negative Matrix Factorization (NMF)

NMF is a group of algorithms in multivariate analysis and linear algebra where a matrix V can be factorized into two matrices W and H, with the property that all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to inspect. Since the problem is not exactly solvable in general, it is commonly approximated numerically and with an acceptable degree of error, the approximation works in place if actual estimation.



With a polynomial time execution time, approximations are better than exact-NMF and its application towards a regression problem i.e. predicting problem seems very feasible. On 10K set (1% MSD), using few random samples NMF seemed to perform well so we decided to pursue NMF as the algorithm of choice for regression problems.

FEATURE SPACE

We used The Million Song Dataset that is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. The core of the dataset is the feature analysis and metadata for one million songs, provided by The Echo Nest. Despite the features already provided, we had to extract and combine features from within the dataset to improve the accuracy of the models. Each sample in the dataset, a song, had different number of features based on it's duration, because of which we had to implement statistical analysis to combine features to make it useful for training a classifier.

Most of the song features that we used have been derived from the Echo Nest "Analyze" music audio analysis tool available as a free public web API and a standalone command-line binary program. These features were already provided in the Million Song Database. We also used some meta-data e.g. year of release as features that have been derived from The Musicbrainz dataset. We describe all these features here.

The Echo Nest API provides following features:

• **Meta data**: analyze, compute, and track information.

• **Track data**

‣ **Time signature**: an estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).

‣ **Key**: the estimated overall key of a track. The key identifies the tonic triad, the chord, major or minor, which represents the final point of rest of a piece. The *key* is a track-level attribute ranging from 0 to 11 and corresponding to one of the 12 keys: C, C#, D, etc. up to B. If no key was detected, the value is -1. The *mode* is equal to 0 or 1 for "minor" or "major" and may be -1 in case of no result. Note that the major key (e.g. C major) could more likely be confused with the minor key at 3 semitones lower (e.g. A minor) as both keys carry the same pitches.

‣ **Mode**: indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.

‣ **Tempo**: the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

‣ **Loudness**: the overall loudness of a track in decibels (dB). Loudness values in the are averaged across an entire track and are useful for comparing relative loudness of

segments and tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude).

‣ **Duration**: the duration of a track in seconds as precisely computed by the audio decoder.

‣ **End of fade in**: the end of the fade-in introduction to a track in seconds.

‣ **Start of fade out**: the start of the fade out at the end of a track in seconds.

• **Sequenced data**: the audio is broken down into musically relevant elements that occur sequenced in time. From smallest to largest those include:

‣ **Segments**: a set of sound entities (typically under a second) each relatively uniform in timbre and harmony. Segments are characterized by their perceptual onsets and duration in seconds, loudness (dB), pitch and timbral content.

‣ **Tatums**: list of tatum markers, in seconds. Tatums represent the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments).

‣ **Beats**: list of beat markers, in seconds. A beat is the basic time unit of a piece of music; for example, each tick of a metronome. Beats are typically multiples of tatums.

‣ **Bars**: list of bar markers, in seconds. A bar (or measure) is a segment of time defined as a given number of beats. Bar offsets also indicate downbeats, the first beat of the measure.

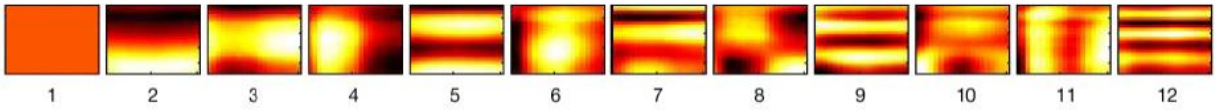
‣ S**ections**: a set of section markers, in seconds. Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc. Each section contains its own descriptions of tempo, key, mode, time_signature, and loudness.

• **Pitch:**

Pitch content is given by a “chroma” vector, corresponding to the 12 pitch classes C, C#, D to B, with values ranging from 0 to 1 that describe the relative dominance of every pitch in the chromatic scale. For example a C Major chord would likely be represented by large values of C, E and G (i.e. classes 0, 4, and 7). Vectors are normalized to 1 by their strongest dimension, therefore noisy sounds are likely represented by values that are all close to 1, while pure tones are described by one value at 1 (the pitch) and others near 0.

• **Timbre:**

Timbre is the quality of a musical note or sound that distinguishes different types of musical instruments, or voices. It is a complex notion also referred to as sound color, texture, or tone quality, and is derived from the shape of a segment's spectro-temporal surface, independently of pitch and loudness. The Echo Nest Analyzer's *timbre* feature is a vector that includes 12 unbounded values roughly centered around 0. Those values are high level abstractions of the spectral surface, ordered by degree of importance. For completeness however, the first dimension represents the average loudness of the segment; second emphasizes brightness; third is more closely correlated to the flatness of a sound; fourth to sounds with a stronger attack; etc. The image below represents the 12 basis functions (i.e. template segments). The actual timbre of the segment is best described as a linear combination of these 12 basis functions weighted by the coefficient values: timbre = c1 x b1 + c2 x b2 + ... + c12 x b12, where c1 to c12 represent the 12 coefficients and b1 to b12 the 12 basis functions as displayed below. Timbre vectors are best used in comparison with each other.



12 basis functions for the timbre vector: x = time, y = frequency, z = amplitude

• **Loudness:**

Loudness information (i.e. attack, decay) is given by three data points, including dB value at onset (*loudness_start*), dB value at peak (*loudness_max*), and segment-relative offset for the peak loudness (*loudness_max_time)*. The dB value at onset is equivalent to the dB value at offset for the preceding *segment*. The last segment specifies a dB value at offset (*loudness*_end) as well.

- **Loudness_start:** indicates the loudness level at the start of the segment
- **Loudness_max_time:** offset within the segment of the point of maximum loudness
- **Loudness_max:** peak loudness value within the segment

Other than these we also have the following metadata within the song dataset that we use to merge and match with true labels scraped from Wikipedia.

• Artist MBID**:** MusicBrainzID associated with artist

• Artist MB tags**:** User defined tags related to artists.

•Artist MB tags count**:** Raw tag count of the tags that the artist received on musicrainz

• Artist Name**:** Artist Name

• Artist PlayMe ID**:** The ID of artist on PlayMe service

• Artist Terms**:** The tags received on a track at the Echo Nest

• Artist Freq**:** Frequency of the Artist terms

• Artist Terms weight**:** Weight of the artist terms from the Echo Nest

• Audio MD5**:** hash code of the audio used for the analysis by The Echo Nest

• Album Release: Album name from which the track was taken, some songs / tracks can come from many albums, in that case only one of the song is presented.

• Release ID**:** The ID of the release (album) on the service 7digital.com

• Similar Artist**:** A list of 100 artists (their Echo Nest ID) similar to the track

• Song Hotnesss**:** The hotness score of a track on scale of 0 to 1 according to The Echo Nest, when downloaded.

• Song ID**:** The Echo Nest song ID

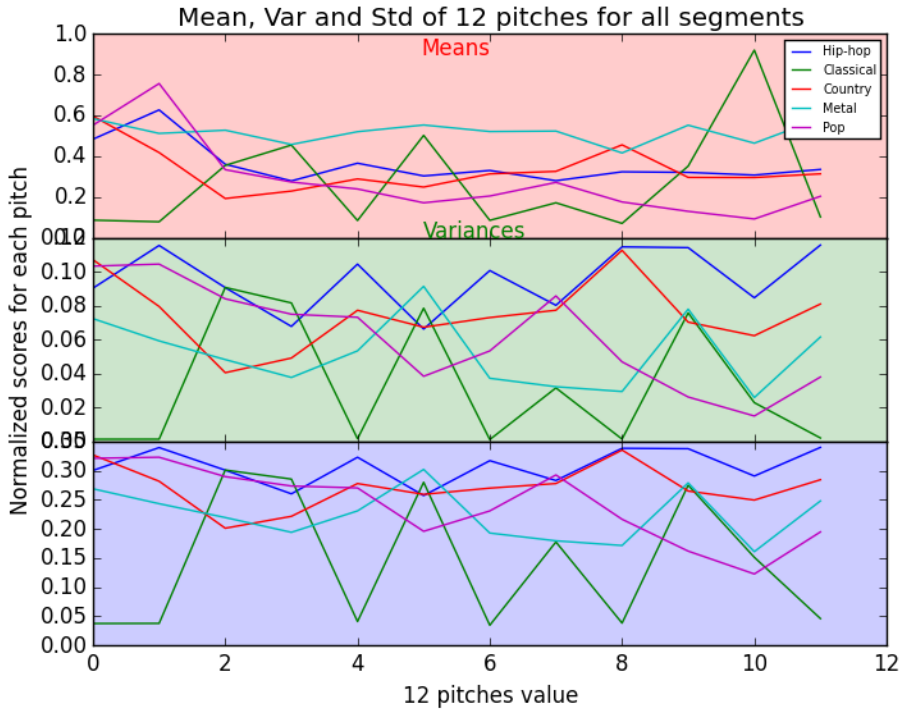
• Start of fade out**:** Start time of the fade out, in seconds, at the end of the song, according to The Echo Nest

• Track ID: The Echo Nest ID of this particular track on which the analysis was done

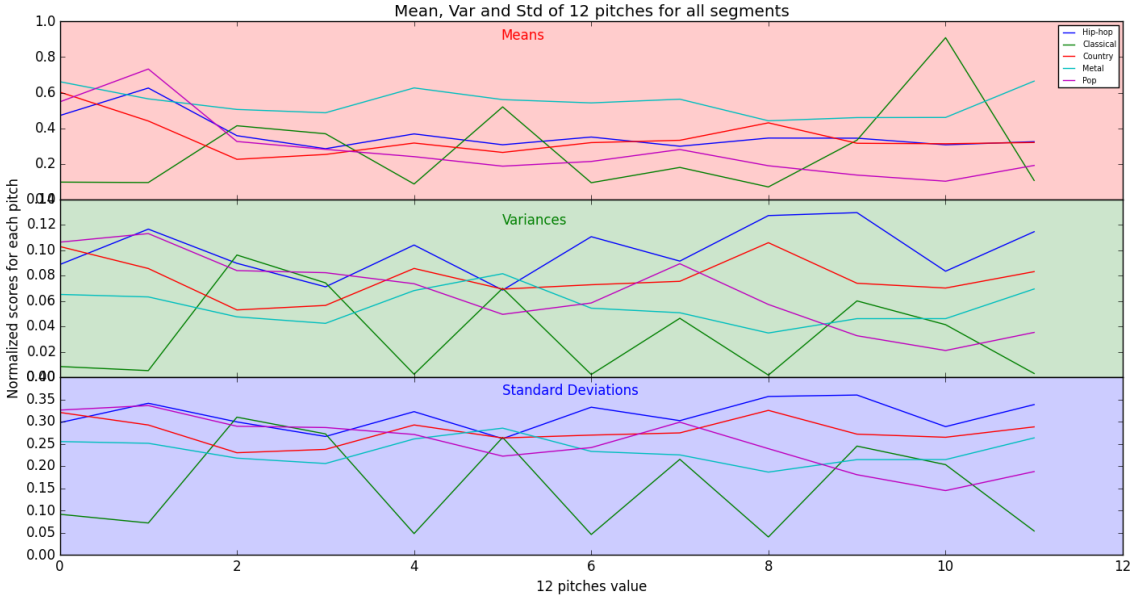
• Year: Year when this song was released, according to musicbrainz.org

PRE PROCESSING AND FEATURE EXTRACTION

Because of the segments described above, the features had different lengths. The pitch of the song is an important feature that could be used to classify genre of a music. We used the statistical quantities, Mean, Variance and Standard Deviations to combine all the segments of a particular track. These have been visualized for 5 different genres for one particular track in the image below.



Although these metrics work well for the genres for one song, we still limit ourselves to only those segments that had a confidence score of more than 50%.



We used the same procedure for other sequenced features that included segments and segment scores as described in Section 1.1. Only the mean and variances were selected as features for segment pitches and timbre values.

The mode of the song, which defines the Major or Minor key that the song is played in was present as a 0 or 1 values respectively. These features were expanded to a 2 dimension feature with one dimension being 1 if the mode is Major and 0 is not and the other dimension being 1 is mode is minor and 0 otherwise. Similar way, the expansion was done for time signature. The time signature values are present as the beats per bar in music theory. These were 6 values from 0-5 for time signature of 3/4, 4/4 and so on. Each of these time signatures were treated as a single binary feature and if the song had either one of the feature, that dimension of time signature was treated as 1 and rest as 0.

TRAINING PROCESS

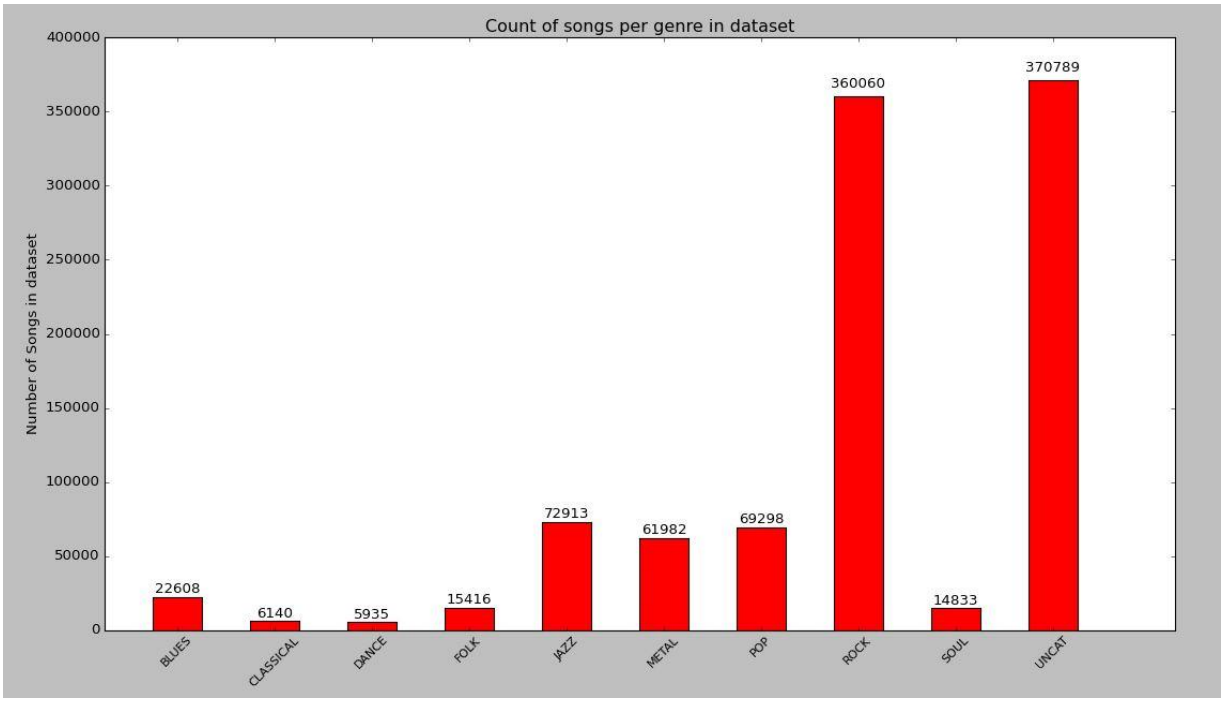
We used various training models to test for accuracy on our dataset. The training dataset come after separating the 20% test set from the full data. The remaining 80% was further divided and 70 % of this data was used for training different models while the remaining 30% for validating the models. Further, the model was created for a particular timeline as an inspiration from Google's Big Picture Music Research Group described in next section.

We already knew that that the data set was not linear, still we tried linear classifiers like Linear discriminant analysis, SVM with linear kernel etc. Even the classification algorithms like quadratic discriminant analysis did not work well enough on our dataset. Here we discuss all the different machine learning techniques and classifiers that we used and report the results that we obtained along with the parameters used to train the model. All models were trained using the algorithms provided by Python Sci-kit learn module. The data set was loaded as Pandas Dataframe from the precompiled python pickle objects for faster computation and saving the memory.

All the samples which had at least one feature missing were ignored and not involved in the training and testing set. These features with NaN values were dropped. This reduced the number of samples from a million to about 600,000. The true labels (genres) were extracted from the histograms of the artist terms. For unavailable genres, we wrote a script to get a Wikipedia album URL link to which the track with missing label belonged to. The true labels were then obtained by our script from the Wikipedia API. This process ran over AWS for several days for all the million songs.

After obtaining the genres from all the sources we could, i.e. human labeled artist tags as well as Wikipedia, we merged the features and genres corresponding to a unique track ID and created our own million song database with our features extracted and combined as a binary file. The tracks for which we could not get the true genre labels were marked as UNCAT (for uncategorized) and saved to dataset. However we did not use the song with UNCAT labels for our training or testing. After ignoring the missing values and ignoring the UNCAT genres we were left with about 550,000 samples to perform training and testing on.

We also ran test to see how our dataset is biased towards Rock genre. We have a lot of Rock samples in our dataset however not enough other types of genres. The figure below shows this distribution.



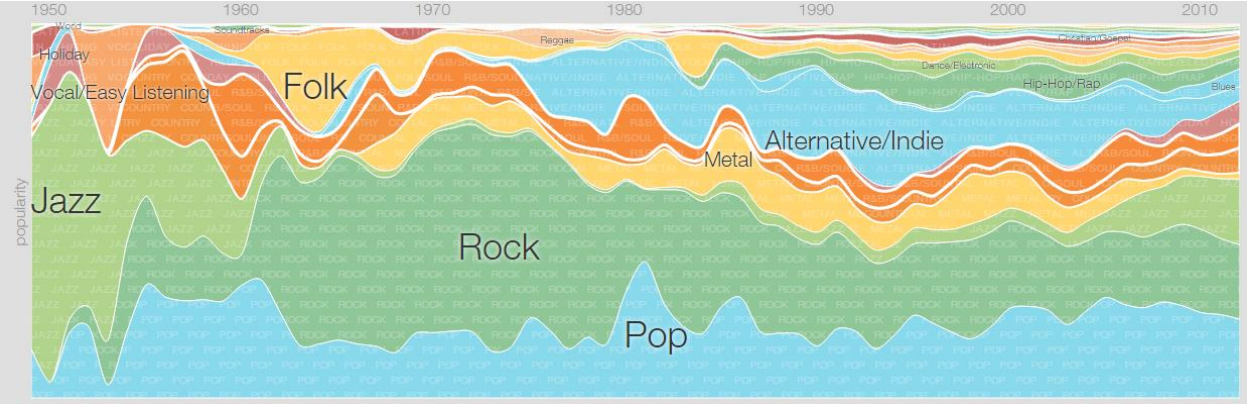
We also notice, despite using the dataset and the Wikipedia parsing, we still have a lot of Uncategorized (marked as 'UNCAT') genres.

The data is also highly skewed with a lot of Rock samples. To prevent the decision boundaries from being biased towards Rock genre, we limit the number of rock samples while training. The process of training. The training was done on further dividing our dataset into 5 timelines based on years and a general dataset for entire training set. We describe this splitting with analysis on in next section.

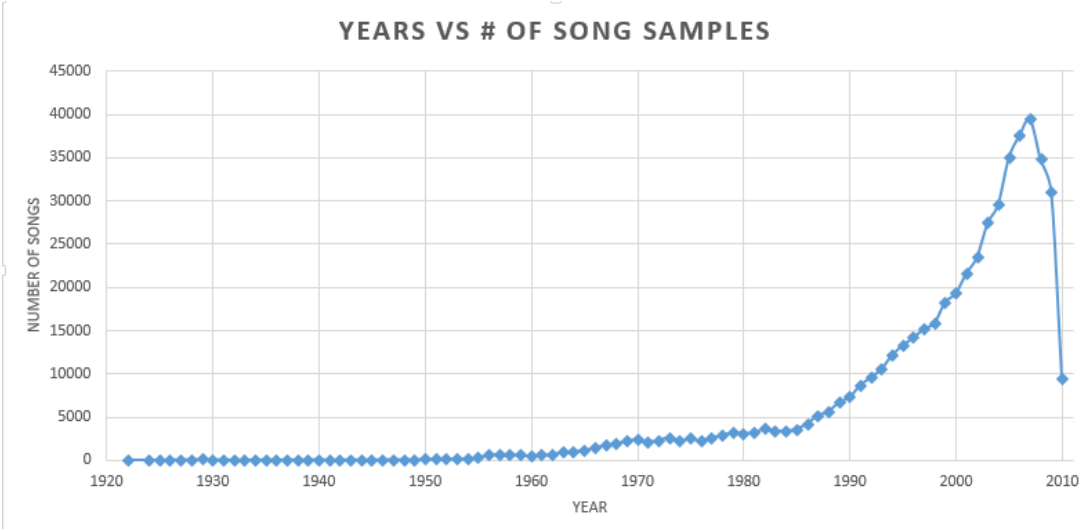
This dataset could further be used by the research community to improve the models for predicting the genre of a music and making a more robust Music Recommendation Engine or Music Information Retrieval system.

TESTING, VALIDATION AND MODEL SELECTION

We used all the models described above to select the best model and improve the model even further. For this purpose, we trained different models depending on the release year of the track. The motivation for this came from [Google Big Picture Music Research]. The visualization below shows the spread of music according to genre through the timeline.



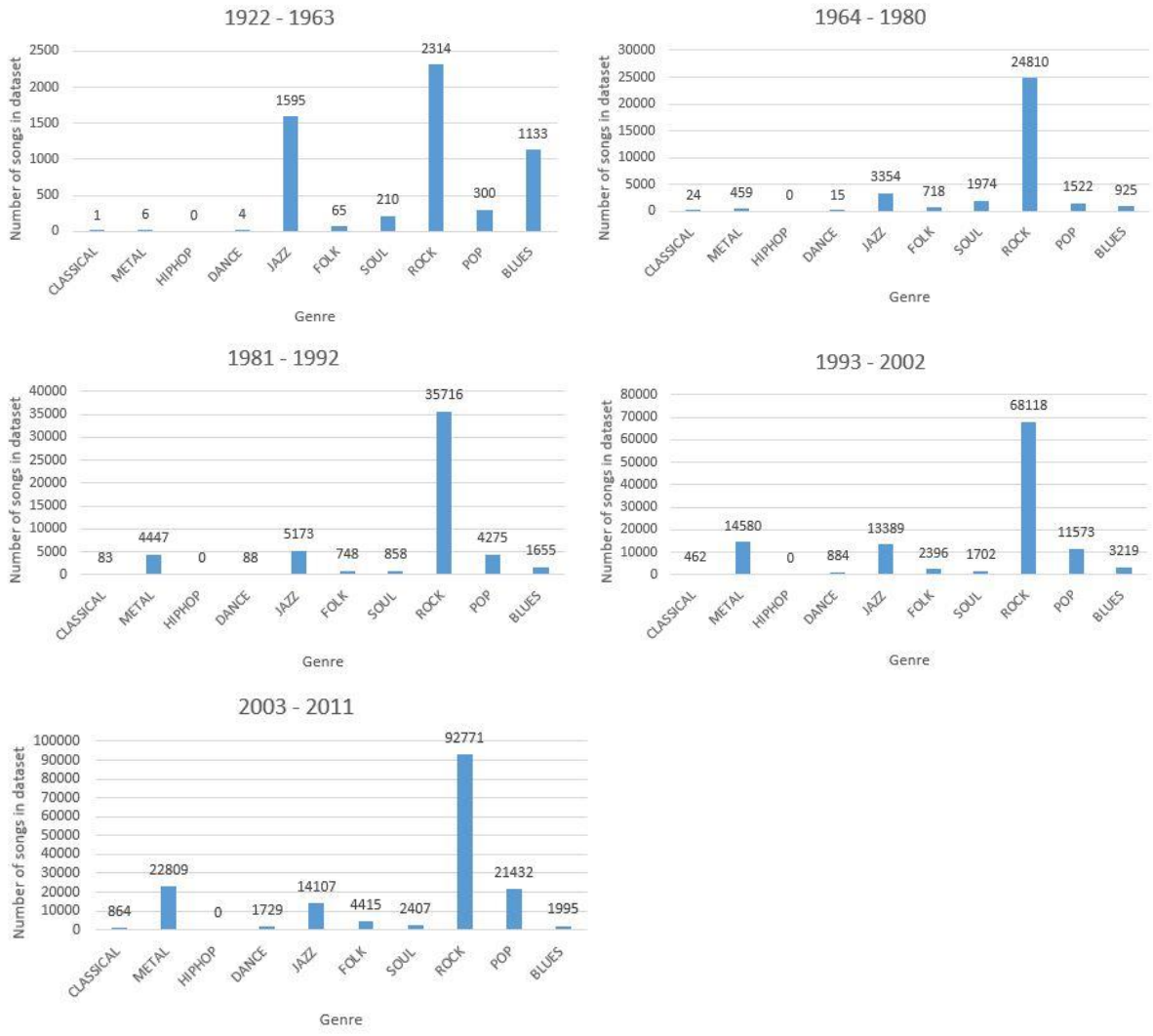
This next visualization shows the number of songs distributed over the years. Out of the million songs, we only had 515,574 (~50%) which had year values. The NaN and 0 values were ignored for the purpose of this visualization. The data is highly skewed and we have very less samples for earlier years.



For this purpose and a motivation from Google's Music Research, we create different models for our data. When training, we filter out the track release year and create different models dependent on the year of release. Our models based on the year release is divided as follows:

1. 1922 – 1963
2. 1964 – 1980
3. 1981 – 1992
4. 1993 – 2002
5. 2003 – 2011

The image below shows the major number of songs in our data set corresponding to particular genre for the major timelines that we considered.



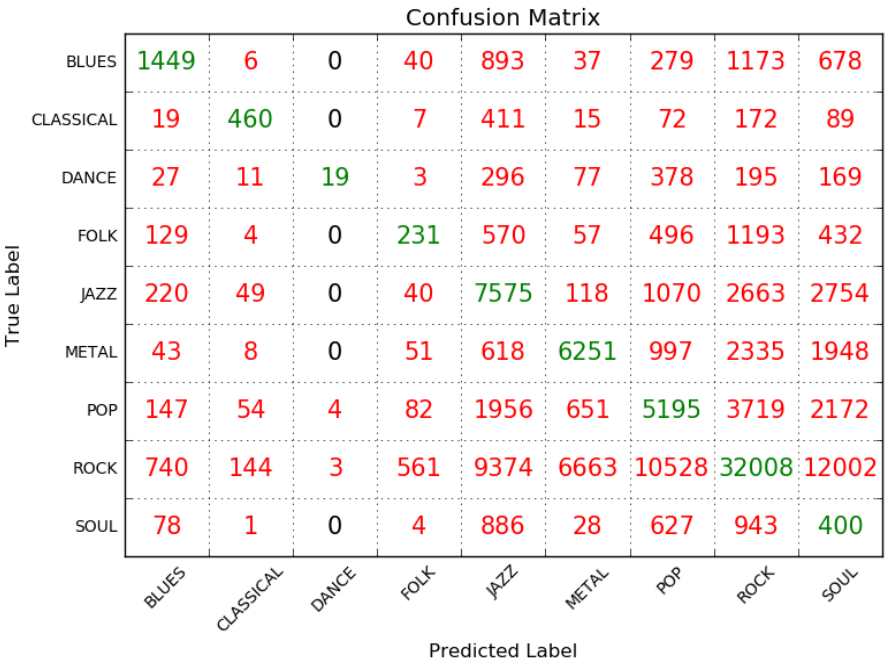
We used multiple machine learning algorithms to create a combinational model. These are a combination of the machine learning algorithms on each of the dataset based on these timelines. We did not use the year as a feature but only as a metadata for separating dataset to build our model.

The validation was performed on 30% of the 80% of training data. The final testing was performed on 20% of the entire dataset. This test set was exclusively used only for testing the final model and nowhere else. The final test result reported are for this test set.

FINAL RESULTS

Model	Error Rates on Validation Data						Comments/Parameters tuned on training set
	1922-1963	**1964-1980**	**1981-1992**	**1993-2002**	**2003-2011**	**Full set**	
LDA	**0.3260**	**0.4310**	**0.5269**	**0.5283**	**0.5422**	**0.5714**	**Using SVD**
QDA	**0.3501**	**0.5213**	**0.6263**	**0.6679**	**0.6920**	**0.6854**	**No priors, regularization=0.001**
SVM	**0.6014**	**0.2627**	**0.3168**	**0.4099**	**0.9220**	**0.9323**	**RBF Kernel, #iterations=10k**
kNN	**0.5436**	**0.7457**	**0.7773**	**0.7718**	**0.7655**	**0.7905**	**n=2, leaf size = 15**
Naïve Bayes	**0.6385**	**0.9385**	**0.8366**	**0.7223**	**0.7292**	**0.7215**	**Non Parametric**
Adaboost	**0.4751**	**0.4747**	**0.6729**	**0.6255**	**0.6156**	**0.6236**	**#estimators=50**
Random Forest	**0.2665**	**0.3578**	**0.4641**	**0.4665**	**0.4182**	**0.5146**	**#trees=500**
K Nearest Centroid	**0.7763**	**0.6048**	**0.7702**	**0.8028**	**0.8020**	**0.8202**	**Non Parametric**

From the above table, based on each of the timelines we select the best model marked in red. For the remaining 20% of the unseen data we extract the year and run them through the model based on the algorithm performing best for that timeline. The error % obtained on our test data is 57.40% which is better than the benchmark performance in paper [1]. This performance cannot be compared with [1] due to highly nonlinear distribution of data. The paper [3] used an additional lyrics features along with bag of words and obtained much better error rate. The confusion matrix on of test data is shown on the right.



	MSE AVG	MSE AVG %	MSE AVG (zeroed)	MSE AVG (zeroed) %	MSE AVG (0.5)	MSE AVG (0.5) %	MSE AVG (fam)	MSE AVG (fam) %
CLASSICAL	0.023176336	2.317633623	0.169505017	16.95050171	0.013246874	1.324687359	0.029823703	2.982370269
METAL	0.022591752	2.259175212	0.170381676	17.03816756	0.005248533	0.52485333	0.029416286	2.941628557
DANCE	0.006877199	0.68771992	0.189424371	18.94243709	0.00290731	0.290730979	0.04868415	4.868414955
JAZZ	0.017911984	1.791198362	0.136911989	13.69119887	0.019379182	1.937918202	0.020092978	2.009297784
FOLK	0.014473743	1.447374309	0.189930975	18.99309746	0.008887124	0.888712416	0.030727126	3.07271257
SOUL	0.011344793	1.134479291	0.220869421	22.08694207	0.002761871	0.276187143	0.040910765	4.091076537
ROCK	0.03935464	3.935463954	0.205184225	20.51842246	0.005798981	0.579898083	0.041116464	4.111646381
POP	0.019018391	1.901839109	0.197328623	19.73286227	0.008851405	0.885140482	0.025204514	2.520451396
BLUES	0.023044488	2.304448774	0.139235495	13.9235495	0.014668569	1.466856915	0.032456147	3.245614676

The above table shows the regression results on artist hotness. The first two columns indicate the average and average % performance and the next two columns indicate the worst performance assuming the artist is not hot at all. The next two show the Bernoulli trial performance where we assume the artist hotness on a binary scale – Hot vs. Not Hot. The last two columns.

The prediction of artist hotness is a proof of concept for prediction of song hotness using NMF. Due to an oversight during data extraction we missed extracting the data for song hotness and due to large amount of computing resources required and time involved we did not opt for re-extracting the data.

INTERPRETATION

During our training phase using 10k set (nearly 1% of MSD) our algorithms performed really well to the tune of nearly 70-85% accuracy. But during the final tests, owing to the non linearity in distribution of data and diverse nature of music spread over almost 10 decades and 9 genres, the accuracy level dropped to 40-50%, which by standards of previously published research work on the same dataset. We attribute this to the extensive data cleansing efforts using novel idea of using Wikipedia as a data imputation source contributed towards a better training set for the models.

As we were focused towards learning the various machine learning algorithms and covering a wide range of such techniques rather than the depth of the algorithms, we were not able to fine tune the models as much as we would have liked. Owing to constrains of time, we could not implement the various variations of every single algorithm.

SUMMARY & CONCLUSIONS

As for scope of improvement, we foresee using features like lyrics and other meta tags towards a better classification. To solve the issue of non linearity of data distribution, we would like to assign weights to the number of songs belonging to each genre. A better approach towards genre tagging involving histogram count, music decades, genre priors and lyrics can provide better results.

For all the algorithms implemented there are several variations and equally high number of parameters which can provide a case by case better accuracy. Fine tuning is a very important part of the project that we would have liked to spend more time on.

In conclusion, our approach is novel as well as competent as compared to previously published research work. As discussed in earlier sections, by Bernoulli trials accuracy > 11% will be considered as baseline performance. Humans with little to none musical training can predict the accuracy of the genre accurately between 50%-70% which is a realistic target for a machine to achieve.

Our accuracy on validation set ranges between 43.6% - 73.4% with an accuracy of 44% on test set which is acceptable as well as matching the realistic targets.

REFERENCES

[1] "Automatic Genre Classification of Music Content [A Survey]", Nicolas Scaringella, Giorgio Zoia, and Daniel Mlynek, IEEE-Signal Processing Magazine (133) Mar 2006

[2] "Music Genre Classification with the Million Song Dataset [15-826 Final Report]", Dawen Liang, Haijie Gu and Brendan O'Connor, Carnegie Mellon University (Dec 03 2011)

[3] "Genre Classification for Million Song Dataset Using Confidence-Based Classifiers Combination", Yajie Hu and Mitsunori Ogihara, SIGIR Aug 12-16, 2012 (ACM 978-1-4503-1472-5/12/08.)

CONTRIBUTIONS TO PROJECT (PARTIAL LIST, NON-REPRESENTATIVE)

NISHANT NATH	ARNAV MENDIRATTA	AHMET CAN OZBEK
Data Cleansing (Wiki Parsing) Data Extraction (using AWS)	Data Merging (Wiki data with MSD extracted data)	Data Splitting & Pickle File Generation
Feature Extraction	Feature Extraction	Feature Extraction
Parametric Models (SVM) & Non-Parametric Models (Naïve Bayes, k-nn, k-nc, k-nCentroid, k-means) & Ensemble Models (Bagging)	Parametric Models (LDA, QDA) & Non-Parametric Models (Mean Shift) & Ensemble Models (Random Forest)	Parametric Models (SVM) & Non-Parametric Models (ball-tree Neighbor Classifier) & Ensemble Models (AdaBoost)
Visualization & Plots	Visualizations & Plots	Visualization & Plots
Report Writing	Report Writing	Report Writing