CS 294-16: Final Project Report

Team: Purple Paraguayans

Michael Ball
Nishok Chetty
Rohan Roy Choudhury
Alper Vural

Problem Statement and Background

Music has always been a form of both personal expression as well as a means for social interaction. More recently, however, music has become a massive industry, with global music sales nearly reaching $15 billion.

We analyze the Million Song Dataset (MSDS)[1] to find correspondences between *hotttnesss* (the dataset's popularity metric) and various features, including *key*, *tempo*, *danceability*, etc. We aim to use machine learning to predict the popularity of a song based on both song characteristics and metadata about a song (such as release date, artist information etc.). From the models, we hope to also get a deeper insight into the features that are most predictive of song popularity, and understand what makes certain songs more popular than others. Our problem statement is thus twofold: (1) *using machine learning, can we predict the popularity of a song* and (2) *can we use our machine learning models to get a deeper insight into features predictive of song popularity.*

To determine success on the dataset we use the prediction accuracy of the various models. We use 5-fold cross validation to tune and optimize our results. We compare models using their 5-fold cross validation accuracy. We define success as a significant improvement in prediction accuracy over the baseline. We use a simple linear SVM model as the baseline. In addition to using accuracy as a metric of performance, we also use the Receiver Operating Characteristic curve. We use the ROC curves to get further insight into the performance of our models by closely examining the tradeoffs between the false positive rate and true positive rate for each model. We define success as a model that has a low false positive rate (<0.2) at some high true positive rate (>0.8).

The MSDS is a database of popular songs spanning decades of music. It has been released to the public for research by The Echo Nest,[2] a company that specializes in music intelligence services. While the music industry likes to portray itself as driven by artistic integrity, it is ultimately an industry that makes a lot of money from making popular music. The music industry (Sony Music Entertainment, Universal Music Group, Warner Music Group, etc.) is highly invested in identifying trending features, and would be especially interested in an algorithmic approach to evaluating the potential popularity of a new song.

While a lot of work has been done on applying machine learning to different facets of the million song dataset, there has been little to no focus on song popularity. Research has focused on predicting the year of a song, recommending songs based on user preferences, etc. There is also a lot of existing literature on attempting to determine the characteristics of a popular song, but they have tended to focus more on evaluative or statistical approaches than a big data or machine learning approach. We present a system that ties together the two aspects -- we build machine learning models that are able to predict song popularity accurately and then use these models to get more insight into the features that are the strongest signals of song popularity.

Methods

Data collection:

We obtained the Million Songs dataset from LabROSA Lab at Columbia[3]. We initially began with a subset of the dataset containing 10,000 songs and then scaled up to the full 1,000,000 song dataset. Both the subset and the full dataset were in the form of compressed HDF5 files. Each file represents one track with all the related information (artist information, release information, audio analysis of the track, etc). We read the data files using the h5py package in python.

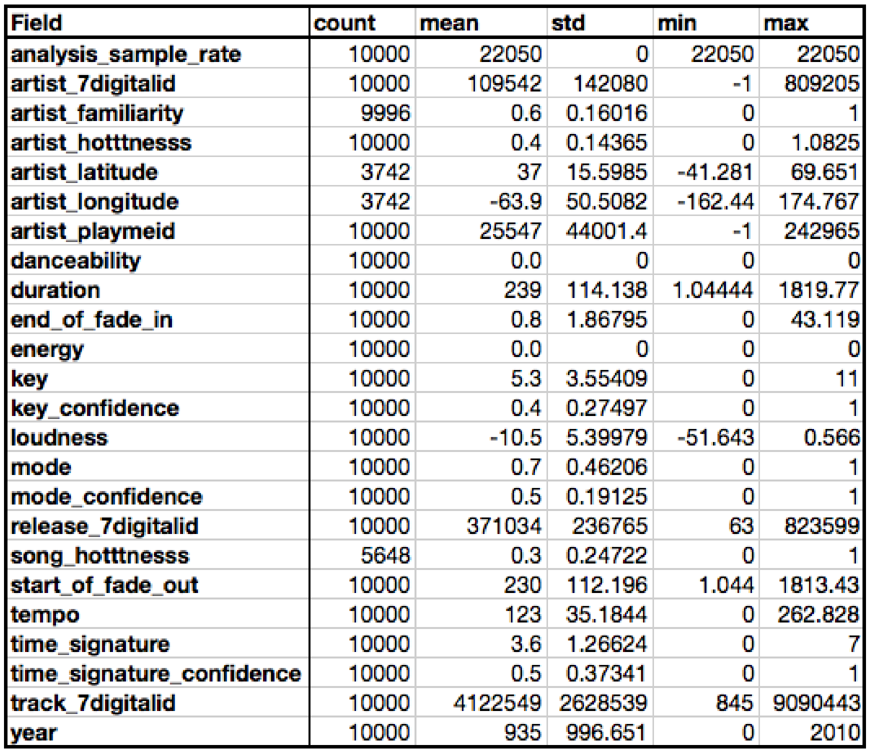
Data wrangling:

Both the subset and the entire dataset are fairly large. The subset is 2.5GB while the full Million Songs dataset is 270GB. We initially tried building our models directly using the HDF5 files. However, that made it difficult to do data cleaning, imputation and statistical analyses. We switched to a different method based on the observation that a lot of the data in the dataset was not relevant to us (such as the actual audio tracks) so we filtered the data from the HDF5 files and converted it into a much more compact CSV representation. We filtered the files by only extracting features relevant to our problem statement. For the 10,000 song subset, these included the following fields: *artist_familiarity, artist_hotttnesss, artist_id, artist_latitude, artist_longitude, artist_location, artist_name, song_id, song_hotttnesss, title, artist_terms, artist_terms_freq, artist_terms_weight, danceability, duration, energy, key, loudness, mode, tempo, year*. By applying this filter, we were able to convert the 2.5GB HDF5-format dataset into a 450MB CSV file.

For the full dataset, we filtered the HDF5 files by extracting only the features that turned out to work well on the 10,000 song subset. These included *artist_familiarity, artist_hotttnesss, artist_latitude, artist_longitude, artist_location, artist_name, song_hotttnesss, title, artist_terms, artist_terms_freq, artist_terms_weight, song_id, key, loudness, mode, tempo, year.* This filtering enabled us to convert the 270GB dataset into a significantly more manageable 1.2GB CSV file.

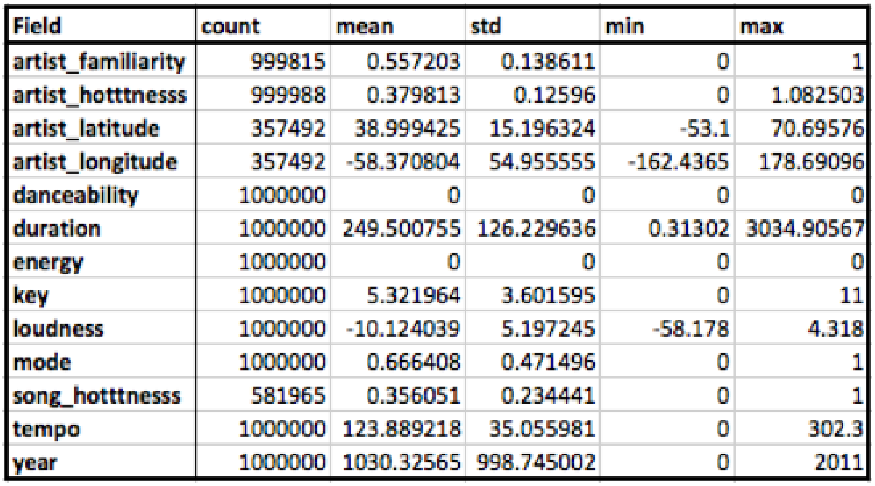
Data exploration:

We began the process of data exploration by examining the dataset in aggregate and computing basic statistics (count, mean, standard deviation, min, max) for each of the integer/float valued fields. This is summarized in Table 1.



Field	count	mean	std	min	max
analysis_sample_rate	10000	22050	0	22050	22050
artist_7digitalid	10000	109542	142080	-1	809205
artist_familiarity	9996	0.6	0.16016	0	1
artist_hotttnesss	10000	0.4	0.14365	0	1.0825
artist_latitude	3742	37	15.5985	-41.281	69.651
artist_longitude	3742	-63.9	50.5082	-162.44	174.767
artist_playmeid	10000	25547	44001.4	-1	242965
danceability	10000	0.0	0	0	0
duration	10000	239	114.138	1.04444	1819.77
end_of_fade_in	10000	0.8	1.86795	0	43.119
energy	10000	0.0	0	0	0
key	10000	5.3	3.55409	0	11
key_confidence	10000	0.4	0.27497	0	1
loudness	10000	-10.5	5.39979	-51.643	0.566
mode	10000	0.7	0.46206	0	1
mode_confidence	10000	0.5	0.19125	0	1
release_7digitalid	10000	371034	236765	63	823599
song_hotttnesss	5648	0.3	0.24722	0	1
start_of_fade_out	10000	230	112.196	1.044	1813.43
tempo	10000	123	35.1844	0	262.828
time_signature	10000	3.6	1.26624	0	7
time_signature_confidence	10000	0.5	0.37341	0	1
track_7digitalid	10000	4122549	2628539	845	9090443
year	10000	935	996.651	0	2010

Table 1: Basic statistics on the 10,000 song subset



Field	count	mean	std	min	max
artist_familiarity	999815	0.557203	0.138611	0	1
artist_hotttnesss	999988	0.379813	0.12596	0	1.082503
artist_latitude	357492	38.999425	15.196324	-53.1	70.69576
artist_longitude	357492	-58.370804	54.955555	-162.4365	178.69096
danceability	1000000	0	0	0	0
duration	1000000	249.500755	126.229636	0.31302	3034.90567
energy	1000000	0	0	0	0
key	1000000	5.321964	3.601595	0	11
loudness	1000000	-10.124039	5.197245	-58.178	4.318
mode	1000000	0.666408	0.471496	0	1
song_hotttnesss	581965	0.356051	0.234441	0	1
tempo	1000000	123.889218	35.055981	0	302.3
year	1000000	1030.32565	998.745002	0	2011

Table 2: Basic statistics on the full 1,000,000 song dataset

Data Visualization:

We plotted some graphs to get a sense of the distribution of different fields in our dataset. As can be seen from Figure 1, key fields as as tempo, artist familiarity, artist hotttnesss, song hotttnesss, song duration all follow a *gaussian distribution*, which is important for distance metric based models such as SVM, k-NN, etc. The only feature that doesn't follow a gaussian distribution is the year.

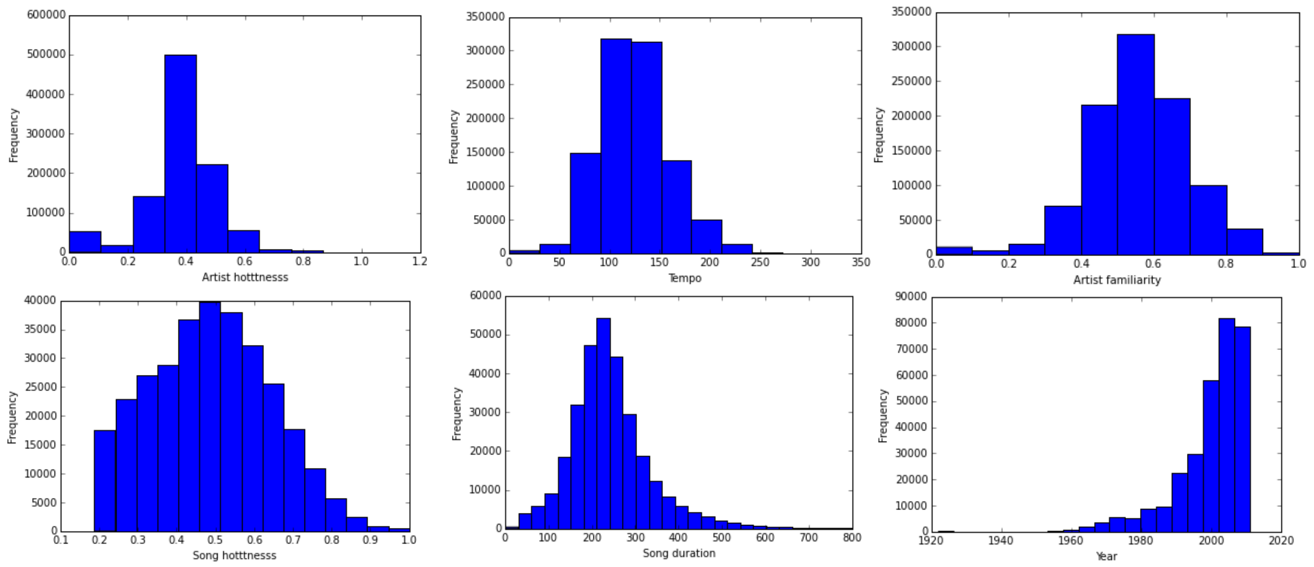
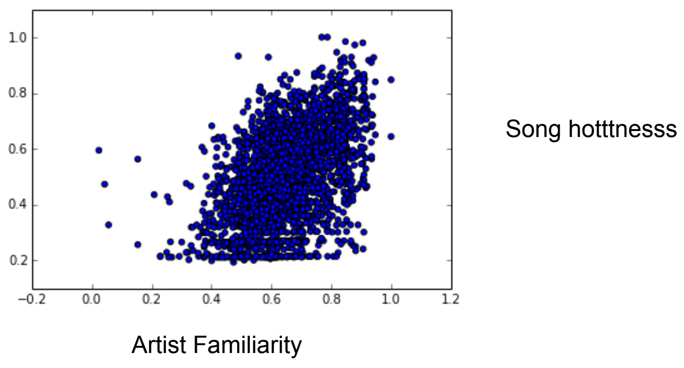


Figure 1: Histograms of key numerical fields of the dataset

We also plotted scatter plots to see examine the correlation between certain fields in the dataset and our target class.



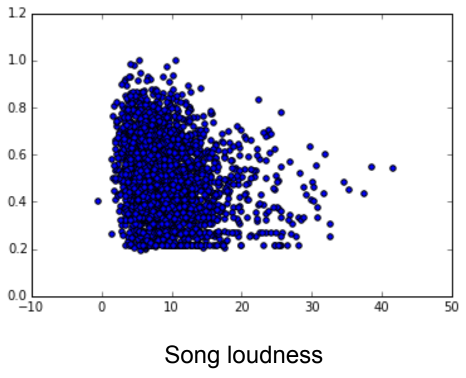


Figure 2: Artist familiarity vs. Song hotttnesss Figure 3: Loudness vs. Song hotttnesss

As evident from Figure 2 and 3, artist familiarity and song loudness weakly correlate with the song hottness. Artist familiarity correlates positively as one would expect, but somewhat surprisingly, song loudness correlates negatively. The expectation was that hotter songs would be louder, but it seems that the opposite it true, since the average loudness for songs in general was .2 higher. In Figure 3, the negative of loudness (to make it positive) is plotted on the x-axis and hotness is plotted on the y. It seems that the reason for hotter songs being quieter is that there are several overly loud songs that bring down the hotness average for songs in general. The average loudness for the top 10% of hottest sons is -7.4 while the overall average is -9.4.

Data cleaning:

We used the basic statistics (Table 1, Table 2) to determine which numeric fields required cleaning. For the non-numeric fields, we examined the number of non-null data points. We found that the *year* field has a significantly large number of missing data (data points with year = 0). We also found that *hotttnesss* was missing from a large number of data points. Attempting to impute or categorically fill a value for year would be futile, and we cannot impute *hotttnesss* since it is what we are trying to predict, so we decided to drop rows in which these fields were missing. After dropping these rows we were left with 3,064 records in the 10,000 song subset and 346,562 records in the full 1 million song dataset.

Some of the less important fields had problems too. We dealt with them in different ways. These fields are described below:

- *energy* and *danceability* had a mean, standard dev, min and max of 0. This suggested all the rows were 0. We verified this and decided to drop these fields.
- *artist_longitude*, *artist_latitude* had a lot of missing data. We decided to impute these fields using a placeholder value of 0. Similarly, *artist_location* had a lot of missing data. We imputed it with the string “unknown”.
- Some additional fields had a few missing values too. However, in most cases only one or two values were missing. Since all the missing values in these other fields were in only 10 different rows in the subset and 119 rows in the full dataset, we decided to drop these rows.

Thus, after removing/imputing missing data, we ended up with a dataset of 346,443 rows (data points) and 19 columns (features).

We checked for duplicate keys using the *song_id* as our unique record identifier. There were no duplicate records. We checked for statistical anomalies using the basic statistics described previously. The only anomalies were in the *energy* and *danceability* columns, which we dropped.

We used pandas’ built-in functionality for most of the data examining, imputing and cleaning. We used the `DataFrame.describe` and `DataFrame.info` functions for aggregate statistics of the dataset. We used pandas’ indexing and selection features to remove rows with missing data. We used the `DataFrame.drop` function to drop columns. We used the `DataFrame.fillna` function to impute missing data. Lastly, we checked for duplicate keys using the `DataFrame.duplicated` function.

Featurization:

Our featurization involves two main components: feature selection and feature engineering. We explain our reasons for the features we have selected and for the features we plan to generate below. These features are a combination of filtering down all features, and generating new ones.

- <u>Feature Selection:</u> We began by using a large subset of the features available (initially selected based on intuition of what features are likely to be predictive, as well as quality and availability of data for that feature) but then used ablation on a Random Forest model to determine the optimal feature set. The optimal feature set included the following features:
    - *artist_familiarity, artist_hotttnesss, bow_artist_terms, duration, tempo, tfidf_song_title, loudness, artist_location, artist_name, genre, decade*
- <u>Feature Engineering:</u> We did some feature engineering to generate features we thought would be predictive of song popularity.
    - Bag-of-Words on *artist_terms: artist_terms* is comprised of tags describing the artist’s work. We applied Bag-of-Words featurization to these tags.

- TF-IDF on *song_title*: We treated each word of a song title as a "term," each song title as a "document" and the collection of all song titles as the "corpus". We used this definition to create a feature based on applying TF-IDF on the song title.
- *decade*: We know that music patterns can be described by decades, so we binned the *year* data into a decade data.
- *genre:* MSDS (surprisingly!) does not include a column for song or artist genre. We categorized each song into an appropriate *genre* based on the content of *artist_terms*.

Models:

We began by building a simple linear SVM model. This was our baseline. We used scikit-learn's `svm.LinearSVC` module to build this model. We then built, tuned, and tested a number of other models, again using the scikit-learn library in Python. These include Neural Network, Random Forest, Adaboost, Logistic Regression, and and k-Nearest Neighbors models. We tuned the parameters of these models using grid search with 5-fold cross-validation. The set of parameters that worked best for each model is shown in Table 3. The parameters in bold are the ones we tuned.

Model	**Parameters** (tuned ones are in bold)
SVM	**C=2**, penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, max_iter=1000
Random Forest	**n_estimators=10**, criterion='gini', **max_depth=10**, **min_samples_split=40**, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, bootstrap=True, class_weight=None
Logistic Regression	**penalty='l2'**, dual=False, tol=0.0001, **C=512**, fit_intercept=True, intercept_scaling=1, class_weight=None, solver='liblinear', max_iter=100, multi_class='ovr'
Decision Tree	criterion='gini', splitter='best', **max_depth=20**, **min_samples_split=40**, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, class_weight=None
Adaboost	base_estimator='DecisionTreeClassifier', **n_estimators=200**, **learning_rate=0.01**, algorithm='SAMME.R', random_state=None
k-NN	**n_neighbors=2**, weights='uniform', **algorithm='kd_tree'**, leaf_size=30, p=2, metric='minkowski'
Neural Network *(new in sklearn version 0.18.dev0)*	**hidden_layer_sizes=(20)**, activation='relu', **algorithm='l-bfgs'**, **alpha=0.001**, batch_size=200, learning_rate='constant', learning_rate_init=0.001, power_t=0.5, **max_iter=500**, tol=0.0001, momentum=0.9, nesterovs_momentum=True

Table 3: Model parameters

Evaluation of Methods:

<u>Methods that worked</u>: filtering features to reduce dataset size from 270GB to 1.2GB, data cleaning and imputation to resolve missing values, model building using scikit-learn and sparkit-learn, 5-fold cross-validation to tune model parameters.

<u>Methods that didn't work as well:</u> directly trying to build models using the HDF5 files. This made it difficult to do data imputation, cleaning and wrangling.

Results

Accuracy

The 5-fold cross-validation accuracy values of our models for both the 10,000 song subset and the full 1 million song dataset are summarized in Table 4 and visualized in Figure 4. These accuracies were generated using the set of features found to be optimal through ablation (described in the *Featurization* section) and the set of parameters found to be optimal through cross-validated grid search (reported for each model in the *Models* section).

As is evident from Figure 4, in terms of accuracy most models performed significantly better than the baseline linear SVM model and majority label baseline (predicting the majority class for all points). In general, the models performed slightly better on the full 1 million song dataset, with lower variation between CV folds. This is as expected since they are able to train on more data. The accuracies of the non-baseline models range from 68% to 79% on the subset and 69% to 82% on the full dataset. Our best performing model, Random Forest, is able to predict song popularity with an **82.4%** accuracy, which is a significant improvement over the baseline.

Model	Subset Accuracy (%)	Full dataset Accuracy (%)
Baseline (majority label)	55.2	**53.4**
Baseline (SVM)	56.3	**57.1**
Neural Network	67.7	**69.2**
kNN	71.2	**72.3**
Logistic Regression	73.4	**71.7**
Decision Tree	72.0	**78.5**
Random Forest	**77.8**	**82.4**
Adaboost	74.6	**72.8**

Table 4: Cross-validation Accuracies

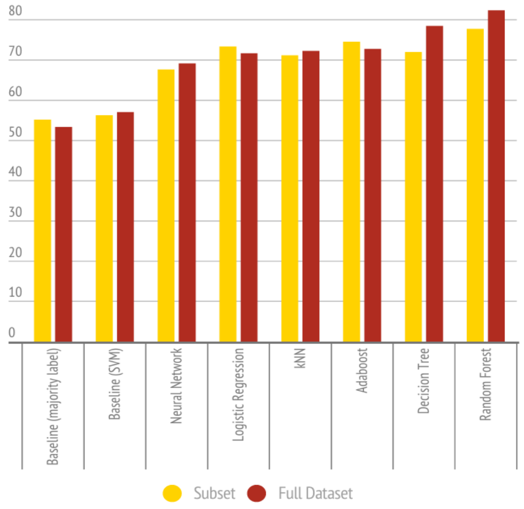


Figure 4: Cross-validation Accuracies

ROC curves and Area Under Curve (AUC)

We also plotted ROC curves to examine the true-positive/false-positive tradeoffs made by our models. False positives are of significant concern for this problem since a false positive would mean a record label wasting significant money on a song that will not be popular. Figure 5 has the ROC curves of our top 3 models and the baseline. As evident from Fig. 5, with the baseline SVM model, to achieve a true positive rate of 0.8, one would have to accept a false positive rate of more than 0.7, which is unacceptable. Conversely, using our best performing model, Random Forest, one can achieve a true positive rate of 0.8 with a false positive rate of less than 0.2.

Feature Importances

Lastly, we extracted the feature importance coefficients from our Random Forest model (These coefficients represent the normalized mean decrease in impurity enabled by each feature across the forest). The top 10 coefficients are shown in Figure 6. The most important features were *artist_familiarity*, *artist_hotttnesss*, and *tempo*, followed by some of the n-grams of *artist_terms*: ‘rock,’ ‘pop,’ ‘guitar,’ ‘metal,’ ‘jazz,’ ‘hip hop’. These feature importances are intuitive. If an artist is popular the songs that he or she releases are probably more likely to be popular. The feature importance coefficients also highlight the importance of genre to song popularity.

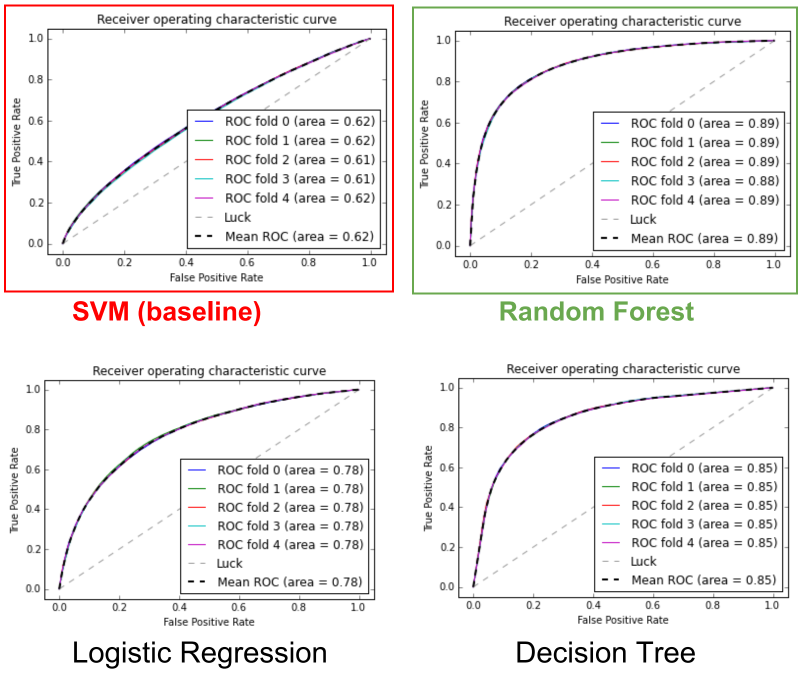


Figure 5: ROC Curves

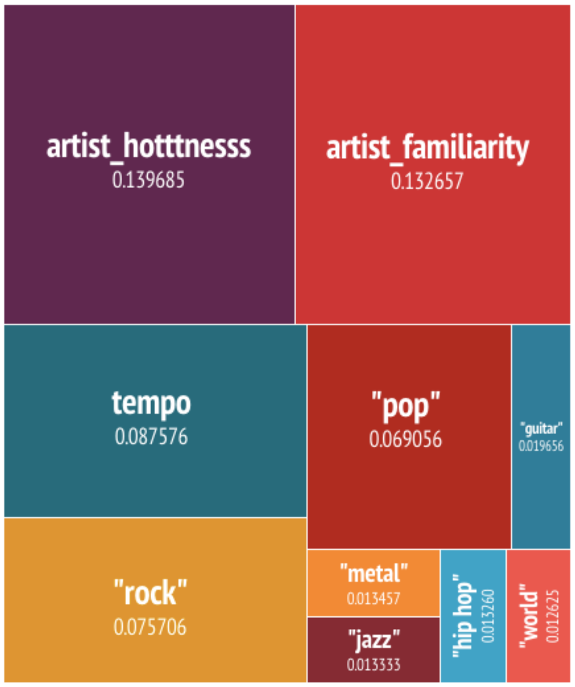


Figure 6: Top ten features

Tools

Pandas, NumPy, SciPy, Matplotlib: We used these libraries for data exploration and data preprocessing. We chose these since they have significant built-in functionality for data processing, analysis and visualization. Pandas in particular has useful functions for cleaning, such as `fillna()` and `notnull()`, as well as for data exploration such as `DataFrame.describe()` and `DataFrame.info()`. Likewise, matplotlib's plotting functionality allowed us to visualize the distribution of variables and relationships between variables and plot our ROC curves. These tools worked quite well; we did not encounter any major issues with them.

H5py: We used the h5py library to read the dataset, since the data was stored in the HDF5 format. We had to take a bit of time to learn how to use this tool, but in the end it accomplished its purpose with no issues.

scikit-learn, sparkit-learn: We used the scikit-learn and sparkit-learn libraries in Python to build our machine learning models. We initially planned on using scikit-learn for the 10,000 song subset and Spark/sparkit-learn on our EC2 instance for the full 1 million song dataset, since we felt that the million songs would take too long to run using scikit-learn. We classified the 10k songs without any significant issues using scikit-learn. As per Professor Canny's advice, we tried running the million songs in scikit-learn (since its vectorization and the C++ implementation of many of its models should theoretically allow it to be as fast as–if not faster than–sparkit-learn, especially considering the overhead from Spark's Java based implementation), but with a higher value for the `n_jobs` parameter to take advantage of multithreading. This worked for certain models, such as Logistic Regression, Decision Tree and Random Forest, but did not work as well for SVM and others, so we used sparkit-learn for those models. The ‘feature importances’ feature of scikit-learn’s Random Forest model was very useful as it allowed us to get more insight into what features were most important in predicting song popularity.

BIDMach: While BIDMach may have given us a significant boost in speed and we did try using it, we ultimately chose not to because the overhead of modifying our entire existing workflow—all developed around the scikit-learn interface—to work with a different framework would have taken significantly more time than the speed gained by using it. Conversely, because of sparkit-learn’s interoperability with scikit-learn, modifying our existing code workflow to work with sparkit-learn was fairly minimal.

Lessons Learned

Our main goals were to (1) use machine learning to predict the popularity of a song and (2) use our machine learning models to get a deeper insight into the features predictive of song popularity. We defined success as a significant improvement in prediction accuracy over the baseline. We use a simple linear SVM model as the baseline. We define success as a model that has a low false positive rate (<0.2) at some high true positive rate (>0.8). We used ROC curves to examine the TPR/FPR tradeoffs for our models.

Using ablation on our Random Forest model, we found the optimal feature set to be the following: *artist_familiarity, artist_hotttnesss, bow_artist_terms, duration, tempo, tfidf_song_title, loudness, artist_location, artist_name, genre, decade.*

We tuned our models using 5 fold cross-validation with grid search for a variety of models, including SVM as a baseline, Random Forest, Logistic Regression, Decision Tree, Adaboost, K-Nearest-Neighbors, and Neural Network. We compared the 5-fold cross-validation accuracy of our models. On our full dataset, the baseline SVM model achieved an accuracy of 57.1% and a mean AUC (area under ROC curve) of 0.62. The frequency-based baseline (predicting the most-frequent class for all data points) had an accuracy of 53.4% Our top three models were Random Forest, with 82.4% accuracy, Decision Tree with 78.5% accuracy, and Adaboost, with 72.8% accuracy. Random Forest also had the highest mean AUC, 0.89. Our models thus performed significantly better than the baseline.

Overall, we were able to correctly predict if a song was popular or not more than 80% of the time, which is a huge success! We extracted the feature importance coefficients from our Random Forest model (These coefficients represent the normalized mean decrease in impurity enabled by each feature across the forest). The most important features were *artist_familiarity*, *artist_hotttnesss*, and *tempo*, followed by some of the n-grams of *artist_terms*: ‘rock,’ ‘pop,’ ‘guitar,’ ‘metal,’ ‘jazz,’ ‘hip hop.’ These feature importances are intuitive. If an artist is popular the songs that he or she releases are probably more likely to be popular. The feature importance coefficients also highlight the importance of genre to song popularity.

We learned from this project how important feature engineering is, since efforts such as the n-grams on *artist_terms* noticeably increased our accuracy. We also learned that accuracy alone is not sufficient to express the quality of a model. False positives and negatives can have a significant impact. In a real application, a false positive would mean a record label wasting significant money on a song that will not be popular. Similarly, a false negative could lead to a potentially popular song being rejected. By plotting ROC curves, we were able to evaluate the true-positive/false-positive trade-offs of our models.

Baseline Model [CS294-16]

For this dataset we created two baseline models. The first baseline model is simple frequency based model. In this model we simply predicted the most common class for each song. In this model **53.4%** of the songs had the correct category assigned to them.

Our second baseline was a linear SVM model. We chose a linear SVM because it's a simple model which is easy to tune and use. This SVM model achieved a 5-fold cross-validation accuracy of **57.1%** and a mean ROC area under curve of **0.62**.

Our best performing "primary" model was a tuned Random Forest. This model achieved a 5-fold cross-validation accuracy of **82.4%** and a mean ROC area under curve of **0.89**. The Random Forest model thus performed significant better than the baseline models, both in terms of raw classification accuracy as well as in terms of minimizing the true-positive/false-positive tradeoff (as demonstrated by the area under curve metric and ROC plots).

The Random Forest model likely performed better than the SVM because the data is not linearly separable. SVMs try to fit a hyperplane that separates two classes. Data that isn't linearly separable often causes SVMs to perform poorly. The non-gaussian distribution of the *year* field also may have affected the performance of the SVM model since it relies on features being normally distributed. The Random Forest model is also able to better reduce bias by taking a majority vote of a large number of Decision Trees. The Random Forest was also better than the SVM in that it allowed us to get insight into how the model was predicting song popularity. We were able to visualize the trees and see what features and thresholds were used to split at different nodes. We were also able to use Random Forest's feature importance coefficients to see which features were most predictive of song popularity. It was also much faster than SVM since it was able to take advantage of our instance's multiple cores to train in parallel, while scikit-learn's SVM implementation is only single threaded. Therefore, overall, the Random Forest worked significantly better than the baseline models, in terms of not only performance, but also usability, interpretability and usefulness.

Team Contributions

Michael Ball: 25% (Data Cleaning, Data Exploration, Presentation, Poster, Writeups)
Nishok Chetty: 25% (Data Preprocessing, Data Exploration, Presentation, Poster, Writeups)
Rohan Roy Choudhury: 25% (Data Preprocessing, Model building, Presentation, Poster, Writeups)
Alper Vural: 25% (Data Cleaning, Model building, Presentation, Poster, Writeups)

Citations

[1] http://labrosa.ee.columbia.edu/millionsong/
[2] http://the.echonest.com
[3] http://labrosa.ee.columbia.edu/millionsong/