Predicting Popularity of Songs Using Spotify Data

Ritu Bahuguna

Problem Statement

- *Spotify* is a Sweden based audio streaming platform.

- *Spotify* has audio tracks that are purchased from record/label producers and independent artists (β version)

- Operates on Freemium model.

- *Our business problem:*

  <u>'Predicting a track as Popular or Not Popular on Spotify'</u>

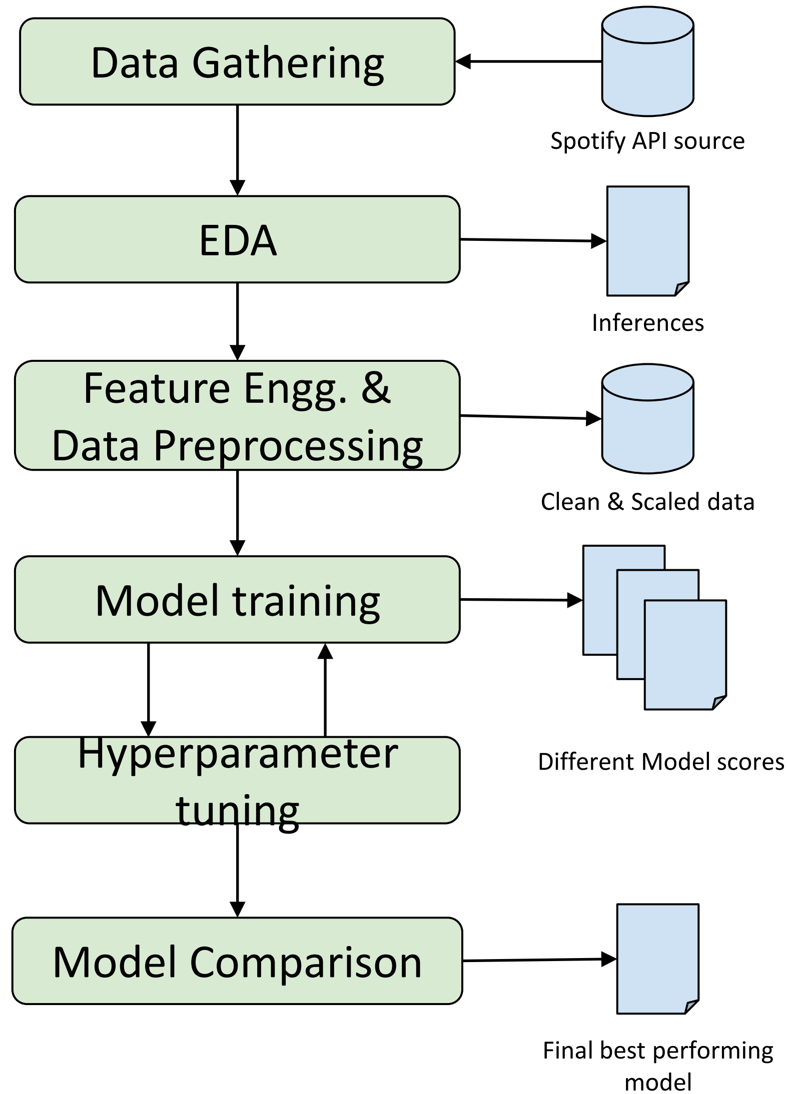
  - Our objective is to build a predictive model that can be used by *Spotify* to predict whether a track will be popular among users after its release ("Hit") or not ("Not Hit").

How will this help?

- Economize the royalties paid to the Labels/Production houses.
    - *"Save money by making better deals for popular songs."*
- Filter new songs for *Spotify* for Artists ™.
    - *"Filter new songs just by prediction, without manual listening"*
- Increase discoverability for a new to-be-hit song.
    - *"Let user reach quickly to to-be-hit predicted song over Spotify"*
- To cluster/group old songs on Spotify.
    - *"Generate new playlists like popular-in-70s, popular-in-80s etc"*

Dataset Information and Procedure

- *Spotify* provides Web-API[1] using which track/song information can be crawled.
- We have used dataset crawled from April 2019 using the API shared on Kaggle[2].
- Data set info:
  - CSV format
  - 1,30,326 tracks
  - Audio features[3] such as acousticness,danceability,loudness, tempo etc
- Other data set considered: Million song dataset[4] -
  - 41 Features
  - 1 Million songs
  - 280 GB , H5 file format

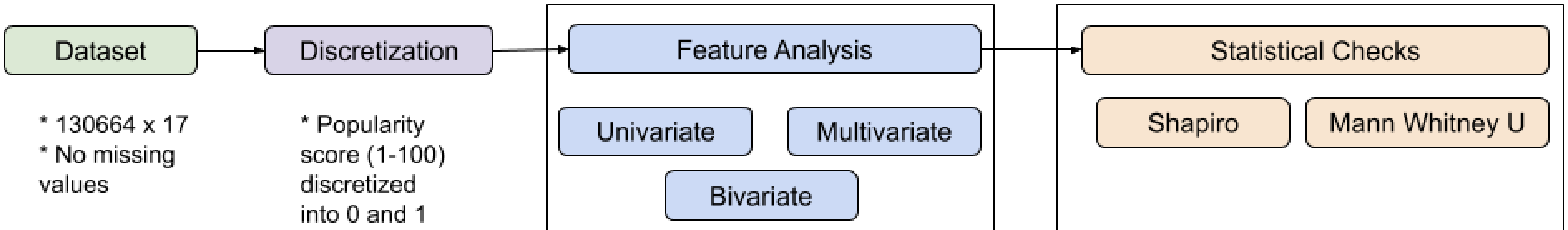


[1] https://developer.spotify.com/documentation/web-api/

[2] https://www.kaggle.com/tomigelo/spotify-audio-features

[3] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

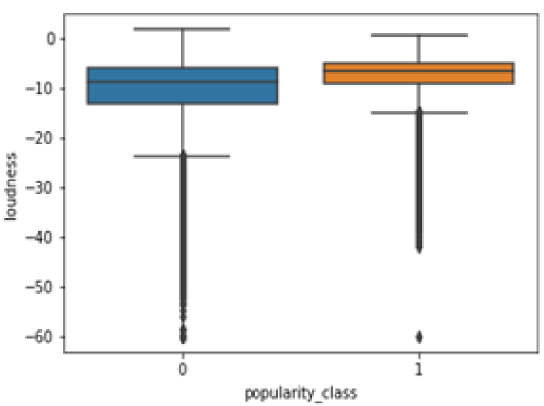
Exploratory Data Analysis

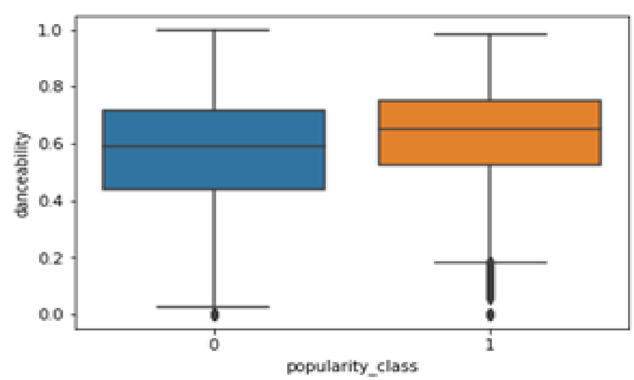


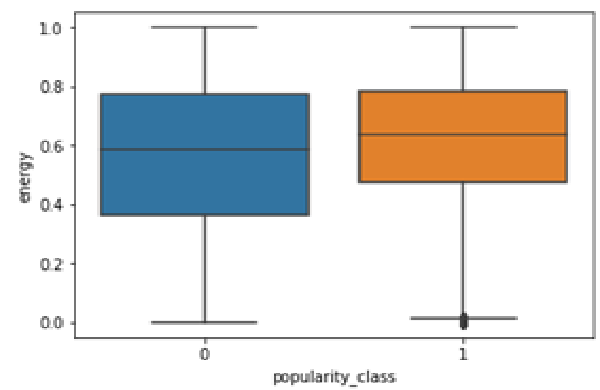
* Analyzed and conclusion drawn with visualizations.

* Conclusion : All the variables are indispensable for model building.

Three most prominent features:







Loudne
ss

Danceabilit
y

Energy

Prominent Features

Energy : measure of intensity and activity, value ranges between 0 to 1

Highest percentage(28.58%) of the popular tracks lie within a range of 0.603 - 0.775.

The most popular, energy value is 0.666, 248 songs have this energy value,of which 175 of those songs are not popular and 73 are popular.

Danceability : Suitability of a track is for dancing , ranges between 0 to 1 .

Maximum chance for a track being popular on basis of danceability is when it falls in range 0.605-0.727. 27% of songs in this range are popular, which is highest in amongst Quartile range

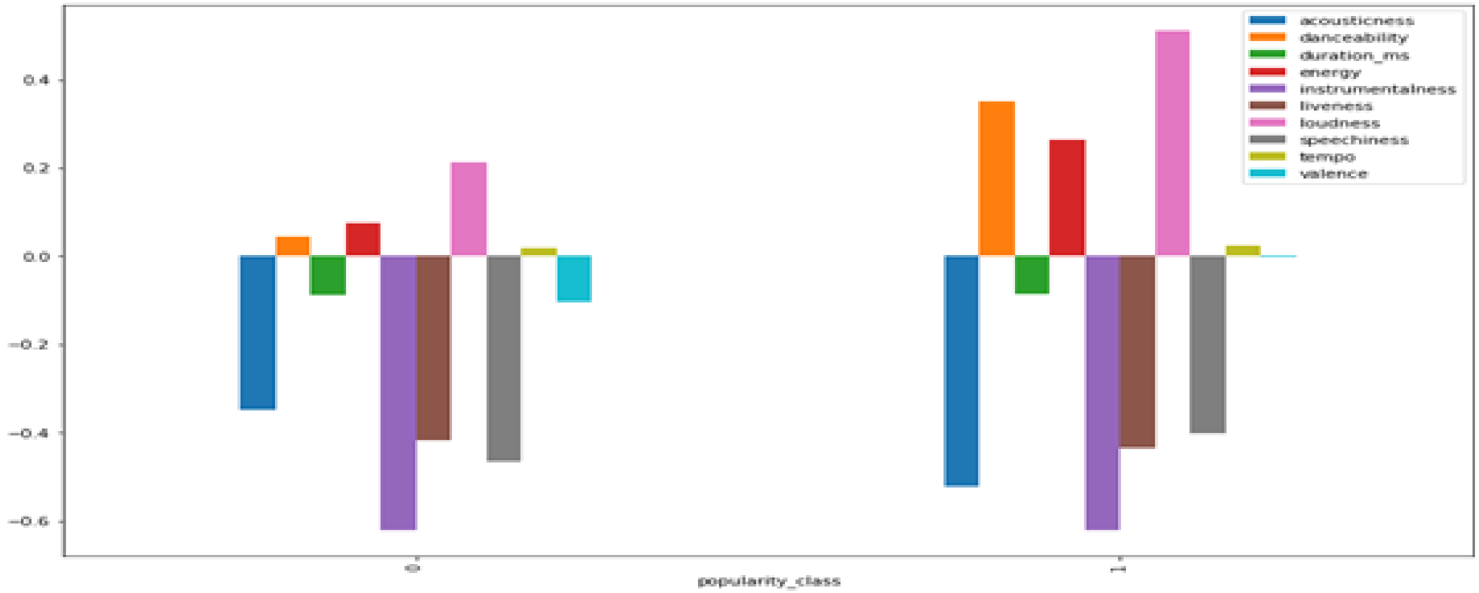
Loudness: overall loudness of a track in decibels (dB), range between -60dB and 0dB.

Highest chance for a track being popular on basis of loudness is when it falls in range -5.684 to 0 db. 34% of songs in this range are popular, which is highest in amongst Quartile range

Multivariate Analysis and Observations

Inferences for Popular songs:

- have higher values of *danceability*, *loudness*, *energy* and *tempo*.
- are less *acoustic*.
- are less popular with increase in *liveliness.*
- are 'happy' than 'sad'. *[f:Valence]*



Indetermental: *Tempo, Duration, Instrumentalness, Speechiness* were identified as weak features for popularity as the spread of data for these features was similar for both classes.

Feature Engineering

- Categorical Feature (Artist name) was converted to numerical variable using the concept of supervised ratio.

- A feature generating mechanism where each Artist name is represented as ratio of popular tracks by total tracks of that artist

$$SR_i = \frac{P_i}{N_i + P_i}$$

- Eg: An artist(YG) was represented as 0.75 (15 popular songs over 20 total songs).

- Highly correlated feature with 0.8511 correlation coefficient with *Popularity*.

Data Preprocessing

* We applied RFECV to find the feature importance for model building and VIF Factor to check the presence of multicollinearity.

RFECV Ranking	Feature
1	('popularity_class', 'mean')
2	speechiness
3	danceability
4	instrumentalness
5	liveness
6	energy
7	acousticness
8	loudness
9	Valencence
10	time_signature
11	mode
12	key
13	tempo
14	duration_ms

	VIF Factor	features
0	3.84	acousticness
1	15.77	danceability
2	4.02	duration_ms
3	16.19	energy
4	2.07	instrumentalness
5	3.17	key
6	2.65	liveness
7	8.56	loudness
8	2.60	mode
9	2.08	speechiness
10	15.98	tempo
11	41.58	time_signature
12	5.53	valence
13	4.79	popularity_class
14	5.30	(popularity_class, mean)

Algorithms Considered

Algorithm	Strength	Weakness
Logistic Regression	* Easy to interpret, efficient implementation	* Ineffective with non-linear boundaries.
Decision Tree	* Can handle non-linear features. * Gives importance to strong features.	* High bias towards training dataset. May overfit.
Random Forest	* Reduces drawbacks of DT by bagging.	`* Complex to interpret and explain.
AdaBoost	* Able to learn difficult samples by assigning them weights	*Complex and may require more data to train weak learners.
KNN	* Fastest to train.	* Equal weightage to each feature.

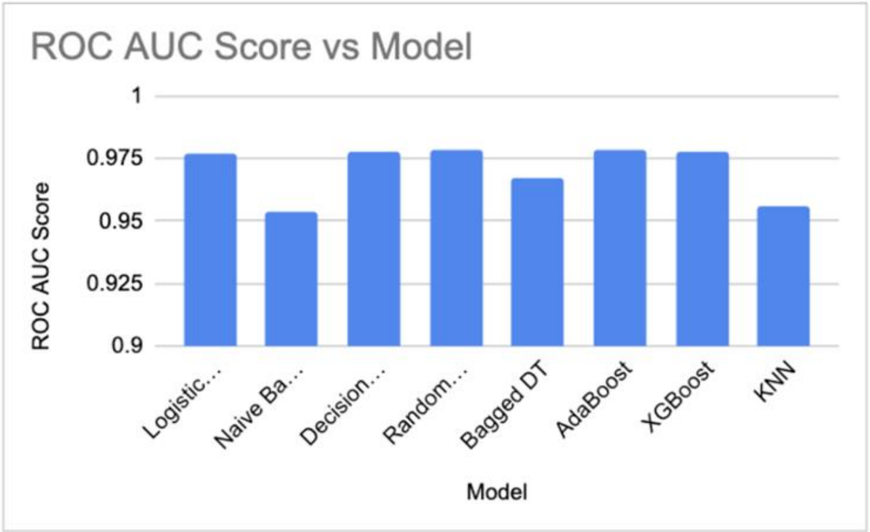
Solution Architecture

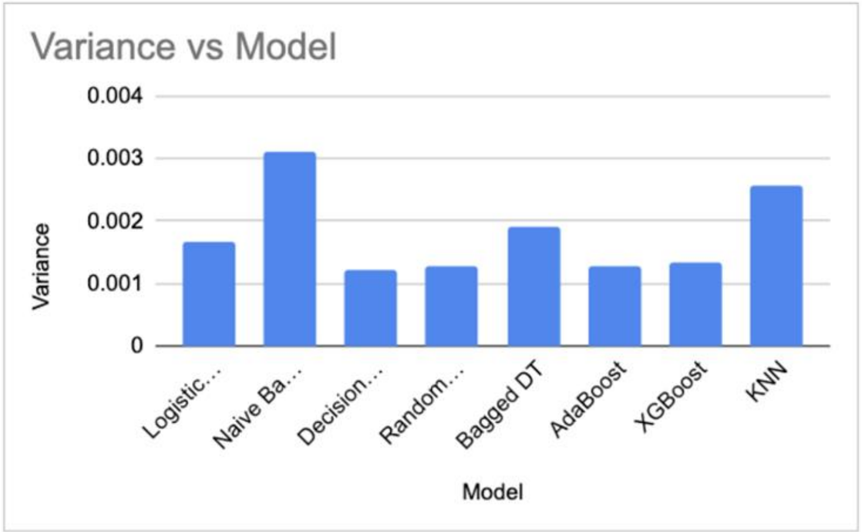
Platform	Python on Jupyter Notebook
Libraries Viz Libraries	Pandas, Numpy, Sklearn (ML Toolkit) Seaborn, Matplot
Hardware	Intel Pentium Core i5 8th Gen/8GB/500GB

Procedure	* CSV operations using Pandas.
	* EDA using Pandas, Numpy
	* EDA visualizations using Seaborn and Matplotlib
	* Feature Engg. done on Pandas dataframe.
	* Data preprocessing using Sklearn.StdScalar
	* ML Models using Sklearn toolkit:
	`-KNeighborsClassifier`
	`-GaussianNB`
	`-LogisticRegression`
	`-DecisionTreeClassifier`
	`-RandomForestClassifier`
	`-BaggingClassifier`
	`-AdaBoostClassifier`
	
	* Model Hyperparameter tuning using:
	`-GridSearchCV`
	
	* Metrics using Sklearn.metrics:
	`- roc_auc_score`
	* Cross Validation using:
	`- cross_val_score`
	`- KFold (splits = 10)`

Results

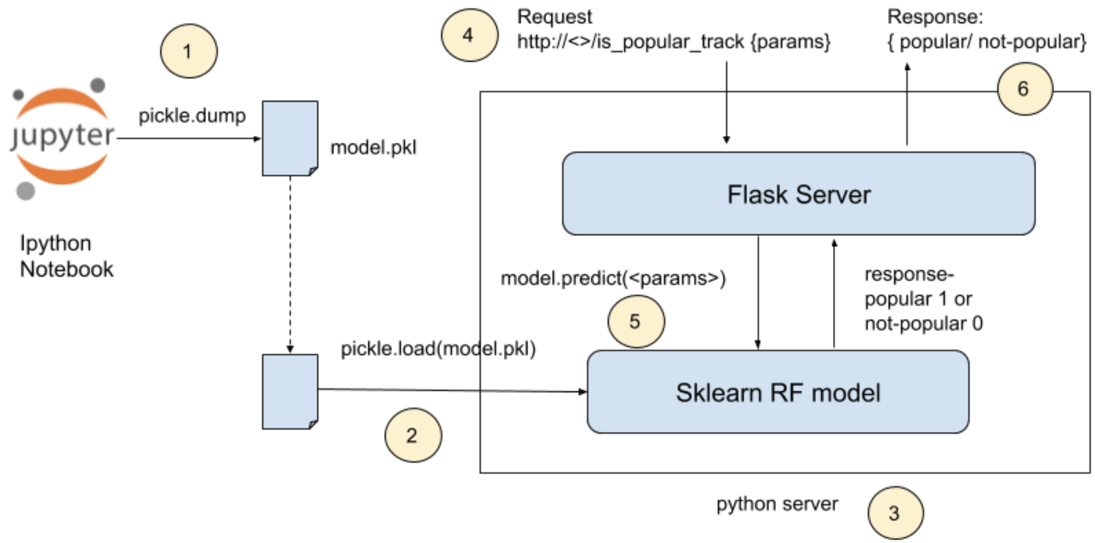
Model	Hyperparameter (with values)	ROC AUC Score	Variance
Logistic Regression	*default*	0.9766691661	0.001670378457
Naive Bayes	*default*	0.9534619438	0.003125650566
Decision Tree	`criterion='entropy',max_depth=5`	0.9779546699	0.001217149749
Random Forest	`n_estimators=573,criterion='entropy'`	0.9785095052	0.001277885704
Bagged DT	`n_estimators=10`	0.9671752292	0.001908651161
AdaBoost	`n_estimators=50`	0.9784513987	0.001277365226
XGBoost	`n_estimators=2600`	0.9777651716	0.001353769988
KNN	`n_neighbors=11,weights='distance'`	0.9558930322	0.002562895114





https://docs.google.com/spreadsheets/d/13m73GMB1wDjQtZWhrP52FJsPr2m1ZGifRtBuftL9X7g/edit?folder=0AFsfQRqEVKbiUk9PVA#gid=310686087

Model To Production



1. First we *dump* our best model (RF) into .pkl file using Pickle[1] library.
2. Then in a different python file, we *load* our model using previous saved file.
3. We then import Flask[2] and Sklearn[3] library in our python file, connect them and start the server.
4. We get request to flask server from browser.
5. We fire *model.predict(<parameters>)* on our model and get the results.
6. Results are then returned back to the browser on the same request.

[1]https://docs.python.org/3/library/pickle.html
[2] https://palletsprojects.com/p/flask/
[3] https://scikit-learn.org/stable/

Conclusion And Way Forward

Conclusion:

- Random Forest is the best performing predictive model for the assigned problem statement.
- Similar performance from AdaBoost, but we choose Random Forest as it is a simpler model than AdaBoost.
- With the best model, we were able to predict "popular" and "not-popular" track with high accuracy and low variance.

Limitations:

- If a new artist is introduced without any past hits in the dataset, model will be unable to predict as it cannot encode the artist name.
- Model will not be able to perform in a different environment other than the training one. Eg: Introduction of new features

Way Forward:

- More features can be crawled from the *Spotify API* example number of followers, genre of the song ,etc to increase/optimize model performance.

Thanks!