Lyrics_Sentiment_Analysis

June 5, 2019

1 Machine Learning Engineer Nanodegree

1.1 Capstone Project

1.2 Project: Lyrics Sentiment Analysis

Can we predict the genre of one song just analysing the lyrics? What are the most frequent words used for each genre? Which are the positive and negative words in each genre? What are the overall sentiments expressed by a music genre?

The goal of this project is to apply supervised Machine learning to predict what is the genre of a specific song.

This is a text classification problem. Machine Learning uses Natural Processing Language (NPL),classifying data based on past observations by using labels examples in the training data, so that the machine learning algorithms can learn.

I got the idea for this project looking at some projects for text classification problems such as, "Spam Detection" and "Movies review", below some examples:

https://medium.com/@martinpella/naive-bayes-for-sentiment-analysis-49b37db18bf8

https://towardsdatascience.com/sentiment-analysis-with-python-part-1-5ce197074184?gi=eb8658aa284

https://towardsdatascience.com/spam-classifier-in-python-from-scratch-27a98ddd8e73

First we scrape the data, then prepare our dataset by removing punctuation, numbers and also change all the text to lower case. After that, we will tokenize and lemmatize our lyrics. After that, we can use Word Cloud to check the most frequent words used in each genre. To solve the problem we will use Supervised Learning techniques/algorithms, such as Naive Bayes Multinomial, Logistic Regression and Support Vector Machine to classify the song genre, given its lyrics. To see if the model is working well, the data will be split into training set and test set; to check the accuracy of the model, the training set will be 80% and the test set 20%. To evaluate the performance, we will check the Accuracy, Precision, Recall and F1 score metrics. Finally, we will perform the hyperparameter tuning to improve the accuracy of the best model.

1.3 Getting Started

The dataset was taken from Genius by using the python wrapper for the Genius API.

To decide which songs to take I checked what are the top 20 artists/bands of all times, for each genre:

Country
Hard-Rock
Rap

Then I chose the 50 most popular songs from those artists.
For a better analysis I replaced the only instrumental songs with songs with lyrics.
In total, the dataset has 3000 samples and 1000 samples by genre.

1.4 Exploring the data

```
In [1]: # Import libraries necessary for this project
        import pandas as pd
        import numpy as np

        # Allows the use of display() for DataFrames
        from IPython.display import display

        # Pretty display for notebooks
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sb

        #Display markdown formatted output
        from IPython.display import Markdown


        #ignore warnings
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: #load the lyrics dataset
        all_lyrics=pd.read_csv('lyrics.csv')

        #preview the first 2 lines of the dataset
        all_lyrics.head(2)
```

```
Out[2]:     artist                                             lyrics         title
        0  50 Cent  [Produced by Dirty Swift]\n\n[Intro: 50 Cent]\...  21 Questions
        1  50 Cent  [Intro]\nFifty, fifty\nFifty, fifty\nFifty, fi...       9 Shots
```

```
In [3]: #information about the dataset
        all_lyrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 3 columns):
artist    3000 non-null object
lyrics    3000 non-null object
title     3000 non-null object
dtypes: object(3)
memory usage: 70.4+ KB
```

The all_lyrics dataset contains 3000 samples with the following columns:
artist: The name of the singer / band.
lyrics: The lyrics of the song.
title: The title of the song.

1.5 Data Preparing / Cleaning

Adding the column genre
Our dataset is missing a label that is the genre of the songs, we need to add the column genre. First, let's check the artists names that we have in our dataset.

```
In [4]: def bold (string):
            display (Markdown (string))
        bold('**List of Artists:**')

        #getting the list of the artists
        all_lyrics.artist.unique()
```

List of Artists:

```
Out[4]: array(['50 Cent', 'AC/DC', 'Aerosmith', 'Alan Jackson', 'Alice Cooper',
               'Alice in Chains', 'Beastie Boys', 'Big Sean', 'Blue Öyster Cult',
               'Bon Jovi', 'Buck Owens', 'Busta Rhymes', 'Charley Pride',
               'Conway Twitty', 'Deep Purple', 'Def Leppard', 'DMX',
               'Dolly Parton', 'Dr.\xa0Dre', 'Drake', 'Eminem', 'Garth Brooks',
               'George Jones', 'George Strait', 'Guns N Roses',
               'Hank Williams Jr.', 'Ice Cube', 'J. Cole', 'JAY-Z', 'Johnny Cash',
               'Kanye West', 'Kendrick Lamar', 'Kenny Chesney', 'Kenny Rogers',
               'Kiss', 'Led Zeppelin', 'Lil Wayne', 'LL Cool J', 'Loretta Lynn',
               'Ludacris', 'Merle Haggard', 'Metallica', 'Method Man', 'Motörhead',
               'Nas', 'Ozzy Osbourne', 'Reba McEntire', 'Ronnie Milsap', 'Rush',
               'Scorpions', 'Snoop Dogg', 'The Notorious B.I.G.', 'Tim McGraw',
               'Van Halen', 'Waylon Jennings', 'Willie Nelson', 'ZZ Top', 'Queen',
               'Alabama', 'Jimi Hendrix'], dtype=object)
```

Adding the column "Genre" containing the genre of each artist:

```
In [5]: #adding a column genrer
        def genre(all_lyrics):
           if all_lyrics['artist'] in ('LL Cool J','Beastie Boys','50 Cent','Drake','Big Sean'
              return 'Rap'
           elif all_lyrics['artist'] in ('Bon Jovi','Queen','Jimi Hendrix','Kiss','AC/DC','Aer
              return 'Rock'
           else:
            return 'Country'

        all_lyrics['genre']=all_lyrics.apply(genre,axis=1)

In [6]: #preview the first 2 lines of the dataset
        all_lyrics.head(2)
```

```
Out[6]:     artist                                             lyrics         title  \
        0  50 Cent  [Produced by Dirty Swift]\n\n[Intro: 50 Cent]\...  21 Questions
        1  50 Cent  [Intro]\nFifty, fifty\nFifty, fifty\nFifty, fi...       9 Shots

          genre
        0   Rap
        1   Rap
```

```
In [7]: #preview the last 2 lines of the dataset
        all_lyrics.tail(2)
```

```
Out[7]:             artist                                             lyrics  \
        2998  Jimi Hendrix  They don't know  \nThey don't know  \nLike I k...
        2999  Jimi Hendrix  Well, you've got me floatin' around and 'round...

                               title genre
        2998               Who Knows  Rock
        2999  You've Got Me Floating  Rock
```

```
In [8]: #count the number of songs by genre
        all_lyrics.genre.value_counts().head()
```

```
Out[8]: Rock       1000
        Rap        1000
        Country    1000
        Name: genre, dtype: int64
```

Now, we have the column genre and we can see that we have 1000 samples by genre.
Cleaning the data
Let's print a sample of our lyrics feature and check how messy it is:

```
In [9]: sample = all_lyrics.iloc[1200]['lyrics']
        print(sample)
```

```
[Verse 1]
I've been thinking about
Thinking 'bout sex
Always hungry for something that I haven't had yet
Well, maybe, baby, you got something to lose
Well, I got something I, got something for you

[Chorus]
My way, your way, anything goes tonight
My way, your way, anything goes to

[Verse 2]
Panties round your knees
With your ass in debris
Doing that grind with a push and a squeeze
```

```
Tied up, tied down, up against the wall
Be my rubber made, baby, and we can do it all

[Chorus]
My way, your way, anything goes tonight
My way, your way, anything goes tonight-i-i-i-i, yeah-yeah
My way, your way, anything goes tonight
My way, your way, anything goes tonight!

[Guitar Solo]

[Chorus]
My way, your way, anything goes tonight
My way, your way, anything goes tonight, oh yeah
My way, your way, anything goes tonight-ay-ay-ay-ay, yeah
My way, your way, anything goes tonight
My way, your way, anything goes tonight

[Outro]
Tonight, tonight, tonight
Anything goes tonight
Oh woah woah woah, woah...
Said anything goes tonight
```

Looking at our lyrics feature, we can see that we have some lyrics extra words that may disrupt our analysis, like "Verse" and "Chorus", therefore before starting our analysis we must get rid of them. Besides, we are also going to convert our text to lowercase, remove the brackets and the digits.

```
In [10]: import string

         # removing the newline regexp
         all_lyrics['lyrics'] = all_lyrics['lyrics'].replace('\n',' ', regex=True)

         #removing everything inside the brackets
         all_lyrics['lyrics'] = all_lyrics['lyrics'].str.replace(r'[\(\[].*?[\)\]]', '')

         #converting to lower case
         all_lyrics.lyrics = all_lyrics.lyrics.apply(lambda x: x.lower())

         #removing digits
         all_lyrics.lyrics = all_lyrics.lyrics.apply(lambda x: x.translate(str.maketrans('',''
```

Lets see now, how our samples looks like:

```
In [11]: sample_clean = all_lyrics.iloc[1200]['lyrics']
         print(sample_clean)
```

```
 i've been thinking about thinking 'bout sex always hungry for something that i haven't had ye
```

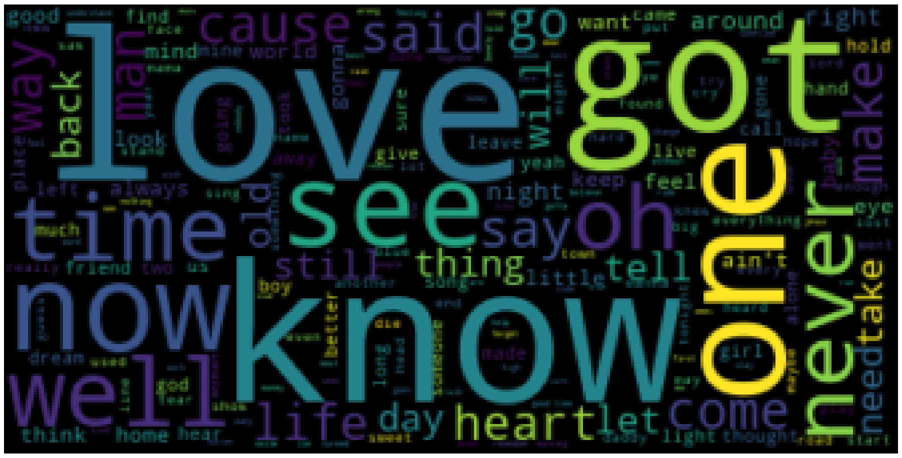
Much better! But we still have a lot of punctuation and stopwords, common words which we want to ignore for our analysis, such as “I”, The”, “Or”, etc. We will not need to remove all of them manually since we will be using the wordcloud package, which is very comprehensive regarding punctuation and stopwords.

For now, we will generate a wordcloud sample and check if our feature lyrics is clean enough to proceed with our analysis.

```
In [12]: #import wordcloud
         from os import path
         from PIL import Image
         from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

```
In [13]: # Groupby by genre
         genre = all_lyrics.groupby('genre')
```

```
In [14]: # Join all lyrics of each of country genre:
         country = " ".join(all_lyrics for all_lyrics in all_lyrics[all_lyrics['genre']=='Coun

         # basic wordcloud generation
         wordcloud = WordCloud().generate(country)
         plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```



Our word cloud is quite nice but if we look carefully, you can see that the words aggregation isn’t really optimal yet. For example, in our word cloud we have the word "said", that for a better analysis should be recognized as belonging to the "say" root. We call this lemmatization . The word cloud package applies only plural normalization, this means that the algorithm only removes the plural "s" from the words.

Token and Lemmatization

Lemmatization is a text normalization technique used in Natural Language Processing (NPL) to prepare the text for future analysis. As already discussed, lemmatization recognize a given words in its root. This means that when we use lemmatization we remove all verbs suffixes ("-ing", "-ed"), for example, the words “playing” and “played” will be both recognized as “play”. There are different algorithms for lemmatization, in this project we are going to use WordNetLemmatizer.

To use lemmatization first we need to tokenize the text. Tokenization will break down a text into words. If we don’t use the tokenization the algorithm WordNetLemmatizer will lemmatize the entire sentence as a word.

```
In [15]: #join all lyrics by genre
         rap = " ".join(lyrics for lyrics in all_lyrics[all_lyrics['genre']=='Rap'].lyrics)
         country = " ".join(lyrics for lyrics in all_lyrics[all_lyrics['genre']=='Country'].ly
         rock = " ".join(lyrics for lyrics in all_lyrics[all_lyrics['genre']=='Rock'].lyrics)
```

```
In [16]: import nltk
         from nltk import word_tokenize

         #tokenizing the lyrics by genre
         tokens_rap = nltk.tokenize.word_tokenize(rap)
         tokens_country = nltk.tokenize.word_tokenize(country)
         tokens_rock = nltk.tokenize.word_tokenize(rock)
```

```
In [17]: #import WordNetLemmatizer
         from nltk.stem.wordnet import WordNetLemmatizer
         wnl = WordNetLemmatizer()

         #lemmatize the tokens lyrics by genre
         lemmatized_rap = [wnl.lemmatize(word, 'v') for word in tokens_rap]
         lemmatized_country = [wnl.lemmatize(word, 'v') for word in tokens_country]
         lemmatized_rock = [wnl.lemmatize(word, 'v') for word in tokens_rock]
```

Words like "I’m", "you’re" were tokenized and were cut as "I", "m", "you", "re", this is not good for our analyses, then we are going to remove the 1 or 2 letters words:

```
In [18]: import re
         #select 1 or 2 letters words
         shortword = re.compile(r'\W*\b\w{1,2}\b')

         #remove the 1 or 2 letters words
         joined_rap = " ".join(lemmatized_rap)
         join_rap = shortword.sub('', joined_rap)

         joined_country = " ".join(lemmatized_country)
         join_country = shortword.sub('', joined_country)
```

```
        joined_rock = " ".join(lemmatized_rock)
        join_rock = shortword.sub('', joined_rock)
```

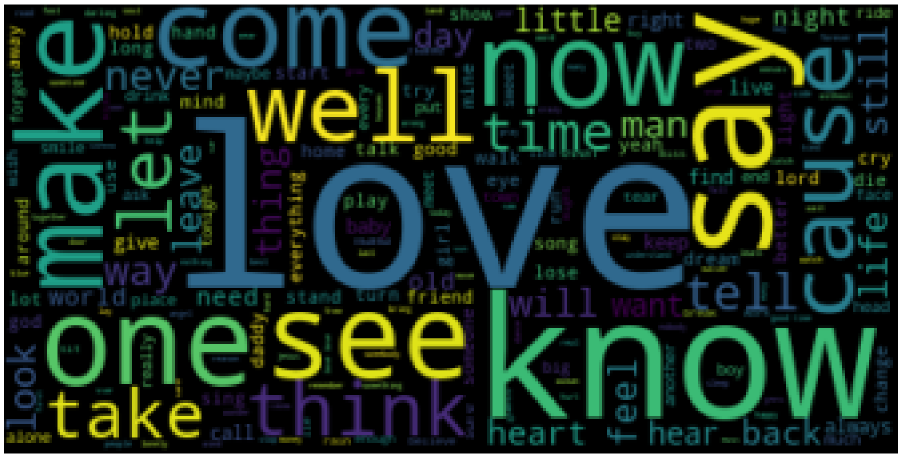
Also the English slang "gonna" and "wanna", had been tokenized as "gon", "na". In this case we are going to replace "gon" for "go" and "wan" for "want".

The lemma also did not recognize fuckin as fuck, then we will do that as below:

```
In [19]: #replace english slang
        lyrics_rap = join_rap.replace("gon ", "go").replace("wan ", "want").replace("fuckin "
        lyrics_country = join_country.replace("gon ", "go").replace("wan ", "want")
        lyrics_rock = join_rock.replace("gon ", "go").replace("wan ", "want")
```

Let's see how our word cloud sample looks like now.

```
In [20]: # basic wordcloud generation
        text = lyrics_country
        wordcloud = WordCloud().generate(text)
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```



As we can see, the word “said” isn't in the word cloud anymore, this means that it was recognized as “say” as per lemmatization step. So, our lyrics are ready to start the analyses. get analysed.

1.6 Exploration Text Analysis

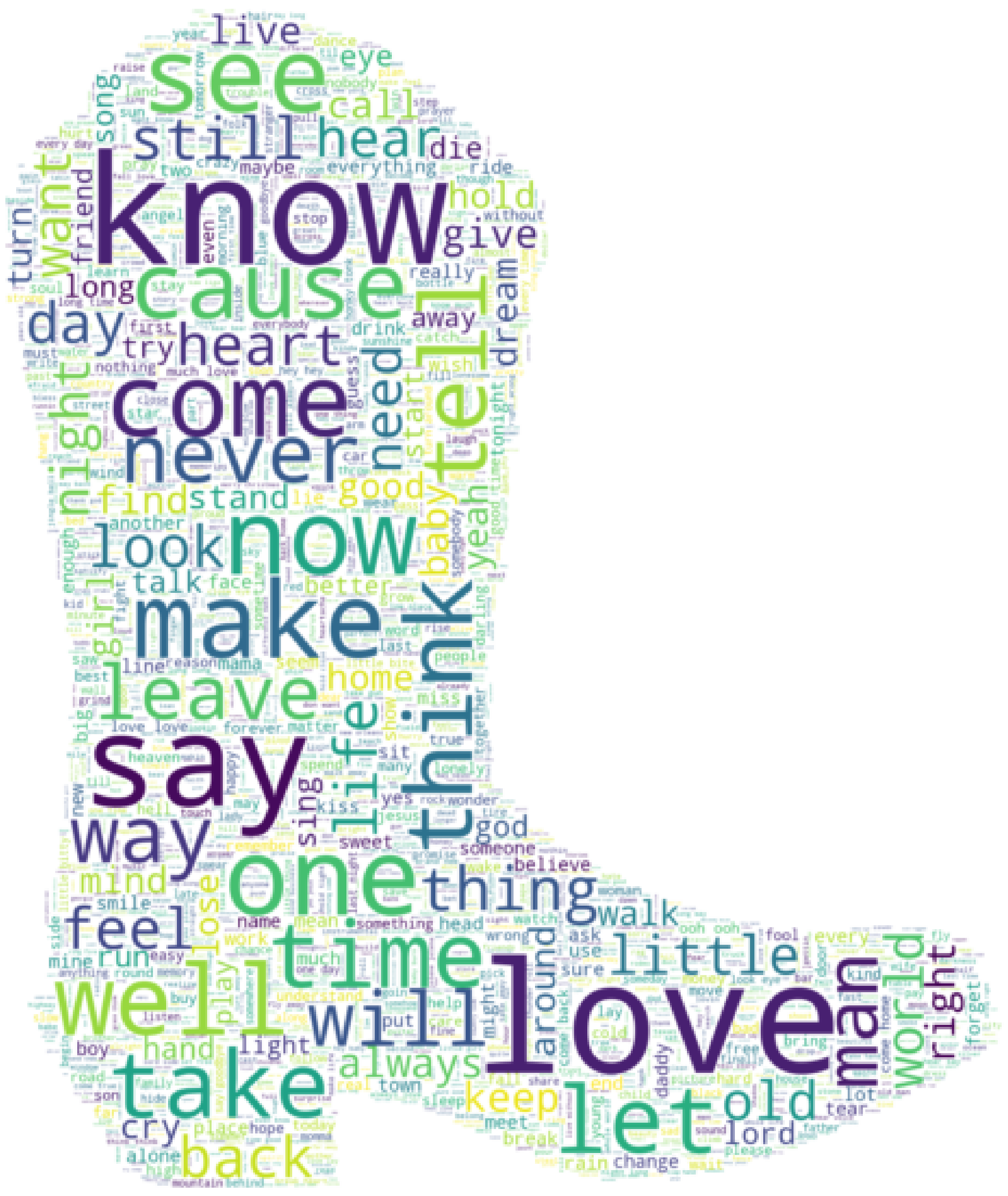
What are the most frequent words used for each genre? Which are the positive and negative words in each genre? The next steps are going to answer those questions.

As mentioned earlier, we will make our analysis with WordCloud package. Word Cloud is a cloud filled with lots of words in different sizes, which represent the frequency of each word.

To make our word cloud nicer, we will change some parameters, like color and shape.
The most frequent words used by Country songs

```
In [21]: #select a mask for country genre
         country_mask = np.array(Image.open("C:\\Users\\jesus\\Desktop\\img\\boots.jpg"))

         #word cloud parameters
         wc = WordCloud(background_color="white", max_words=10000, mask= country_mask, random_s

         # Generate a wordcloud
         wc.generate(lyrics_country)

         bold('**Country Word Cloud:**')

         # show
         plt.figure(figsize=[20,10])
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
         image_colors = ImageColorGenerator(country_mask)
```

Country Word Cloud:



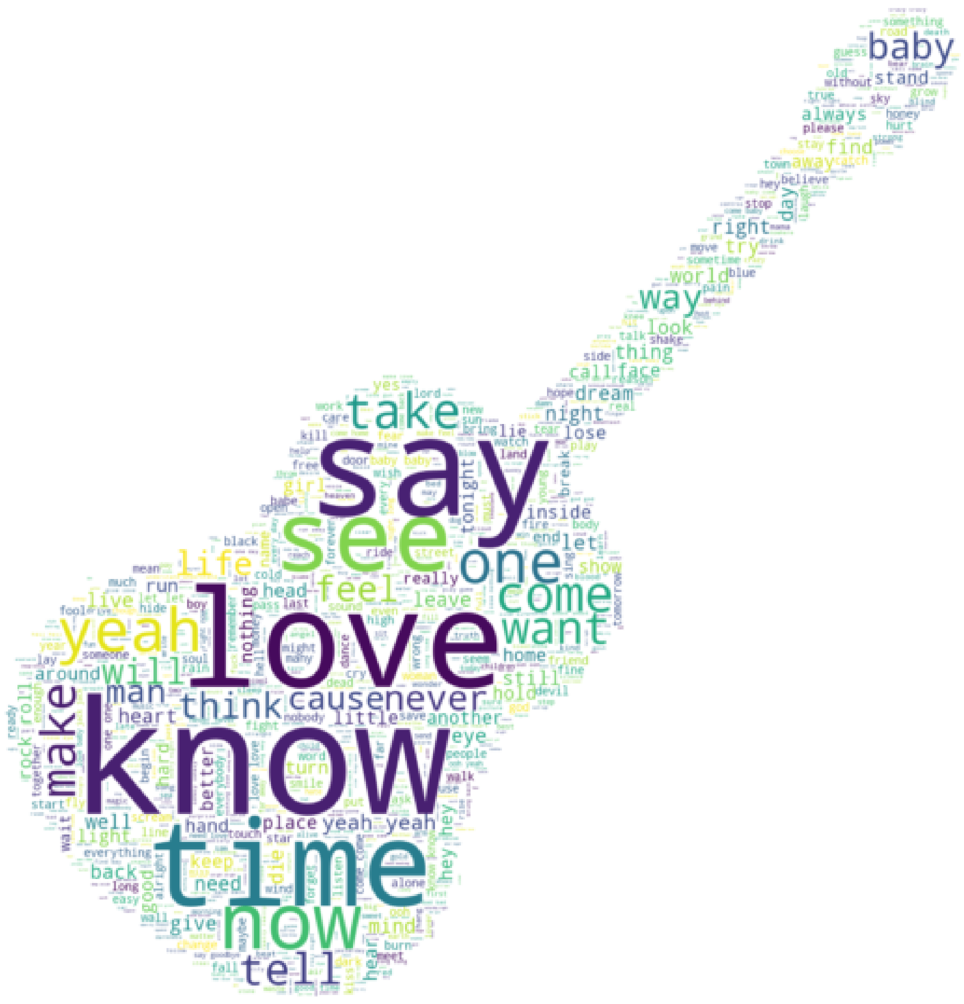
We can see that country mentions "love" frequently, and uses “know” a lot, followed by words as “come”, “think”, “make”, “say” and so on.

The most frequent words used by Rock songs

```
In [54]: #select a mask for country genre
         rock_mask = np.array(Image.open("C:\\Users\\jesus\\Desktop\\img\\rock.jpg"))

         wc = WordCloud(background_color="white", max_words=10000, mask= rock_mask, random_sta

         # Generate a wordcloud
```

```
wc.generate(lyrics_rock)

bold('**Rock Word Cloud:**')

# show
plt.figure(figsize=[20,10])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
image_colors = ImageColorGenerator(rock_mask)
```

Rock Word Cloud:

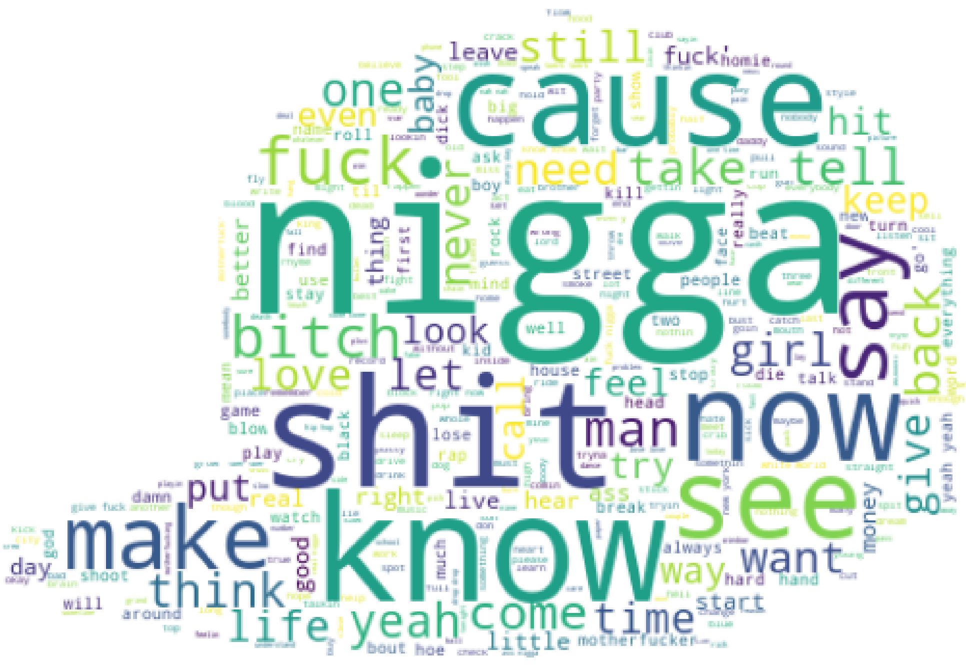


The most frequent words used by rock are quite similar to country. Here we have a lot of “love” and “know”. And also “say” appears frequently. The words “time”, “now”, “come”, “want”, “baby” appear with quite some frequency too.

The most frequent words used by Rap songs

```
In [55]: #select a mask for rap genre
         rap_mask = np.array(Image.open("C:\\Users\\jesus\\Desktop\\img\\cap_.jpg"))

         wc = WordCloud(background_color="white", max_words=10000,mask=rap_mask, random_state=

         # Generate a wordcloud
         wc.generate(lyrics_rap)

         bold('**Rap Word Cloud:**')

         # show
         plt.figure(figsize=[20,10])
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
         image_colors = ImageColorGenerator(rap_mask)
```

Rap Word Cloud:



The rap word cloud is very different from the ones for country and rock. Here we have very offensives words that appear with high frequency.

Word Count

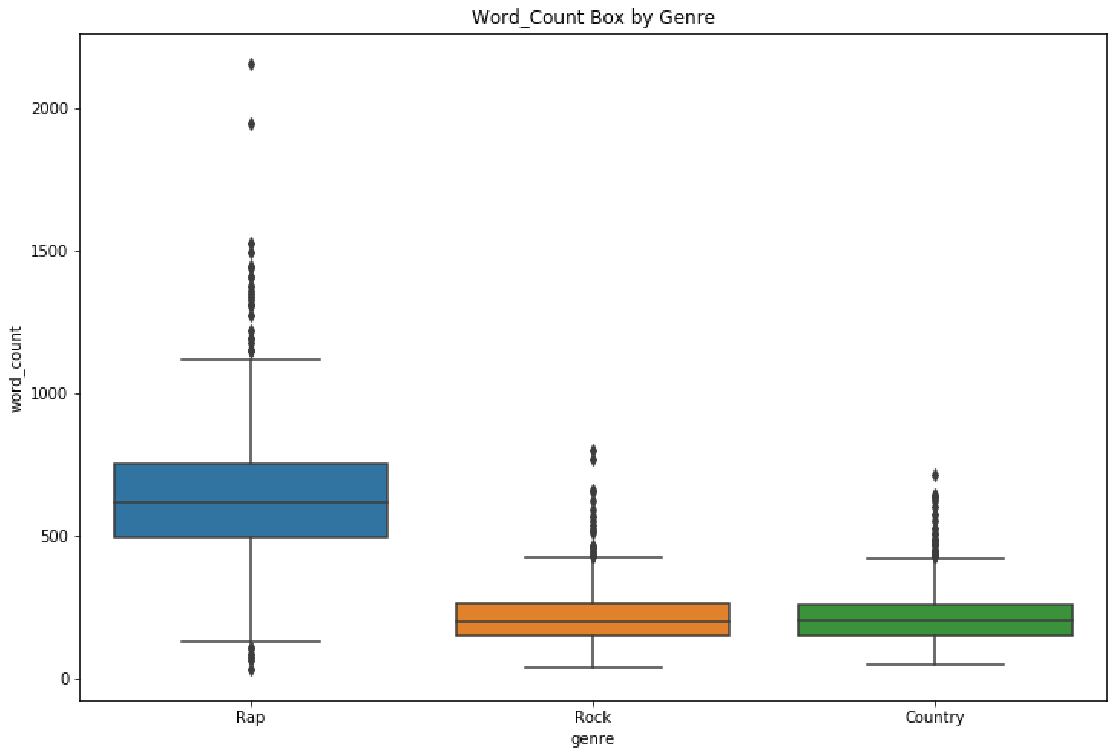
Another interesting data to look into could be how many words a song has on average, per genre: is rap the genre using the biggest number of words, or is it country? How do they stack against each other? Let's analyse our dataset to find out.

```
In [22]: #function to count the words
         def word_count(text_string):
             '''Calculate the number of words in a string'''
             return len(text_string.split())
```

```
In [23]: #insert a column word_count in our dataset
         all_lyrics['word_count'] = all_lyrics['lyrics'].apply(word_count)
         #the statics for word_count
         all_lyrics['word_count'].describe()

Out[23]: count    3000.000000
         mean      353.906667
         std       246.815384
         min        31.000000
         25%       173.000000
         50%       256.000000
         75%       508.000000
         max      2158.000000
         Name: word_count, dtype: float64
```

The mean of the words per song of our dataset is 353 and the maximum is 2158. Let's check now, which genre has more words:

```
In [24]: #boxplot graph
         f,ax = plt.subplots(figsize = (12,8))

         sb.boxplot (x='genre', y='word_count', data=all_lyrics)

         ax.set_title('Word_Count Box by Genre')

Out[24]: Text(0.5, 1.0, 'Word_Count Box by Genre')
```



As we expected rap has uses the most words. And, country and rock have very similar number of words by song.

Positive and Negative Lyrics

To decide if a song is more positive or negative, we will use the Afinn algorithm. Afinn scores in a range between -5 (negative) and +5 (positive). First Afinn preprocessed the text by removing the punctuation and converting to lowercase then it assign the score.

```
In [25]: #import afinn
         from afinn import Afinn

         #set the language to English
         afinn = Afinn(language='en')

         bold('**Affin Score example for love:**')

         #affin score example
         afinn.score('love.')
```

Affin Score example for love:

```
Out[25]: 3.0
```

```
In [26]: #adding the column afinn_score in our dataset
         all_lyrics['afinn_score'] = all_lyrics['lyrics'].apply(afinn.score)
```

```
In [27]: #affin_score statics
         all_lyrics['afinn_score'].describe()
```

```
Out[27]: count    3000.000000
         mean      -12.982667
         std        53.725896
         min      -726.000000
         25%       -22.000000
         50%         0.000000
         75%        14.000000
         max       166.000000
         Name: afinn_score, dtype: float64
```

When we apply Afinn in a long text this may get higher score, just because it contains more words. To fix it, we are going to add the column afinn_adjusted that is the result of afinn_score divided by the count of words.

```
In [28]: #adding the column afinn_adjusted in our dataset
         all_lyrics['afinn_adjusted'] = all_lyrics['afinn_score'] / all_lyrics['word_count'] *

         #affin_score statics
         all_lyrics['afinn_adjusted'].describe()
```

```
Out[28]: count    3000.000000
         mean       -0.496870
         std        12.090328
         min      -100.409836
         25%        -6.705339
         50%         0.000000
         75%         6.060606
         max        73.451327
         Name: afinn_adjusted, dtype: float64
```

```
In [29]: #columns to display
         columns_to_display = ['artist', 'lyrics','title','genre','afinn_adjusted']
```

```
In [30]: #set the most negative songs
         all_lyrics.sort_values(by='afinn_adjusted')[columns_to_display].head(3)
```

```
Out[30]:           artist                                              lyrics  \
         361     Big Sean     ass, ass, ass, ass, ass ass, ass, ass, ass,...
         2506  Snoop Dogg   fuck them other niggas 'cause i'm down for my...
         2171   Motörhead  if you squeeze my lizard i'll put my snake on ...

                          title genre  afinn_adjusted
         361        Dance (A$$)   Rap     -100.409836
         2506  Down 4 My Niggas   Rap      -88.536585
         2171   Killed by Death  Rock      -71.597633
```

```
In [31]: #set the most positive songs
         all_lyrics.sort_values(by='afinn_adjusted')[columns_to_display].tail(3)
```

```
Out[31]:             artist                                              lyrics  \
         1146  George Jones  when with the savior we enter the glory land w...
         2935       Alabama  yes, jesus loves me    yes, jesus loves me   ye...
         2857         Queen  funny how love is everywhere just look and see...

                                    title    genre  afinn_adjusted
         1146  Won't It Be Wonderful There?  Country       54.248366
         2935               Jesus Loves Me  Country       68.907563
         2857           Funny How Love Is     Rock       73.451327
```

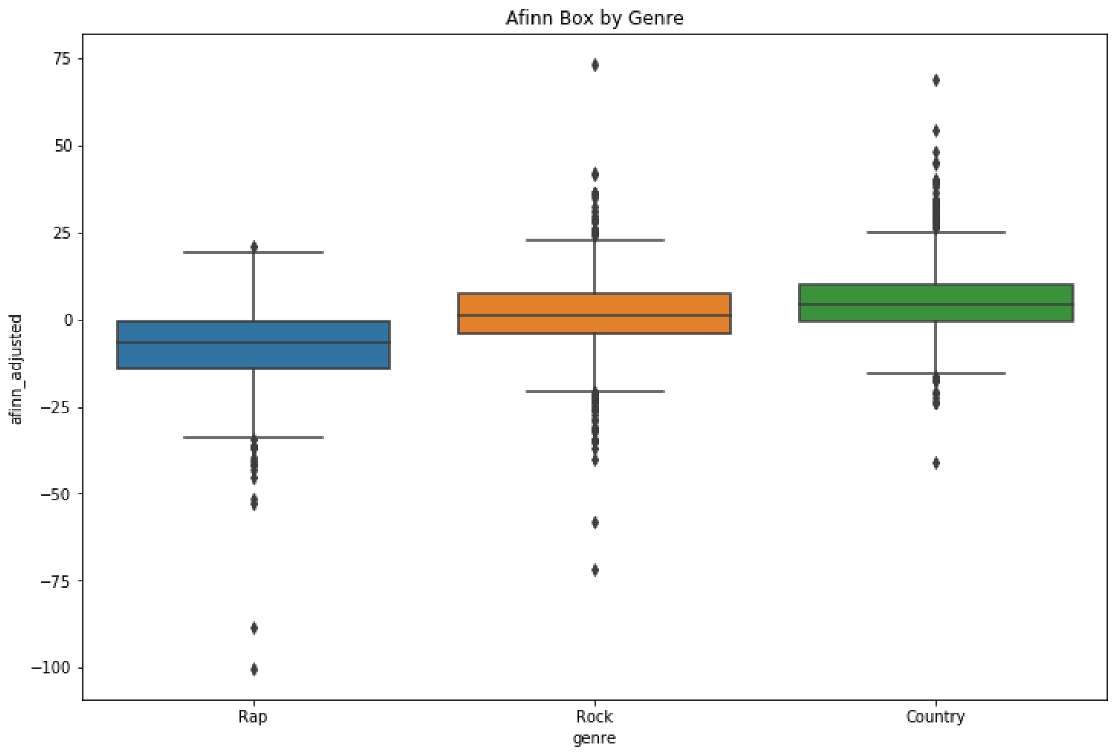
```
In [32]: #affin_score statics by genre
         all_lyrics.groupby('genre')['afinn_adjusted'].describe()
```

```
Out[32]:           count      mean        std         min        25%       50%  \
         genre
         Country  1000.0  5.408293   9.904624  -41.200000  -0.476029  4.275790
         Rap      1000.0 -8.049333  11.009734 -100.409836 -14.082627 -6.835487
         Rock     1000.0  1.150429  11.170763  -71.597633  -4.104698  1.379376

                        75%        max
```

```
genre
Country  10.110969  68.907563
Rap      -0.662252  20.797227
Rock      7.153550  73.451327
```

Considering the mean of afinn_adjusted the Country genre has the most positive songs (5.40), while rap songs has the most negative songs (-8.04). Let's generate a boxplot graph to better visualize the results:

```
In [33]: #boxplot graph
         f,ax = plt.subplots(figsize = (12,8))

         sb.boxplot (x='genre', y='afinn_adjusted', data=all_lyrics)

         ax.set_title('Afinn Box by Genre')

Out[33]: Text(0.5, 1.0, 'Afinn Box by Genre')
```



As mentioned, rap has the most negative songs, while country has the most positive songs. And, Country and Rock are quite similar.

Adding a "feelings" column. Where:

-Score <0 is negative

-Score = 0 is neutral

-Score >0 is positive

```
In [34]: #set the edges to cut
         bin_edges = [-101,0,1,np.inf]

         #set the labels
         bin_names = ['negative', 'neutral', 'positive']

         #adding the column feeling
         all_lyrics['feelings'] = pd.cut (all_lyrics['afinn_adjusted'], bin_edges, labels = bi

         #display first 5 samples
         all_lyrics.head()
```

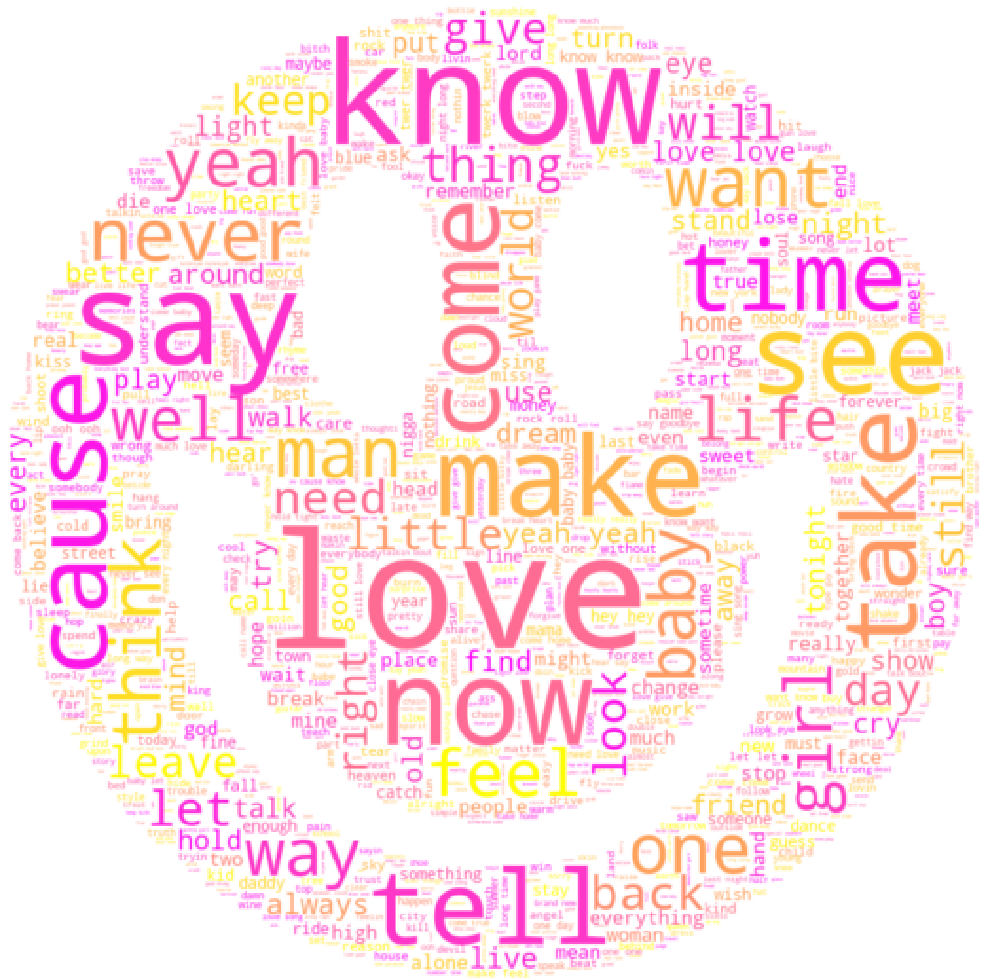
```
Out[34]:     artist                                             lyrics  \
         0  50 Cent     i just want to chill and twist the lye catc...
         1  50 Cent   fifty, fifty fifty, fifty fifty, fifty ferrar...
         2  50 Cent   she loves me, she loves me not yeah, she love...
         3  50 Cent     somethin' special, unforgettable  cent, cen...
         4  50 Cent   it's easy to see when you look at me if you l...

                          title genre  word_count  afinn_score  afinn_adjusted  \
         0         21 Questions   Rap         611        102.0       16.693944
         1              9 Shots   Rap         359        -34.0       -9.470752
         2  A Baltimore Love Thing   Rap         843         16.0        1.897983
         3       Ayo Technology   Rap         716         -3.0       -0.418994
         4            Back Down   Rap         722        -69.0       -9.556787

           feelings
         0  positive
         1  negative
         2  positive
         3  negative
         4  negative
```

Let's check which are the most frequent words used in positives and negatives songs. For that, we are going to use word cloud once again. First, we need to tokenize and lemmatize the text.

```
In [35]: #join lyrics by feeling
         positive = " ".join(lyrics for lyrics in all_lyrics[all_lyrics['feelings']=='positive
         negative = " ".join(lyrics for lyrics in all_lyrics[all_lyrics['feelings']=='negative

         #tokenizing the lyrics by feeling
         tokens_positive = nltk.tokenize.word_tokenize(positive)
         tokens_negative = nltk.tokenize.word_tokenize(negative)

         #lematize the tokens by feeling
         lemmatized_tokens_positive = [wnl.lemmatize(word, 'v') for word in tokens_positive]
         lemmatized_tokens_negative = [wnl.lemmatize(word, 'v') for word in tokens_negative]
```

```
        #remove the 1 or 2 letters words
        joined_positive = " ".join(lemmatized_tokens_positive)
        joined_negative = " ".join(lemmatized_tokens_negative)
        lyrics_long_positive = shortword.sub('', joined_positive)
        lyrics_long_negative = shortword.sub('', joined_negative)

        #replace English slang
        lyrics_positive = lyrics_long_positive.replace("gon ", "go").replace("wan ", "want")
        lyrics_negative = lyrics_long_negative.replace("gon ", "go").replace("wan ", "want").

In [36]: #select mask for positive songs
        positive_mask = np.array(Image.open("C:\\Users\\jesus\\Desktop\\img\\happyb.jpg"))

        #set parameters for positive word cloud
        wc = WordCloud(background_color="white", max_words=10000,colormap="spring", mask= pos

        # Generate a wordcloud
        wc.generate(lyrics_positive)

        bold('**Positive Word Cloud:**')

        # show
        plt.figure(figsize=[20,10])
        plt.imshow(wc, interpolation='bilinear')
        plt.axis("off")
        plt.show()
        image_colors = ImageColorGenerator(positive_mask)
```

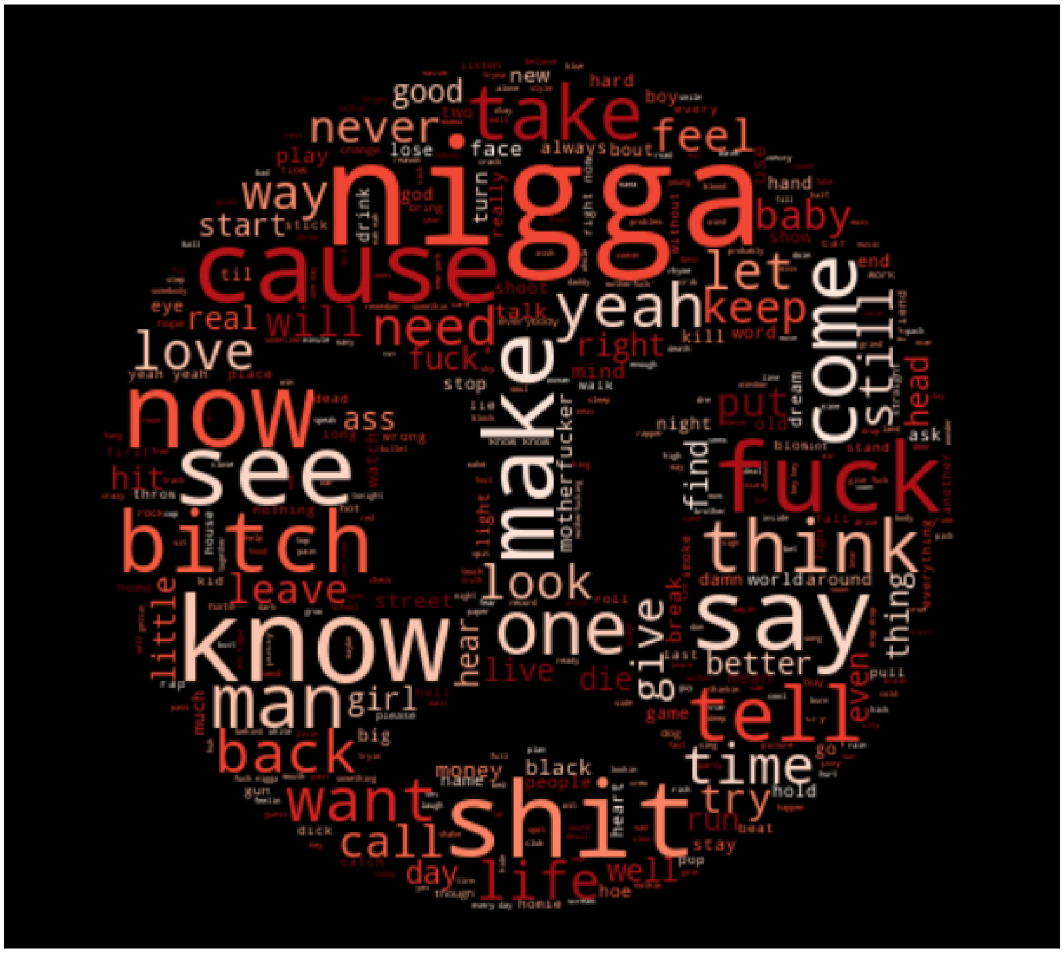
Positive Word Cloud:



The words used most frequently in positive songs are : “know”, “love”, “say”, and “tell”.

```
In [73]: #set parameters for negative word cloud
         negative_mask = np.array(Image.open("C:\\Users\\jesus\\Desktop\\img\\angry.jpg"))

         #set parameters for positive word cloud
         wc = WordCloud(background_color="black", max_words=10000,colormap="Reds", mask= negat

         # Generate a wordcloud
         wc.generate(lyrics_negative)

         bold('**Negative Word Cloud:**')

         # show
         plt.figure(figsize=[20,10])
         plt.imshow(wc, interpolation='bilinear')
         plt.axis("off")
         plt.show()
         image_colors = ImageColorGenerator(negative_mask)
```

Negative Word Cloud:



And for negative songs we have a lot of offensive words.

1.7 Evaluate Algorithm

Model Building

The algorithms below will be evaluated:

- Naive Bayes – Natural Language Processing
- Logistic Regression
- Support Vector Machine

Before we start building our models, let's convert our labels to binary variables:

- 0 for 'country'
- 1 for 'rock'
- 2 for 'rap'

This step is important since sklearn works better with numerical values.

```
In [37]: all_lyrics['genre'] = all_lyrics.genre.map({'Country':0, 'Rock':1, 'Rap':2})
         print(all_lyrics.shape)
         all_lyrics.head()# returns (rows, columns)
```

```
(3000, 8)
```

```
Out[37]:     artist                                             lyrics  \
         0  50 Cent     i just want to chill and twist the lye catc...
         1  50 Cent   fifty, fifty fifty, fifty fifty, fifty ferrar...
         2  50 Cent   she loves me, she loves me not yeah, she love...
         3  50 Cent     somethin' special, unforgettable  cent, cen...
         4  50 Cent   it's easy to see when you look at me if you l...

                           title  genre  word_count  afinn_score  afinn_adjusted  \
         0          21 Questions      2         611        102.0       16.693944
         1               9 Shots      2         359        -34.0       -9.470752
         2  A Baltimore Love Thing      2         843         16.0        1.897983
         3        Ayo Technology      2         716         -3.0       -0.418994
         4             Back Down      2         722        -69.0       -9.556787

           feelings
         0  positive
         1  negative
         2  positive
         3  negative
         4  negative
```

Training and Testing Sets:

```
X_train: Is our training data for the lyrics column
y_train: Is our training data for the label column (genre)
X_test: Is our test data for the lyrics column
y_test: Is our test data for the label column (genre)
```

The training set will be 80% and the test set 20%.

```
In [38]: X = all_lyrics['lyrics']
         y = all_lyrics['genre']
```

```
In [79]: from sklearn.model_selection import train_test_split

         # Split the data into train and test data
         X_train,X_test,y_train,y_test = train_test_split (X,y, test_size = 0.20, random_state
         print('Number of rows in the total set: {}'.format(all_lyrics.shape[0]))
         print('Number of rows in the training set: {}'.format(X_train.shape[0]))
         print('Number of rows in the test set: {}'.format(X_test.shape[0]))
```

```
Number of rows in the total set: 3000
Number of rows in the training set: 2400
Number of rows in the test set: 600
```

Word Count with CountVectorizer

Before using the machine learning algorithm we have to convert text into numbers, we call this process Bag of Words or BoW. The BoW take pieces of the text and counts the frequency of the words in that text.

To implement the BoW we can use the CountVectorizer that will tokenize the string and give a integer ID to each token and count the frequency of those token.

CountVectorizer will also convert all of our data to lowercase, remove all punctuation and stop words.

```
In [80]: from sklearn.feature_extraction.text import CountVectorizer

         #Instantiate the CountVectorizer method
         count_vector = CountVectorizer()

         #Fit the training data and then return the matrix
         training_data = count_vector.fit_transform(X_train)

         #Transform testing data and return the matrix
         testing_data = count_vector.transform(X_test)
```

Evaluation Metrics

Precision, Recall, Accuracy and f1_score will be the metrics to evaluate the performance of a classifier, since considering only accuracy is not a very good metric to check how good the model is doing, especially when we have imbalanced data. Imbalanced data is when there are a lot more samples for one class than for the other class. For example, in spam detection it is common to get a dataset with high samples of not spam emails and lower samples for spam email. In this situation you could get 98% of accuracy and would think that you model did very well, and in fact the 98% accuracy belongs to one class. Our dataset is balanced, we have 1000 samples for each class (Country, Rock, and Rap), and only accuracy wouldn't be a big problem but is interesting to check recall, precision and f1 score to make sure that our model is really doing well. For all four metrics the range is between 0 and 1, value closer to 1 means that our model is doing well. Below, the definition of the metrics used in this project:

- Accuracy measures how much text was predicted correctly given the total number of predictions.

- Precision measures how much text was classified correctly (True Positives) to all positives (True Positives + False Positives): [True Positives/(True Positives + False Positives)]
- Recall (Sensitivity) measures how much text was classified correctly (True Positives) to all text that should be classified as Positive (True Positives + False Negatives): [True Positives/(True Positives + False Negatives)]
- F1_score: Is the harmonic mean between recall and precision.

Multinomial Naive Bayes

The first model that we will build is the Multinomial Naive Bayes that is simple and works very well with text classification. The Multionomial Naive Bayes is our benchmark model.

Naive Bayes is frequently used in text classification problems, because this algorithm gets very high accuracy with small data like our dataset. Naive Bayes is based on Bayes's Theorem. Bayes's Theorem calculates the probability of a hypothesis, based on other probabilities that are related to this hypothesis. In our problem, Bayes's Theorem calculates the probability of a song being country, rock or rap based on the appearance of words in a song.

Bayes formula:

"*P(h|d) = (P(d|h)* P(h)) / P(d)*

P(h|d) is the probability of hypothesis h given the data d. This is called the posterior probability.

P(d|h) is the probability of data d given that the hypothesis h was true.

P(h) is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h. P(d) is the probability of the data (regardless of the hypothesis)"

Retrieved from: https://machinelearningmastery.com/naive-bayes-for-machine-learning/

```
In [81]: #Import Multinominal Naive Bayes
         from sklearn.naive_bayes import MultinomialNB

         #Define the model
         naive_bayes = MultinomialNB()

         #Fit the model
         naive_bayes.fit(training_data, y_train)

         #Make predictions on the test data
         pred_nb = naive_bayes.predict(testing_data)
```

```
In [82]: #Import the metrics from sklearn - accuracy_score, precision_score, recall_score, f1_
         from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, r

         #Calculate the accuracy, precision, recall and f1_score
         acc_nb = accuracy_score (y_test, pred_nb)
         precision_nb = precision_score (y_test, pred_nb, average='weighted')
         recall_nb = recall_score (y_test, pred_nb, average='weighted')
         f1_nb = f1_score (y_test, pred_nb, average='weighted')

         #Print the accuracy, precision, recall and f1_score
         print('Accuracy Score Naive Bayes is', (acc_nb))
         print('Precision Score Naive Bayes is', (precision_nb))
         print('Recall Score Naive Bayes is', (recall_nb))
         print('F1 Score Naive Bayes is', (f1_nb))
```

```
Accuracy Score Naive Bayes is 0.858333333333
Precision Score Naive Bayes is 0.857861843703
Recall Score Naive Bayes is 0.858333333333
F1 Score Naive Bayes is 0.857466492346
```

Logistic Regression

Logistic Regression also works very well in text classification problems with a small dataset. It is frequently used in binary classification but it also works well in Multinomial classification, where multiple classes are present in target, like our dataset that contains 3 classes: country, rock and rap. Logistic Regression describes and estimates the relationship between the variables. This model takes the inputs and checks the probability of this input to belong to the class 0. If the probability is greater than 0.5 the output will be predicted for class 0.

```
In [83]: #Import Logistic Regression
         from sklearn.linear_model import LogisticRegression

         #Define the model
         logistic_regression= LogisticRegression()

         #Fit the model
         logistic_regression.fit(training_data, y_train)

         #Make predictions on the test data
         pred_lr = logistic_regression.predict(testing_data)
```

```
In [84]: #Calculate the accuracy, precision, recall and f1_score
         acc_lr = accuracy_score (y_test, pred_lr)
         precision_lr = precision_score (y_test, pred_lr, average='weighted')
         recall_lr = recall_score (y_test, pred_lr, average='weighted')
         f1_lr = f1_score (y_test, pred_lr, average='weighted')

         #Print the accuracy, precision, recall and f1_score
         print('Accuracy Score Logistic Regression is', (acc_lr))
         print('Precision Score Logistic Regression is', (precision_lr))
         print('Recall Score Logistic Regression is', (recall_lr))
         print('F1 Score Logistic Regression is', (f1_lr))
```

```
Accuracy Score Logistic Regression is 0.815
Precision Score Logistic Regression is 0.819088130131
Recall Score Logistic Regression is 0.815
F1 Score Logistic Regression is 0.816593813532
```

Support Vector Machine

Support Vector Machine (SVM) works by creating “line” or “hyperplane” that separates the observations considering the labels. If a number of features is 2, the hyperplane just uses a line. The hyperplane becomes a two-dimensional plane for 3 features. Support Vector is a series of data points that are closer of the hyperplane. With support vector we maximize the margin of the classifier. If we remove the support vector, the position of the hyperplane will change and help us to build our algorithm. The optimal hyperplane is the one that has the maximum margin in a space that separated classifies data points.

```
In [85]: #Import SVM
         from sklearn import svm

         #Define the model
         svm = svm.SVC()

         #Fit the model
         svm.fit(training_data, y_train)

         #Make predictions on the test data
         pred_svm = svm.predict(testing_data)

In [86]: #Calculate the accuracy, precision, recall and f1_score
         acc_svm = accuracy_score (y_test, pred_svm)
         precision_svm = precision_score (y_test, pred_svm, average='weighted')
         recall_svm = recall_score (y_test, pred_svm, average='weighted')
         f1_svm = f1_score (y_test, pred_svm, average='weighted')

         #Print the accuracy, precision, recall and f1_score
         print('Accuracy Score SVM is', (acc_svm))
         print('Precision Score SVM is', (precision_svm))
         print('Recall Score SVM is', (recall_svm))
         print('F1 Score SVM is', (f1_svm))
```

```
Accuracy Score SVM is 0.758333333333
Precision Score SVM is 0.764247063461
Recall Score SVM is 0.758333333333
F1 Score SVM is 0.760578972282
```

The Best Model

Between our 3 models: Multinominal Naives Bayes, Logistic Regression and Support Vector Machine, Multinomial Naives Bayes perform better, with a accuracy of 85%.

Model Tuning

Using GridSearch to try to improve our model.

```
In [87]: # TODO: Import 'GridSearchCV', 'Pipeline', and others necessary libraries
         from sklearn.pipeline import Pipeline
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
         from sklearn.model_selection import train_test_split, GridSearchCV
```

```
        from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, r

        # TODO: Initialize the classifier
        text_clf = Pipeline([('vect', CountVectorizer()),
                             ('tfidf', TfidfTransformer()),
                             ('clf', MultinomialNB())])

        #parameter list to tune
        tuned_parameters = {
            'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
            'tfidf__use_idf': (True, False),
            'tfidf__norm': ('l1', 'l2'),
            'clf__alpha': [1, 1e-1, 1e-2]
        }
```

```
In [108]: #import classification report
          from sklearn.metrics import classification_report


          clf_1 = GridSearchCV(text_clf, tuned_parameters, cv=10, scoring="accuracy")
          clf_1.fit(X_train, y_train)

          print(classification_report(y_test, clf_1.predict(X_test), digits=4))
```

```
              precision    recall  f1-score   support

           0     0.8241    0.8396    0.8318       212
           1     0.8324    0.7660    0.7978       188
           2     0.9147    0.9650    0.9392       200

   micro avg     0.8583    0.8583    0.8583       600
   macro avg     0.8570    0.8569    0.8562       600
weighted avg     0.8569    0.8583    0.8569       600
```

Changing the KfoldCV from 10 to 6 to validate the model:

```
In [110]: #import classification report
          from sklearn.metrics import classification_report


          clf_2 = GridSearchCV(text_clf, tuned_parameters, cv=6, scoring="accuracy")
          clf_2.fit(X_train, y_train)

          print(classification_report(y_test, clf_2.predict(X_test), digits=4))
```

```
              precision    recall  f1-score   support
```

```
           0     0.8170    0.8632    0.8394       212
           1     0.8391    0.7766    0.8066       188
           2     0.9307    0.9400    0.9353       200

   micro avg     0.8617    0.8617    0.8617       600
   macro avg     0.8622    0.8599    0.8605       600
weighted avg     0.8618    0.8617    0.8611       600
```

```
In [111]: # score our model on testing data
          predicted_opt = clf_2.predict(X_test)

          np.mean(predicted_opt == y_test)
```

```
Out[111]: 0.86166666666666669
```

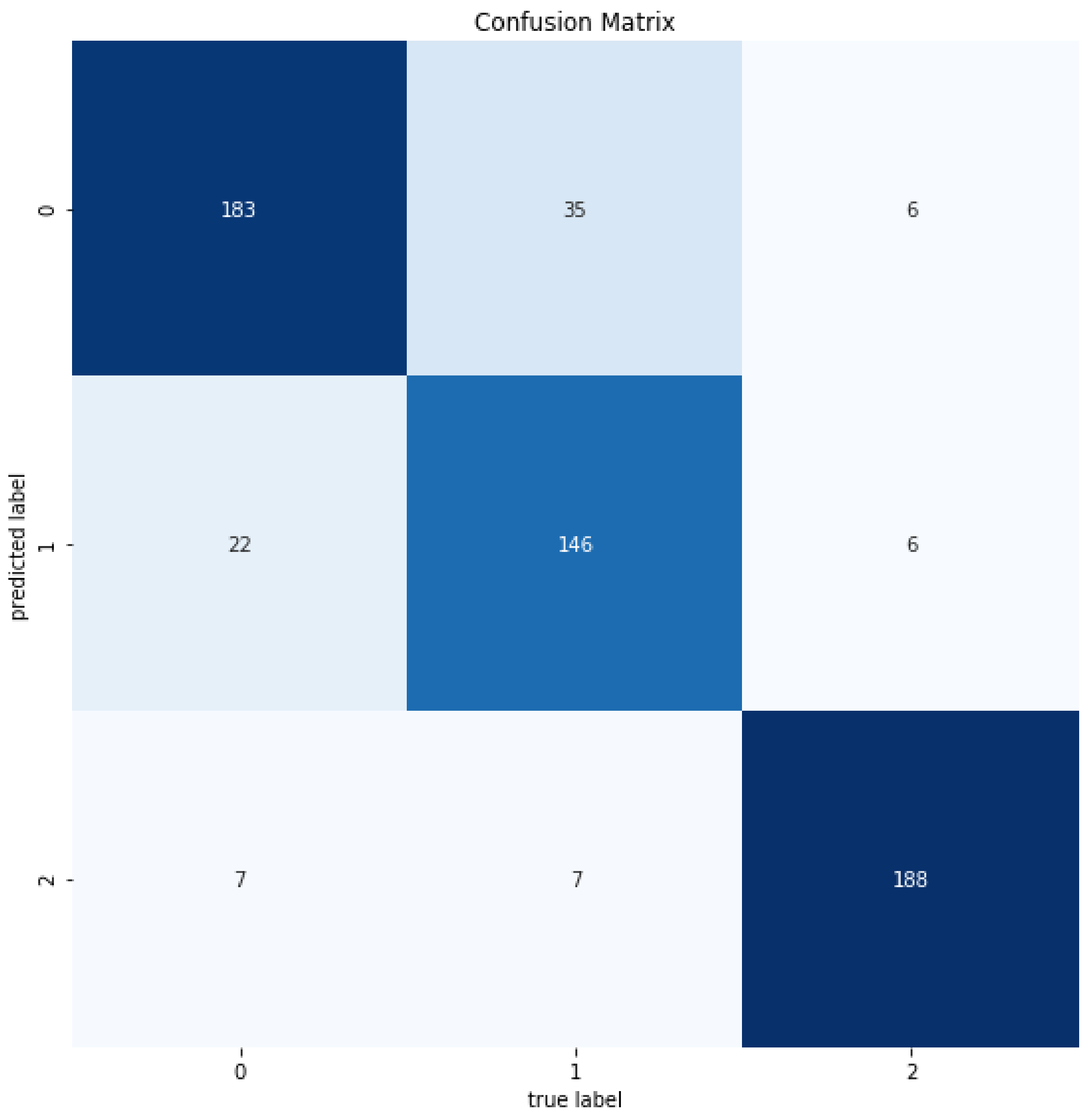
```
In [112]: # Report the before-and-afterscores
          print("Unoptimized model\n------")
          print("Accuracy score on testing data: {:.3f}".format(accuracy_score(y_test, pred_nb
          print("Precision score on testing data: {:.3f}".format(precision_score(y_test, pred_
          print("Recall score on testing data: {:.3f}".format(recall_score(y_test, pred_nb, ave
          print("f1-score on testing data: {:.3f}".format(f1_score(y_test, pred_nb, average='we

          print("\nOptimized Model\n------")
          print("Accuracy score on testing data: {:.3f}".format(accuracy_score(y_test, predicte
          print("Precision score on testing data: {:.3f}".format(precision_score(y_test, predic
          print("Recall score on testing data: {:.3f}".format(recall_score(y_test, predicted_op
          print("f1-score on testing data: {:.3f}".format(f1_score(y_test, predicted_opt, avera
```

```
Unoptimized model
------
Accuracy score on testing data: 0.858
Precision score on testing data: 0.858
Recall score on testing data: 0.858
f1-score on testing data: 0.857

Optimized Model
------
Accuracy score on testing data: 0.862
Precision score on testing data: 0.862
Recall score on testing data: 0.862
f1-score on testing data: 0.861
```

The metrics in the Optimized Model are better than Unoptimized model. The Optimized model did a bit better in all four metrics. So, our model will be the Optimized model.

```
In [113]: from sklearn.metrics import confusion_matrix
          fig, ax = plt.subplots(figsize=(8,8))         # Sample figsize in inches
```

```
mat = confusion_matrix(y_test, predicted_opt)

sb.heatmap(

    mat.T, square=True, annot=True, fmt='d', cbar=False,cmap = 'Blues',

)


plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title('Confusion Matrix')
plt.tight_layout();
```



- Country (label 0): We got 183 songs that were corrected classified and 41 that were misclassified.
- Rock (label 1): We got 146 songs that were corrected classified and 28 were misclassified.
- Rap (label 2): We got 188 songs correct classified and 14 were misclassified.

Complications

The biggest complication that I had in this project was scraping the data, since it was my first time. I tried to use the python library "Beautiful Soup" without much success. Then I found the python wrapper for the Genius API as mentioned earlier, that really helped me, but I had some issues. The first issue was that this takes quite a lot time to search the songs. Then, when I had started my analyses. I found out that the some results weren't correct. It wouldn't always return the songs for the selected artists, for example when I tried to get the Queen songs, the API return the Beyoncé songs. In this case, I found a dataset containing the Queen songs that I concatenated with my dataset.

Reflection

This project was really interesting and I had a lot fun doing it. Of all I learnt how to scrape the data. Then, I wanted to know how my data would look like in a word cloud. I tried to create a word cloud without preparing the data and it didn't look very nice, for example I got a big size "Chorus" keyword since I hadn't removed it yet. I tried to create a word cloud again but this time, I had removed those useless words like "Chorus" but the word cloud still wasn't very nice, and I removed the verbs suffixes ("-ing", "-ed") for the algorithm to recognize the words in their root format. To accomplish this result, I used tokenization and lemmatization. (Before using the lemmatization technique, I tried stemming with PorterStemmer but I realized that WordNetLemmatizer was doing better for my dataset). I used Afinn algorithm that would return a range between -5 to +5, then based on this score I created a "feelings" column contains "positive", "negative" or "neutral". My initial ideal was to classify my songs using feeling, like "happy", "angry", "nervous", "in love"... But I couldn't figure out how to do that. In the first try, my predictions got a accuracy of 82%,so I dug into my dataset and I found some rock songs which were instrumental tracks with ones with lyrics. Also, as I initially had problems to scrap the data, I was running my model with 800 songs for rock, 750 for rap and 950 for country, so I added more data and I got 1000 sample for each genre, improving my model's accuracy improved from 82% to 86% accuracy.

1.8 Conclusion

As we expected our model did better to classify rap than country our rock. This was expected, since in your analyses we saw rap has very different words compared with rock and country. Furthermore, the rap songs are much more negative than rock and country songs.

When we compared the rock and country word clouds, we got quite similar words and the feeling of the songs is also quite similar.

For future projects, we can try to include music into the analysis, since this would be very different between rock and country.