ALGO-RHYTHM
Million Song Classification

Andrew Hill ahill6@ncsu.edu
Guanxu Yu gyu9@ncsu.edu
Dustin Lambright dalambri@ncsu.edu
Fuxing Luan fluan@ncsu.edu
Marc Quaintance miquaint@ncsu.edu
Yuchen Sun ysun34@ncsu.edu

***Abstract*—It has often been remarked that popular music hit songs all sound similar. But is there a way to quantify what it is that makes a hit song? We apply outlier detection techniques, classification methods, and clustering to the Million Song Database to train a model that can predict whether a given song is a hit.**

***Keywords*—Classification, Music, Supervised Learning.**

I. Introduction

If a song reaches the the Billboard top 100 chart, it has a much higher chance of being played on the radio, in stores, and online. Many a music snob claims that they hate popular music because it all sounds the same. Is there validity to that statement? Is there a way to break down a song to engineering units and compare the popular songs with the unpopular songs? Our efforts focus on being able to predict the probability of a song being successful (reaching the Billboard Top 100 chart for at least one week, in our case) by using supervised and unsupervised learning to identify key points in songs that contribute to the success of a song. We gathered data from the LabROSA Million Songs Dataset (MSD)[5] and combined that data with a history from Billboard The Hot 100[6] songs to create our labeled data set for classification.

Our end goal in this undertaking was to be able to take a song and predict if it would reach the Top 100 chart, using only the attributes available in the LabROSA dataset. Each of us provided a classification algorithm to compare our results. Some of the algorithms were written completely from scratch, and some were written using available libraries, such as scikit-learn or numpy. Our goal was to compare and contrast the efficacy of each algorithm, and draw reasonable conclusions about the current state of similarity of songs in the top 100. Our github can be found in the first reference[1].

II. Background and Related Work

There is already some related work in song hotness prediction. Stephen Mussmann, John Moore and Brandon Coventry[2] had worked on the influence of lyrics to the hotness of a song and they come to the conclusion that lyrics slightly improve ranking predictions. On the other hand, Julien Kawawa-Beaudan and Gabriel Garza's research[3] revealed that Gaussian discriminant model is pretty good for prediction using metadata of songs. In our work, we try to use kNN, decision tree, Naive Bayes, regression and neural network with metadata to do the song hotness prediction.

In the work of Julien Kawawa-Beaudan and Gabriel Garza[3], their goal was to find a way of classifying popular songs using the LabROSA dataset, but many of their parameters differed from ours. Rather than analyzing the entire million songs database, they focused on a subset of 10 thousand songs. Their feature set was smaller than ours, but also included features about the song data, including another section with pitch and timbre data attached to their songs. They encountered similar issues with an imbalanced data set. Their most accurate results came from analyzing the pitch and timbre data rather than the metadata that we analyzed.

III. Data Description

Our data combined the MSD [5] for feature data and a manually-compiled record of Billboard Top 100 Songs for the years 2000-2011[6]. The original dataset contains 54 features and one binary class variable.

However, many of the features were metadata tags, lists, or strings which could not be easily binned. As a result, it was decided that our dataset should be preprocessed to remove lists and irrelevant string data. We managed to shrink down the number of attributes we will use from 54 to 19. The final list of attributes can be found in Table 1.[1]

Name	Type	Example
Track ID	string	TRAXLZU12903D05F94
Title	string	Never Gonna Give You Up
Artist	string	Rick Astley
Release	string	Big Tunes - Back 2 the 80s
Year	int	1987
Key	int	1
Key Confidence	float	0.324
Time Signature	int	4
Time Sig. Conf.	float	0.634
Mode	int	1
Mode Confidence	float	0.434
End FadeIn	float	0.139
Start FadeOut	float	198.536
Energy	float	0
Duration	float	211.696
Danceability	float	0.0
Song Hotness	float	0.86
Tempo	float	113.36
Loudness	float	-7.75
Top 100	bool	yes

TABLE I

The features which are unclear have further descriptions in Table 2. Additionally, it should be noted that it was discovered during experimentation that the "danceability" and "energy" value were 0 for all songs.

[1]The example column comes from the examples given at https://labrosa.ee.columbia.edu/millionsong/pages/field-list

Name	Description
Duration	Length of song in seconds
Release	Name of Album
Key	Musical key (e.g. C) encoded as integer
Time Signature	Time signature (e.g. 4/4) encoded as integer
Mode	"Major" or "Minor" as 0 or 1

TABLE II

IV. Preprocessing

A. Feature Selection

Many of the features were strings or didn't pertain to the song itself (ids, information on the audio files, etc). Since these are not characteristics of the song, they were removed leaving 20 features that were relevant to be used by the algorithms.

B. Label Assignment

Since the original MSD doesn't have the label we need to do the classification, we managed to create the label by ourselves. We decide to categorize a song into the 'yes' class based on the fact that this song was ever in the Billboard Top 100 list. Otherwise we put a 'no' label for this song. To do so, for each song record in the MSD, we check whether this song is showed up in our top 100 dataset by comparing the song title and artist name. If it matches a record in the top 100 dataset then we put 'yes' label on it and vice versa.

The lack of unique identifiers in the Billboard top 100 history provided an obstacle in connecting the songs from the Billboard dataset to the Million Songs dataset. The only connecting link between the two datasets was an exact match of the artist name and song name, which opens the door to precision loss. As far as we can tell, there was not a statistically significant precision loss.

C. Undersampling

One means of dealing with the class imbalance problem is undersampling the majority class. The most naive form of undersampling is simply to take a random sample of majority class(es) instances equal to some multiple of the size of the minority class. Undersampling was tried with multipliers 0.25, 0.50, 0.75, 1, 3, 5, and 10. [2] Figure 4 shows the results of this experiment using Naive Bayes and kNN classifiers. Note that each of the unlabeled columns are the kNN run with the same settings as the NB column directly to their left. Equal frequency undersampling was the most effective in these tests.

V. Outlier Detection

Because the problem of outlier detection is very closely related to the problem of finding minority classes in unbalanced data, it was natural to compare classifier performance to the results from outlier detection methods. We chose to use one-class SVM and Isolation Forest.

[2]That is to say, a multiplier of 1 means that a random sample was taken such that there would be an equal number of majority and minority instances in the training set.

A. One-Class SVM

The One-Class Support Vector Machine (OC-SVM) is a variant on the standard Support Vector Machine (SVM) which is used for anomaly detection. The standard SVM maps the input data to a space interpreted as spatial coordinates, and then attempts to find a separating hyperplane which maximizes the margin between the two classes. Because the OC-SVM is an anomaly detector, it does not have class labels, but learns the boundary of a single class, then classifies items as either inside or outside of the class boundary. Like normal SVM, it has a C parameter, which was set to the default of $C = 1$ for this experiment.

B. Isolation Forest

Isolation Forest is an outlier detection technique which determines how "normal" an item is by calculating the average number of random splits required to isolate it. Because this is calculated by making a tree each time (multiple trees is called a forest), and the calculation is a measure of how many of these random splits are needed to isolate an instance, the method is called *isolation forest*. The logic behind such a technique is that, on average, it should require significantly fewer splits to "isolate" an outlier. Our experiment used the default value of 100 for the number of trees created and averaged to find this dissimilarity value.

VI. Methodology

The features that we had selected fro mteh Million Song Database were all numerical values in order to accomodate the needs of our classifiers. The one exception to this is the artists name, which was hashed. This was to allow the possibility that the artist would have a large impact on whether a song will be successful.

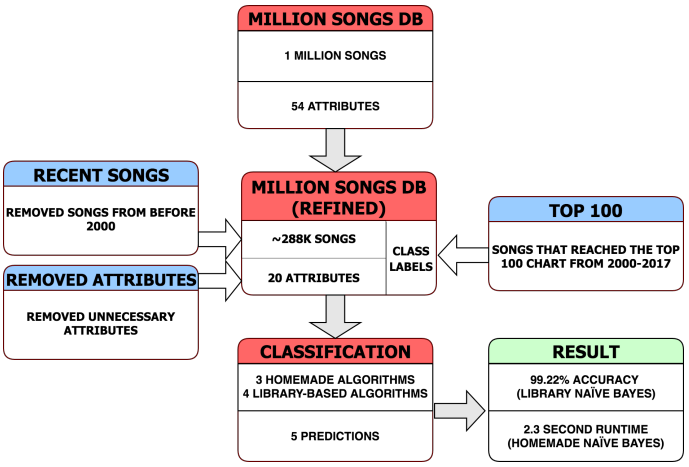

Fig. 1: Overall Process

Because the presence of a song in the Billboard Top 100 was not a feature in MSD, it was necessary to manually add the class labels to MSD to perform our calculations. Because the creation of this second dataset included manual search, only songs from 2000-2011 were considered. Since class labels were only added for these years, when the class labels were added to the MSD entries containing years outside that range

were removed reducing the size of the dataset to approximately 280,000 songs.

A. Procedure

Cleaned and preprocessed data was provided to each group. This data was provided as distinct files for hit songs and non-hit songs (to facilitate undersampling if needed). The data was split 7:3 into training and testing groups. Each group used the data to train a different classifier using machine learning libraries in Python or R. Additionally, the group split into halves, with each half implementing either kNN or Naive Bayes.

For each classifier, a confusion matrix was generated and used to calculate Precision, Recall, and F measures. Training time and prediction time were also collected.

B. Implementation Details

1) kNN: k-Nearest-Neighbors (kNN) is a simple classification method that uses the k nearest neighbors in the training data to decide the class label of a new object. Unlike most classification methods, it does not generate a model based on the training data and instead reuses the training data every iteration to determine the class of a given object.

The k nearest neighbors can be determined based on many different similarity measures (Euclidean, Mahalanobis, Cosine Similarity, etc), the choice is usually based on the data that is being used.

While easy to implement, kNN is expensive in both terms of storage and computation time. Because kNN does not generate a model, the training data must be stored for as long as the method is used and the distances must be recalculated for each new point (very time consuming on large datasets).

For the library implementation, a k-value of 1 was decided on since it resulted in the best f-measure. For our own implementation Manhattan distance was used with the hopes of making the calculation as easy as possible to save time.

2) Decision Tree: Decision tree is a simple classifier that uses a tree like structure where every node in the tree is a judgement of a specific attribute. Each leaf is a class label and every path from root to leaf represents a classification rule.

When the tree model is established, decision tree works super fast but it suffers from the overfitting problem. Overfitting happens when the decision tree continues to growth in order to reduce training data set error but caused an increased test set error. Generally, there are two ways to handle this problem: pre-purning and post-purning. In this project, we use pre-pruning (control the maximum depth of the tree) to overcome this shortcoming.

3) Naive Bayes: Naive Bayes is a probabilistic classifier which makes use of Bayes Theorem to calculate the probability of the class label given a certain set of attribute values. The method then makes use of Bayes Theorem to calculate a posterior probability based on the a class prior probability and the conditional probability.

$$P(C|X_1X_2\cdots X_n) = \frac{P(X_1X_2\cdots X_n|C)P(C)}{P(X_1X_2\cdots X_n)}$$

This posterior probability is used to test the likelihood of new data points of being assigned to each class, with the most common decision method being Maximum A Posterior likelihood (MAP). In order to simplify calculations, Naive Bayes classifiers assume that the input attributes are conditionally independent when we compute the conditional probability.

$$P(X_1X_2\cdots X_n|C) = P(X_1|C)P(X_2|C)\cdots P(X_n|C)$$

Despite this very strong requirement often being violated, NB regularly performs well even in the presence of significant collinearity. Furthermore, the independence assumption allows an implementation to treat each portion of the calculation as independent, which effectively neutralizes due to the curse of dimensionality. This results in NB often getting good results even in situations for which there are no theoretical guarantees, while at the same time being one of the fastest classification algorithms.

For the group's implementation of Naive Bayes, there were two major design decisions: how to estimate the probability of each attribute value, and how to handle missing values.

In order to estimate the probability of each attribute value, there are different approaches for different types of attributes. For discrete attribute, we counted the times of appearance of each possible values in the training set as their probabilities. In our dataset, for example, artist name, key, time signature and mode are discrete attributes. In the other side, for continuous attributes, typically there are two approaches to estimate the probability. One is using probability density estimation, another one is discretize the attributes into several ranges and treat each range as a discrete value. In our implementation, we choose the latter method to estimate probabilities. In addition, we introduce our human common sense in the process of discretization to make the value ranges more reasonable. For example, when we decide the ranges for the duration of a song, we deem that 120 seconds to 270 seconds will be the normal length for a song and beyond 480 seconds would be a hideously long duration. And it turns out the distribution justify the assumption we make above.

In term of handling missing values, there are two methods typically. One is treating a value that is not in the training data as a distinct value[7]. Another way is ignoring the missing value[8]. In our implementation, we have tried two different ways. First, if a value is missing in either one class conditional probability, then we set it to 1 and set a factor for another class conditional probability. If this value is missing in both class, then we set both of them to 1. Second way, we simply ignore the attribute when a value is missing. The results of both method are identical. The reasonable explanation is that if a value is missing in one class, then it is missing in another class as well.

4) Regression: Linear regression makes use of least squares and Maximum Likelihood Estimator (MLE) techniques to derive parameters after assuming that the class variable is a linear function of the input data, potentially with a normally distributed error. Generalized linear models expand this by removing the requirement that the error be normally distributed and allowing it to come from any of the family of exponential distributions, which includes the normal distribution.

LASSO (Least Absolute Shrinkage and Selection Operator) is an additional regularization scheme which can be added to various techniques, but for the purpose of this project will only be discussed with reference to generalized linear models. LASSO as used here combines a generalized linear model with regularization and variable selection. Variable selection is accomplished because - unlike many regression techniques such as ridge regression - LASSO can set coefficients on some input variables to zero (effectively removing them). To limit this behavior to only when the values were truly zero, the ϵ parameter was set to 0.001.

5) Neural Network: A neural network is patterned after the neurons and dendrites in the human brain, using graph nodes with weighted edges. The nodes and edges were optimized using back propagation with gradient descent. The input layer is the collected data in a matrix, hidden layer(s) is weighted to provide accuracy, and the output layer is the actual classification.

VII. Results

A. Library vs Implementation

1) Naive Bayes: The group-implemented Naive Bayes performed better than nearly all other classifiers. It has a precision of nearly 45% and overall accuracy over 99%. It is likely because of manual optimizations by considering columns individually when creating it. The standard library functions often struggled with the dataset's unique mix of continuous and discrete attributes.

The library method performed exceptionally well, especially when used with undersampling. This is something of a mystery, because the library method assumed independence and normally distributed data, both of which assumptions are strongly violated.

2) k-Nearest Neighbors: The implementation of kNN which did not make use of a tree structure did not finish executing,[3] so results cannot be compared to kNN. kNN was integrated with a tree structure which a group member had previously used, but the results for this combination were surprisingly poor compared to the library kNN (Figure 2). One major lesson learned is that for a large dataset, kNN will not work without a tree structure to limit the number of distance calculations needed, as well as the need to do initial testing on smaller datasets.

The run time of using library kNN algorithms was also much greater than the other algorithms. The R (programming language) library implementation of kNN took approximately 5 minutes to complete. This is substantially longer than the other algorithms, library or self-made, which ran for only a few seconds. Using k values from one to five resulted in an average f-measure of .06, the highest was .0599 for a k of one.

B. Other Classifiers

1) Decision Tree: In decision tree, we used both Python (sklearn) and R (party) library. After repeating test, we found both libraries are sensitive to the radio between the different labels.The result mostly depends on the ratio of the number

[3]Runtime is in weeks

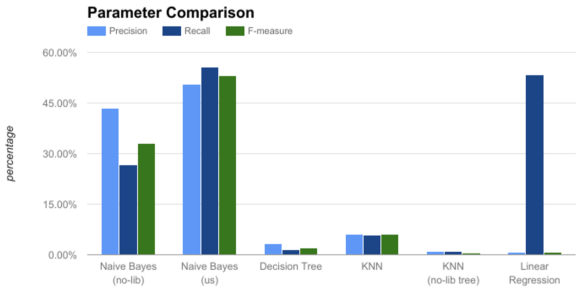

Fig. 2: Results

of 'yes' and 'no' instances in the training data. As long as the number of 'no' data points does not exceed three times the number of 'yes' points, the result stays optimistic with a decent precision with party. However, when it reaches four time as the number of 'yes' data points, the result obtained by party worsen dramatically and the precision drops to 0. With sklearn, when the ratio between "yes" and "no" is 0.2, it still maintain a prediction with a precision of 0.054, recall of 0.015 and f-measure of 0.008.

2) Neural Network: We created a neural network patterned after the implementation found at Welch Labs [9]. The requirements for the data structure using this method were extremely restrictive, and the data had to be heavily modified in order to make the code run. An issue arose in this algorithm with the scipy.optimize function. It was critical to the accuracy of the algorithm to run this function, but we were unable to diagnose why there was an issue with this particular function. Upon review, it was determined that our efforts were better spent elsewhere, and this implementation was shelved.

C. Undersampling

Let an undersampling rate be the ratio of majority to minority instances in the training set. Naive Bayes (nb) and k-Nearest Neighbor (kNN) were each run with undersampling rates of $0.25, 0.5, 0.75, 1, 3, 5, and 10$. Figure XX shows the results of this test. As can be seen, undersampling rates which cause a class imbalance of more than 20% significantly bias both methods in favor of the dominant class. In the case of class imbalance above 50%, classifiers functionally predicted only a single class (e.g. always predicted yes).

D. Outlier Detection

Both the One-Class SVM and the Isolation Forest had considerable difficulty with this dataset. Both worked well on the initial 10,000 item subset that was used for initial work, but had major problems with the full dataset. Isolation Forest was unable to finish computation because of its computational requirements and the size of the dataset, and One-Class SVM consistently predicted all items as outliers.

The lack of scalability in both methods are known issues, and a lesson learned is that additional time should be spent researching methods in the beginning to avoid such problems. By the time scalability issues were discovered (One-Class SVM cannot efficiently process more than a few tens of thousands of datapoints in its sklearn implementation), it was too late to change techniques.

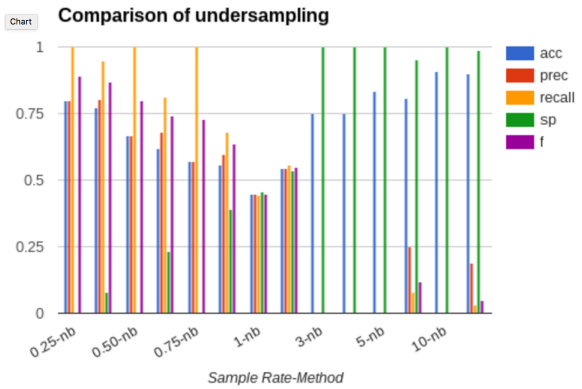

Fig. 3: Undersampling

Furthermore, One-Class SVM's classification of all items as outliers suggests that the data does not have a neat division between songs which are and are not hits. This would explain why some of the classifiers were able to achieve modest success, but the SVM - both in its separating hyperplane and kernel variants - had difficulty characterizing a separate space for non-hit songs. It may well be that hits cannot be separated in this way.

Another possibility is that something about the hashed strings (artists) caused a problem for the One-Class SVM outlier detection. When run on a dataset with all strings removed, this method had recall of over 67%. This is not good in itself, but is relatively promising compared to many of the other methods predicting only the majority category (i.e. recall of 0.00).

VIII. Threats to Validity

The data used was reduced to songs from the years 2000-2011 and may leave out longer-term trends (or may overlap multiple trends) which could complicate classification. Likewise, trends found in this data may not generalize to larger claims about trends in hit songs.

The Naive Bayes in sklearn used is Gaussian NB, which assumes not only that the conditional probabilities are normally distributed, but is also used primarily for continuous variables. Since this data is may not be normally distributed and is mixed continuous and discrete rather than purely continuous. Nonetheless, the other standard library version of NB is Multinomial, which was tried and performed significantly worse. This is not surprising since MultiNB is mostly used for text analysis. Also, the fact that NB consistently performed among the best techniques validates this design decision to some extent.

Because of the differences in how each method works, each group member needed to tweak the input data so that it would fit requirements for methods which (for example) can only take numerical data. While the group made an effort to maintain uniform data during training and testing, it is possible small discrepancies were introduced.

IX. Conclusion/ Future Directions

Our results with the Naive Bayes classification proved to be the most successful at classifying the songs. Stephen Mussmann, John Moore and Brandon Coventry's research reveals that lyrics will improve the prediction, so we can incorporate some natural language processing with lyrics into the data set.

Lyrical considerations could have application for future song writers as it could allow them to craft their song to be a hit. This would only appeal to some since many music writers enjoy creating music that is their own rather than crafting it to be popular. Also, since the best precision and recall values were not even 70%, it is unlikely that the prediction would help to a significant degree

In the future it could prove useful to test the effect of the time period selected on data patterns. This would allow longer time-based trends to be seen as well as patterns on a short time-scale that are overshadowed by the large range. Moreover, we could refine our label generation process. The employment of regular expression to recognize the co-artist could be helpful on improving the accuracy of label assignment.

In addition, it could be interesting if we utilize Zachary's Karate Club into our projects in the future. Zachary's Karate Club is is a well-known social network of a university karate club described in "An Information Flow Model for Conflict and Fission in Small Groups" paper by Wayne W. Zachary, which uses Graph Theory to make it possible to find connections or relevance between artists. This will enable us to figure out whether an singer's success is related to another singer who has similar styles and genres.

References

[1] Github - Million Song Classification
`https://github.ncsu.edu/ysun34/CSC522_MillionSongClassification`

[2] Stephen Mussmann, John Moore, Brandon Coventry, *Using Machine Learning Principles to Understand Song Popularity*

[3] J. Kawawa-Beaudan and G. Garza, *Predicting Billboard Top 100 Songs*, Stanford University, 2015.

[4] T. Bertin-Mahieux, D. P.W. Ellis, B. Whitman, P. Lamere. *The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, 2011.

[5] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. *The Million Song Dataset*. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[6] Billboard - The Top 100
`http://www.billboard.com/charts/hot-100`

[7] Lowd, Daniel, and Pedro Domingos. "Naive Bayes models for probability estimation." *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005.

[8] Kononenko, Igor. "Semi-naive Bayesian classifier." *Machine Learning-EWSL-91*. Springer Berlin/Heidelberg, 1991.

[9] Welch Labs - Neural Networks Demystified
`http://www.welchlabs.com/blog/2015/1/16/neural-networks-demystified-part-1-data-and-architecture`