Introduction to Logistic Regression Model - Visualizng Data

In this section, we aim to interpret and visualize the logistic regression model.

Necessary Libraries

We will be using NLTK, an opensource NLP library, for collecting, handling, and processing Twitter data. In this lab, we will use the example dataset that comes alongside with NLTK. This dataset has been manually annotated and serves to establish baselines for models quickly.

In [1]:
```
# Natural Language Process ToolKit
# NLTK is a platform for building Python programs to work with human language data.
# It includes a suite of text processing libraries for:
#     classification, tokenization, stemming, tagging, parsing, and semantic reasoning.
# https://www.nltk.org/
import nltk
from nltk.corpus import twitter_samples

# Pandas is an open source data analysis and manipulation tool,built on top of the Python programming language.
# It will be used as a library for Dataframes
# https://pandas.pydata.org/
import pandas as pd

# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# It will be used as library for visualization
# https://matplotlib.org/
import matplotlib.pyplot as plt

# NumPy is an open source project aiming to enable numerical computing with Python.
# It will be used as a library for math functions.
# https://numpy.org/
import numpy as np

# Python method getcwd() returns current working directory of a process.
from os import getcwd
```

In addition, we are going to declare our previously implemented funtions, for:

- Pre - Processing
- Building a freqs dictionary

In [3]:

```
# ============== PRE - PROCESS FUNCTION ==============
def preProcess_tweet(tweet):
    # Part 1: Clean redundancy
    # ------------------------
    # Remove re-tweets
    data = re.sub(r'^RT[\s]+', '', tweet)

    # Remove hyperlinks
    data = re.sub(r'https?://[^\s\n\r]+', '', data)

    # Remove hashtags by only removing the hash # sign from the word
    data = re.sub(r'#', '', data)

    # Part 2: Tokenize
    # ------------------------
    # Instantiate tokenizer class
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)

    # tokenize tweets
    data_tokens = tokenizer.tokenize(data)

    data_clean = []
    for word in data_tokens: # Go through every word in your tokens list
        if (word not in stopwords_english and  # remove stopwords
            word not in string.punctuation):  # remove punctuation
            data_clean.append(word)

    # Part 3: Stem
    # ------------------------
    # Instantiate stemming class
    stemmer = PorterStemmer()

    # Create an empty list to store the stems
    data_stem = []

    for word in data_clean:
        stem_word = stemmer.stem(word)  # stemming word
        data_stem.append(stem_word)  # append to the list
```

```
    return data_stem
```

In [5]:
```
# ============== BUILD FREQUENCIES FUNCTION ==============
def build_freqs(tweets_dataset, ylabels):

    # Convert np array to list since zip needs an iterable.
    # The squeeze is necessary or the list ends up with one element.
    # Also note that this is just a NOP if ys is already a list.
    ylabelslist = np.squeeze(ylabels).tolist()

    # Start with an empty dictionary and populate it by looping over all tweets
    # and over all processed words in each tweet.
    freqs = {}
    for y, tweet in zip(ylabelslist, tweets_dataset):
        for token in preProcess_tweet(tweet):
            pair = (token, y)
            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1

    return freqs
```

Load the necessary data set

Given that we have already downloaded the ntlk twiter data sample via the "nltk.download('twitter_samples')" command, all we have to do is load the samples in the necessary lists:

In [6]:
```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

We now unify all the necessary tweets in one structure:

In [7]:
```
# Concatenate the lists.
tweets = all_positive_tweets + all_negative_tweets
```

We set the necessary labels (1s and 0s) corresponding to positive and negative sentiment tweets respectively.

In [10]:
```
# how many positive - sentiment tweets do we have
positive_count = len(all_positive_tweets)

# how many negative - sentiment tweets do we have
negative_count = len(all_negative_tweets)

# create a nx1 (positive + negative) label matrix
# we put np.ones as positive labels
# we put np.zeros as negative labels
# np.append(positive_labels, negative_labels, axis)
# https://numpy.org/doc/stable/reference/generated/numpy.append.html
labels = np.append(np.ones((positive_count,1)), np.zeros((negative_count,1)), axis = 0)
```

Load the training and testing data

In this section, we are going to use the provided data set, to generate two sets of data:

- Training data
- Testing data in order for us to train our classifier

In [13]:
```
# split the data into training data
# 4,000 positive training items
training_positive  = all_positive_tweets[:4000]
# 4,000 negative training items
training_negative  = all_negative_tweets[:4000]
```

In [15]:
```
training_data = training_positive+training_negative
```

Load Features

In [20]:
```
data = pd.read_csv('./data/logistic_features.csv'); # Load a 3 columns csv file using pandas function
data.head(10) # Print the first 10 data entries
```

Out[20]:

	bias	positive	negative	sentiment
0	1.0	3020.0	61.0	1.0
1	1.0	3573.0	444.0	1.0
2	1.0	3005.0	115.0	1.0
3	1.0	2862.0	4.0	1.0
4	1.0	3119.0	225.0	1.0
5	1.0	2955.0	119.0	1.0
6	1.0	3934.0	538.0	1.0
7	1.0	3162.0	276.0	1.0
8	1.0	628.0	189.0	1.0
9	1.0	264.0	112.0	1.0

In [21]:
```
print(type(data))
```

```
<class 'pandas.core.frame.DataFrame'>
```

We are now going to convert data to numpy arrays.

In [23]:
```
# Convert X Features:
# Each X - feature is labeled as:
#     bias, positive, negative
# We will only get the numerical values of the dataframe
X = data[['bias', 'positive', 'negative']].values
```

In [26]:
```
print(type(X))
print(X.shape)
print(X[:5])
```

```
<class 'numpy.ndarray'>
(8000, 3)
[[1.000e+00 3.020e+03 6.100e+01]
 [1.000e+00 3.573e+03 4.440e+02]
 [1.000e+00 3.005e+03 1.150e+02]
 [1.000e+00 2.862e+03 4.000e+00]
 [1.000e+00 3.119e+03 2.250e+02]]
```

In [27]:
```
# Convert Y Labels:
# We will only get the numerical values of the dataframe
Y = data[['sentiment']].values
```

In [29]:
```
print(type(Y))
print(Y.shape)
print(Y[:5])
```

```
<class 'numpy.ndarray'>
(8000, 1)
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

Load Model

A pre-trained logistical - regression model, has concluded with the following θ parameter:

In [31]:
```
theta = [6.03518871e-08, 5.38184972e-04, -5.58300168e-04]
```

Graphical representation of Training Data

In this section, we are going to represent all of our training data in a plane. Given that we are refering to a 2D vector (if we do not take under consideration the 1 - bias), a point can be represented as a point in a plane.

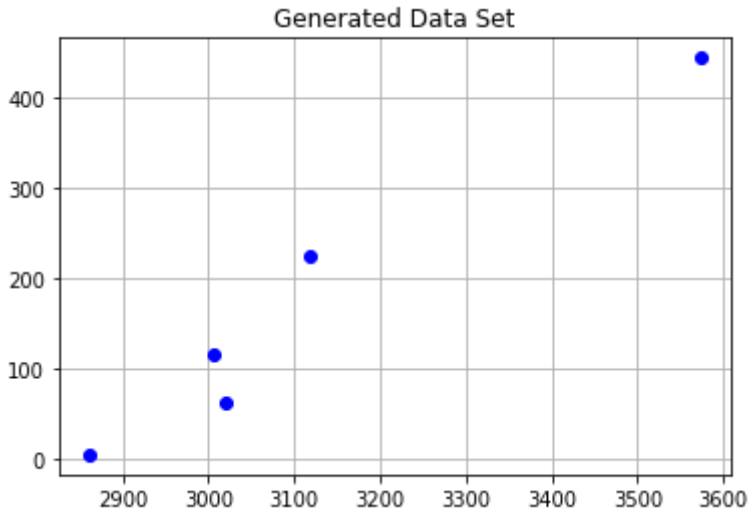
In [42]:
```
vectors =[]
for i in range(5):
    point = (X[i][1], X[i][2])
    vectors.append(point)
```

In [44]:
```
print(vectors[0:])
```

```
[(3020.0, 61.0), (3573.0, 444.0), (3005.0, 115.0), (2862.0, 4.0), (3119.0, 225.0)]
```

In [62]:
```
# Plot the prementioned five samples
coordinates = vectors
```

```
for i in range(len(coordinates)):
    plt.plot(coordinates[i][0], coordinates[i][1], 'bo')

# Displaying grid
plt.grid()

# Adding title
plt.title('Generated Data Set')

# Displaying plot
plt.show()
```

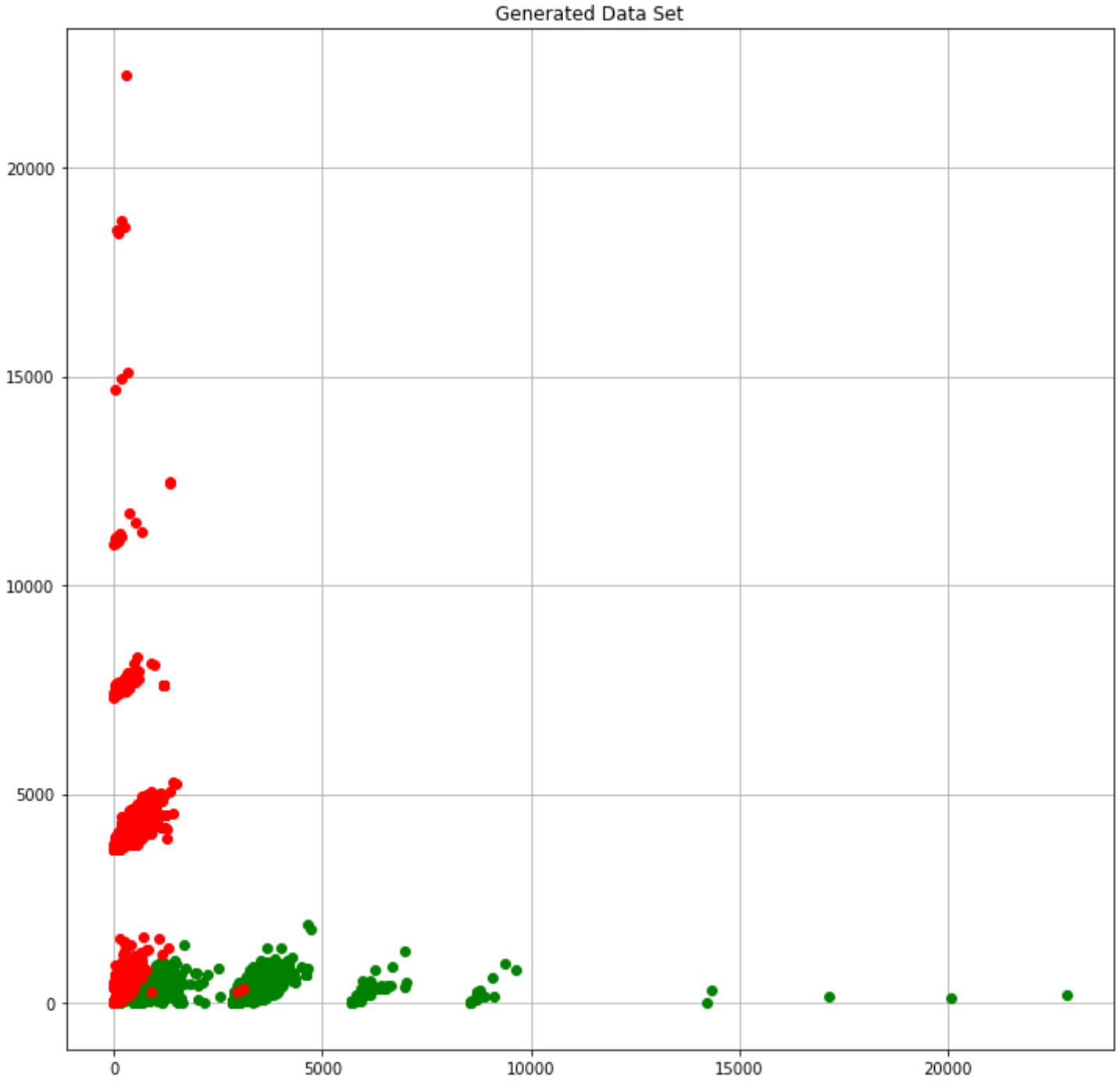


The prementioned representation, indicates the number of occurancies of a token in positive dictionary opposed to in negative dictionary.

Let us now use the same methodology, to represent all training tokens, alongside their representative color

In [84]:
```
vectors =[]
for i in range(len(X)):
    # positive occurancies, negative occurancies, label
    point = (X[i][1], X[i][2], Y[i][0])
    vectors.append(point)

# Plot the prementioned five samples
plt.subplots(figsize = (12, 12))
for i in range(len(vectors)):
    point = vectors[i]
```

```
    if(point[2]==1.0):
        plt.plot(point[0], point[1], 'go')
    else:
        plt.plot(point[0], point[1], 'ro')

# Displaying grid
plt.grid()

# Adding title
plt.title('Generated Data Set')

# Displaying plot
plt.show()
```


Generated Data Set
20000
15000
10000
5000
0
0
5000
10000
15000
20000

From the plot, it is evident that the features that we have chosen to represent tweets as numerical vectors allow an almost perfect separation between positive and negative tweets. So we may expect a very high accuracy for this model!