RESEARCH PAPER

SPOTIFY MUSIC ANALYSIS USING MACHINE LEARNING

By

RUTA LAD

NORTHEASTERN UNIVERSITY

UNDER GUIDENCE OF

PROF. NICK BROWN

ABSTRACT: A machine learning approach to classify music by mood based on song lyrics and many other factors as listed above. This project is about building a music recommendation system for users who want to listen to customized playlists. Such a system can not only be used to brighten up one's mood but also could be used to spread positive mood among people. Spotify data was compiled as part of my project that uses machine learning to classify songs based on audio features associated with it. The collection of data started by outing together playlist details form Spotify. Spotify web API was used for the collection of the playlist of each category. Song title, and artist names were used to extract low level and high-level Audio features like danceability, energy, liveness, valence, tempo etc. of the songs. Data was also curated using Spotify's audio analysis API. A larger set of songs is part of this data set.

I. INTRODUCTION

Spotify is a music, podcast, and video streaming service that was officially launched on 7 October 2008. It is developed by startup Spotify AB in Stockholm, Sweden. It provides DRM–protected content from record labels and media companies. Spotify is a freemium service; basic features are free with advertisements or limitations, while additional features, such as improved streaming quality and music downloads, are offered via paid subscriptions. Spotify pays royalties based on the number of artists' streams as a proportion of total songs streamed on the service, unlike physical or download sales, which pay artists a fixed price per song or album sold. They distribute approximately 70% of total revenue to rights holders, who then pay artists based on their individual agreement.

II. RELATED WORK

The best part about the data wrangling is the minimal data cleaning and wrangling. When I passed through the list features variable into a panda's data frame, it returned a clean and neatly ordered dataset absent of null values. In addition, the only feature engineering to be done was converting the duration column from milliseconds to minutes. The only thing left for me to do was label the two different datasets (1 for good, 0 for bad) and concatenate them. Given that this is my first time working with kind of data, I knew I needed to undergo a thorough EDA to attain a significant level of familiarity with the data. I refrained from making any sort of hypotheses before diving into the data. For my EDA, my two goals were understanding the variance of the features and to see which of the features correlate the best with the target variable (liked/disliked song.) Though there is evidence of slight correlations between certain features (acousticness/energy and acousticness and loudness), I observed just an iota of correlation between the selected features and the target variable. This meant that my task of devising a competent classifier was a difficult one and that a linear classifier would most likely not be enough. Before moving onto the modeling section, it was imperative that I observe the data in 2-dimensional form. Decision tree is the clear winner in this graphic, its accuracy score of 0.71 and Naive Bayes was lowest with 0.58. To see if I could improve my scores, I opted to go with a more high-powered algorithm: Random Forest. However much to my surprise, the best cross-validated score I could come up with was a 0.73. I intended to practice my newly acquired insights into seaborn visualization, I used the dataset which contains the daily ranking of the 200 most listened songs in 53 countries from 2017 and 2018 by Spotify users. It contains more than 2 million rows, which comprises 6629 artists, 18598 songs for a total count of one hundred five billion streams count. To get a first insight into seaborn I implemented it, as I step-by-step, my process to get insights into the dataset and meaningful visualizations. Visualizing streams by region, track name, and date Visualizing a track's daily rank (+/- yesterday's rank) Visualizing position vs streams Find most popular artist per region (by time on chart, total stream count, average position and highest position) Evaluate lag of popular tracks in neighboring countries.

III. DATA SOURCES

Data Sources for this Kernel

- Spotify's Worldwide Daily Song Ranking :

  This dataset contains the daily ranking of the 200 most listened songs in 53 countries from 2017 and 2018 by Spotify users. It contains more than 2 million rows, which comprises 6629 artists, 18598 songs for a total count of one hundred five billion streams count. The data spans from 1st January 2017 to 9th January 2018 and will be kept up-to-date on following versions. It has been collected from Spotify's regional chart data.

- Audio features of song :

  This dataset includes the names, artists, Spotify URIs and audio features of tracks from playlist. It covers attributes such as danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration, time_signature.

IV. PROPOSED ALGORITHMS

TRADITIONAL MACHINE LEARNING ALGORITHM

- GAUSSIAN NAÏVE BAYES ALGORITHM
  Naive Bayes classifier. In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naïve Bayes has been studied extensively since the 1950s.

- DECISION TREE
  A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements

- Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by

constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

DEEP LEARNING USING NEURAL NETWORKS

ARTIFICIAL NEURAL NETWORK:
Firstly, it helps us understand the impact of increasing / decreasing the dataset vertically or horizontally on computational time. Secondly, it helps us understand the situations or cases where the model fits best. Thirdly, it also helps us explain why certain model works better in certain environment or situations

Formulation Neural network:

Following is the framework in which artificial neural networks (ANN) work:

STEP 1: ASSIGN RANDOM WEIGHTS TO ALL THE LINKAGES TO START THE ALGORITHM.

STEP 2: USING THE INPUT AND THE LINKAGES FIND THE ACTIVATION RATE OF OUTPUT NODES.

STEP 3: USING THE ABOVE FINDING THE ACTIVATION RATE OF OUTPUT NODES.

STEP 4: FINDING THE ERROR RATE AT THE OUPUT NODE.

STEP 5: USING THE WEIGHTS AND ERRORFOUND AT OUTPUT NODE, CASCADE DOWN THE ERROR TO HIDDEN NODES.

STEP 6: RECALIBRATE THE WEIGHTS BETWEEN HIDDEN NODE AND INPUT NODE.

STEP 7: REPEAT THE PROCESS THE CONVERGENCE CRTIRIA IS MET.

STEP 8: USING THE FINAL LINKAGE WEIGHT SCORE THE ACTIVATION RATE OF THE OUTPUT NODES.

V. CODE DOCUMENTATION

TRAING THE CLASSIFICATION ALGORITHM

In [3]:
```
from sklearn.model_selection import train_test_split

# drop string features (like names)
df.drop(['song_title','artist'], axis=1, inplace=True)

# the songs, "X" axis
X = df.drop(['target'], axis=1)

# and their label/class, "y" axis
y = df['target']

# split the data into train sets (X_train, y_train) and test sets (X_test, y_test).
# the testing sets are usually smaller than the training sets.
# this time I'll use 20% for testing and 80% for training.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [4]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

dtc = DecisionTreeClassifier()
nbc = GaussianNB()
rfc = RandomForestClassifier()

dtc.fit(X_train, y_train)
nbc.fit(X_train, y_train)
rfc.fit(X_train, y_train)
```

Out[4]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0
```

ACCURACY OF TRAIN AND TEST SET

In [5]:
```
tree_score = dtc.score(X_test, y_test)
naive_score = nbc.score(X_test, y_test)
random_forest_score = rfc.score(X_test, y_test)

print('Decision Tree score:', tree_score)
print('Naive Bayes score:', naive_score)
print('Random Forest score:', random_forest_score)
```

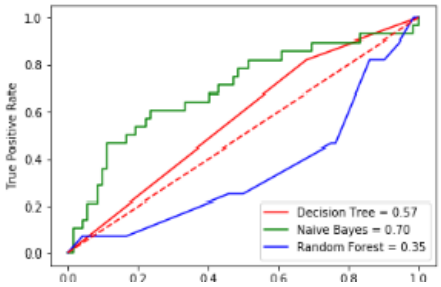
```
Decision Tree score: 0.668316831683
Naive Bayes score: 0.611386138614
Random Forest score: 0.759900990099
```

In [7]:
```
test_playlist = pd.read_csv('C:/Users/Chandu Lad/Desktop/spotify2.csv')
test_playlist.drop(['id', 'album', 'analysis_url', 'name', 'track_href', 'type', 'uri'], axis=1, inplace=True)

test_playlist_X = test_playlist.drop(['class'], axis=1)
test_playlist_y = test_playlist['class']

test_tree_score = dtc.score(test_playlist_X, test_playlist_y)
test_naive_score = nbc.score(test_playlist_X, test_playlist_y)
test_random_forest_score = rfc.score(test_playlist_X, test_playlist_y)

print('Decision Tree score:', test_tree_score)
print('Naive Bayes score:', test_naive_score)
print('Random Forest score:', test_random_forest_score)
```

```
Decision Tree score: 0.46
Naive Bayes score: 0.74
Random Forest score: 0.3
```

ROC CURVE

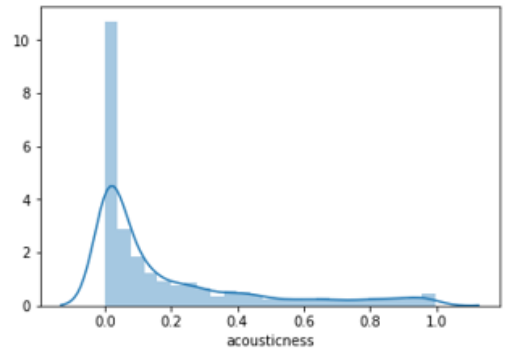
In [8]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

actual = test_playlist['class']
decision_tree_prob = [p[1] for p in dtc.predict_proba(test_playlist_X)]
naive_bayes_prob = [p[1] for p in nbc.predict_proba(test_playlist_X)]
random_forest_prob = [p[1] for p in rfc.predict_proba(test_playlist_X)]

dt_false_pos, dt_true_pos, _ = roc_curve(actual, decision_tree_prob)
dt_auc = auc(dt_false_pos, dt_true_pos)

nb_false_pos, nb_true_pos, _ = roc_curve(actual, naive_bayes_prob)
nb_auc = auc(nb_false_pos, nb_true_pos)

rf_false_pos, rf_true_pos, _ = roc_curve(actual, random_forest_prob)
rf_auc = auc(rf_false_pos, rf_true_pos)

# plot
plt.plot(dt_false_pos, dt_true_pos, 'r', label='Decision Tree = %0.2f'% dt_auc)
plt.plot(nb_false_pos, nb_true_pos, 'g', label='Naive Bayes = %0.2f'% nb_auc)
plt.plot(rf_false_pos, rf_true_pos, 'b', label='Random Forest = %0.2f'% rf_auc)

plt.legend(loc='lower right')
plt.plot([0,1], [0,1], 'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



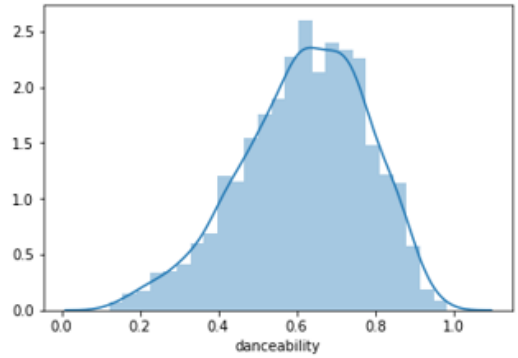
Means of Audio Features

Acousticness and Danceability

```
Mean value for acousticness: 0.18759003442241004
```

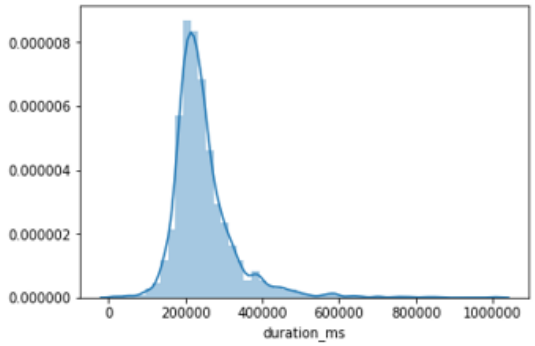



```
Mean value for danceability: 0.6184219137332657
```

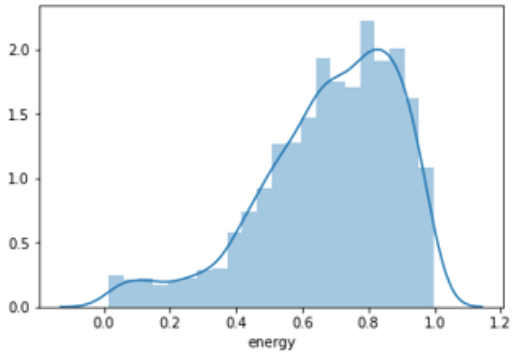


Duration and Energy

Mean value for duration_ms feature: 246306.19732275658

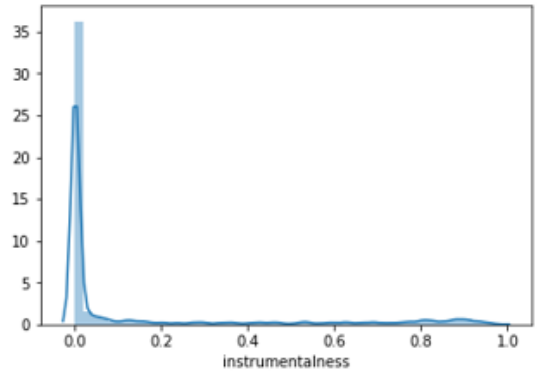


Mean value for energy: 0.6815771442736742

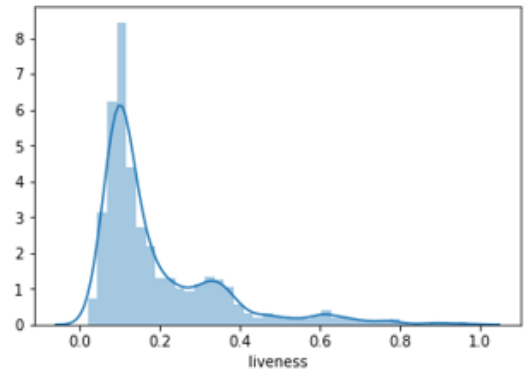


Instrumentalness and liveness

Mean value for instrumentalness: 0.13328552863163118

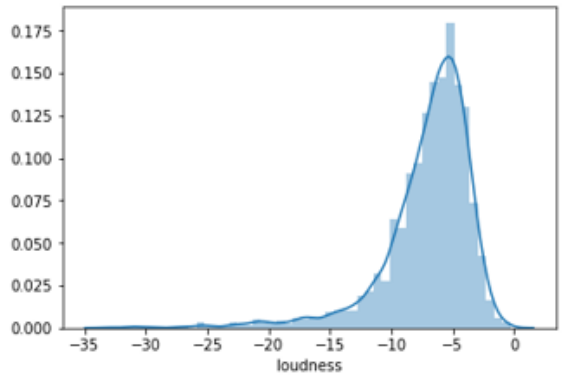


Mean value for liveness:valen 0.19084402578086268

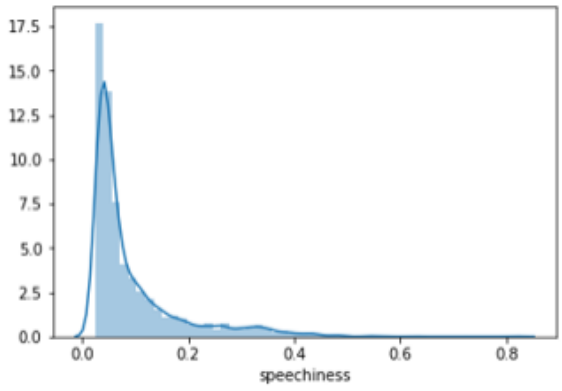


Loudness and Speechiness

Mean value for loudness: -7.085624194348036

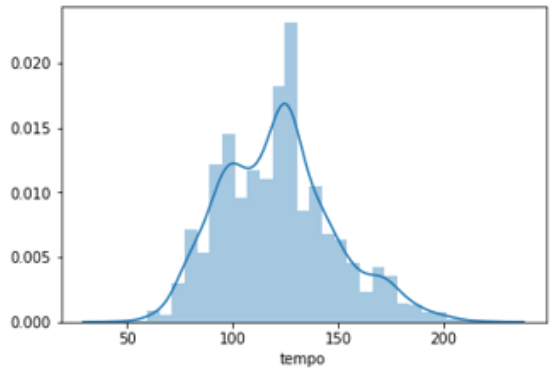


Mean value for speechiness: 0.0926642538423404

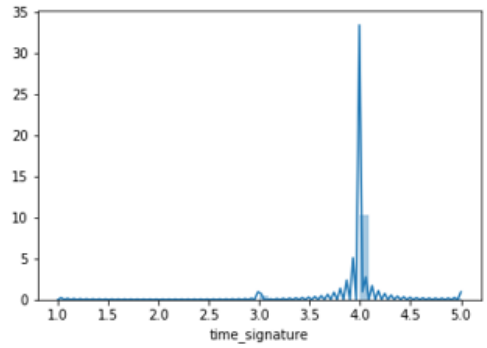


Tempo and Time_Signature

```
Mean value for tempo feature: 121.60327169062967
```

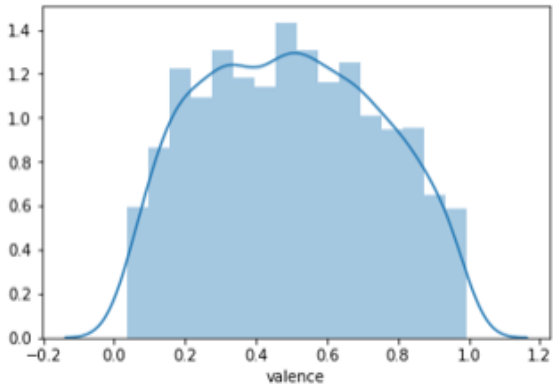



```
Mean value for time_signature feature: 3.9682697074863658
```

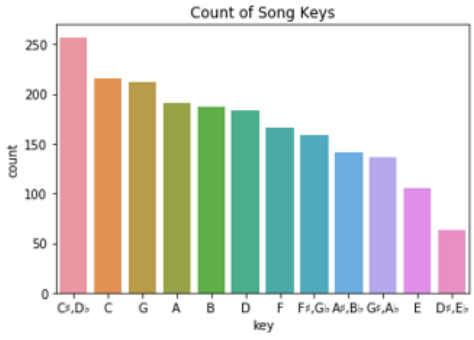


Valence

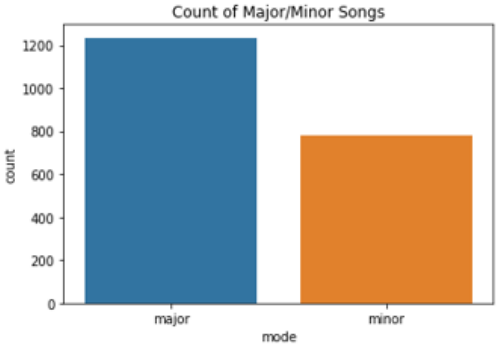
```
Mean value for valence feature: 0.4968150223103621
```



Key and Mode




```
Mean value for mode feature: 0.6122954883490332
```



EDA(Exploratory data analysis)

Count of appearances of artists

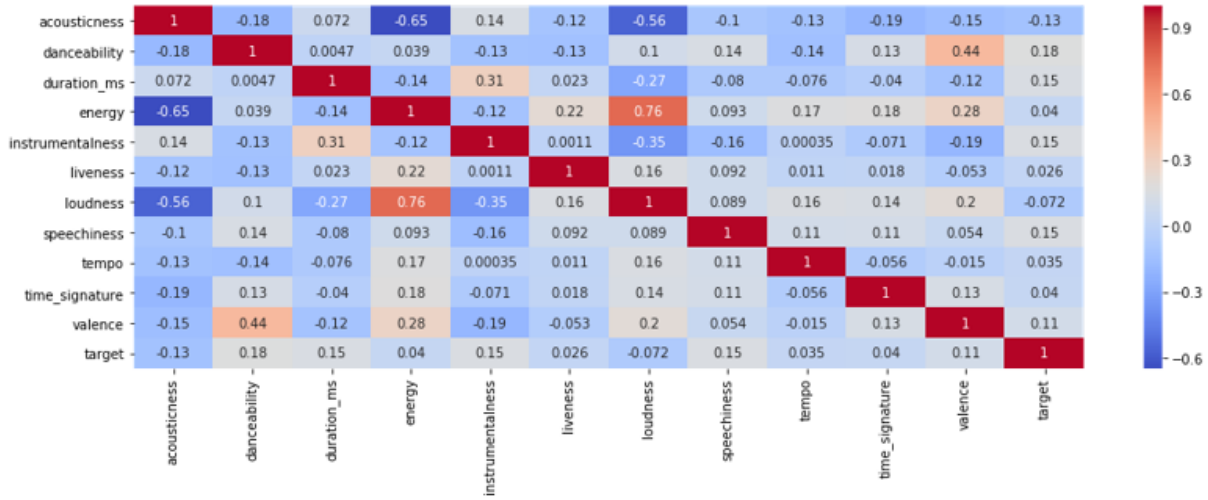
In [25]:

```
speaker_df = data.groupby('artist').count().reset_index()[['artist', 'target']]
speaker_df.columns = ['artist', 'appearances']
speaker_df = speaker_df.sort_values('appearances', ascending=False)
speaker_df.head(10)
```

Out[25]:

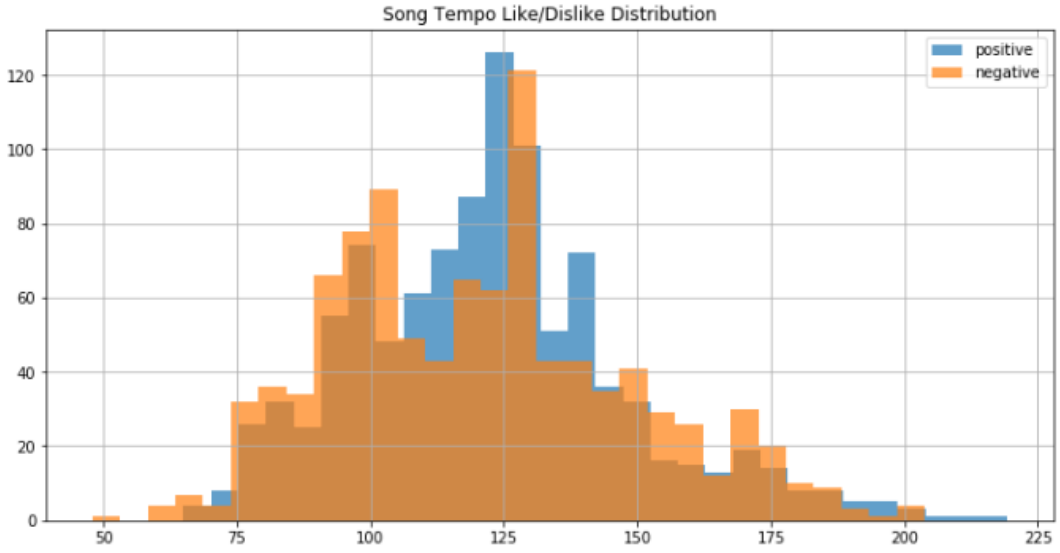
	artist	appearances
356	Drake	16
960	Rick Ross	13
345	Disclosure	12
81	Backstreet Boys	10
1273	WALK THE MOON	10
405	FIDLAR	9
277	Crystal Castles	9
1	*NSYNC	8
409	Fall Out Boy	8
788	Michael Jackson	8

HeatMap for correlation



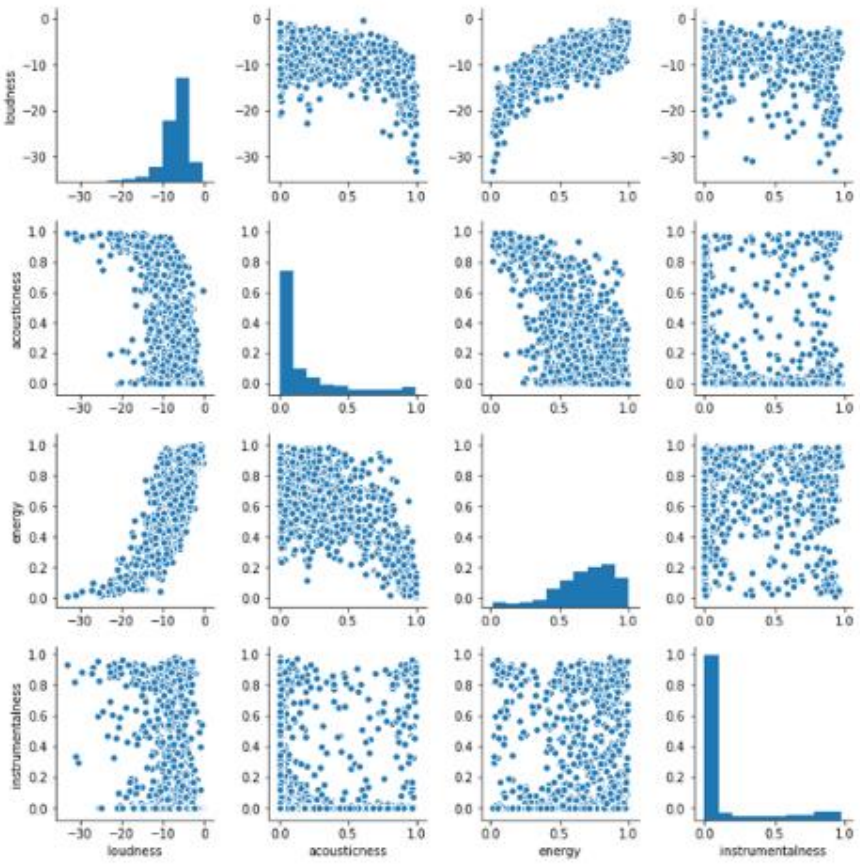
Like/Dislike Distribution

```
Out[64]: <matplotlib.legend.Legend at 0x2014db32668>
```



PairPlot

```
Out[51]: <seaborn.axisgrid.PairGrid at 0x2014dc2d6a0>
```



Analysis with respect to Artist and Location

[6]:
```
Top_song = data.loc[data['Streams'] == data['Streams'].max() ,:]
Top_song
```

t[6]:

	Position	Track Name	Artist	Streams	URL	Date	Region
3145443	1	Despacito (Featuring Daddy Yankee)	Luis Fonsi	11381520	https://open.spotify.com/track/4aWmUDTfIPGksMN...	2017-05-27	global

Deep Learning (Artificial Neural Network)

In [17]:
```
#Initialising ANN
#Defining ANN as a squence of layers
classifier = Sequential()
#Adding the input layer and the first hidden layer
classifier.add(Dense(7, activation='relu', input_shape=(13,), name = 'DenseLayer1'))
#Adding 2nd hidden layer
classifier.add(Dense(7, activation='relu', name = 'DenseLayer2'))
#Adding the output layer
classifier.add(Dense(1, activation='sigmoid', name = 'DenseLayer3'))
#Compiling the ANN model
classifier.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
#Fitting the model
classifier.fit(X_train, y_train, epochs=100, batch_size=100)
```

```
1613/1613 [==============================] - 0s 29us/step - loss: 0.5102 - acc: 0.7626
Epoch 93/100
1613/1613 [==============================] - 0s 28us/step - loss: 0.5093 - acc: 0.7632
Epoch 94/100
1613/1613 [==============================] - 0s 15us/step - loss: 0.5089 - acc: 0.7619
Epoch 95/100
1613/1613 [==============================] - 0s 19us/step - loss: 0.5084 - acc: 0.7619
Epoch 96/100
1613/1613 [==============================] - 0s 29us/step - loss: 0.5079 - acc: 0.7619
Epoch 97/100
1613/1613 [==============================] - 0s 19us/step - loss: 0.5073 - acc: 0.7657
Epoch 98/100
1613/1613 [==============================] - 0s 29us/step - loss: 0.5070 - acc: 0.7626
Epoch 99/100
1613/1613 [==============================] - 0s 29us/step - loss: 0.5061 - acc: 0.7619
Epoch 100/100
1613/1613 [==============================] - 0s 10us/step - loss: 0.5061 - acc: 0.7613
```

Out[17]: <keras.callbacks.History at 0x1d9bf5e5588>

ANN with different hyperparameters

n [36]:

```
#Initialising ANN
#Defining ANN as a squence of layers
classifier = Sequential()
#Adding the input layer and the first hidden layer
classifier.add(Dense(7, activation='tanh', input_shape=(13,), name = 'DenseLayer1'))
#Adding 2nd hidden layer
classifier.add(Dense(7, activation='tanh', name = 'DenseLayer2'))
#Adding the output layer
classifier.add(Dense(1, activation='sigmoid', name = 'DenseLayer3'))
#Compliling the ANN model
classifier.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = ['accuracy'])
#Fitting the model
classifier.fit(X_train, y_train, epochs=100, batch_size=100)
```

```
1613/1613 [==============================] - 0s 30us/step - loss: 0.6938 - acc: 0.4761
Epoch 93/100
1613/1613 [==============================] - 0s 20us/step - loss: 0.6938 - acc: 0.4923
Epoch 94/100
1613/1613 [==============================] - 0s 17us/step - loss: 0.6934 - acc: 0.5046
Epoch 95/100
1613/1613 [==============================] - 0s 17us/step - loss: 0.6937 - acc: 0.5071
Epoch 96/100
1613/1613 [==============================] - 0s 17us/step - loss: 0.6937 - acc: 0.4842
Epoch 97/100
1613/1613 [==============================] - 0s 20us/step - loss: 0.6941 - acc: 0.4885
Epoch 98/100
1613/1613 [==============================] - 0s 17us/step - loss: 0.6936 - acc: 0.4985
Epoch 99/100
1613/1613 [==============================] - 0s 22us/step - loss: 0.6937 - acc: 0.4935
Epoch 100/100
1613/1613 [==============================] - 0s 27us/step - loss: 0.6935 - acc: 0.5059
```

ut[36]: <keras.callbacks.History at 0x1d9c4381a20>

VI. RESULT AND EXPLANATION

- Though all three models have its unique characteristics which makes them useful in different situations, Random Forest classifier performed the best with the highest accuracy value
- From the exploratory analysis , I determined danceability and energy audio feature is contributing more to the likeness of the song.
- For ANN, the accuracy changes with different parameters, here relu works better than tanh, as the epochs increases the accuracy also increases, more layers it results in overfitting.

VII. REFERENCES

[1] www.researchgate.net/post/Is_there_any_documentation_about_Spotifys_audio_features

[2]www.opendatascience.com/a-machine-learning-deep-dive-into-my-spotify-data/

[3] www.kaggle.com

[4] www.spotify.com

[5] www.Wikipedia.com