Spotify Music Analysis

Alankrit Joshi JOSHI.A@HUSKY.NEU.EDU
Harshdeep Singh SINGH.H@HUSKY.NEU.EDU

1. Abstract

With the rise of music streaming services like Spotify, understanding music and recommending songs based on their similarity and user preferences is an essential part of these services' business model. Various techniques are employed on large scale using hefty models trained on millions of songs and collaborative filtering systems are updated periodically on highly sparse high-dimensional data. This project aims to approach this problem in an alternative way by designing a methodology to utilize smaller supervised learning models which can run on client-side to eventually get feedback from results of unsupervised learning results generated on back-end for presenting similar songs. The supervised learning techniques discussed in this paper reached accuracy up to 88% while the unsupervised learning techniques gave mixed results in terms of localizing sections of similar music.

2. Introduction

Scoring music similarity and song recommendation is a massive part of online music streaming services like Spotify, Apple Music and Google Play Music. Significant computing resources are spent on training machine learning models and tuning collaborative filtering systems. In this paper, we present an alternative way to approach this problem by considering the potential of client-side model generating frameworks like TensorFlow Lite, TensorFlow Mobile and TensorFlow.js.

A simple application, in the form of a web or mobile app, can be deployed that can train a supervised machine learning model for the user on the client-side to get the user's likeability. This likeability can be used to map user's affinity to particular classes of songs by using the unsupervised machine learning clusters that would be learned and maintained on a back-end service. Such a mapping can be achieved using multiple techniques, simplest of which is using Inverse Transform Sampling. This would allow in building a scalable system employing user-specific models to get likeability and sample similar songs from the clusters learned using music properties.However, the scope of this project is to study different supervised learning techniques and measure their performance on the data set(Spotify data for a user) using standard performance measuring metrics and suggest a model that can be used in the application described above(i.e. predicting the class likeable or not of a song given its attributes).

We used Spotify's Song Attributes API to build a dataset of songs and their music attributes. Each song available on Spotify for streaming has 10 music attributes like speechiness, danceability and tempo which we consider as our feature set. For training of a supervised learning model, we hand-labelled songs that we personally liked using a Boolean indicator of 'likeability'. Additionally, for future reference we also saved another target variable of 'genre' to understand and correlate our clusters with the dominant music genres of the clustered songs.

With an array of supervised learning techniques in our hand learned in the duration of this course, we were able to compare the performance of Logistic Regression, Random Forests Classifier, Support Vector Machines and a Deep Neural Network in classifying a new song as likable or not. Furthermore, we also employed multiple unsupervised learning techniques to cluster our dataset like kNN, k-means, Gaussian Mixture Models, DBSCAN and agglomerative clustering. We used the best model learnt and the most meaningful clustering to demonstrate the learning and sampling similar songs as discussed above.

3. Background

The paper can be read with prior of knowledge of machine learning techniques mentioned above and some data analysis experience.

It would be useful to read about Inverse Transform Sampling which we utilized towards the end of the project.

The project was worked out on a Jupyter Notebook running on a Python 3.6.4 kernel.

To understand the code in the notebook, it would be worthwhile to take a look at the documentation for some of the Python packages that were used. Multiple machine learning and data processing libraries were used like scikit learn, SciPy, TensorFlow, Keras, NumPy and Pandas. For visualizations, we took advantage of matplotlib and plot.ly to generate 2D and 3D plots respectively. Finally, it is worth mentioning the use of Spotipy, without which using Spotify's API and aggregating paginated results would have been a painful task.

4. Related Work

4.1. Recommendation Systems

These systems are a subclass of Information Filtering systems which aim to find out the preferences or ratings of a user. The most famous techniques in this approach are Collaborative Filtering and Content-based Filtering. Both techniques have advantages and drawbacks owing to things like regularity of updates, new items, new users, sparse or dense data and scalability.

We chose not to go with this approach because it is a problem usually dealt at scale. Our problem in hand has more to do with applying classical machine learning techniques that can be used to solve smaller problems in order to achieve similar results.

4.2. Natural Language Processing

Natural language processing can be applied on data scanned by web crawlers for particular songs (like blogs, articles, twitter feed, etc.) or by performing analysis on song's lyrics. Since this would have required an NLP dataset or curation of one, we chose to not venture into this field.

4.3. Audio Models

Audio models come closest to our approach, or rather, the processing part of such audio models. One can use libraries like Essentia to analyze songs and numerically label them for certain kinds of properties by analyzing pitch, beats, etc. Fortunately for us, Spotify does it already for us. Our part involves actually correlating these audio modelling results to try to recommend similar songs, although often audio models are solely used to categorize new songs in a recommendation system relying heavily on collaborative filtering.

5. Project Description

5.1. Data

The data we have used in our project is a collection of song attributes or features that are mode available by Spotify, a online music playing website. There are about 2550 songs available to us and each songs has 11 features or attributes. A song in our data set is represented as:

acousticness	danceability	duration_ms	energy	instrumentalness
0.051200	0.688	200640	0.868	0.00121

liveness	loudness	mode	speechiness	tempo	valence
4	0.4690	-4.183	0.0396	127.990	0.322

Table 1. Song attribute or feature values before performing normalization.

An addition feature named 'Likeability' was added to represent whether a song is liked by the user or not. Likeability 1 means songs is liked by user and 0 means not liked by user. This feature acts as target variable in algorithms later.

Each of the feature has a specific description, that explains the significance and also the value range of that feature(Available on Spotify's documentation). For e.g. speechiness tells whether lyrics are dominant in the songs or music. More the value of speechiness, more is the verbal the song and less music and has a value range 0 to 1. Like in case of a rap song, the verbal content is dominant as compared to music, so it will have more speechiness quotient.
The data is normally distributed between 0 and 1 for the features. In Fig.1. are the plots for some features showing their distributions:

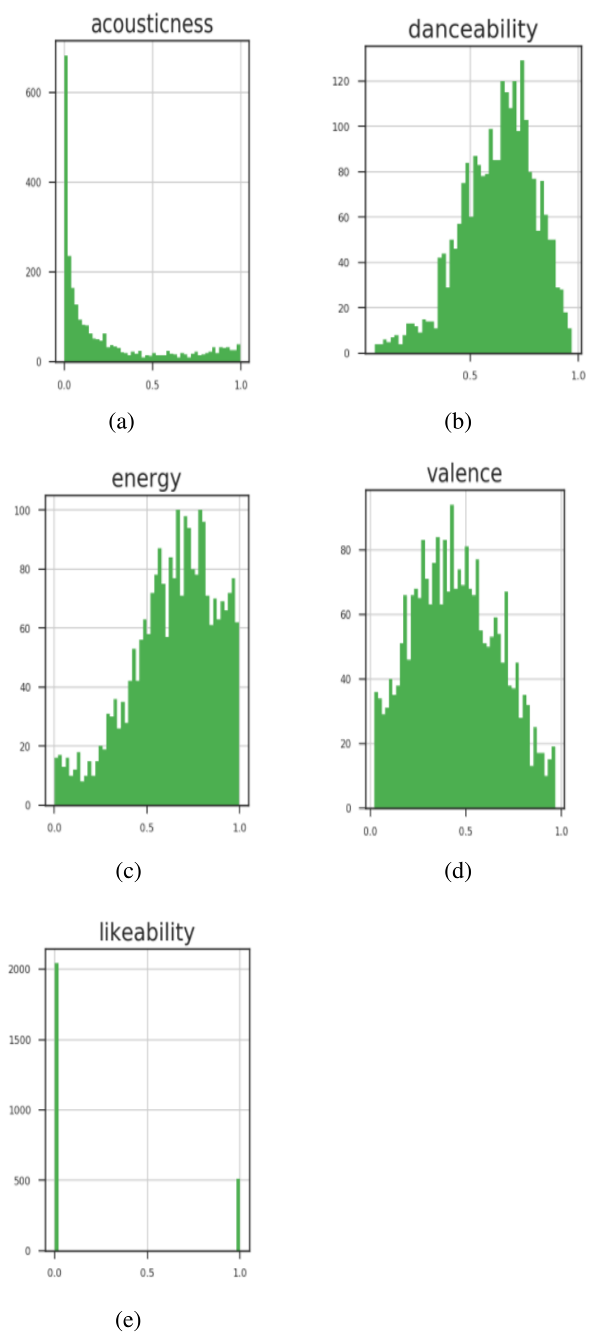


Figure 1. Distribution of some features in the data set.

5.2. Data Collection :

Now that we have an idea of how our data looks like, we need to get a collection of songs so that we can run our algorithms on that data set. There is an external library named as Spotipy that allows users to interact with Spotify's web API. With Spotipy we get full access to all of the music data provided by the Spotify platform. To access a specific users library, one must have proper authorization and its achieved by using the Client_ID and Secret Token provided by Spotify.

Once authorized to extract songs via Spotify's API, the process of creating a collection follows steps:

1. First step involves getting the playlist ids and the user name or user ids. Using these details we send request to API and extract the songs from from the given playlist. We have 2 playlists one consists of the songs that are liked by the user and the other consists of the songs ot liked by user

2. After getting the songs from the playlists, we clean the data by removing some of the features that we do not need to run our algorithm. Features like song name, artist name, external_urls etc are removed. Only the features mentioned above in the table are kept for future processing

3. Now we normalize the feature so as to maintain consistency. All the features are normalized and are in range of 0 to 1

4. After feature normalization, we merge the songs from the playlists and create a single data frame, which is used in future for performing all the algorithms

After cleaning the data and merging it together into a single data frame, we know proceed ahead to perform analysis on the data using both supervised and unsupervised learning.

5.3. Supervised Learning

5.3.1. Feature Selection and Elimination(RFE)

Before we start implementing our regression and classification algorithms, it is useful to know about feature selection and feature elimination. Sometimes, our data has large number of features associated with it and not all the features contribute equally towards the data. In order to make the algorithms work more effectively and efficiently, it becomes useful to select only those features that have most effect on the data. An algorithm to show how Recursive Feature Elimination works can been in Algo.1.

Algorithm 1 Recursive Feature Elimination

```
1: Tune/train the model on the training set using all predictors
2: Calculate model performance
3: Calculate variable importance or rankings
4: for each subset size $S_i,\ i = 1...S$ do
5:     Keep the $S_i$ most important variables
6:     [OPTIONAL] Pre-process the data
7:     Tune/train the model on the training set using $S_i$ predictors
8:     Calculate model performance
9:     [OPTIONAL] Recalculate the rankings for each predictor
10: end for
11: Calculate the performance profile over the $S_i$
12: Determine the appropriate number of predictors
13: Use the model corresponding to the optimal $S_i$
```

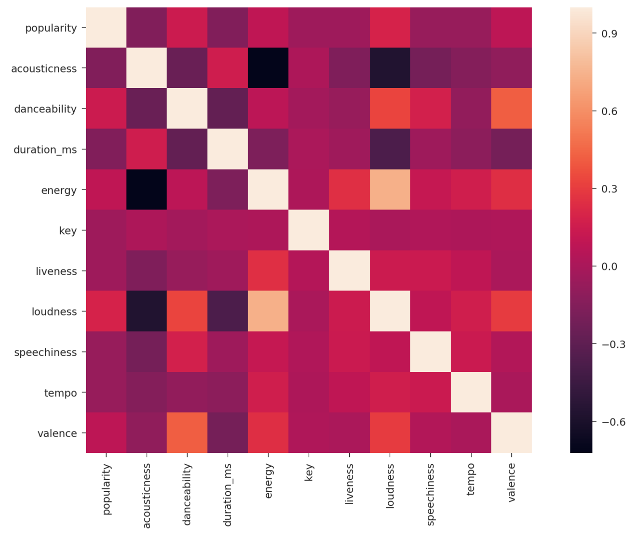


Figure 2. Co-relation between the features in our data set.

In fig.3, we can see how the features are correlated. Darker the color in the matrix, more negatively they are correlated. There are very few features that have highly positive correlation with other feature.

Later in the analysis we will see how does feature elimination work and whether it is actually good to do it or not.

5.3.2. Logistic Regression

Logistic regression is a regression analysis most commonly used when the dependent variable is binary(i.e when we have binary classification problem). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. In other words, Logistic regression predicts the probability of a data point belonging to a class and returns the class that has maximum probability of including that point. In our case, it predicts the chances of a user liking a song. A predictive analysis like linear regression can be converted to binary classification problem (logistic regression) by making two changes. First we replace the distribution for dependent variable with a Bernoulli distribution, which is more appropriate for the case when the response is binary. i.e.

$$p(y|x, w) = Ber(y|\mu(x))$$

, where

$$\mu(x) \ = E[y|x] = p(y = 1|x)$$

Second, we compute a linear combination of the inputs, as before, but then we pass this through a function that ensures $0 <= \mu(x) <= 1$ by defining

$$\mu(x) = sigm(w^T x)$$

where sigm(n) is a **sigmoid** function or **logistic** function. The **logistic function**, also called the sigmoid function an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits. A plot to show sigmoid function can be seen in fig.4.

Algorithm 2 Performing Logistic Regression using existing machine learning libraries.

```
1: Given data is classified as 0 or 1
2: Split data into Training and Test sets. Also, split target or true labels accordingly
3: Using sklearn's linear_model library, create a logistic regression model
4: Learn the Logistic Regression model using training data
5: Using the model learned above, fit test data.
6: Measure performance of the model on test data using sklearn's metrics module like plotting average precision-recall, roc_curve_auc etc.
```

5.3.3. Support Vector Machines

Support Vector Machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.. An SVM model is a representation of the

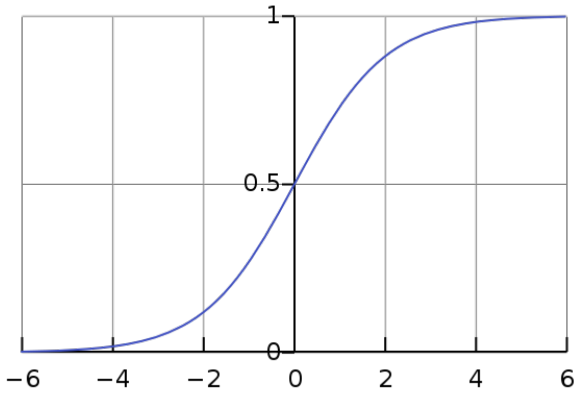


Figure 3. According the image above, any value greater than 0.5 is classified as 1 and any value less than 0.5 is classified as 0.

examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

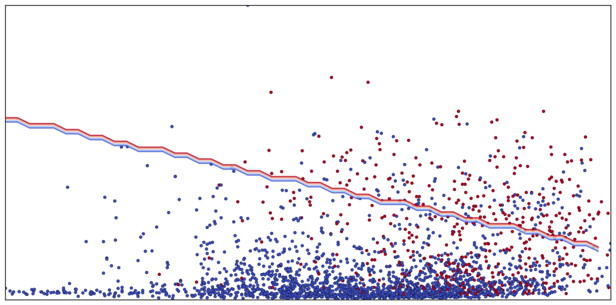


Figure 4. SVM with linear kernel performed on data with top 2 features.

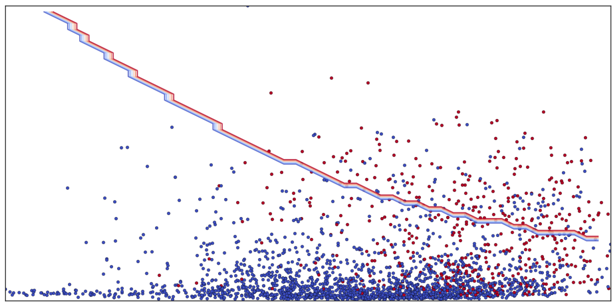


Figure 5. SVM with Gaussian (RBF) kernel performed on data with top 2 features.

Margin SVM When the data is linearly separable, then we can divide points that belong to different classes by a Hyperplane as shown in Fig.7. The hyperplane is bounded by boundary given by equations $w.x_i - b = 1$ and $w.x_i - b = -1$. For data point x, if $w.x - b >= 1$ then it is classified as 1 else if $wx - b <= -1$ then it is classified as 0. These constraints state that each data point must lie on the correct side of the margin. The width of the hyperplane is determinedby the support vectors or the data points that lie on the margins (as shown in Fig.7).

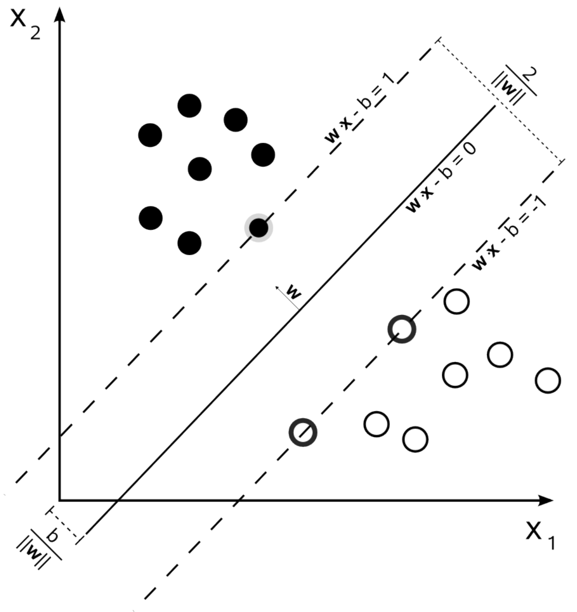


Figure 6. SVM Margins and Margin Hyperplane with data points plotted.

If the data is not linearly separable, then we use kernels to perform SVM. Kernels transform the high dimensional data into low dimensions. Most popular Kernels are Gaussian kernel (RBF) and is given by equation where n is n-dimensions :

$$(1/\sqrt{(2.pi.\sigma)^n}).exp(-(|x|^2)/(2\sigma^2))$$

Algorithm 3 Performing SVM using existing machine learning libraries.

```
1: Given data is classified as 0 or 1
2: Split data into Training and Test sets. Also, split target or true labels accordingly
3: For each kernel (linear and rbf), set up the parameter values i.e. set multiple C and $\gamma$ values
4: Using sklearn's linear_model library, create a SVM (SVC) model
5: Learn the SVM model using training data for each of the parameters chosen in above step
6: Using the model learned above, fit the test data
7: Measure performance of the model on test data using sklearn's metrics module like plotting average precision-recall, roc_curve_auc etc
```

5.3.4. Random Forests

Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

In performing Random Forest Classifier using sk-learn library, the main parameters that help us in model selection are n_estimator and max_features. n_estimators is the number of trees in the forest. More the number of trees better is the result but it does increase computational time. The max_features parameters decides the size of the subset of features to consider while splitting a node. Lower size of the subset means greater reduction in variance but increase in bias.

One more advantage of using sk-learn ensemble.RandomForestClassifier is that we can find out the importance. The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable.

In the analysis, we also take a look at the results given by RFE and Random Forest's feature importance results and how the classes behave in each case.

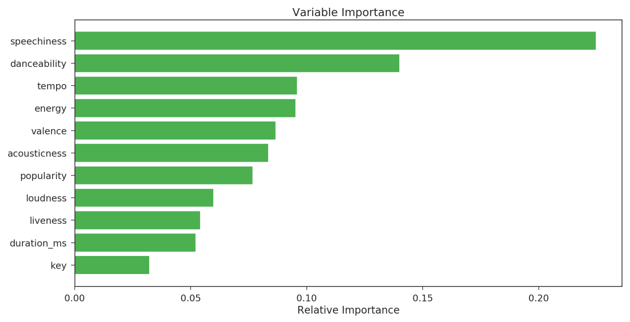


Figure 7. Feature importance generated by Random Forest Classifier

Algorithm 4 Performing Random Forest Classification using existing machine learning libraries.

```
1: Given data is classified as 0 or 1
2: Split data into Training and Test sets. Also, split target or true labels accordingly
3: Set parameters with different n_estimators and max_features for each of the model
4: Using sklearn's linear_model library, create a RFC model
5: Learn the RFC model using training data for each of the parameters chosen in above step
6: Using the model learned above, fit the test data
7: Measure performance of the model on test data using sklearn's metrics module like accuracy_score,roc_curve_auc etc
```

5.4. Unsupervised Learning

5.4.1. k-means

k-means is the first choice of clustering technique that we use for its ability to find clusters with comparable spatial extent. Since it partitions n observations into k clusters where each observation would belong to the cluster with the nearest mean, it can explain the spatial correlation of normalized features, similar to ones present in our dataset.

k-means tends to partition data space into Voronoi cells and it was an expectation of ours that a reasonable partitioning can be achived in our dataset. k-means is an NP-hard algorithm, however, the package we used provided the option to specify random seed and number of random nationalizations which helped us get near-optimal results.

Formally, if given a set of observations $(x_1, x_2, \cdots x_n)$, where each observation is a d-dimensional vector, k-means clustering can partition n observations into k ($\leq$ n) sets $\mathbf{S} = \{S_1, S_2, \cdots S_n\}$ to minimize the per-cluster sum-of-squares (or variance).

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n - \mu_k||^2$$

$$\textit{with } r_{nk} = \{1 \textit{ if } x_n \in S_k \; 0 \textit{ otherwise}\}$$

Figure 8. k-means objective function

$$S_i^{(t)} = \{x_p : \|x_p - \mu_i^{(t)}\|^2 \leq \|x_p - \mu_j^{(t)}\|^2 \,\forall j, 1 \leq j \leq k\}$$

Figure 9. k-means E-step: Assign

Algorithm 5 Performing k-means using existing machine learning libraries.

```
1: Set parameter with different n_clusters and 'minkowski' distance metric
2: Using sklearn's cluster library, create a KMeans model
3: Fit and run the KMeans model using dataset for all choices of n_clusters (aka $k$)
4: For each $k$, calculate and save the distortions using cdist method from scipy's spatial distance package
5: Plot the distortions and choose $k$ based on bend in the graph
```

K-means is Hard clustering procedure because we just assign a point to single cluster at all times, we do not consider probabilities of that belonging to other clusters. At all times, we assign the point to the nearest centroid or mean.

5.4.2. Gaussian Mixture Models

A Gaussian mixture model (GMM) is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.In this model, each base distribution in the mixture is a multivariate Gaussian with mean μ_k and covariance matrix Σ_k. Thus the model has the form:

$$p(x_i, \theta) = \sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \Sigma_k)$$

Gaussian Mixture model is a soft clustering procedure, it is because it calculates the probability of a point belonging to given clusters or distributions. Gaussian Mixture model can be converted to k-means clustering by tuning some of the parameters.

As stated above under GMM that we initially do not know the parameters, so the main task here is to identify the parameters that can produce best clustering of our data. In

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|}\sum_{x_j \in S_i^{(t)}} x_j$$

Figure 10. k-means M-step: Update

order to achieve this an approach known as Expectation Maximization (EM) is used.

EM for GMM has 2 steps: **E step:** In E-step, we update the responsibility vector for each point given the parameters. The E-step can be written as following mathematically:

$$r_i k = \pi_k p(x_i|\theta_k^{(t-1)}) / \sum_{k'} \pi'_k p(x_i|\theta_{k'}^{(t-1)})$$

M step: We optimize the Auxilary function Q w.r.t π are weights and θ_k parameters like mean and covariance matrix. The auxilary function can be represented as $Q(\theta, \theta^{(t-1)} = \sum_i \sum_k r_i k log \pi_k + \sum_i \sum_k r_i k log p(x_i|\theta_k)$ where $r_i k$ is responsibility of cluster k for point i and π_k is weight for cluster k.

The parameters μ_k and $\sum_k$ represent means and covariance matrix respectively. These parameters are also updated and new estimates are given by :

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T$$

Figure 11. Equations for new parameter estimates in M-step

Algorithm 6 Performing Gaussian Mixture with EM using existing machine learning libraries.

```
1: Consider the data for top features based on feature importance.
2: Set the number of components or number of clusters you want the data to be grouped into.
3: Using sklearn's mixture library, create a Gaussian Mixture model (GMM) with parameter component set to the number of cluster determined in previous step.
4: Fit and run the GMM model.
5: Plot the clustering result by GMM.
```

The Sklearn's Gaussian Mixture Model performs EM by itself, so we do not need to perform it explicitly.

5.4.3. DBSCAN

Since most of the data points are closely packed owing to the spatial proximity of certain features, it makes sense to try out Density-based spatial clustering of applications with noise (DBSCAN). DBSCAN groups together points that are closely packed and marks the outliers into a cluster of their own.

There are kinds of designations that each point gets during the DBSCAN algorithm: *core point, (density-)reachable point and outlier*.

1. A point p is a *core point* if at least minPts are within ϵ distance of the point p

2. A point q is directly reachable from point p if q is ϵ distance from p and p is a core point

3. A point q is reachable from point p if there is a path from p to q and all points, except possibly q, are directly reachable from previous point

4. Points which are not reachable from any point are outliers

5. Each cluster contains at least 1 core point

6. All points within cluster cluster are mutually density-connected

7. If a point is density-reachable from any other point in the cluster, it is part of the cluster too

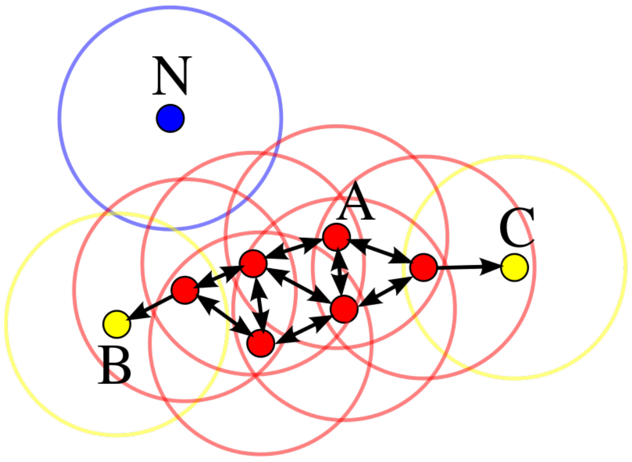


Figure 12. Simple DBSCAN cluster: red are core, yellow are reachable, blue is noise

Algorithm 7 Performing DBSCAN using existing machine learning libraries.

```
1: Set parameters with different eps, min_samples and
   'minkowski' distance metric
2: Using sklearn's cluster library, create a DBSCAN
   model
3: Fit and run the DBSCAN model using dataset for all
   choices of eps and min_samples
4: Infer appropriate parameters based on domain knowl-
   edge or, alternatively, try OPTICS
```

5.4.4. Agglomerative Clustering

Hierarchical clustering can build a hierarchy of clusters based on linkages. There are two kinds: *agglomerative* and *divisive*. Although the clustering is not guaranteed to find the optimum solution, it works well on our data set where genres can be inferred in a hierarchical manner as the data points are just music properties.

We use Agglomerative clustering for the ease of simplicity for choosing the clusters after observing a dendogram. A measure of dissimilarity between sets of observations tells us which clusters should be combined. This can be achieved using a distance metric and a linkage criterion that specifies the dissimilarity of sets as a function of pairwise distances of observations in the sets.

The Ward metric was selected after comparing results from the single, complete and average linkage metrics. Ward's method measures the increase in variance for the cluster being merged. Sklearn also lets us provide a parameter which can stop clustering when sufficiently small number of clusters begin forming.

One can also find the cophentic distance to find cophentric correlation. Cophenetic correlation is a measure of how faithfully a dendrogram preserves the pairwise distances between the original unmodeled data points.

Algorithm 8 Performing Agglomerative Clustering using existing machine learning libraries.

```
1: Set parameter for linkage and choose a distance metric
   like 'ward'
2: Using scipy's cluster library, create a dendogram with
   the linkage
3: Fit and generate the dendogram using dataset
4: Infer dendogram cut off distance based on dataset and
   number of the clusters generated
5: Note the cophenetic correlation distance
```

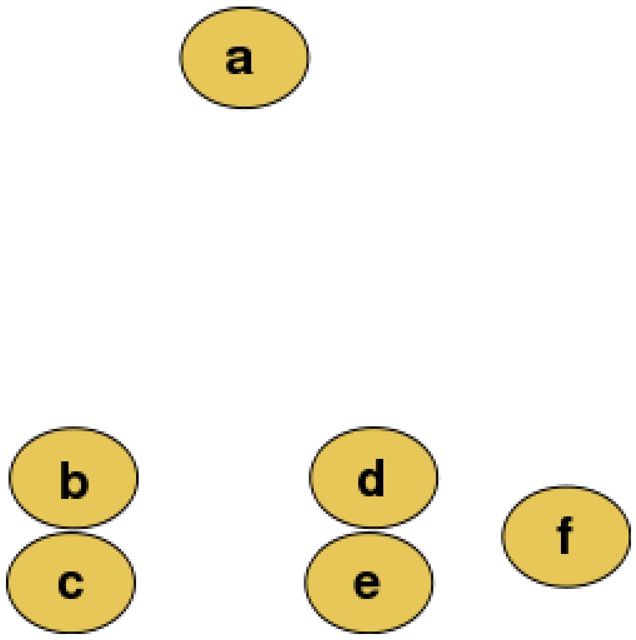


Figure 13. Agglomerative clustering in work

6. Experiments

We executed the above mentioned supervised and unsupervised techniques on our data set in order to achieve targets described in introduction section of this report.

6.1. Supervised Learning for classifying song

6.1.1. Training and Test data

We divided our data into 70:30 ratio. 70% portion was used as Training set and the remaining 30% was used as Test set. This step was common through out the supervised learning techniques we used in our analysis. We performed each algorithm on 2 sets of data. In the first set, we included all the features available to us and in the second set we only included the important features as suggested by RFE and Random Forest's feature importance outputs.

6.1.2. Setting Parameters

According to the algorithm being used is done by specifying C, γ and kernel values for the Support Vector Classifier(SVC). In case of Random Forests Classifier, parameters used are n_estimators and max_features.

6.1.3. Parameter selection using GridSearchCV

This has been used for model selection in each of the technique/algorithm. GridSearchCV is an exhaustive search over specified parameter values for an estimator.GridSearchCV implements a fit and a score method. It also implements predict, predict_proba for score calcualtion of each class, decision_function, transform and inverse_transform if they are implemented in the estimator used. GridSearchCv returns the best estimator based on the model parameters set for the model. We then select the best estimator provided by GridSearchCV to predict the class of the song in test set. The scoring function can be specified at the time of model creation. We have used 'average_precision' as scoring function in our experiment.

6.1.4. Performing Algorithms

The first supervised classification technique we performed is Logistic regression. Following the steps mentioned in Algorithm 1 above, we first trained our model on the training set and then made prediction using the trained model on the test data.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid as explained above.

6.2. Unsupervised learning for clustering similar songs

6.2.1. Analyzing distribution of data

On further expanding our exploratory analysis we noticed skewed distribution of data points that was happening due to certain features which were not normally distributed e.g., acousticness. This led us to assert that to understand our data better we would have to reduce the dimensions and visualize it.

6.2.2. Principal Component Analysis

We tried reducing the number of components to n = 4, 3 and 2 respectively so that we could choose a number which could best explain our data. We also used PCA components several times to just visualize our clustering results as a projection on 2- and 3-dimensions. However, there was also an inherent need to drop certain features which were skewing the distribution regardless of applying PCA and that followed.

6.2.3. Clustering

With PCA done, we began our first clustering task using k-means. As k-means requires several random restarts to converge on a optimal solution, we tweaked the parameters as such and ran the algorithm for $k = 1 \cdots Length(genres)$. We choose the k using the graph for distortion which would give us a bend in the plotted line.

We also tried multivariate Gaussian Mixture Models to generate better clusters as the spatial uniformness had to be minimized. We tried out GMMs for several number of components and, similar to k-means, we visualized the clusters in 2- and 3-dimensional plots.

With not so exciting results as we had hoped with the above clustering techniques, we went ahead and tried DBSCAN owing to the dense local proximity of data points and also agglomerative clustering which could help us given an alternate perspective on music properties and their hierarchical nature.

For each clustering, we compared it with the clustering that was achieved using already available genre labels that we had got when we constructed our dataset.

6.3. Sampling

Finally, we chose the best trained model for our 'user' and assigned affinity values to clusters that we generated and used Inverse Transform Sampling on these values to sample songs for the user from the clusters.

6.4. Results

6.4.1. Logistic Regression

As we can see from the results above, the accuracy scores of logistic regression is almost same. So, reducing the number of features to top-3 features does not help in improving score. Even though the accuracy score is high but the average precision score is medium.

6.4.2. Support Vector Machines

For **Support Vector Machine (SVM)** with linear kernel, we perform the steps described above in algorithm 2. The C values used are = [1.0,10.0,100.0]. Since this uses a linear kernel, we do not need to set γ parameter. The results of performing SVM with linear kernel are mentioned below:

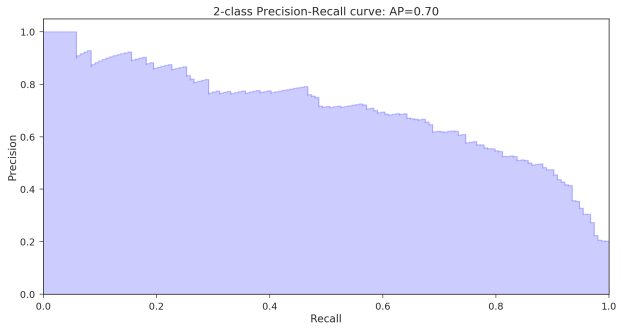
From the results above, we can see that accuracy score of SVM with linear kernel is high when we used full features as compared to the case when we used top-3 features. Also, just like logistic regression the average_precision is not so good and precision decreases rapidly when the number of features are reduced.

For **Support Vector Machine (SVM)** with rbf kernel, we perform the steps described above in algorithm 2. The C values used are = [1.0,10.0,100.0]. Now we also setγ parameter to following values = [0.001, 0.01, 0.1, 1, 2, 5]. The results of performing SVM with rbf kernel are mentioned in table 4 and 5.

From the results in table 4 and 5, we can see that accuracy score of SVM with rbf kernel is high when we used full features as compared to the case when we used top-3 features. Also, just like logistic regression the average_precision is higher in case of all features and precision decreases rapidly when the number of features are reduced.

For data set with all features:

```
'Accuracy score of Logistic
    Regression on test set:
    0.86'
'ROC AUC: 0.88'
```



For data set with top-3 features:

```
'Accuracy score of Logistic
    Regression on test set:
    0.85'
'ROC AUC: 0.87'
```

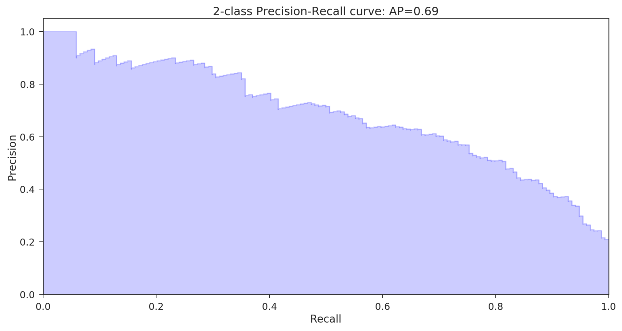


Table 2. Logistic regression results and Precision-Recall curve plots

6.4.3. Random Forests Classifier

For **Random Forests Classifier (RFC)** we perform the steps described above in algorithm 3. The n_estimators values used are = [10,30,40,50,100] and the max_features parameter has following values = ['auto','sqrt','log2']. The results of performing RFC are mentioned below:

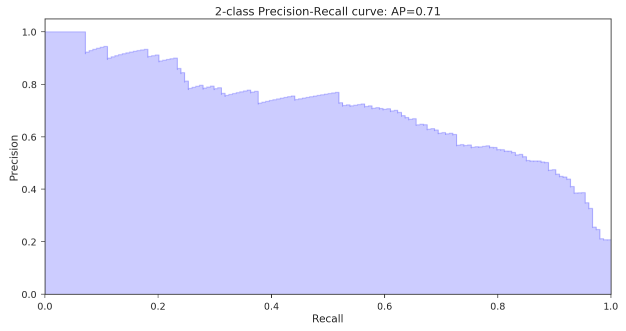
From the results above, we can see that accuracy score of RFC is high when we used full features as compared to the case when we used top-3 features.

6.4.4. Comparing all models

To compare all the models stated above, we have used roc_aoc_score. ROC(Receiver Operating Characteristic) curves are typically used in binary classification to study the output of a classifier. ROC curves feature true positive

For data set with all features:

```
"Best params: {'C': 100.0}"
'Accuracy score of SVM on test
     set: 0.87'
'ROC AUC: 0.89'
```



For data set with top-3 features:

```
"Best params: {'C': 10.0}"
'Accuracy score of SVM on test
     set: 0.84'
'ROC AUC: 0.86'
```

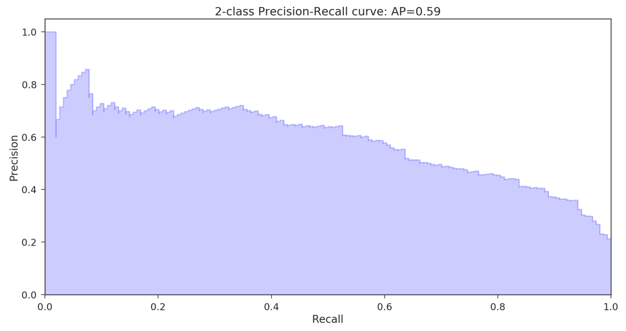


Table 3. SVM results and Precision-Recall curve plots

For data set with all features:

```
"Best params: {'C': 1.0, '
    gamma': 2}"
'Accuracy score of SVM on test
     set: 0.86'
'ROC AUC: 0.89'
```

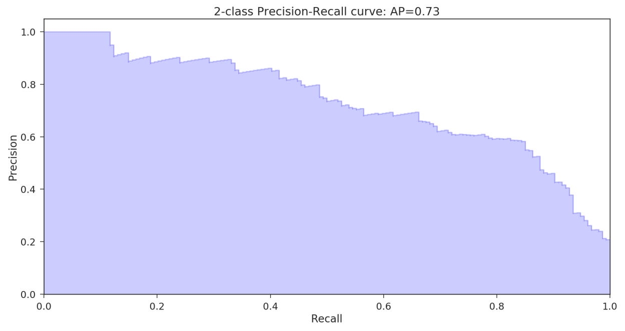


Table 4. SVM with rbf kernel results and Precision-Recall curve plots

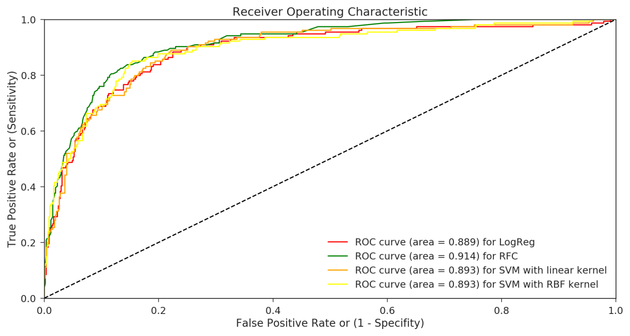


Figure 14. ROC Curve

rate on the Y axis, and false positive rate on the X axis. Basically, it is true positive rate vs false positive rate plot for a model. True positive means the data point that are correctly classified as 1s and false positive means a point which is wrongly classified as 1 by the model. In a roc_curve, the left most point is the ideal point with minimum false rate and maximum true positive rate. The area under the curve tells us about how well a model distinguish between two classes. Greater the area under the curve, better the model is in distinguishing the classes.

The curve above shows that RFC has most roc_auc_score 91% which means that is best among all the models we have in distinguishing the classes. Logistic regression is worst with 88%.

6.4.5. Principal Component Analysis

After applying PCA on our entire feature set with number of components = 4, 3 and 2, we decided that 4 components were good enough to explain over 89% of our data. However, even after PCA the data seemed to be heavily skewed, at least visually, by a couple of features, one of them being Accousticness. In order to fix that, we removed certain features and then applied PCA again on reduced feature set.

Throughout the clustering process, we projected the clustering results onto the 2D and 3D projections on PCA components for n = 2 and n = 3 respectively.

We observed, as above in Fig.15-17, that there was a pattern in the distribution of dominant features like Danceability, Speechiness and Valence.

We also projected the clustering based on an already available label of 'genre' that we had acquired on the creation of our dataset. Throughout the remaining clustering results,

For data set with top-3 features:

```
"Best params: {'C': 1.0, '
    gamma': 1}"
'Accuracy score of SVM on test
    set: 0.84'
'ROC AUC: 0.85'
```

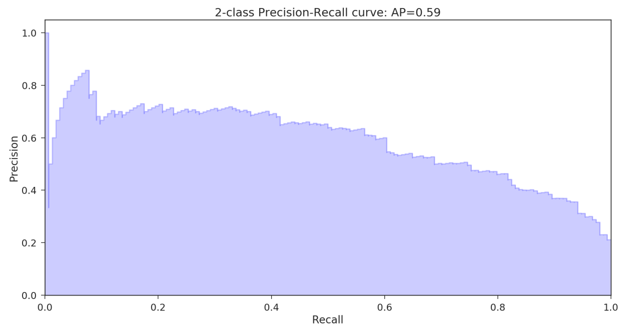


Table 5. SVM with rbf kernel results and Precision-Recall curve plots

For data set with all features:

```
"Best params: {'max_features':
    'sqrt', 'n_estimators':
   100}"
'Accuracy score of RFC on test
    set: 0.88'
'ROC AUC: 0.91'
```

For data set with top-3 features:

```
"Best params: {'max_features':
    'auto', 'n_estimators':
   100}"
'Accuracy score of RFC on test
    set: 0.86'
'ROC AUC: 0.89'
```

Table 6. RFC results and Precision-Recall curve plots

we compared the clusters with this genre clustering to get us an idea of the correlation between our clusters based on music properties and conventional genre labelling that Spotify does based on their models.

6.4.6. k-MEANS

We ran k-means for upto 15 clusters and for each we saved the distortion that was occurred. The distortion vs k graph as shown in Fig.19, we were able to observe a bend but it wasn't sharp as expected. This told us that k-means didn't work well on our data as clustering based on spatial uniformity (like circles or spheres) wasn't going to be enough to find patterns. As expected, on project cluster results for k=2, which we chose from the bend in Fig.19, we saw the clusterings as seen in Fig.20-21.

k-means was able to group together clusters which were highly danceable to but was confused for the rest of the spectrum. The choice of the remaining two clusters was largely owed to the uniform dense structure that the points in those regions were making. k-means decided to naively split them into two sections. Coincidentally, that did segregate into certain higher-level partitioning of genres, if compared with the genre clustering in Fig.18.

6.4.7. GAUSSIAN MIXTURE MODELS

The disappointing performance of k-means prompted us to try out Gaussian Mixture Models by inference rather than using some Bayesian priors. Surprisingly, in higher dimensions the correlation of clusterings with genres was increased as compared to k-means.

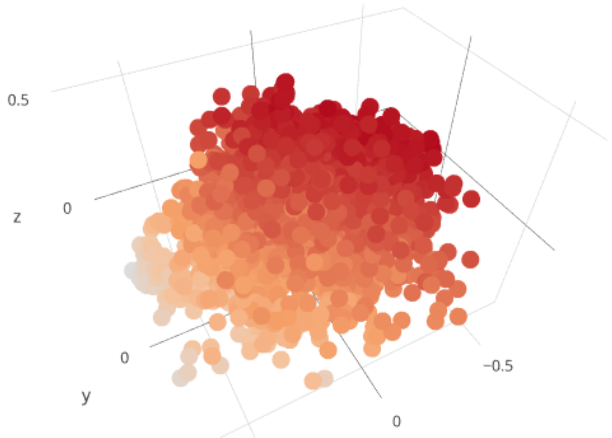


Figure 15. Danceability visualized on PCA plot with n_components = 3

As observed in Fig.22 and Fig.23, projections of GMM cluster for n=5 components became very close to the genre clustering in Fig.18.

Since GMMs allow the degree of variance and has stronger resistance to noise, it performed significantly better than k-means.

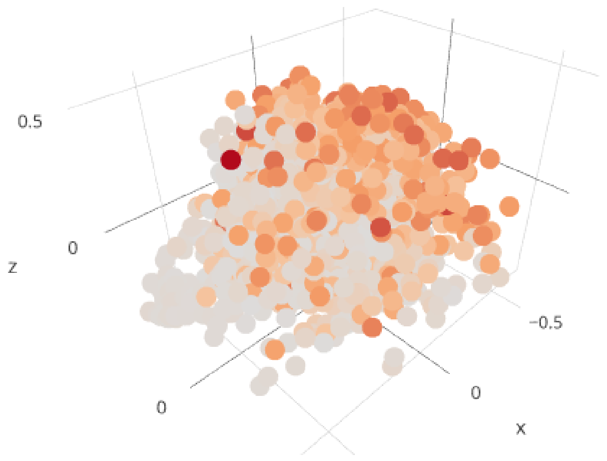


Figure 16. Speechiness visualized on PCA plot with n_components = 3

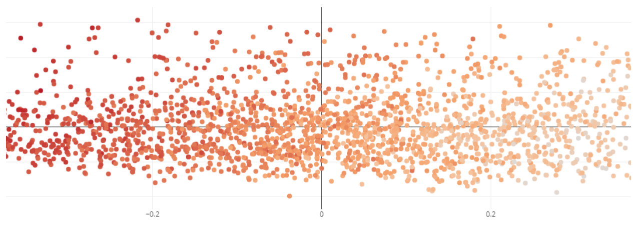


Figure 17. Valence visualized on PCA plot with n_components = 2

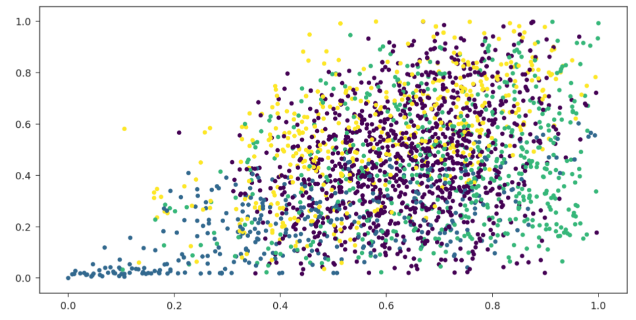


Figure 22. GMM 2D projection

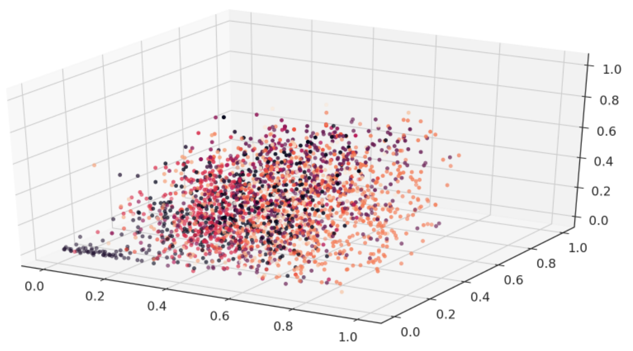


Figure 18. Projection of clustering based on genres

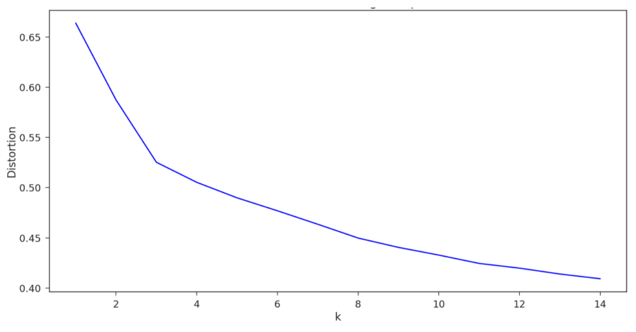


Figure 19. k-means elbow

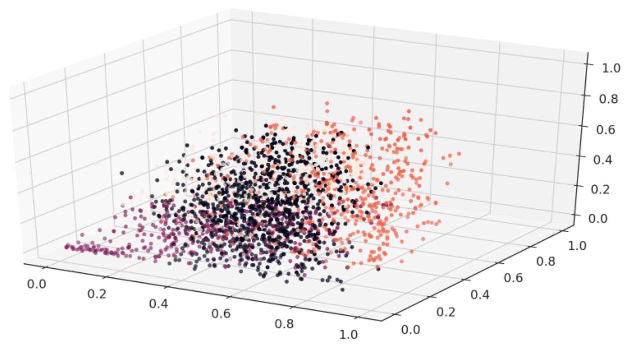


Figure 23. GMM 3D projection

6.4.8. DBSCAN

Although pretty satisfied with the clustering results that GMMs gave us, we still wanted to explore the dense nature of our datapoints, even in higher dimensions (if measured through Minkowski distance metric).

We tried out DBSCAN for the same and the clustering results were significantly different. On closer inspection, we found that DBSCAN was clustering the data points in the shape of the overall dense structure that majority of data points were forming (something that confused k-means).

Fortunately it seemed to work better than k-means, how-

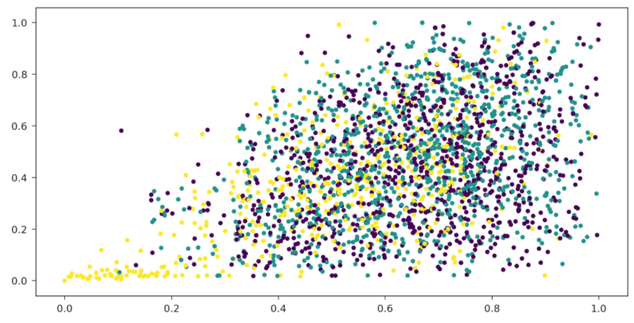


Figure 20. k-means 2D projection

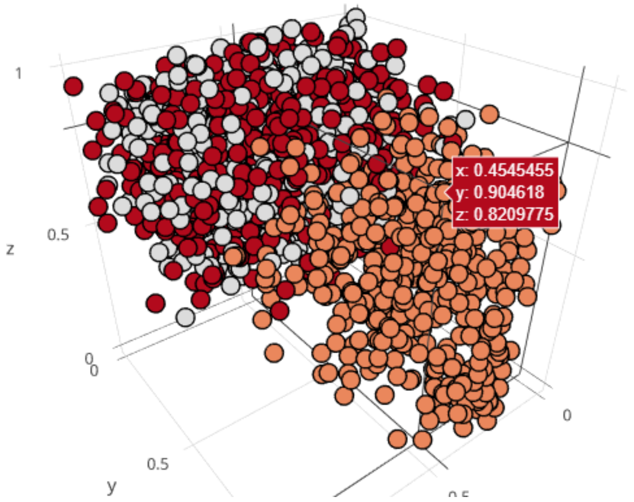


Figure 21. k-means 3D projection

ever we did have to fiddle with *minPts* and *epsilon* values as there is no standard way of inferring the parameters except perhaps OPTICS. There was no significant code in Python for OPTICS so left it out and tuned the parameters based on our knowledge of the data set.

6.4.9. Agglomerative Clustering

As a final step in the clustering adventure, we tried to understand the hierarchical nature of genres or the music properties in this case. It proved to be a daunting task to make sense of it although we were able to apply standard techniques to find the cutoff distance candidates in the dendogram as (12.5, 6.5).

There is definitely some hierarchical pattern to decipher, however, how it comes out to be, we were unable to figure out at least at the time of this report.

The cophentic correlation in this clustering came out to be 0.60. While not ideal, it is close to one and acts as a strong indication of a correlation in the clusters.

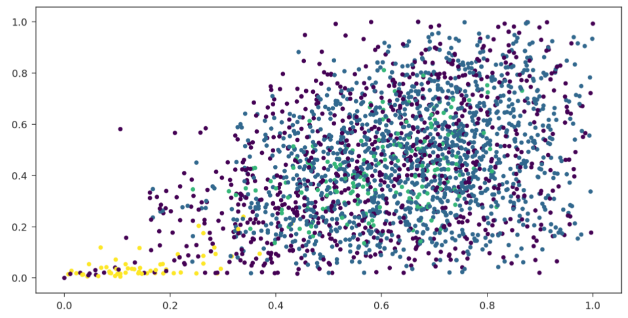


Figure 24. DBSCAN 2D projection

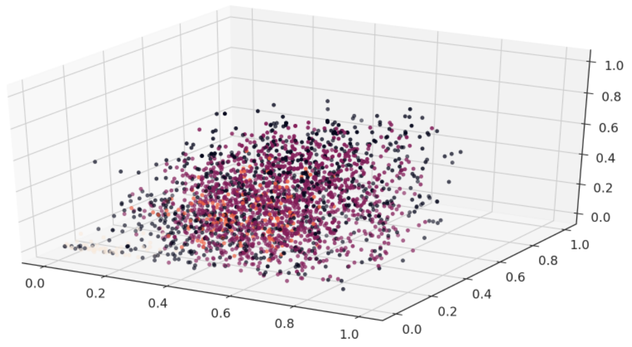


Figure 25. DBSCAN 3D projection

6.4.10. Clustering compared

```
Best clustering: GMM
GMM - 65% similar to genre
    clustering
DBSCAN - 55% similar to genre
    clustering
k-means - 35% similar to genre
     clustering
Agglo - *% similar to genre
    clustering
* Did not calculate
```

Table 7. Clustering compared

6.4.11. Sampling

After we chose the best model, which was the Random Forests Classifier, we found the affinity values (percentage of likeability to a cluster based on frequency) for each of our 5 clusters generated by Gaussian Mixture Model.

We used Uniform Inverted Sampling on these clusters based on the proximity to the cluster centers to sample songs in weighted probability of the affinity values.

The songs listed were approximately 74% similar to the user's tastes.

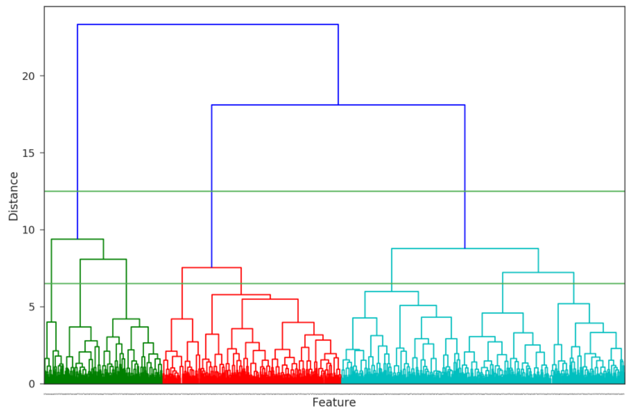


Figure 26. Agglomerative Clustering with cutoffs

7. Conclusion

We were able to demonstrate the power of classical machine learning techniques in both domains of supervised and unsupervised learning to solve small-scale subproblems as a precursor to building a scalable client-side model learning recommender.

Our supervised learning techniques reached upto 88% accuracy and that can only be improved with application of deep learning models, which we tested but not to the extent of covering in this report. Random Forests Classifier proved to be the strongest, with SVMs just trailing behind, as a worthy classifier capable of classifying multiple labels.

With a variety of clustering techniques, we were able to reach our end goal of generating our own determination of 'genres'. However, further investigation and work needs to be done to understand in deeper detail about what is happening in higher dimensions on clustering. We suspect that a divergent dataset would have better clustering than the small dataset we were able to accumulate. Gaussian Mixture Models worked surprising well for this kind of data, although DBSCAN was very close to giving us really interesting results and agglomerative clustering is something we would like to explore more.

Finally, we were immensely happy with the sampling test we did by combining the two domains of learning and are confident in building a web app using Tensorflow.js to demonstrate the same in near future.

References

Essentia. Essentia documentation. http://essentia.upf.edu/documentation/. Accessed: 2018-02-26.

George McIntire, ODSC. A machine learning deep dive into my spotify data. https://opendatascience.com/blog/a-machine-learning-deep-dive-into-my-spotify-data
2017. Accessed: 2018-02-26.

Kevin P. Murphy. Machine learning a probablistic perspective.

Nikhil Sonnad, Quartz. The magic that makes spotifys discover weekly playlists so damn good. https://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-pla
2015. Accessed: 2018-02-26.

Paul Lamere. Spotify api. http://spotipy.readthedocs.io/en/latest/.

Sander Dieleman. Recommending music on spotify with deep learning. http://benanne.github.io/2014/08/05/spotify-cnns.html, 2014. Accessed: 2018-02-26.

scikit-learn.org. scikit-learn. http://scikit-learn.org.

Sophia Ciocca, Hacker Noon. Spotify discover weekly: How machine learning finds your new music. https://hackernoon.com/spotifys-discover-weekly-how-machine-learning-fin
2017. Accessed: 2018-02-26.

Spotify. Spotify api. https://beta.developer.spotify.com/documentation/web-api/.

Stackexchange. Dbscan parameter tuning. https://stats.stackexchange.com/questions/88872/a-routine-to-choose-eps-and-minpts-for-dbscan.

Tiago Ramalho. Inverse transformation sampling. http://www.nehalemlabs.net/prototype/blog/2013/12/16/how-to-do-inverse-transformation-sampling-in-scip

Tristan Jehan, David DesRoches, The Echo Nest. Analyzer documentation. http://docs.echonest.com.s3-website-us-east-1.amazonaws.com/_static/AnalyzeDocumentation.pdf, 2014. Accessed: 2018-02-26.

Wikipedia. Wikipedia. https://en.wikipedia.org.

(Kevin P. Murphy) (Wikipedia) (scikit-learn.org) (Spotify) (Paul Lamere) (Tiago Ramalho) (Stackexchange) (George McIntire, 2017) (Sophia Ciocca, 2017) (Sander Dieleman, 2014) (Nikhil Sonnad, 2015) (Tristan Jehan, David DesRoches, 2014) (Essentia)