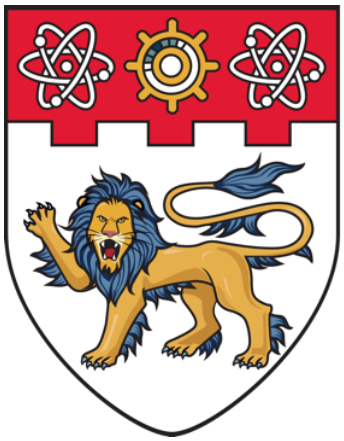


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4032 Data Analytics and Mining

Top Song Analysis and Prediction

Eugene Yeo (U1621627G)	16.67%
Li Huimin (U1620337G)	16.67%
Liu Fangbing (U1622143H)	16.67%
Wu Qinxin (U1522835L)	16.67%
Xie Zesheng (U1621162D)	16.67%
Yang Can (U1520582J)	16.67%

1. Abstract

At the end of each year, Spotify compiles a playlist of the songs streamed most often over the course of that year. Several questions thus arise: What do these top songs have in common? Do these popular songs share a common pattern so that they are more likely to spread and circulate? Why do people like them? Therefore, we come out with this study, aiming to analyze the characteristics as attributes of these top songs in order to examine the common attributes as well as to predict the model to produce the next top song. Data mining processes were developed in Python and MATLAB by experimenters. Analysis using real world data mining tools Weka was also performed. For this study, after careful consideration, four different classification algorithms, i.e. K-Nearest Neighbor (K-NN), Naïve Bayes Classifier (NBC), Artificial Neural Networks (ANN) and Decision Tree were utilized. In conclusion, this study extracts a common pattern of the top songs and provides a quick way to predict the popularity of a song, as well as constructs a model to compose a song in a concise and efficient way so as to achieve high prevalence.

2. Problem Description

The dataset obtained from Kaggle records down 600 top songs and non-top songs. There is a total of 12 attributes describing the song features. We aim to find a common model by extracting the characteristics od these attributes and utilizing the model to predict whether a song will be popular or not using a KDD ((Knowledge Discovery in Database) process on the song data. Here is a rough introduction of the song attributes:

- Danceability

Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.

- Energy

Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

- Key

The value of the key is integer map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. The minimum is 0, the maximum is 11.

- Loudness

The loudness means the overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.

- Mode

The mode means the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.

There are 316 instances in minor mode and 398 instances in major mode.

- Acousticness

A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.

- Speechiness

Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

- Instrumentalness

Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

- Liveness

Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.

- Valence

The valence of a song is defined as a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive such as happy, cheerful and euphoric, while tracks with low valence sound more negative such as sad, depressed and angry. And it is usually been classified into three class where valence 0 to 0.4 as sad songs, valence 0.4 to 0.6 as neutral songs and valence 0.6 to 1 as happy songs.

- Tempo

The tempo of a song represents the beats per minutes. For example, the tempo of a song is 100, that means the song is playing at 100 beats per minute. This is used to determine a song slow or fast. The classification as below:

Largo (very slow) is 40–60 BPM. Larghetto (less slow) is 60–66 BPM.

Adagio (moderately slow) is 66–76 BPM. Andante (walking speed) is 76–108 BPM.

Moderato (moderate) is 108–120 BPM. Allegro (fast) is 120–168 BPM.

Presto (faster) is 168–200 BPM. Prestissimo (even faster) is 200+ BPM.

- Duration

The duration of a song represents the play time of the song.

3. Pre-processing

3.1 Removal of Null Values

Firstly, we get the top song data file from Kaggle. Each row contains a ranking position on a specific day in 2017 for a song. For instance, the first 200 rows present the ranking for the 1st of January in Argentina. We read the *data.csv* file as dataframe *'data'* which contains 3441197 rows and some cells in the column "Trace Name", "Artist" and "URL" are empty. Hence, firstly we drop rows that contains null.

```
In [49]: data.isnull().any()

Out[49]: Position        False
         Track Name       True
         Artist           True
         Streams         False
         URL              True
         Date            False
         Region          False
         dtype: bool
```

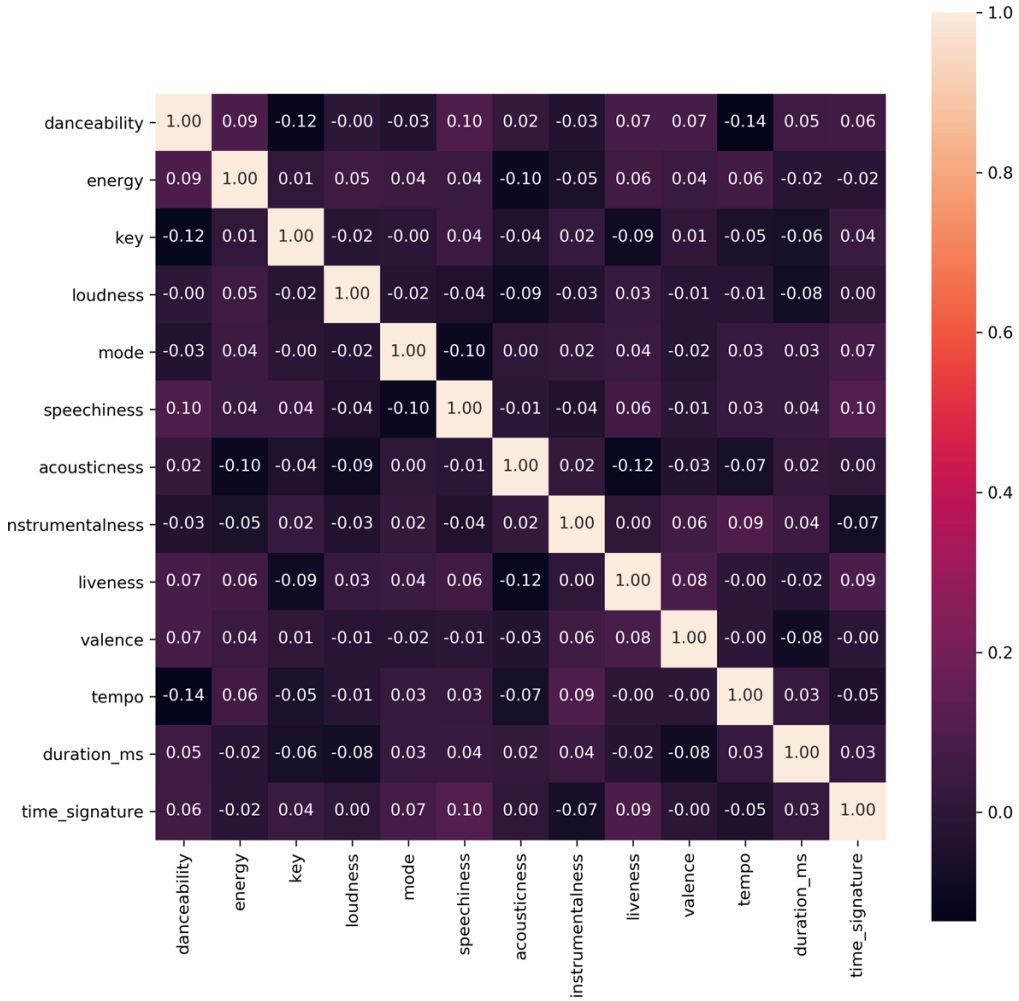
```
In [44]: data = data.dropna()
         data.isnull().any()

Out[44]: Position        False
         Track Name      False
         Artist          False
         Streams         False
         URL             False
         Date            False
         Region          False
         dtype: bool
```

3.2 Define top songs and non-top songs

To define top songs, we group the data by “song title” and add all the streams up as “total stream”. Then divide “total streams” by the unique days that this song is on the list, to get the average worldwide stream per day. We sort these data according to the average stream per day and classify the top 140 songs which streams more than 1.64×10^6 times per day as top songs. Besides 140 top songs, 460 non-top songs are also collected training dataset. We fill in the song titles manually and collect descriptions of audio features from the Spotify Web API.

3.3 Correlation Heatmap for Song Attributes

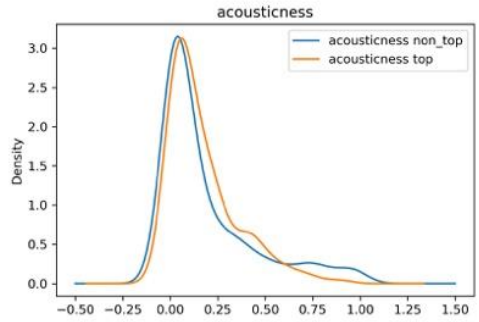


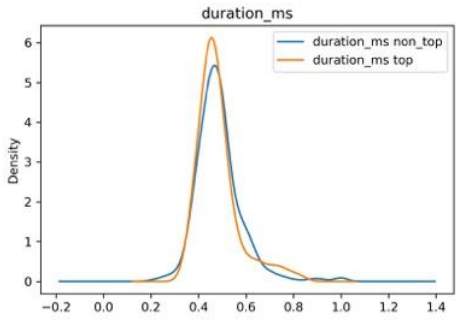
Above is the correlation between each pair of the song attribute, with value scaling from '-1' to '1' where '1' indicates a directly proportional correlation and '-1' indicates perfectly negative correlation. There seems to be little correlation between each pair of feature attributes, as most of index are lower than 0.1. What we found from the last row is that whether a song is popular, highly correlates to danceability with an index of 0.44. Besides "danceability", Top attributes also has relatively higher correlation with "energy", "loudness", "speechiness", "liveness" and "valence", whose index is higher than 0.1. Therefore, we take these six attributes for the following analysis.

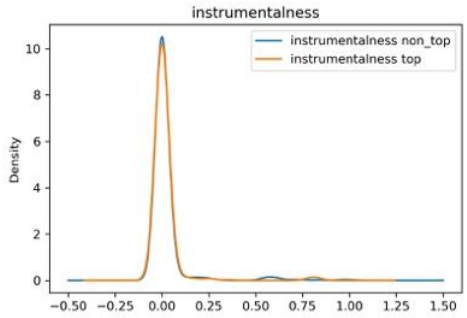
3.4 Comparison of Attributes

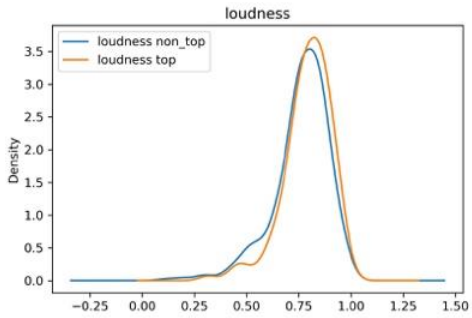
In [4]:
```
for i in range(3,16):
    df = pd.DataFrame()
    df[str(Topdf.columns[i])+" top"] = Nontopdf[Nontopdf.columns[i]]/ \
    Nontopdf[Nontopdf.columns[i]].loc[Nontopdf[Nontopdf.columns[i]].abs().idxmax()].astype(np.float64)
    df[str(Nontopdf.columns[i])+" non_top"]  = Topdf[Topdf.columns[i]]/\
    Topdf[Topdf.columns[i]].loc[Topdf[Topdf.columns[i]].abs().idxmax()].astype(np.float64)
    df.plot(kind='kde', title = Topdf.columns[i])
    plt.savefig('OneDrive - Nanyang Technological University/'+Topdf.columns[i]+'.jpg', dpi=300)
```

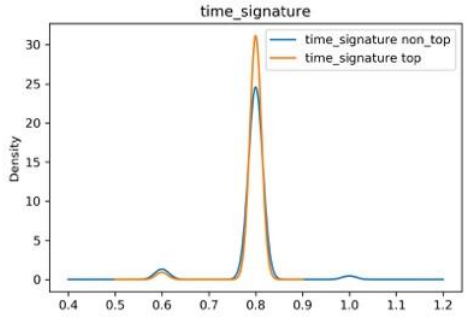
To compare top song and non-top song, we first normalize each column, and then plot the density distribution of top songs and non-top songs with respect to the same feature in one graph.



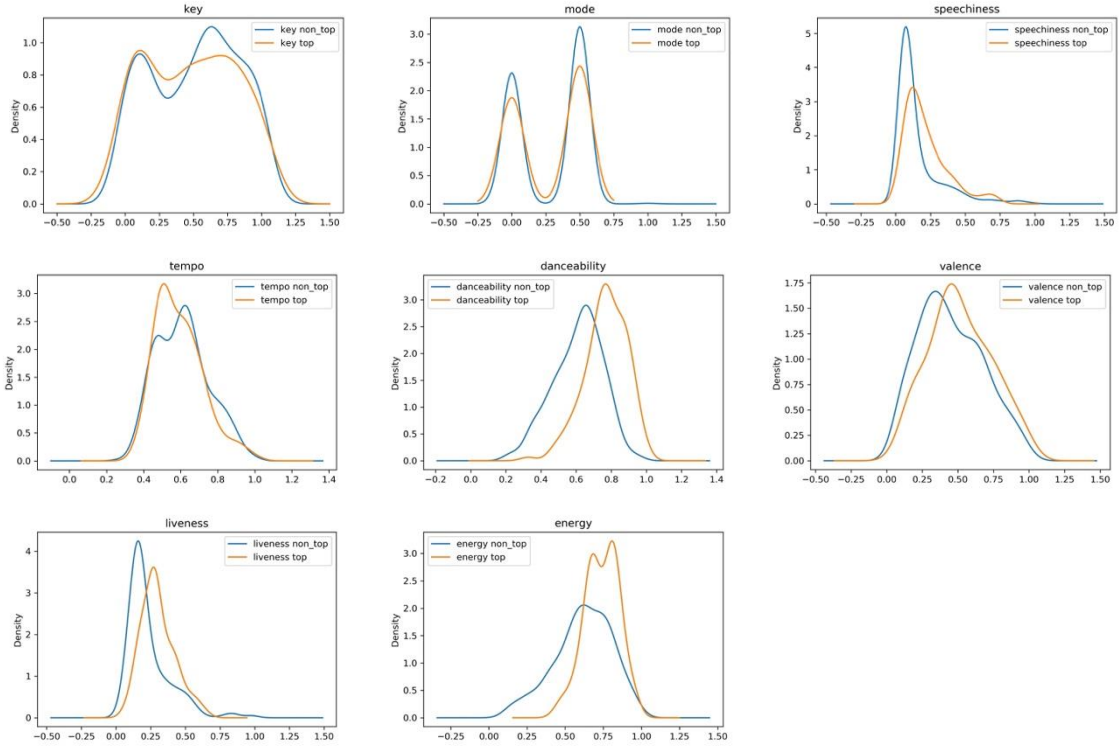








Regarding to some features, there is no significant difference between top songs and non-top songs, such as acousticness, duration, instrumentalness and loudness. As a result, we can drop these columns in the following analysis.



While looking at the figures above, we can see that top songs performs better than non-top songs on features such as danceability, valence, liveness and energy. Therefore, a danceable, energetic, positive song with live audience, is more likely to be popular among the public.

4. Prediction of Top Songs by Multiple Classification Techniques

4.1 Artificial Neural Network Classification

4.1.1 Algorithm

Artificial neural network (ANN) classification is the process of learning to separate samples into different classes by finding common features between samples of known classes. It is an example of Supervised Learning. Known class labels help indicate whether the system is performing correctly or not. This information can be used to indicate the desired response, validate the accuracy of the system, or be used to help the system learn to behave correctly. The known class labels are a supervising learning process; the term is not meant to imply that you have some interventionist role. Also, it is generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning. A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs with each layer fully connected to the next one. MLP utilizes a supervised learning technique called back-propagation for training the network. It seeks to classify an observation as belonging to some discrete class as a function of the inputs.

4.1.2 Implementation

Software Tools

In this project, Weka, Pandas, Keras, Tensorflow and Scikit-Learn are chosen to test data mining approaches as its character of containing a collection of visualization tools and algorithms for predictive modeling, which matches our aim to predicting mushroom based on analyzing results of training data. Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand.

Pandas is an open source, a BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing proper research.

TensorFlow is an open source software library for high-performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms and from desktops to clusters of servers to mobile and edge devices.

Scikit-Learn is a Simple and efficient tool for data mining and data analysis. It Built on NumPy, SciPy, and matplotlib. Also, it is accessible to everybody and reusable in various contexts.

Attribute Transformation (For ANN Classification)

In ANN Classification, the values of datasets must be a numerical value, and it has to be normalized to a numerical value between 0 and 1. Take training set and normalize numerical valued attribute to zero mean and unit standard deviation. For the test dataset, the same normalization is performed as to the training dataset using the same parameters calculated from the training datasets. Following python code is written to perform this data normalization.

```
scalar = MinMaxScaler()
scalar.fit(train)
train = scalar.transform(train)
scalar.fit(test)
test = scalar.transform(test)
```

Data Analysis (ANN Classification)

To analyze data with artificial neural network method, we used Multilayer Perceptron method of classifier under functions folder. The results did not really change when the settings changed. In the results section, the results were derived from the settings of learning rate = 0.2, momentum = 0.2 and training time = 20.

Model Construction (ANN Classification)

To solve this classification problem, a model of the dataset is needed. Models in Keras are defined as a sequence of layers. The first thing to get right is to ensure the input layer has the right number of inputs. This can be specified when creating the first layer with the input_dim argument and setting it to 13 for the 13 input variables as there are 13 columns in dataset needs to be inputted. Next, a fully-connected network structure with three layers is built for the model. Fully connected layers are defined using the

Dense class. It specify the number of neurons in the layer as the first argument, the initialization method as the second argument as init and specify the activation function using the activation argument. After which, the rectifier ('relu') activation function is used on the first two layers and the sigmoid function in the output layer. It used to be the case that sigmoid and tanh activation functions were preferred for all layers. These days, better performance is achieved using the rectifier activation function. We use a sigmoid on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5. Overall, the model have the first layer with 13 neurons and expects 13 input variables. The second hidden layer has 4 neurons and finally, the output layer has 1 neuron to predict the class (onset of top song or not). Following python code is written to build the classification model:

```
model = Sequential()
model.add(Dense(13, input_dim=13, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

After the building the model, compile of the model is needed. Compiling the model uses the efficient numerical libraries under the covers which is TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed. When compiling, some additional properties are required when training the network such as "loss" function, "optimizer" function and ""metric" function. The "loss" function used to evaluate a set of weights, the "optimizer" used to search through different weights for the network and "metrics" used to collect and report during training. In this case, we will use logarithmic loss, which for a binary classification problem is defined in Keras as "binary_crossentropy". We will also use the efficient gradient descent algorithm "adam" for no other reason that it is an efficient default. Finally, because it is a classification problem, we will collect and report the classification accuracy as the metric. Following python code is written to compile the model:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Up to now, the model structure is done and it is time to execute the model on the top song dataset. To train the model, calling the "fit()" function on the model. The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument. We can also set the number of instances that are evaluated before a weight update in the network is performed, called the batch size and set using the batch_size argument. For this project, the epochs is set to 5000 and batch_size is set to default. Following python code is written to train the model:

```
history = model.fit(X, Y, epochs=5000, batch_size=32, verbose=0)
```

Lastly, the evaluation of the model needs to be done. This will give us an idea of how well we have modeled the dataset. After the evaluation, the accuracy rate of 93.67% is obtained. Following python code is written to evaluate the model:

```
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Following image is the evaluation results of the model:

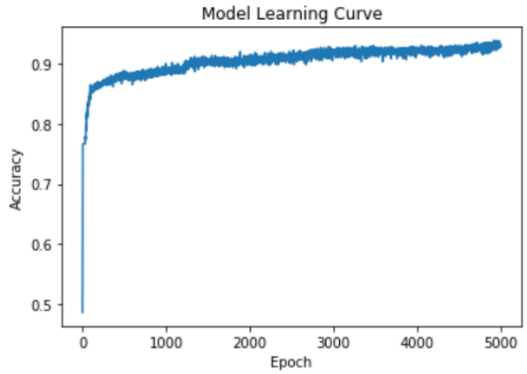
```
600/600 [==============================] - 0s 62us/step

acc: 93.67%
```

Additionally, Keras also allows to plot the learning curve and it shows the accuracy rate of the training process on the model. Following python code is written to plot the learning curve:

```
from matplotlib import pyplot
pyplot.plot(history.history['acc'])
pyplot.title('Model Learning Curve')
pyplot.ylabel('Accuracy')
pyplot.xlabel('Epoch')
pyplot.show()
```

Following graph is the learning curve of the accuracy rate of the model:



Results (ANN Classification)

Now is time to predict whether a song will be a top song. First of all, the values in test dataset need to be normalized. After which, the class prediction will be used for the test dataset. A class prediction is given the finalized model and one or more data instances, predict the class for the data instances. Following python code is written to predict the class of the dataset (0: not a top song; 1: top song):

```
scalar.fit(W)
W = scalar.transform(W)
predictions = model.predict_classes(W)
Y = test.iloc[:,1:2].values
for i in range(len(predictions)):
    print("song=%s, Predicted=%s" % (Y[i], predictions[i]))
```

While predicting the classes of the songs, ANN Classification takes very short time to perform the classification. After the prediction, the following result is obtained:

```
song=['Shape of You'], Predicted=[0]
song=['Despacito - Remix'], Predicted=[0]
song=['Despacito (Featuring Daddy Yankee)'], Predicted=[0]
song=['Something Just Like This'], Predicted=[0]
song=["I'm the One"], Predicted=[0]
song=['HUMBLE.'], Predicted=[0]
song=["It Ain't Me (with Selena Gomez)"], Predicted=[0]
song=['Unforgettable'], Predicted=[0]
song=["That's What I Like"], Predicted=[0]
song=['I Don’t Wanna Live Forever (Fifty Shades Darker) - From "Fifty Shades Darker (Original Motion Picture Soundtra
ck)"'], Predicted=[0]
song=['XO TOUR Llif3'], Predicted=[1]
song=['Paris'], Predicted=[0]
song=['Stay (with Alessia Cara)'], Predicted=[0]
song=['Attention'], Predicted=[0]
song=['Mask Off'], Predicted=[0]
song=['Congratulations'], Predicted=[0]
song=['Swalla (feat. Nicki Minaj & Ty Dolla $ign)'], Predicted=[0]
song=['Castle on the Hill'], Predicted=[0]
song=['Rockabye (feat. Sean Paul & Anne-Marie)'], Predicted=[0]
song=['Believer'], Predicted=[1]
song=['Mi Gente'], Predicted=[0]
song=['Thunder'], Predicted=[0]
song=["Say You Won't Let Go"], Predicted=[0]
song=["There's Nothing Holdin' Me Back"], Predicted=[0]
song=['Me Rehúso'], Predicted=[0]
song=['Issues'], Predicted=[0]
song=['Galway Girl'], Predicted=[0]
song=['Scared to Be Lonely'], Predicted=[0]
song=['Closer'], Predicted=[0]
song=['Symphony (feat. Zara Larsson)'], Predicted=[0]
song=['I Feel It Coming'], Predicted=[0]
song=['Starboy'], Predicted=[1]
song=['Wild Thoughts'], Predicted=[0]
song=['Slide'], Predicted=[1]
song=['New Rules'], Predicted=[0]
song=['1-800-273-8255'], Predicted=[0]
song=['Passionfruit'], Predicted=[0]
song=['rockstar'], Predicted=[0]
song=['Strip That Down'], Predicted=[0]
song=['2U (feat. Justin Bieber)'], Predicted=[0]
song=['Perfect'], Predicted=[1]
song=['Call On Me - Ryan Riback Extended Remix'], Predicted=[0]
song=['Feels'], Predicted=[0]
song=['Mama'], Predicted=[0]
song=['Felices los 4'], Predicted=[0]
song=['iSpy (feat. Lil Yachty)'], Predicted=[0]
song=['Location'], Predicted=[0]
```

Last but not least, besides the classification of the songs, the probability of song will be a top song also could be computed by ANN Classification. Following python code is written to predict the probability of a song that is a top song:

```
predictions = model.predict_proba(W)
Y = test.iloc[:,1:2].values
for i in range(len(predictions)):
    print("song=%s, Predicted=%s" % (Y[i], predictions[i]))
```

After the prediction, the following result is obtained:

```
song=['Shape of You'], Predicted=0.0000
song=['Despacito - Remix'], Predicted=0.0001
song=['Despacito (Featuring Daddy Yankee)'], Predicted=0.0045
song=['Something Just Like This'], Predicted=0.0001
song=["I'm the One"], Predicted=0.0000
song=['HUMBLE.'], Predicted=0.0029
song=["It Ain't Me (with Selena Gomez)"], Predicted=0.0000
song=['Unforgettable'], Predicted=0.0006
song=["That's What I Like"], Predicted=0.0000
song=['I Don't Wanna Live Forever (Fifty Shades Darker) - From "Fifty Shades Darker (Original Motion Picture Soundtra
ck)"'], Predicted=0.0000
song=['XO TOUR Llif3'], Predicted=0.0005
song=['Paris'], Predicted=0.0000
song=['Stay (with Alessia Cara)'], Predicted=0.0001
song=['Attention'], Predicted=0.0000
song=['Mask Off'], Predicted=0.8157
song=['Congratulations'], Predicted=0.0009
song=['Swalla (feat. Nicki Minaj & Ty Dolla $ign)'], Predicted=0.2084
song=['Castle on the Hill'], Predicted=0.0002
song=['Rockabye (feat. Sean Paul & Anne-Marie)'], Predicted=0.0784
song=['Believer'], Predicted=0.0277
song=['Mi Gente'], Predicted=0.0001
song=['Thunder'], Predicted=0.0000
song=["Say You Won't Let Go"], Predicted=0.0000
song=["There's Nothing Holdin' Me Back"], Predicted=0.0005
song=['Me Rehúso'], Predicted=0.0000
song=['Issues'], Predicted=0.0000
song=['Galway Girl'], Predicted=0.0116
song=['Scared to Be Lonely'], Predicted=0.0016
song=['Closer'], Predicted=0.0000
song=['Symphony (feat. Zara Larsson)'], Predicted=0.0024
song=['I Feel It Coming'], Predicted=0.0175
song=['Starboy'], Predicted=0.0033
song=['Wild Thoughts'], Predicted=0.0001
song=['Slide'], Predicted=0.3404
song=['New Rules'], Predicted=0.0310
song=['1-800-273-8255'], Predicted=0.0000
song=['Passionfruit'], Predicted=0.3118
song=['rockstar'], Predicted=0.0063
song=['Strip That Down'], Predicted=0.0000
song=['2U (feat. Justin Bieber)'], Predicted=0.0009
song=['Perfect'], Predicted=0.0000
song=['Call On Me - Ryan Riback Extended Remix'], Predicted=0.0060
song=['Feels'], Predicted=0.0002
song=['Mama'], Predicted=0.0000
song=['Felices los 4'], Predicted=0.0531
song=['iSpy (feat. Lil Yachty)'], Predicted=0.2524
song=['Location'], Predicted=0.0000
song=['Chantaje'], Predicted=0.1379
song=['Bad and Boujee (feat. Lil Uzi Vert)'], Predicted=0.3374
song=['Havana'], Predicted=0.0000
song=['Solo Dance'], Predicted=0.4176
song=['Fake Love'], Predicted=0.0172
song=['Let Me Love You'], Predicted=0.0003
song=['More Than You Know'], Predicted=0.0008
```

4.1.3 Analysis

Confusion Matrix and Measurement for ANN Classification

Predict Class			
Actual Class		Class = 1	Class = 0
	Class = 1	127	13
	Class = 0	9	451
Accuracy	96.33%		
Precision	0.9338(Top=1); 0.972 (non-top=0)		
Recall	0.9071 (Top=1); 0.9804 (non-top=0)		
Cost	127*0 + 13*50 + 9*50 + 451*0 = 1100		
F-measure	2*127/(2*127+13+9) = 92.03%		

From the result, there are 140 top songs and 460 non-top songs. However over these 140 top songs there are 13 song actually is not a top song. Also, in the of non-top songs list, there also have 9 songs never been mark out. Overall, result have 13 false positive and 9 false negative and the accuracy is 96.33 percentage. With this, we could conclude that Artificial Neural Networks is a flexible classifier to solve the non-linear dataset. As a component of an ANN fails, the net continues to operate. It performs very fast after training as well as gives a high accuracy rate.

4.2 Naïve Bayes Classifier

4.2.1 Methodology

Naïve Bayes Classifier Unlike k-NN which makes predictions only based on local information, Naïve Bayes Classifier is a probabilistic framework that derives the class for unknown record based on the global distribution of data. The core rationale of Bayesian Classifier is to calculate the probability of unknown record belonging to certain class, given its other attributes. The class with largest probability is assigned to the unknown record. Such probability can be denoted as following
$P(C|A_1, A_2, \ldots, A_n)$, where,

- $P(A|C)$ is the probability of hypothesis A given the event C, a posterior probability.
- $P(C|A)$ is the probability of event C given that the hypothesis A is true.
- $P(A)$ is the probability of hypothesis A being true (regardless of any related event), or prior probability of A.
- $P(C)$ is the probability of the event occurring (regardless of the hypothesis).

According to Bayes theorem, It is equivalent to maximize $P(A_1, A_2, \ldots, A_n|C)P(C)$in which $P(A_1, A_2,\ldots, A_n|C) = P(A_1|C)\, P(A_2|C) \ldots P(A_{n-1}|C)\, P(A_n|C)\, P(C)$. $P(C)$ is simply the frequency of class C in global scope. To calculate $P(A_i|C)$ in which A_i corresponds to the values of song attributes. In our case, the class consists of top song and non-top songs. The songs were assigned to one of the classes according to the attribute values. However, Naïve Bayes Classifier may not be so accurate for this dataset, because independence among different attributes cannot be proved.

4.2.2 Algorithm: Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It is specifically used when the features contain continuous values other than categorical values. It is also assumed that all the features are following a gaussian distribution i.e. normal distribution.

4.2.3 Implementation

Importing Python Machine Learning Libraries

Python 3 together with Scikit-Learn are used to implement the Naive Bayes Classifier.

```
# To model the Gaussian Navie Bayes classifier
from sklearn.naive_bayes import GaussianNB
import pandas as pd
```

Data Slicing

We can easily perform the data slicing using sklearn's train_test_split() method which randomly slices the data into training and test set

```
from sklearn.model_selection import train_test_split
songs_train, songs_test = train_test_split(concated_songs,
                                           test_size = 0.20,
                                           random_state = 10)
```

Using above code snippet, we have divided the data into training and test set, with 80% of the data used for training and 20% used for testing.

Furthermore, to improve the accuracy and data utilization, we incorporated the k-fold cross validation process to randomly partition the data set into *5* equal sized subsamples.
Of the 5 subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k – 1 = 4 subsamples are used as training data. The cross-validation process is then repeated *5* times, with each of the *5* subsamples used exactly once as the validation data, such that each data is utilized for both training and testing.

```
concated_songs = pd.concat([top,non_top])
concated_songs = shuffle(concated_songs)
for i in range (0,5):
songs_train1 = concated_songs[0:120*(i):]
songs_train2 = concated_songs[120*(i+1):600:]
songs_train = pd.concat([songs_train1,songs_train2])
songs_test = concated_songs[120*(i):120*(i+1):]
```

Training

Training process involves finding a model for class attributes as a function of the class label. It sets the probability for each class label and counts the total times a concept occurs in the training set for each class label. Then the probability of a concept can be obtained by a simple division. After that, the conditional probability for each attribute under each concept is calculated.

```
gnb.fit(songs_train[used_features].values,
        songs_train["top"].values)
```

Using the above code snippets which makes use of the fit() function of the GaussianNB library, we conducted supervised training for the training data against the target value, i.e. Class 1 = top song, Class 0 = non-top song.

Testing

The other important process in classification is to assign a class label to an unseen record. This process finds a class label for an input instance. It calculates the probability for each testing instance under each class label. The probability is obtained by (1) multiply together each attribute's probability with a specific value in an instance for the class label and then (2) multiply the probability of the class label. After that, the class label with largest probability will be assigned to the instance.

```
prediction = gnb.predict(songs_test[used_features])
```

Using the above code snippets which makes use of the predict() function of the GaussianNB library, we assign the respective class label to the testing data so as to predict whether the input song is a top song or a non-top song.

4.2.4 Results

1) Results of *Naïve Bayes Classifier* using the train_test_split data slicing method:
   Number of mislabeled points out of a total 120 points: 24, performance 80.00%

```
non_top_song=pd.read_csv('non_top_460.csv', header=0)
non_top_song["top"] = 0
non_top = non_top_song[features]

top_song = pd.read_csv('top_140.csv', header=0)
top_song['top'] = 1
top = top_song[features]

concated_songs = pd.concat([top,non_top])

songs_train, songs_test = train_test_split(concated_songs, test_size = 0.20, random_state = 10)

gnb.fit(songs_train[used_features].values,
        songs_train["top"].values)

prediction = gnb.predict(songs_test[used_features])

# Print results
print("Number of mislabeled points out of a total {} points : {}, performance {:05.2f}%"
        .format(
            songs_test.shape[0],
            (songs_test["top"] != prediction).sum(),
            100*(1-(songs_test["top"] != prediction).sum()/songs_test.shape[0])
))

prediction
```

```
Number of mislabeled points out of a total 120 points : 24, performance 80.00%

array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0], dtype=int64)
```

Figure 1 – Implementation and results of Naïve Bayes Classifier using train_test_split data slicing

2) Results of Naïve Bayes Classifier using 5-fold cross validation:
   For the five iterations:
   Number of mislabeled points out of a total 120 points: 19, performance 84.17%
   Number of mislabeled points out of a total 120 points: 14, performance 88.33%
   Number of mislabeled points out of a total 120 points: 28, performance 76.67%
   Number of mislabeled points out of a total 120 points: 29, performance 75.83%
   Number of mislabeled points out of a total 120 points: 21, performance 82.50%
   An average of 81.67% accuracy is obtained.

```
non_top_song=pd.read_csv('non_top_460.csv', header=0)
non_top_song["top"] = 0
non_top = non_top_song[features]

top_song = pd.read_csv('top_140.csv', header=0)
top_song['top'] = 1
top = top_song[features]

concated_songs = pd.concat([top,non_top])
concated_songs = shuffle(concated_songs)

for i in range (0,5):

    songs_train1 = concated_songs[0:120*(i):]
    songs_train2 = concated_songs[120*(i+1):600:]
    songs_train = pd.concat([songs_train1,songs_train2])

    songs_test = concated_songs[120*(i):120*(i+1):]

    gnb.fit(songs_train[used_features].values,
            songs_train["top"].values)

    prediction = gnb.predict(songs_test[used_features])

    # Print results
    print("Number of mislabeled points out of a total {} points : {}, performance {:05.2f}%"
          .format(
              songs_test.shape[0],
              (songs_test["top"] != prediction).sum(),
              100*(1-(songs_test["top"] != prediction).sum()/songs_test.shape[0])
    ))

    print(prediction)
```

Figure 2 – Implementation of Naïve Bayes Classifier using train_test_split data slicing

```
Number of mislabeled points out of a total 120 points : 19, performance 84.17%
[0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0
 1 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1
 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0
 0 0 0 1 0 0 1 0 0]
Number of mislabeled points out of a total 120 points : 14, performance 88.33%
[0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 1 0 0]
Number of mislabeled points out of a total 120 points : 28, performance 76.67%
[0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1
 1 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 0
 0 1 0 1 0 0 0 0 0]
Number of mislabeled points out of a total 120 points : 29, performance 75.83%
[0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0
 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1
 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0
 0 0 0 0 1 1 0 0 0]
Number of mislabeled points out of a total 120 points : 21, performance 82.50%
[0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1
 0 0 0 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 1
 0 0 0 0 0 1 0 0 0]
```

Figure 3 –Results of Naïve Bayes Classifier using train_test_split data slicing

4.2.5 Analysis

Confusion Matrix and Measurement

Predict Class				
Actual Class		Class = 1	Class = 0	Weight Average
	Class = 1	125	15	23.3%
	Class = 0	95	365	76.6%
Accuracy	81.67%			
Precision	0.568 (Top=1); 0.960 (non-top=0); 0.791 (Weight Average)			
Recall	0.893 (Top=1); 0.793 (non-top=0); 0.809 (Weight Average)			
Cost	125*0 + 15*50 + 95*50 + 365*0 = 5500			
F-measure	2*125/(2*125+95+15) = 69.44%			

Naïve Bayes algorithm identified instances correctly at a rate of 81.67% within 0.04 second. This implies that the selected attributes are independent enough to have a decent level of accuracy but still have some dependences.

The inaccuracy may be caused by that the assumption of the independence of all the attributes may not hold. For example, from the results of the attributes' inter-relationship heatmap, the loudness and the energy level are positively correlated.

4.3 Decision Tree

4.3.1 Methodology

For this project, decision tree builds classification model in the form of a tree structure. The final result is a tree with decision nodes and leaf nodes. The advantage of using decision tree is to produce a tree that people can understand. The aim is to get the smallest tree and top down tree induction methods use some kind of heuristic and the most popular heuristic to produce pure nodes is an information theory based heuristic. The idea of information theory is to use entropy to measure information in bits. The formula is:

entropy(P1,P2,…,Pn) = -P1logP1-P2logP2…-PnlogPn

Information gain (amount of information gained by knowing the value of the attribute) = (Entropy of distribution before the split) – (entropy of distribution after the split)

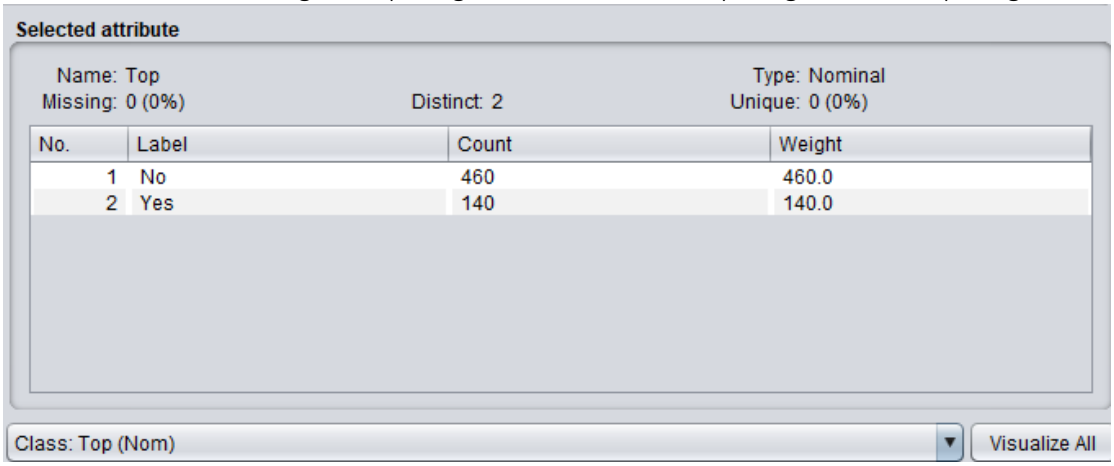
Classifier:

In this project, we use J48 tree classifier. In the WEKA data mining tool, J48 is an open source Java implementation of the C4.5 algorithm. The basic steps in the algorithm are: [1]

1. In case the instances belong to the same class the tree represents a leaf so the leaf is returned by labeling with the same class.
2. The potential information is calculated for every attribute, given by a test on the attribute. Then the gain in information is calculated that would result from a test on the attribute.
3. Then the best attribute is found on the basis of the present selection criterion and that attribute selected for branching.

4.3.2 Implementation and analysis:

1. First combine the top and non-top songs data into one csv file. Add one more column named "Top" to show whether the song is a top song. We have 460 non-top songs and 140 top songs.



2. Under the classifier, choose J48, use the data to train and select 10 folds cross-validation to test. Use the default pruning and other default settings, here is the result we can get:

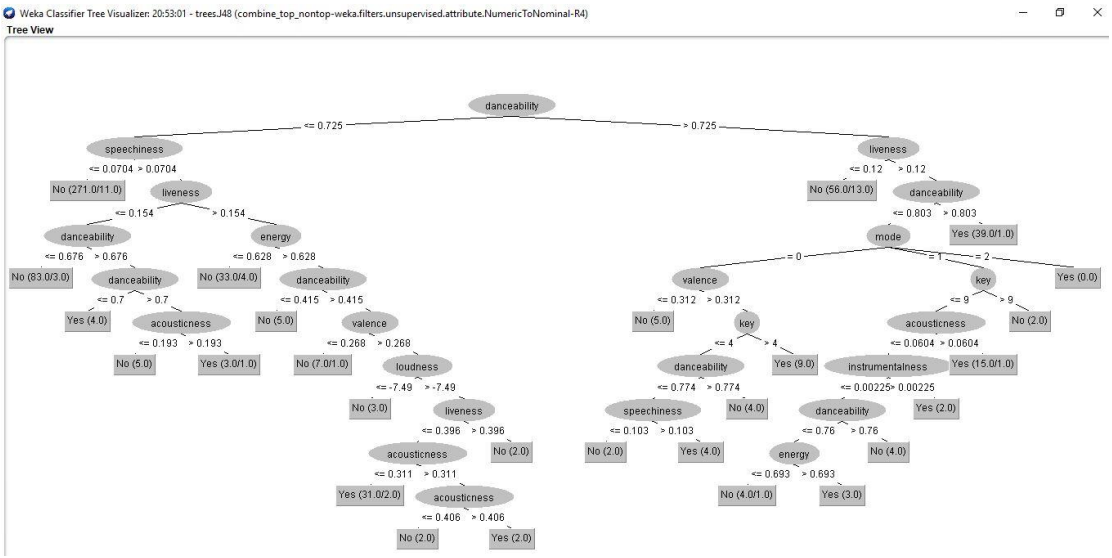
```
Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         491               81.8333 %
Incorrectly Classified Instances       109               18.1667 %
Kappa statistic                          0.4832
Mean absolute error                      0.2022
Root mean squared error                  0.3952
Relative absolute error                 56.4367 %
Root relative squared error             93.4292 %
Total Number of Instances              600
```

We get an 81.8% accuracy in 0.02 seconds. Compare to the simplest method "always choose the most" which provides 460/600=76.7% accuracy, J48 has great improvement.

In addition, we can easily visualize how the decision tree looks like as shown below:



If there are two numbers in one bracket, for example (271.0/11.0), the former number 271.0 means the correctly predicted number of instances, the latter number 11.0 means the wrongly predicted number of instances. The tree has 27 leaves and the number of the tree is 52.

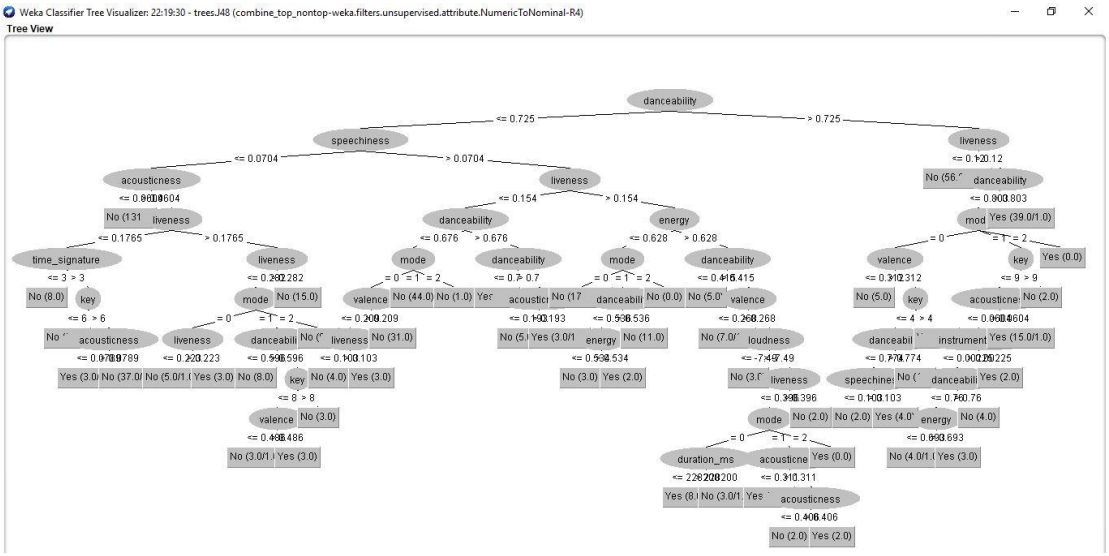
3. If we don't use prune, here is the result:

```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         490               81.6667 %
Incorrectly Classified Instances       110               18.3333 %
Kappa statistic                          0.4824
Mean absolute error                      0.1944
Root mean squared error                  0.4065
Relative absolute error                 54.267  %
Root relative squared error             96.1073 %
Total Number of Instances              600
```

We can get an 81.7% accuracy which does not have much difference compare to pruned model. However, the tree is much more complicated as shown below:



The tree has 50 leaves and the size of the tree is 94.

4.3.3 Analysis

Confusion Matrix and Mesurement

	Predict Class		
Actual Class		Class = No	Class = Yes
	Class = No	409	51
	Class = Yes	58	82
Accuracy	81.83%		
Precision	0.876 (Class = No); 0.617 (Class = Yes)		
Recall	0.889 (Class = No); 0.586 (Class = Yes)		
Cost	409*0+51*50+58*50+82*0= 5450		
F-measure	2*409/(2*409+58+51) = 88.24%		

Pruning allows us to avoid over-fitting. In this case, simplifying a decision tree gives us a slightly better result. Overall the J48 classifier gives an 81.8% accuracy.

4.4 K Nearest Neighbour

4.4.1 Methodology

In pattern recognition, the k-nearest neighbors' algorithm (k-NN) is a non-parametric method used for classification. In both cases, the input consists of the k closest training examples in the feature space. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

4.4.2 Implementation

1. Importing Python Machine Learning Libraries

Python 3 together with Scikit-Learn are used to implement the K Nearest Neighbors Classifier. Together with other libraries to split, normalize the training data and classification report.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

2. Data Slicing

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 11].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
```

The above code divides the data into training and testing dataset.

3. Data normalization

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

StandardScaler transforms data such that its distribution will have a mean value 0 and standard deviation of 1. Given the distribution of the data, each value in the dataset will be normalized to a 0-1 scale using the above code. Ensuring normalised feature values implicitly weights all features equally in their representation.

Template matching & interpolation

KNN does not use training data points to do any generalization unlike any other supervised machine algorithm. In other words, there is no explicit training phase, or it is kept to the minimal. KNN keeps all the training data and use them for the testing phase. It is one of the most time and space consuming classification method.

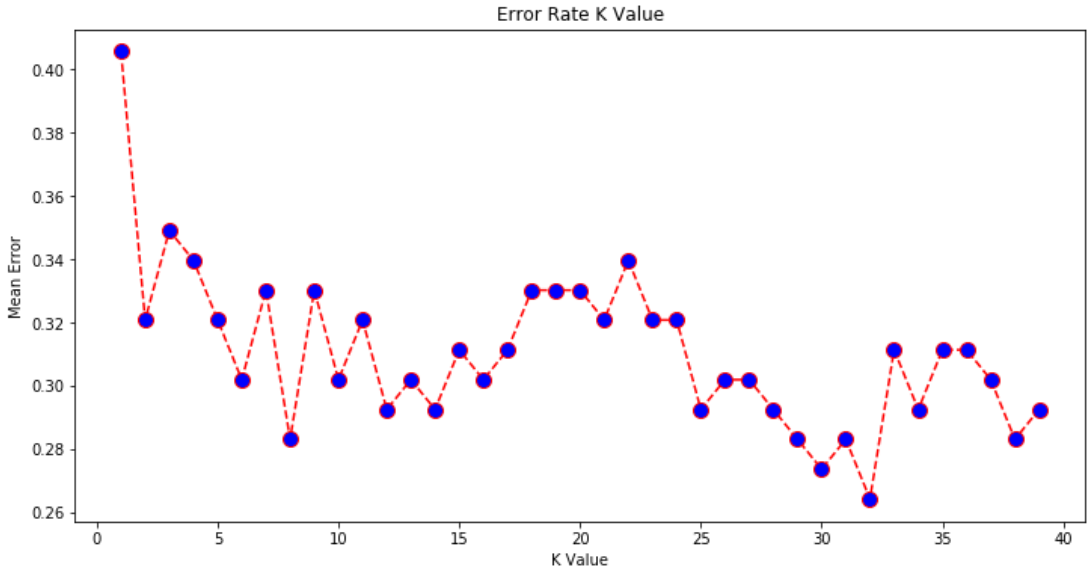
```
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Using the above code snippets, we implemented the KNN classifier and train the classifier with songs data.

4. Comparing Error Rate with the K Value

To get the best K value that yields the best result, we must find out error rate of each K value and make a comparison. One way to find the best value of K is to plot the graph of K value and the corresponding error rate for the dataset.

```
error = []
# Calculating error for K values between 1 and 40
for i in range(1, 40):
   knn = KNeighborsClassifier(n_neighbors=i)
   knn.fit(X_train, y_train)
   pred_i = knn.predict(X_test)
   error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```



Given the graph above, the mean error rate is the lowest when K = 32. When K = 32 we have the highest accuracy when classifying songs.

5. Testing

Testing is an important process in classification is to assign a class label to an unseen record. This process finds a class label for an input instance. It looks for the number of neighbors given a point and the K value. For instance, if K=5, KNN looks for 5 nearest neighbor and classify the test data based on most of the classes it finds.

```
y_pred = classifier.predict(X_test)
```

Using the above code snippets which makes use of the predict () function of sklearn.svm.libsvm library, we classify and assign a class label for the testing data to predict whether the input song is a top song or a non-top song.

6. Results

Results of KNN Classifier using the train_test_split data slicing method:
Number of mislabeled points out of a total 600 points: 34, performance 71.66%

```
[0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0
 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1]
```

4.4.3 Analysis

Confusion Matrix and Measurement

Predict Class			
Actual Class		Class = 1	Class = 0
	Class = 1	101	39
	Class = 0	127	333
Accuracy	72%		
Precision	0.443 (Top=1); 0.105 (Top =0)		
Recall	0.721(Top=1); 0.276 (Top=0)		
Cost	a*0 + b*50 + c*50 + d*0 = 8300		
F-measure	2*101/(2*101+127+39) = 55.89%		

KNN algorithm identified instances correctly at a rate of 72 % within 2 second. This implies that the selected features of the song might be overlapped. As we have many features, it is very possible that some songs of different class contain similar feature values causing KNN algorithm to misclassify.

5.Comparison of Classification Techniques

5.1 Comparison Table

	Accuracy	Precision	Recall	Cost	F-measure
ANN	0.963	0.952	0.943	1100	0.920
NBC	0.817	0.791	0.809	5500	0.694
Decision Tree	0.818	0.876	0.889	5450	0.882
KNN	0.720	0.443	0.721	8300	0.559

Comparing the result of our own implemented classification algorithms, ANN has achieved the best performance with highest accuracy, precision, recall and lowest cost. ANN also achieves a highest F1 score which is the harmonic mean of precision and sensitivity. This might be because of the powerfulness of neural networks being capable of solving most of the classification problems well.

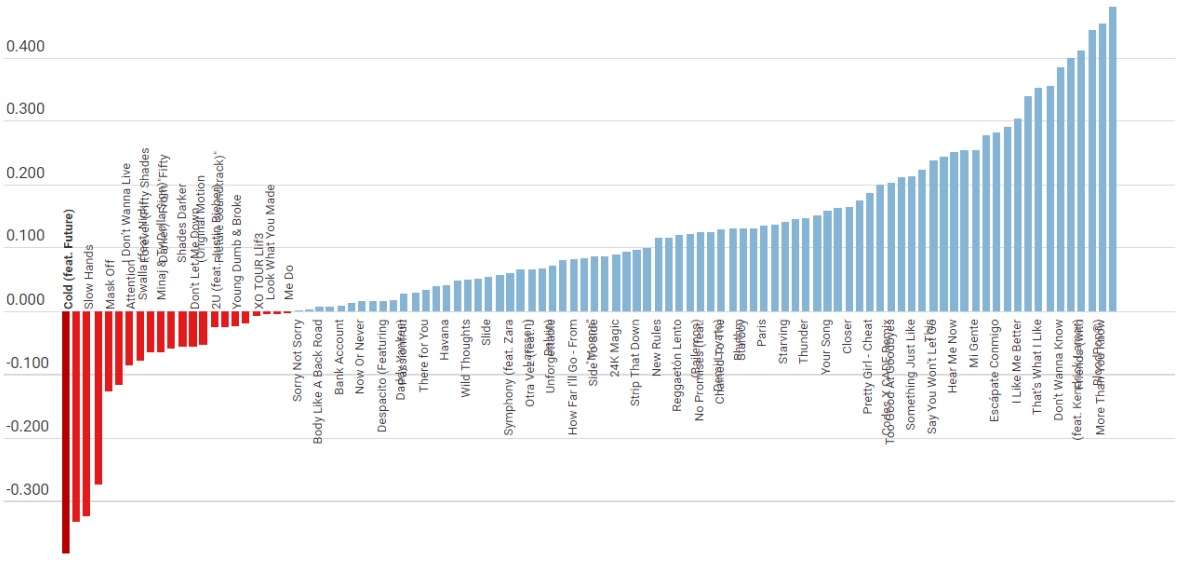
5.2 Discussion of Pros and Cons

ANN	ANN is flexible to solve the non-linear dataset and detect complex non-linear relationships since when one of the components of an ANN fails, the next continues to operate. Neural Networks train the neurons on the entire training set, hence is very robust and unsusceptible to noises and outliers in the training data. In our experiment, ANN outperforms the other techniques in terms of all indicators. It performs very fast after training but training the network may require a large amount of time. In order to realize cost-efficient, ANN is suitable for dataset with large number of input features. However, it is usually hard to train neural networks with different combinations of hyperparameters, including learning rate, number of hidden neurons and number of validation checks, to find the optimal combination of the hyperparameters for our problem. Neural networks are like black box, making it difficult to troubleshoot and hard to understand how it solves the problem.
NBC	NBC allows each attribute to contribute towards the final decision. It is time and space efficient, with: - Training Time Complexity: O(n*d) (n training instances, d dimensions) - Testing Time Complexity: O(c*d) (c class labels, d dimensions) - Space Complexity: O(c*d*m) (c class labels, d dimensions, m max number of values of one attribute) We can look up all probabilities with a single scan. It can handle real and discrete data well. However, assumption about independencies between attributes seldom holds.
Decision Tree	- Training Time Complexity: O((m!)^n) (n attributes, m values in each attribute) - Testing Time Complexity: O(h) (h tree levels) - Space Complexity: O(K^d) (K: average branching factor, d dimensions) Decision tree has simple and efficient testing, and relatively decent performance with high accuracy and low cost. And all values of all attributes are taken into

	considerations. One of the most significant advantages of classification tree is that its result can be represented visually in a tree structure, making it easy to read, interpret and explain. The most important attributes are at the top of the classification tree. Classification tree is also fast to make prediction as it considers the attributes in the decreasing order of their importance. However, the main disadvantage is the calculation of gain split and gain ratio with the entropy. All split conditions should be calculated should be calculated. Therefore, the training is time consuming. And it is prone to over-fitting, having more than necessary final nodes in the classification tree in order to fit the training data better. Pruning techniques need to be incorporated.
KNN	- Training Time Complexity: O(1) - Testing Time Complexity: O(m*n*d*nlgn) (m testing, n training instances, d dimensions, nlgn sortings) - Space Complexity: O((m+n)*d) ((m testing, n training instances, d dimensions) - The k-NN classifier is simple and powerful with no extra effort for tuning complex parameters to build a model. Training is easy and efficient as the classifier for KNN does not need to learn the model, thus the cost of learning process is zero and no assumptions about the characteristics of the concepts to learn have to be made. However, testing is expensive and slow. At each prediction, the classifier needs to compute and sort the distances for all the training data. Besides, the value of K has a strong impact on results for predicting classes. If K is small, KNN model is sensitive to noise. If K is large, many points from other classes will included.

6. Sentiment Analysis

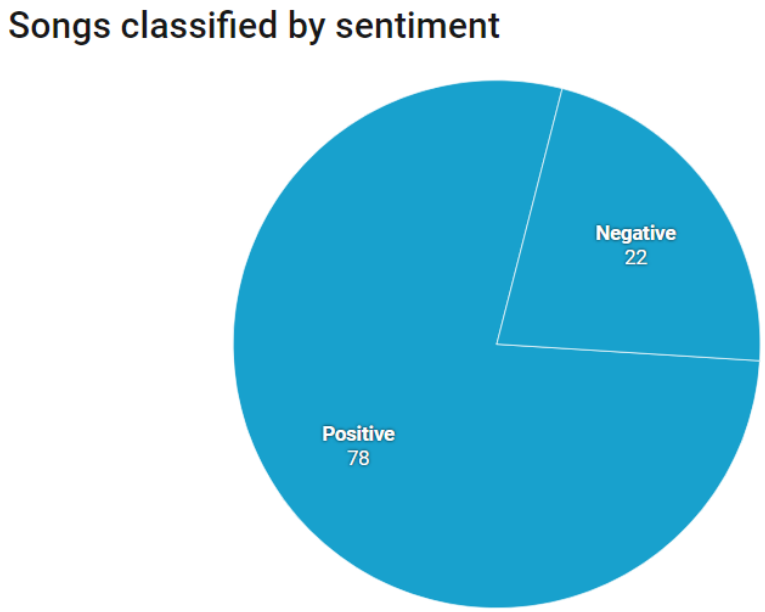
Sentiment Analysis



In this sentiment analysis TextBlob is used because it handles modifier words like: “very”, “extremely”, etc.
Example:

```
TextBlob("great").polarity
## Sentiment(polarity=0.8, subjectivity=0.75)

TextBlob("not great").polarity
## Sentiment(polarity=-0.4, subjectivity=0.75)
TextBlob("very great").polarity
## Sentiment(polarity=1.0, subjectivity=0.9750000000000001)
```



22 songs out of the top 100 songs in the sentiment analysis have a polarity of between -0.003333333 to -0.382389937. These songs contain mostly negative words.
78 of the songs have a polarity between 0 to 0.418. These songs contain mostly positive words.
Conclusion
The top songs sentiment is mainly positive, with polarity more than 0. The average polarity of the top songs is 0.095465718, which also suggest that top songs generally uses more positive words than negatives ones.

7. Word Cloud

Also, we have crawled from the internet for the lyrics of these top 100 songs. First of all, we did some data cleaning on the lyrics such as change all the letter to lower case and remove stop words. Stop words are words such as "and," "the," "where," "how," "what," and "or" and so on. These words do not have much meaning and also will slow down out process on the dataset. Figure 1 is the code of data cleaning for the lyrics.

```
df["lyrics"] = df['lyrics'].str.lower().str.cat(sep=' ')
stop = stopwords.words('english')
df["lyrics"] = df['lyrics'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
a = df['lyrics'].str.lower().str.cat(sep=' ')
```

Figure 1. Data Cleaning

After which, we were finding the most frequent used words among all the lyrics and use the word cloud to represent it. Figure 2 is the code to build a word cloud for the lyrics.

```
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(words))
fig = plt.figure(
    figsize = (40, 30),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Figure 2. Building Word Cloud

After the process, we obtain the following word cloud:

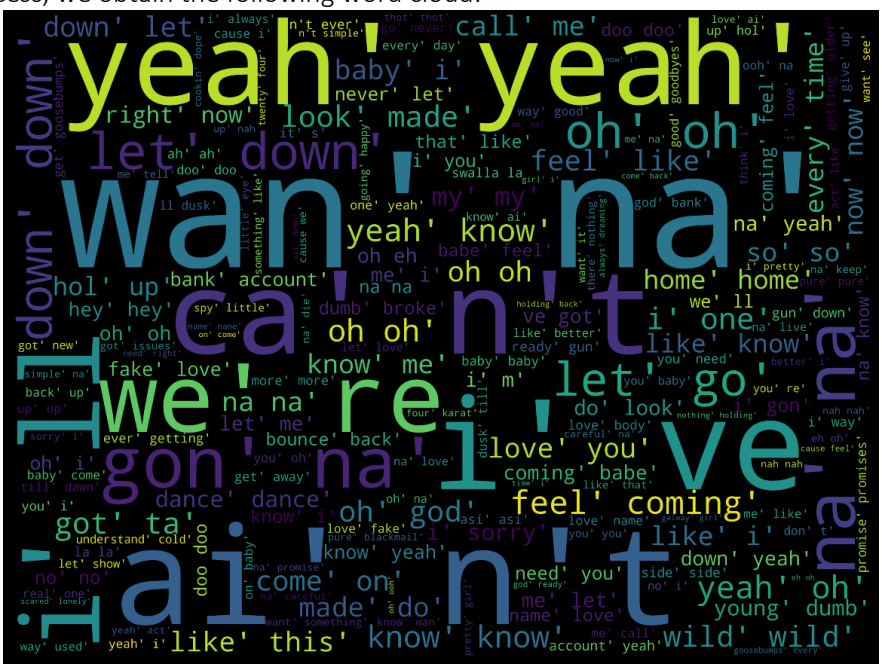


Figure 3. Word Cloud of Top Songs' Lyrics

From the word cloud, we could have a clear view of the frequently used words in the top songs' lyrics. Furthermore, we have computed the top 20 frequent words for all the lyrics. Figure 3 is the code to find the top 20 frequent words.

```
words = nltk.tokenize.word_tokenize(a)
word_dist = nltk.FreqDist(words)
rslt = pd.DataFrame(word_dist.most_common(20),columns=['Word', 'Frequency'])
print(rslt)
```

Figure 4. Search for Top 20 Frequent Words

After process with the Lyrics datasets, we obtain the following results:

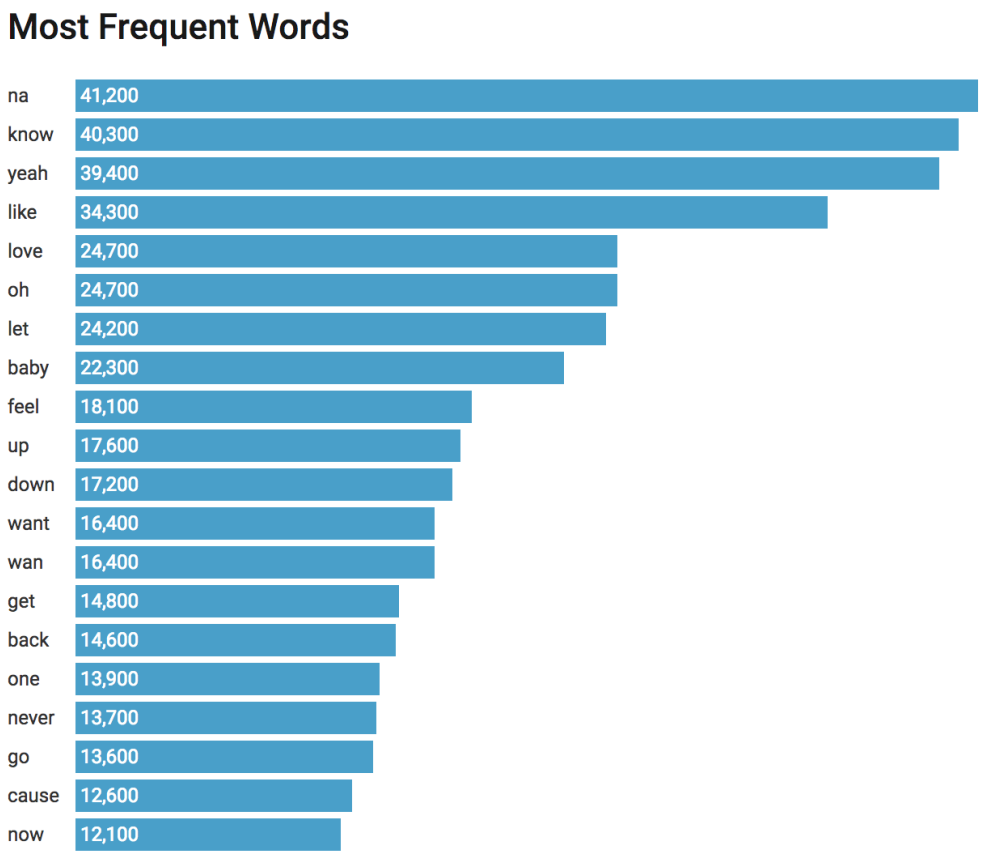


Figure 5. Top 20 Frequent Words

8. Conclusion

In this project, song features dataset was analyzed and mined. Specifically, classification of the dataset was studied and the Classifiers used to do the mining were Artificial Neural Network, k-NN, Naïve Bayes Classifier and Decision Tree. ANN has outperformed the others with ideal accuracy. While the other techniques do yield decent results but not so perfect as the ANN. This may due to the respective constraints of the classification techniques as analyzed before, but also may due to the fact that the classification pattern from top song and non-top song is not that significant. Other than the features from the song itself, in the meanwhile the popularity is also partially dependent on the reputation of the singer, the publicity measures, and the subjective perceptions of the general public. Nonetheless, through this experiment procedure, different classification algorithms were tested, and knowledge discovery process was familiarized, and these strengthen the foundation to conduct data mining task in the future in other popular song composition.