Predicting Song Popularity on Spotify

By Anantha Rao, Xingrong Chen, and Desmond Fung

Introduction

- With the increasing prevalence of streaming platforms for music, learning about the music industry becomes more and more about listener behavior on these platforms.

- Our project involves a data set from spotify with 19,000 songs and 13 features related to the song quality(tempo, key, danceability, runtime,etc..) as well as a rating of how popular the song is on spotify(related to the number of streams).

Motivation/Goals

- These questions may offer insight for record labels and artists in deciding to publish songs. It may also reveal patterns in human music consumption that are interesting for other reasons.

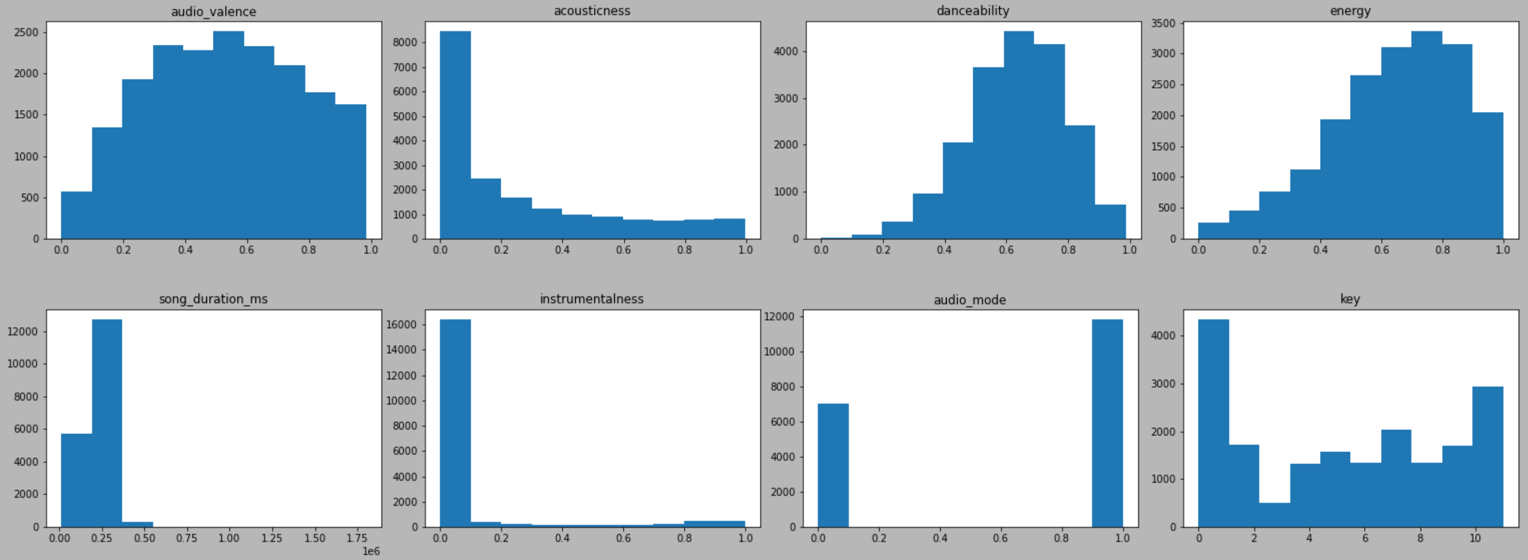
- (How well) Can we predict whether a song will be popular? What are the acoustic qualities of a song that make it popular?

Our Dataset

- We collected the data from website https://www.kaggle.com/edalrami/19000-spotify-songs?select=song_data.csv
- 14 feature with 18835 observations
- 1 feature measure the popularity of songs, other features describe the characterization of songs like song duration, acousticness, loudness, tempo, etc.
- 3 categorical variables (like keys, time signature and audio mode) and 11 numeric variables

	song_name	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_mode
0	Boulevard of Broken Dreams	1	262333	0.005520	0.496	0.682	0.000029	8	0.0589	-4.095	1
1	In The End	0	216933	0.010300	0.542	0.853	0.000000	3	0.1080	-6.407	0
2	Seven Nation Army	1	231733	0.008170	0.737	0.463	0.447000	0	0.2550	-7.828	1
3	By The Way	1	216933	0.026400	0.451	0.970	0.003550	0	0.1020	-4.938	1
4	How You Remind Me	0	223826	0.000954	0.447	0.766	0.000000	10	0.1130	-5.065	1

Feature distribution



Preprocessing

- How to define popularity?

  If a song’s popularity measurement is greater than 80% songs in our dataset, then we treat it as a ‘popular’ song. Otherwise, the song is ‘unpopular’

- What are we trying to predict?

  We try to use features except ‘song_popularity’ in the dataset to identify whether a song is popular or not by machine learning algorithm.

Preprocessing

- Splitting the 85% of original data set as train set while the rest are test set.

- Keep the distribution of popular song in two subsets are the same as the original dataset by using stratify method

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
                                                    shuffle=True,random_state=123, stratify=y)
```

Models

- Logistic Regression
- kNN
- Decision Tree
- Random Forest

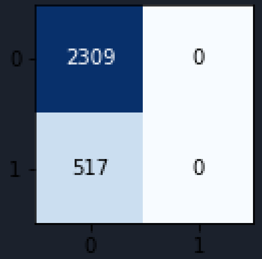
Logistic regression

- Since after our preprocessing, we only have the song with ‘popular’ label and the song with ‘unpopular’ label. So we treat it as a binary data, and try to use logistic regression model to estimate the probability of a song will be popular, using the default threshold in sklearn, if the estimated probability greater than 0.5, then the model will predict the song is popular

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_{13} x_{13}$$

Result

- The logistic model predict all the songs are unpopular
- All the songs has less 0.5 probability to be popular



- Since the prediction given by logistic model is not quite useful, so we look at other result given by this model.

```
Coefficients:
                    Estimate Std. Error z value Pr(>|z|)    
(Intercept)       -8.947e+00  1.002e+02  -0.089  0.92882    
song_duration_ms   7.014e-07  3.717e-07   1.887  0.05917 .  
acousticness      -6.825e-01  1.031e-01  -6.618 3.65e-11 ***
danceability       1.744e+00  1.571e-01  11.102  < 2e-16 ***
energy            -1.815e+00  1.869e-01  -9.713  < 2e-16 ***
instrumentalness  -3.399e+00  2.683e-01 -12.669  < 2e-16 ***
key1               1.975e-01  7.805e-02   2.531  0.01139 *  
key10             -7.988e-03  9.421e-02  -0.085  0.93243    
key11              4.872e-02  8.728e-02   0.558  0.57675    
key2              -2.388e-01  9.055e-02  -2.637  0.00838 ** 
key3              -4.587e-01  1.580e-01  -2.903  0.00369 ** 
key4              -1.394e-01  9.768e-02  -1.427  0.15353    
key5              -5.533e-02  8.978e-02  -0.616  0.53771    
key6               7.206e-02  9.103e-02   0.792  0.42861    
key7              -2.550e-01  8.614e-02  -2.960  0.00307 ** 
key8              -1.564e-01  9.345e-02  -1.673  0.09428 .  
key9              -1.811e-01  9.013e-02  -2.010  0.04447 *  
liveness          -2.929e-01  1.452e-01  -2.017  0.04373 *  
loudness           1.773e-01  1.105e-02  16.049  < 2e-16 ***
audio_mode1       -8.382e-02  4.235e-02  -1.979  0.04777 *  
speechiness        5.531e-02  1.969e-01   0.281  0.77875    
tempo              5.438e-04  7.291e-04   0.746  0.45569    
time_signature1    9.030e+00  1.002e+02   0.090  0.92816    
time_signature3    9.254e+00  1.002e+02   0.092  0.92638    
time_signature4    9.380e+00  1.002e+02   0.094  0.92538    
time_signature5    9.100e+00  1.002e+02   0.091  0.92760    
audio_valence     -8.346e-01  9.599e-02  -8.695  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 17931  on 18834  degrees of freedom
Residual deviance: 16452  on 18808  degrees of freedom
AIC: 16506

Number of Fisher Scoring iterations: 10
```

KNN

- Our group also attempted to utilize K -nearest neighbors (KNN) to predict song popularity

- Advantages
  - The algorithm is simple and easy to implement.
  - There’s no need to build a model, tune several parameters, or make additional assumptions.
  - The algorithm is versatile and can used for both classification, regression

KNN

```
#KNN prediction

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
prediction = knn.predict(x_test)
print('With KNN test accuracy is: ', knn.score(x_test,y_test))
```

```
With KNN test accuracy is:  0.8163481953290871
```

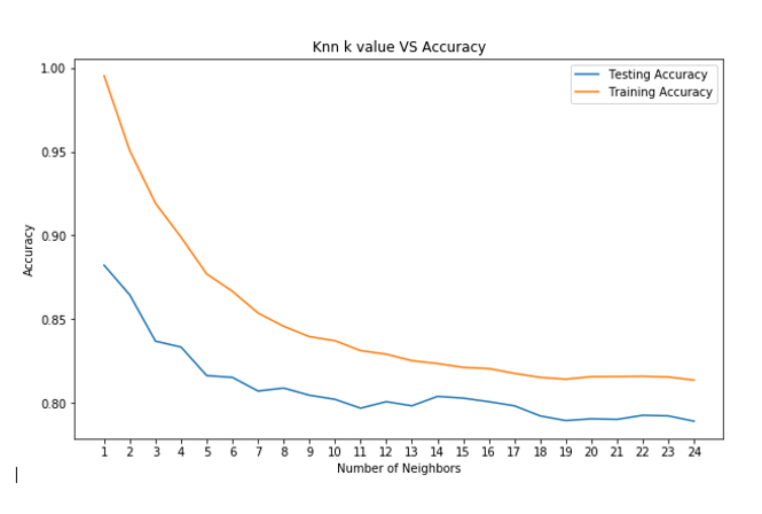


Figure with KNN performance on test set

Potential drawback: Curse of Dimensionality

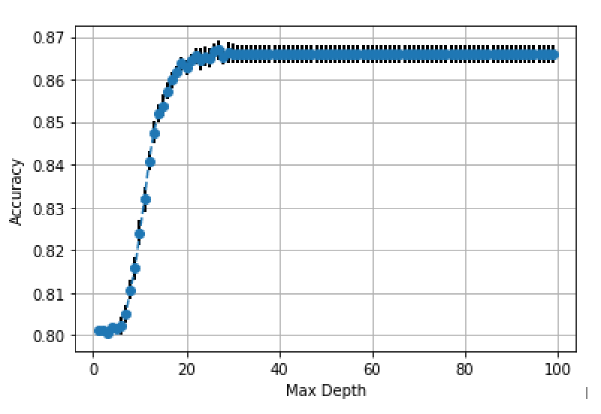
- becoming significantly slower as the size of the data grows
- take a large portion of the hypervolume into consideration to find k nearest neighbor

Decision Trees

- The Decision Tree model was build to predict song popularity based on the given features
- Set max_depth = 15
- Model performed better than KNN, logistic regression perform the worst
- Accuracy tends to stabilize as max_depth grow larger than 20

```
from sklearn.tree import DecisionTreeClassifier


dt= DecisionTreeClassifier(random_state = 123, max_depth = 15)
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
DecisionTree_score=dt.score(x_test,y_test)
print("Train ccuracy of decision tree:",dt.score(x_train,y_train))
print("Test accuracy of decision tree:",dt.score(x_test,y_test))
```

```
Train ccuracy of decision tree: 0.9592729089886939
Test accuracy of decision tree: 0.8598726114649682
```



Decision Tree Performance with different max_depth

Ensemble Methods

Methods explored:

- Bagging
- Random Forests

Bagging

We carried out bagging with 500 Decision trees with the same depth as our base DT estimator before(depth = 15).

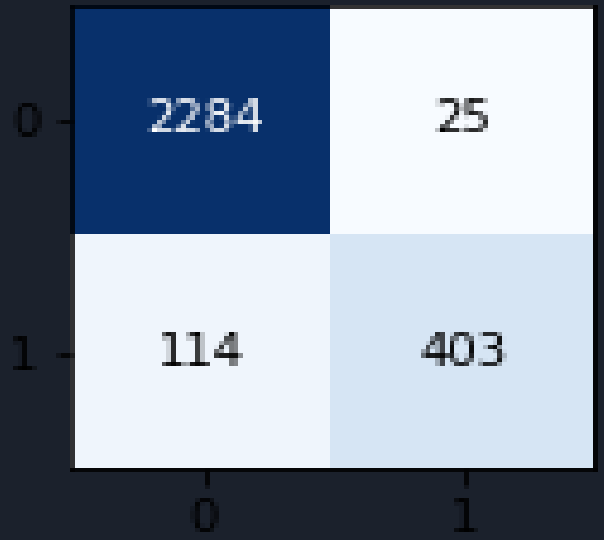
Decided on hyperparameter manually using the '%timeit' function

```python
# Use a bagging classifier
from sklearn.ensemble import BaggingClassifier

base = DecisionTreeClassifier(max_depth = 15,
                              criterion = 'entropy',
                             random_state = 1)

bag = BaggingClassifier(base,
                       n_estimators = 500,
                       random_state = 1)


bag.fit(X_train,y_train)
np.mean(bag.predict(X_test) == y_test)
```

```
0.9249823071479123
```

Random Forest

- 100 trees in the forest
- Using Gini impurity for information gain
- No max depth of trees
- Default number of features to consider the split



- 93.98% accuracy on test set.

Grid Search

Grid Search is used to see if there was a way to improve the accuracy of our base models

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11]}

gs = GridSearchCV(estimator=knn,
                  param_grid=param_grid,
                  refit=True,#default
                  cv=10,#k-fold cross validation, strtified, accuracy
                  n_jobs=-1)

gs.fit(x_train, y_train)

print('Best Accuracy: %.2f%%' % (gs.best_score_*100))#available after fitting
print('Best Params:', gs.best_params_)#get the parameter combination
```

```
Best Accuracy: 87.11%
Best Params: {'n_neighbors': 1}
```

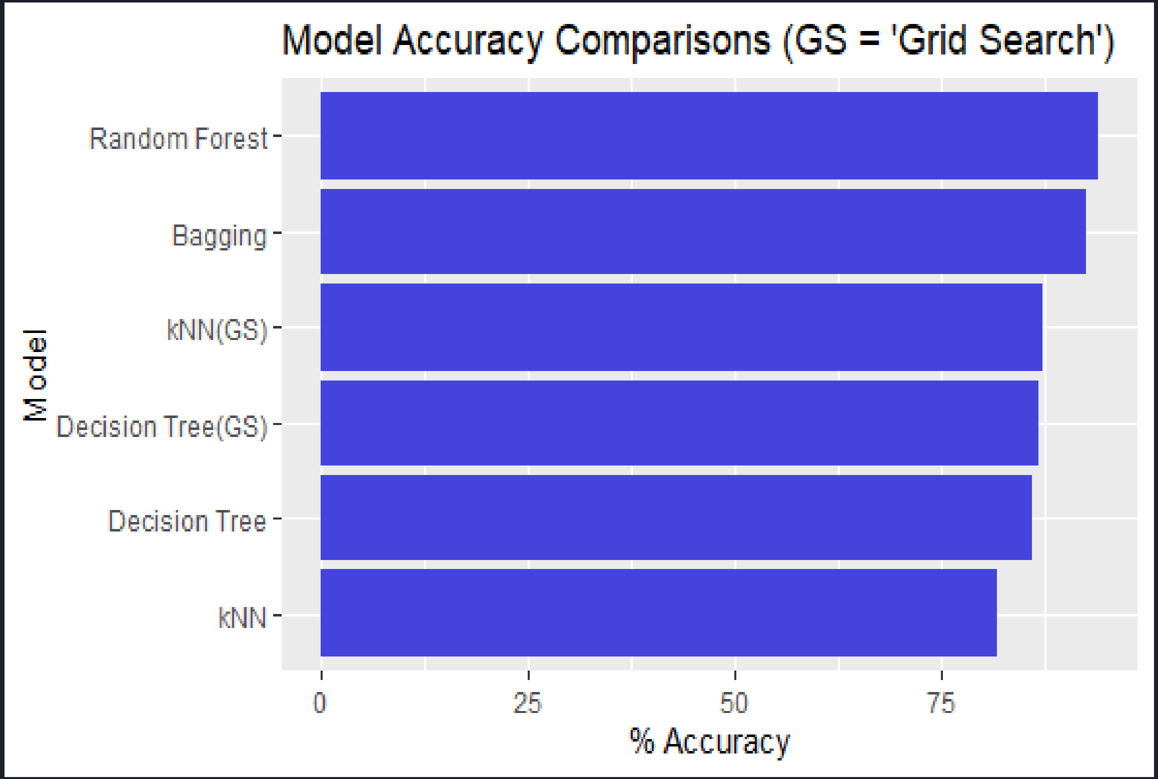
KNN with Grid Search

```
from sklearn.model_selection import GridSearchCV


param_grid = [{'max_depth': [1, 2, 10, 20, 25, 30, 65, None],
               'criterion': ['gini', 'entropy']}]

gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state = 123),
                  param_grid=param_grid,
                  refit=True,
                  cv=20,
                  n_jobs=-1)

gs.fit(x_train, y_train)

print('Best Accuracy: %.2f%%' % (gs.best_score_*100))
print('Best Params:', gs.best_params_)
```

```
Best Accuracy: 86.93%
Best Params: {'criterion': 'entropy', 'max_depth': 25}
```

Decision Tree with Grid Search

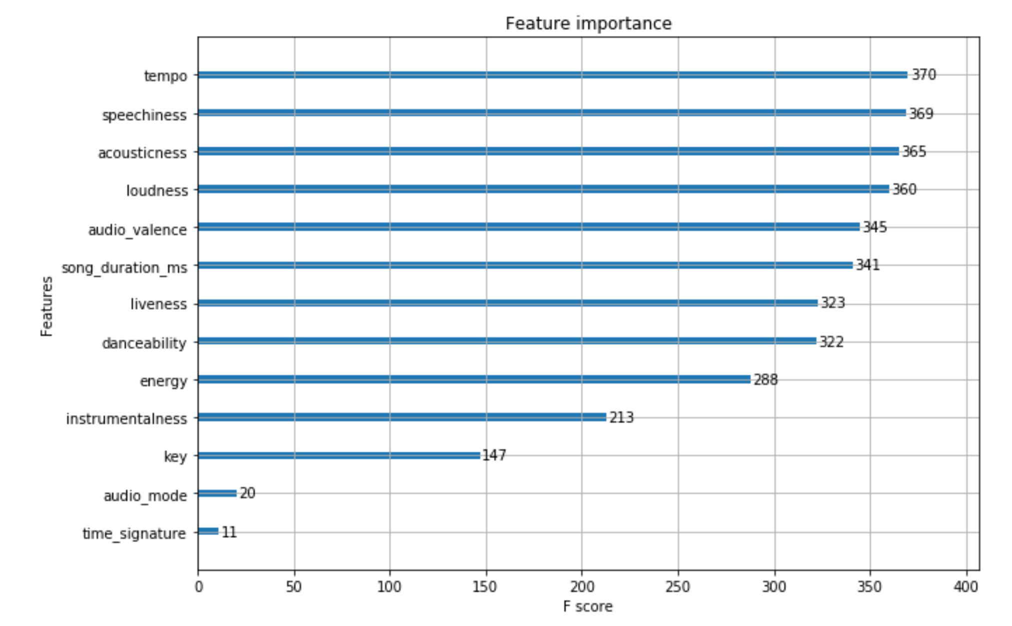
Model Comparisons (Van)



Feature importance(Van)

In asking about which acoustic qualities are most important in determining song popularity, we implicitly require a level of interpretability

We analyze feature importance with XGboost python package and the plot_importance function to gain an understanding of the popular song qualities



Conclusion

The model that performed best was :
Random Forest Classifier

Examining other analysis of this data (such as on Kaggle)supported this classifier as well showing power of this model

Other Considerations:

How prevalent is a song on social media (Tik Tok, etc.)?