Spotify Music Recommendation System

Krina Shah
NJ, US
kshah119@stevens.edu

Vaishnavi Shirsath
NJ, US
vshirsat@stevens.edu

Raj Shah
NJ, US
rshah97@stevens.edu

I. Introduction

With the growth of the internet, a large amount of music data is available on the internet. Hence, it becomes necessary to develop a recommendation system. For the recommendation system to be an ideal one should have maximum user satisfaction and minimum user effort. The music recommender must recommend user's songs of his favourite list each time. The recommendation service is to recommend items that users may be interested in based on users' predefined preferences or users' access histories. In this project, we will built a music recommendation system on Spotify data-set. Since it uses various ways to recommend and with around 100 million of users the variety and amount of data available is also very large. The objective of this project is to recommend songs to the user. For simplicity, we predict whether a particular song matches the user's interest or not based on the different features on the data-set. If it matches the user's interest then it will recommend the song to the user.

There are three main types of recommendation system:

1) Content Based Recommendation
2) Collaborative Filtering Technique
3) Popularity Based Recommendation

1. Content Based Recommendation System: Content Based System works without user's evaluation or ratings. It uses machine language to acquire information and has several algorithms such as decision trees, neural networks or vector-based methods. The objects are defined by characteristics and related attributes. Knowledge-based recommendation uses demands and preferences of the user and decides predictions with functions and features of objects. In the content-based filtering approach, the user profiles are first formed by extracting features of the data items which have been accessed in the past. Based on the user profiles, the system recommends only the data items that are highly relevant to the user profiles by computing the similarities between the features of the data items and the user profiles.

2. Collaborative Filtering Technique: Collaborative filtering is the most common method for recommender. It uses the K-nearest neighbor technique (KNN). The music taste of users calculates the distance between different users. The method searches for neighbor users who share similar interest in music and recommend the content based on these information. Therefore, the recommender predicts users preferences with by user-item relationship. In daily life it can be compared to a friend's recommendation.

3. Popularity Based Recommendation: As the name suggests Popularity Based recommendation involves recommending items rated highest by users. We can see its implementation in Spotify as the trending playlist which is basically list of tracks most liked by all users on the platform. In this project we have Implemented all three methods of Recommendation and their comparison.

II. Related Work

Most of the recommendation systems rely on user interactions and user feedback to recommend music. This approach also known as collaborative filtering generally encounters a cold start problem as we don't have any usage information when we start recommending songs to new users or when we add new songs to the system that have not been rated by any user. To deal with this problem Spotify has introduced additional audio features of the songs in addition to the user interactions to make recommendations based on the audio features of the song. This method is called content-based recommendation. There is another type of approach in recommendation known as popularity based recommendation. It is a type of recommendation system which works on the principle of popularity and or anything which is in trend. Ryan et al. [1] uses artist embedding with the help of cosine similarity for making recommendations. They have used Convolution Neural Network (CNN), Long Short Term Memory (LSTM) and Recurrent Neural Network (RNN) for implementation. Chou et al. [2] has proposed an Adaptive linear mapping model (ALMM) model to deal with the cold start problem. Bogdanov et al. [3] propose three content-based recommendation approaches (1) Semantic distance from the mean (SEM-MEAN, (2) Semantic distance from all tracks (SEM-ALL), (3) Semantic Gaussian mixture model (SEM-GMM. After implementing these approaches, they conclude that although the considered content-based approaches to music recommendation do not reach the satisfaction and novelty degree of the collaborative filtering approach, the difference in performance diminishes to a great extent while using semantic descriptors. We know that collaborative filtering will almost always outperform content based filtering. But collaborative filtering suffers from a number of factors like the cold start problem, scarce usage data, and even a very big usage data. Whereas content- based filtering limits the recommendations to the users current liking and does not help with expanding users interest. Popularity based recommendation is a more generalized recommendation system; it will not help with personalized suggestions. In this project we will be exploring all three approaches in recommendation systems mentioned above and compare their performances.

III. Our Solution

A. Description of Dataset

The link of the dataset:

https://www.kaggle.com/huanntran100/spotify-song-popularity-prediction/notebook

We capture our dataset from kaggle. Our dataset contains about 18 features and around 228,000 training samples. Some of the important features are genre, artist name, track name, track id, popularity, acousticness, danceability, duration(ms), energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time signature and valence.

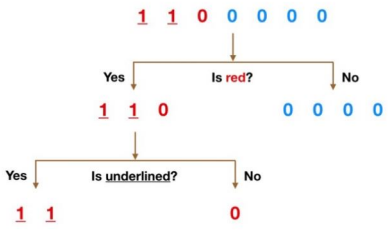
```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 228159 entries, 0 to 228158
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   genre             228159 non-null  object
 1   artist_name       228159 non-null  object
 2   track_name        228159 non-null  object
 3   track_id          228159 non-null  object
 4   popularity        228159 non-null  int64
 5   acousticness      228159 non-null  float64
 6   danceability      228159 non-null  float64
 7   duration_ms       228159 non-null  int64
 8   energy            228159 non-null  float64
 9   instrumentalness  228159 non-null  float64
 10  key               228159 non-null  object
 11  liveness          228159 non-null  float64
 12  loudness          228159 non-null  float64
 13  mode              228159 non-null  object
 14  speechiness       228159 non-null  float64
 15  tempo             228159 non-null  float64
 16  time_signature    228159 non-null  object
 17  valence           228159 non-null  float64
dtypes: float64(9), int64(2), object(7)
memory usage: 31.3+ MB
```

Out of all features we have selected those features that are highly correlated. Genre, artist name, track name attributes contain character values while most other attributes contain numerical values. Our Target value "popularity" contained continuous integer values. We have performed various data preprocessing methods before running any Machine learning algorithm. Data preprocessing implementation is explained in the later part of the report. To train and test machine learning models we have divided our training data and test data into 80-20 percent.

B. Machine Learning Algorithms

(1) Decision Tree:

The Decision Tree Model is a non-parametric supervised learning model which is used primarily for classification. The main goal of this model is to create a target value for a certain target variable based on simple decision based rules from the features in a dataset.



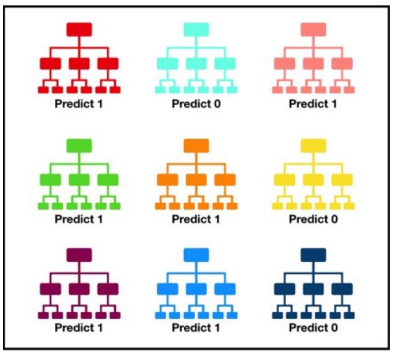
Simple Decision Tree Example

The decision tree (DT) classifier works well with our dataset as we are trying to find relationships between the features in our dataset, like "CASESTATUS" "EMPLOYER" for example. Therefore the nature of the decision tree makes it an easy choice to implement on our dataset. However the disadvantage of using decision trees is that the classifiers can become very complex depending on the data set and might take an extremely long time to complete their classification. In other words the decision tree classifier might be a little complex when running the classifier locally. The "depth" parameter specified in the DT- classifier, is also an important characteristic of the decision tree. Using this parameter we can specify the depth to which each decision tree will create it's nodes and branches. We used a depth of "10" on our dataset. It's probably much easier to understand how a decision tree works through an example. Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this? Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, "Is it red?" to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don't go down the No branch. The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, "Is it underlined?" to make a second split. The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decision tree was able to use the two features to split up the data perfectly.

We tested the decision tree classifier on the over-sampled, unsampled and under-sampled data on a local machine as well as on Amazon EMR instances (3 and 5 instances specifically)and noted the results. We then performed a comparative analysis on how the decision tree performed.

(2) Random Forest:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction



Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same

direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1) There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2) The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

(3) Naïve Bayes:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other

$$\textbf{Naive Bayes classifier: } v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i|v_j)$$

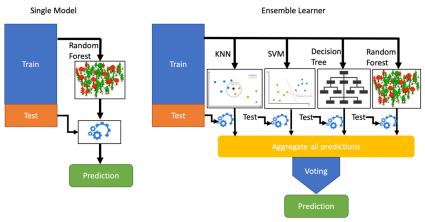
Working of Naive Bayes:.

1) Read training data-set T.
2) Calculate mean and standard deviation of predictor variables in each class.
3) Calculate the probability of f(i) using Gauss density equation in each class for all predictor variables.
4) Calculate likelihood for each class.
5) Choose the greatest likelihood.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

(4) XGBoost:

XGBoost is a decision-tree based ensemble Machine Learning algorithm that uses a gradient boosting framework. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. Bagging and boosting are two widely used ensemble learners. Though these two techniques can be used with several statistical models, the most effective usage has been with decision trees. XGBoost provides a parallel tree boosting that solves many data science problems in a fast and efficient way. XGBoost allows users to define custom optimization objectives and evaluation criteria. This adds a whole new dimension to the model and there is no limit to what we can do.



XGBoost Parameters:.

1) General Parameters: Guide the overall functioning.
2) Booster Parameters: Guide the individual booster (tree/regression) at each step.

The XGBoost library implements the gradient boosting decision tree algorithm.

(5) SGDClassifier:

Stochastic Gradient Descent (SGD) Classifier is a linear classifier (Support Vector Machine (SVM), logistic regression) optimized by the SGD.

SGD is basically an optimization method and svm and logistic regression are the linear machine learning algorithms. So SGD is used to minimize or maximize the loss/cost function defined by the svm or logistic regression. In other words, it is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression.

Gradient descent is used to minimize a cost function. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. But we can also use these kinds of algorithms to optimize our linear classifier such as Logistic Regression and linear Support Vector Machines.

There are three well known types of gradient decent:.

1) Batch gradient descent.
2) Stochastic gradient descent.
3) Mini-batch gradient descent.

- Batch gradient descent computes the gradient using the whole dataset to find the minimum located in it's basin of attraction.
- Stochastic gradient descent (SGD) computes the gradient using a single sample.
- Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples.

SGDClassifier is usually used along with GridsearchCV() to tune all the parameters of the model and the best model for your data-set is returned with the parameters that gave the best model. All the parameters used in the training are listed in Fig. 2.

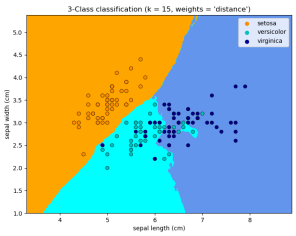
```
from sklearn.linear_model import SGDClassifier
s = SGDClassifier()
s.get_params()
```

```
{'alpha': 0.0001,
 'average': False,
 'class_weight': None,
 'early_stopping': False,
 'epsilon': 0.1,
 'eta0': 0.0,
 'fit_intercept': True,
 'l1_ratio': 0.15,
 'learning_rate': 'optimal',
 'loss': 'hinge',
 'max_iter': 1000,
 'n_iter_no_change': 5,
 'n_jobs': None,
 'penalty': 'l2',
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'tol': 0.001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

(6) K-Nearest Neighbor (KNN):

The KNN algorithm assumes that similar things exist in close proximity with each other. In other words, similar things are near to each other. The Fig. 2 shows the well known

example of iris flowers and how 'versicolor' and 'virginica' are similar and 'setosa' is very different from the others.



KNN uses this assumption of similar points being closer together for classification.

working of KNN:

1) Initialize K to your chosen number of neighbors.
2) Calculate the distance between the query example and the current example from the data.
3) Add the distance and the index of the example to an ordered collection.
4) repeat step 2 and 3 for every example in the data-set.
5) Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances.
6) Pick the first K entries from the sorted collection
7) Get the labels of the selected K entries.
8) If regression, return the mean of the K labels.
9) If classification, return the mode of the K labels.

Choosing the right value of K is very important in KNN. To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before. KNN can be used for classification as well as regression.

K stands for the number of data set items that are considered for classification. K-nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure in this case distance between closely grouped points.

(7) K Means Algorithm :

The K Means algorithm is an iterative technique that attempts to classify a dataset into K separate non-overlapping subgroups (clusters), each of which contains only one data point. It attempts to make intra-cluster data points as near as possible while maintaining clusters as distinct as possible. It distributes data points to clusters in such a way that the sum of the squared distances between them and the cluster's centroid is as small as possible. Within clusters, the less variance there is, the more homogenous the data points are.

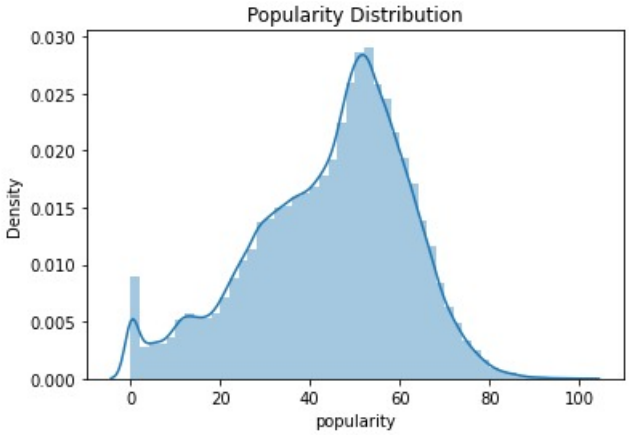
The following is how the k means algorithm works:

1) k is the number of clusters to specify.
2) Initialize the centroids by picking K data points at random for the centroids without replacing them.
3) Continue iterating until the centroids do not change. i.e. the clustering of data points does not change.
4) Calculate the total of all data points squared distances from all centroids.
5) Assign each data point to the cluster that is closest to it (centroid).
6) Calculate the cluster centroids by averaging all of the data points that correspond to each cluster.

C. Implementation Details

Data Preparation: We initially looked for any missing or null values in our data-set. We found out there were no missing or null values in our data-set. We performed data cleaning on our data-set to remove duplicate values, special characters in track name and artist names. we also converted some string valued features like "mode" and "key" to integer values. we have also converted the "popularity" feature from continuous to discrete making it our target variable.The figure below shows the data-set after cleaning and preparation.

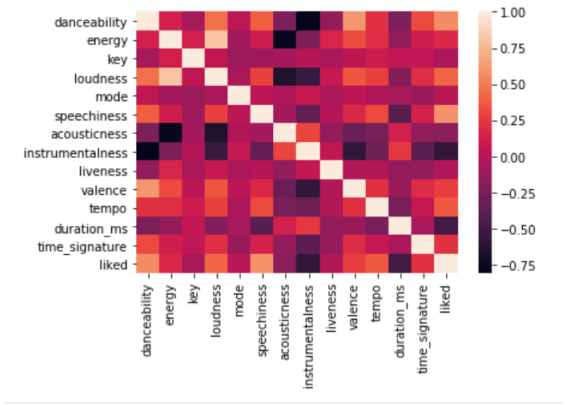
Further we converted our target variable = "popularity" that was in a continuous range of integers (0-100) to discrete two values.



For that we plotted density graphs for popularity and we found that highest density was observed when popularity value was above 50 thus, popularity whose value was above 50 was given value of 1 and rest where given value of 0. We then looked for duplicated data entries in our data set, and removed all the duplicate rows. Additionally, for ease of computation we converted many character values to numerical values like in attribute column mode. Mode contains two values Major and Minor. We label all major values as 1 and all minor values as 0. Moreover, there were some special characters in artist name and track name, we got rid of those characters too. The figure below shows the data-set after cleaning and preparation.

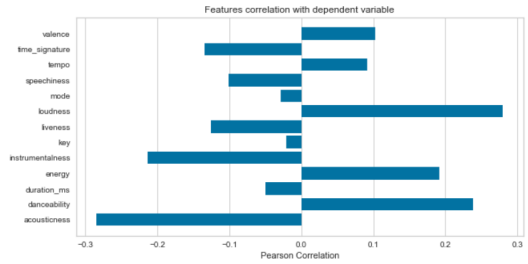
	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	mode	key	time_signature	speechiness	tempo	valence
112081	Darell	Vamo Alla	1	0.2420	0.863	237227	0.748	0.000273	0.116	-4.480	0	7	0	0.1780	89.958	0.838
7174	Foster The People	Sit Next to Me - Stereotypes Remix	1	0.1730	0.680	190720	0.738	0.000360	0.370	-4.881	0	5	0	0.0331	105.015	0.675
151695	Cody Johnson	Never Go Home Again	1	0.4340	0.429	252147	0.675	0.000000	0.186	-5.296	0	9	0	0.0463	168.003	0.418
130528	Alexis y Fido	Donde Estés Llegaré	1	0.0661	0.829	228813	0.904	0.000002	0.343	-3.510	0	7	0	0.0535	94.021	0.595
119341	Tommy Ryman	Divorce and Lesbians	0	0.8960	0.674	151144	0.582	0.000000	0.794	-8.378	0	3	0	0.9240	77.215	0.681

For features importance we plotted correlation heat-map.



As seen in Figure above, acoustic, loudness, instrumentalism have the highest correlation.

We then use Pearson correlation plot to visualize the association of all predictor variables with dependant variable.



As we can see valence, tempo, loudness, energy and danceability are all positively correlated with popularity. wheras time, speechiness, mode, liveness, key, instrumentalness, duration and acousticness is negatively correlated with popularity. We don't have a feature in our dataset which is not linearlly correlated with our target variable 'popularity'.

- $ContentBasedRecommendation$:

1. Naive Bayes: The first algorithm we implemented was Naive Bayes. We used sklearn's GaussianNB to implement our model. We normalized the training data with standardscaler() before training the model on training data. We split the dataset into 80% training and 20% test and setting 'Popularity' as our target varibale. Then we evaluated the model on test set using sklearn metrics's accuracy_score() method. We got accuracy of 71% for Naive Bayes model which is pretty good for a start. We also see that Naive Bayes outperforms KNN classifier and Decision Tree Classifier.

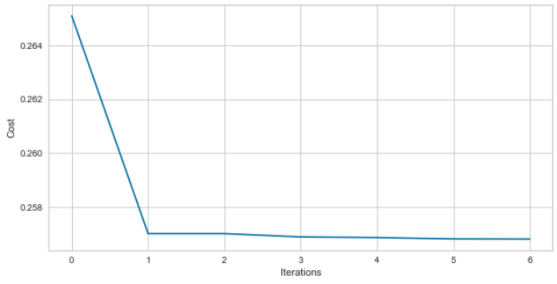
2. Decision Tree Classifier: The second algorithm we implemented was Decision Tree classifier. We used sklearn's DecisionTreeClassifier() to implement this. After evaluation we found the accuracy of Decision Tree Classifier to be 68%. Which was not good enough as we got an accuracy of 71% with Naive Bayes. So we move on to our next model.

3. K nearest Neighbours Classifier: We implemented K Nearest Neighbour using Sklearn's KNeighborsClassifier() model. Using 'Popularity' as our target variable with two classes '0' and '1' where 0 means the track is unpopular and 1 means the track is popular. This models gave us an accuracy of 68% same as Decision Tree. But we had reduced F1_score of 28%. Which tells us that K Nearest Neighbour was not able to identify the distinct classes in the dataset.

4. extreme Gradient Boosting (XGBoost): We then moved on to a much popular machine learning model XGBoost. We used Xgboost's XGBClassifier() model for our project with binary: logistic as the objective function. we ran this model with n_estimator = 10. After evaluation we got an better accuracy of 75% Thus we finally outperform Naive Bayes model by using Gradient Boosting. But our F1_score of 32% was still less meaning the model was not able to distinguish between the classes of our model and was more biased toward one particular class. So we move onto our next model.

5. Random Forest Classifier (RFC): We implemented Decision tree for our model but failed to get a good model accuracy but we got a good F1_score for Decision tree so we decided to use Random Forest Classifier as improvement to Decision Tree for better accuracy as well as F1_score.We used sklearn's RandomForestClassifier() to implement this model. After implementation and evaluation we got an accuracy of 75% which was similar to what we got in XGBoost but unlike XGBoost we got a better F1_score with RFC which was 40%. So RFC outperforms XGboost in terms of distinguishing the two classes [0,1] in our model. Our accuracy is still not that great so we move on to our next model to improve our model further.

6. Stochastic Gradient Descent: We thought of using optimization techniques to further improve our model so we decided to implement SGDClassifier.We used sklearn's SGDClassifier with SVM linear model for our problem. We also use GridSearchCV to tune our model for optimal learning rate with a range of {"alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]} and found the optimal value of learning rate to be 0.001. And the reported accuracy for this model was 88% which is highest among all other models implemented. Thus we improved the accuracy of our model by using optimization technique.



The iterations vs cost plot shows how SGD optimizes the model by updating cost at every iteration and thus result in improved model in less time.

- $PopularityBasedRecommendation$:

To Implement Popularity Based recommendation we used cosine similarity. Cosine similarity computes angle (cos theta) between two vectors. Smaller the angle between two vectors greater is its cosine. so any two similar items will have greater cosine value.

$$cosine(x, y) = \frac{x \cdot y^T}{||x|| \cdot ||y||}$$

For our project we create a vector of popular songs from our data-set then split them into training and testing sets. then we compute the similarity between previously heard and liked songs by user and the test set(assuming that it includes songs that were not heard by the user). Then we sort the similarity vector in descending and use the top 5 tracks as the most similar to previously heard popular tracks as recommendation to the user.

```
Top 5 new tracks for users with their similatirity to popular songs :

1. Powers  ,Target: 1

2. On And On  ,Target: 1

3. Two Tickets to Paradise  ,Target: 1

4. Tricks on Me  ,Target: 1

5. I Wanna Love You - Album Version (Edited)  ,Target: 0
```

As shown in the figure above we can see the recommended tracks and its true target value, 1 being popular and 0 being unpopular like we explained in data preparation. Here we see that cosine similarity helps us recommend tracks that were not previously rated by using similarity between previously rated songs with unrated new songs.

- $Collaborative filtering Based Recommendation$:

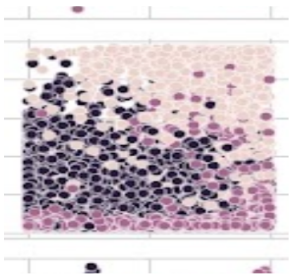
To implement collaborative filtering based Recommendation system we have used K Means algorithm. To implement K Means, We have first used PCA (Principal component analysis) to reduce the dimensionality of the data. Dimensionality Reduction is simply reducing the number of features (columns) while retaining maximum information. PCA is a linear dimensionality reduction technique which converts a set of correlated features in the high dimensional space into a series of uncorrelated features in the low dimensional space. These uncorrelated features are also called principal components. PCA is an orthogonal linear transformation which means that all the principal components are perpendicular to each other. It transforms the data in such a way that the first component tries to explain maximum variance from the original data. Our input features contain 10 attributes, we reduce it to 5. As we reduced the data, the data value was going into exponential form. Thus, we used ML lib. To normalize the data using the StandardScalar() function. After normalization, to find the ideal number of clusters (K) values, we use hyper parameter tuning. We run the K Means algorithm by initializing the value of K from 2 to 15 and to find the highest score we used silhouette score. We found out that when K=3, our score was highest (0.60).

The degree of separation between clusters can be determined via silhouette analysis:

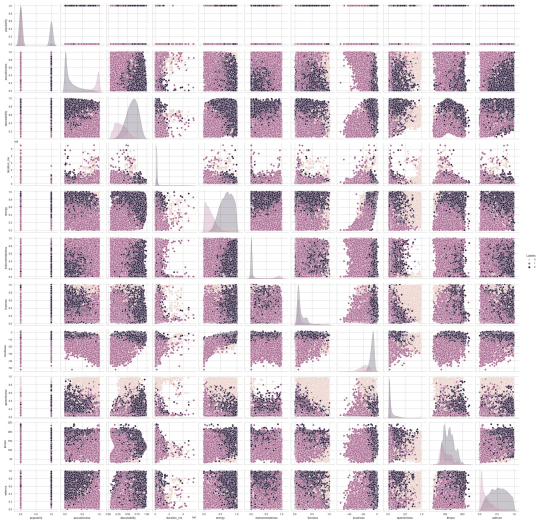
1) Calculate the average distance between all the data points in a cluster (ai).
2) Calculate the average distance between all data points in the cluster nearest to you (bi).
3) Compute the coefficient using.

$$\frac{b^i - a^i}{max(a^i, b^i)}$$

The coefficient can take values in the interval [-1, 1]. If it is 0, the sample is very close to the neighboring clusters. If it is 1, the sample is far away from the neighboring clusters. If it is -1, the sample is assigned to the wrong clusters. Therefore, we want the coefficients to be as big as possible and close to 1 to have a good cluster. As our score was 0.6 which is nearest to 1, our model is pretty good.



By zooming into our pair plot for speechiness vs popularity, we can see that we can distinctly identify 3 clusters.



IV. Comparison

The comparison of all the models with respect to Accuracy and F1_score is shown in the table below.

	Model	Accuracy	F1_Score
5	SGDClassifier	0.886953	0.779912
0	RandomForestClassifier	0.756132	0.406650
3	XGBClassifier	0.751487	0.324076
4	NaiveBayes	0.715059	0.626741
1	KNeighborsClassifier	0.683659	0.280199
2	DecisionTreeClassifier	0.682814	0.423579

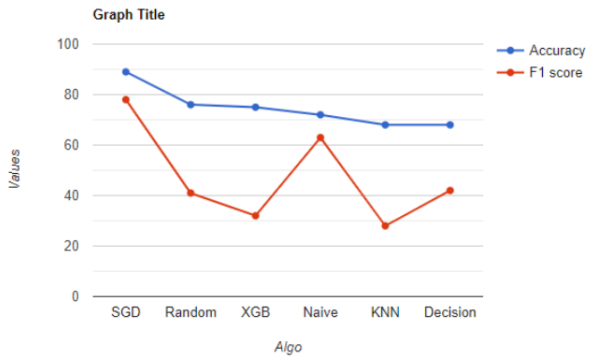
Compared to other algorithms Decision Tree requires less efforts for data preparation during pre-processing. A decision tree does not require any normalization of data and does not require scaling of data also. The main disadvantage of decision tree is that a small change in the data can result in the large change in the structure of the tree. Even the calculation are far complex when compared with other algorithms. The computational cost of Decision tree is O(n*log(n)).Random Forest algorithm is less prone to over fittings as compared to Decision Tree and other algorithms. It also outputs the importance features which is very useful.The main disadvantage of

Random Forest is that the algorithm may change considerably when small change in data. The computational cost of Random Forest is O(n*log(n)). When assumption of independent predictors holds true, a Naive Bayes classifier performs better as compared to other models.Naive Bayes requires a small amount of training data to estimate the test data. So, the training period is less.Main limitation of Naive Bayes is the assumption of independent predictors. Naive Bayes implicitly assumes that all the attributes are mutually independent. In real life, it is almost impossible that we get a set of predictors which are completely independent. The computational cost of Naive Bayes is O(n). The main advantage of SGD is that it is easier to fit in the memory due to a single training example being processed by the network.It is computationally fast as only one sample is processed at a time.Due to frequent updates, the steps taken towards the minima are very noisy. This can often lean the gradient descent into other directions.Also, due to noisy steps, it may take longer to achieve convergence to the minima of the loss function. The computational cost of SGD is O(knp). KNN stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training.Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.The computational cost of KNN is O(n).XG Boost as compared to other algorithms is fast and efficient. The ateliers have a minimal impact and the large size data sets are handled well. The disadvantage is that the visualization is tough. There might be chances of over fitting if the parameters are not tuned properly.The computational cost of XG Boost is O(npn).

V. Future Directions

This project can be further extended in many ways. For instance since we have developed a Spotify Recommendation System we could use this information to suggest to artist that what type of the features make the song popular. Using this insight the artists can keep making Hit songs!. Further in this project Hybrid type of filtering techniques can be used to improve recommendations. Hybrid filtering technique consist of both collaborative filtering and content based filtering, that uses advantages of both method and suppresses its disadvantages.

VI. Conclusion



The purpose of this project was to be able to implement various Machine Learning algorithms and compare their performance. In this research we implemented and evaluated performance of seven Machine Learning algorithms as shown in the graph above. We come to the conclusion that SGDClassifier performed best with our data-set in classifying popular and unpopular songs. We also explored all three methods of Recommendation System (Content Based, Collaborative filtering and Popularity Based) that is applicable in music recommendation. For future work we propose to use Hybrid models (combination of content based and collaborative filtering methods) to further improve the accuracy of the music recommendation system.

References

[1] Whitell, Ryan, "Content-Based Music Recommendation using Deep Learning" (2019). All Regis University Theses.

[2] Chou, Szu-Yu, Yi-Hsuan Yang, Jyh-Shing Roger Jang and Yu-Ching Lin. "Addressing Cold Start for Next-song Recommendation." Proceedings of the 10th ACM Conference on Recommender Systems (2016)

[3] Bogdanov, Dmitry Haro, Martin Fuhrmann, Ferdinand Gomez, Emilia Herrera, Perfecto. (2010). Content-based music recommendation based on user preference examples.

[4] Xiaoyuan Su Taghi M. Khoshgoftaar. "A Survey of Collaborative Filtering Techniques"(2009).

[5] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13). Curran Associates Inc., Red Hook, NY, USA, 2643–2651.

[6] Malcolm Slaney, Kilian Q. Weinberger, and William White. Learning a metric for music similarity. In Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR), 2008

[7] Music Recommendation System Spotify - Collaborative Filtering by Mithun Madathil, July 5, 2017

[8] A Survey on Recommendation System for Bigdata using MapReduce Technology (ICCMC 2019)

[9] Mobile Recommendation System Using Mapreduce (ICCES 2021)

[10] Research on Collaborative Filtering Algorithm Based on MapReduce. (2016 9th International Symposium on Computational Intelligence and Design)

[11] Scalable Collaborative Filtering Recommendation Algorithm with MapReduce (2014 IEEE)

[12] Simulation of Genre based Movie Recommendation system using Hadoop MapReduce Technique. (ICECDS-2017)

[13] Building Hybrid Recommendation System Based on Hadoop Framework (ICEEOT) - 2016

[14] Recommender System using Hybrid Approach (ICCCA2016)