**1.Load and simplify the dataset**

Our SMS text messages dataset has 5 columns if you read it in pandas: v1 (containing the class labels ham/spam for each text message), v2 (containing the text messages themselves), and three Unnamed columns which have no use. We’ll rename the v1 and v2 columns to class\_label and message respectively while getting rid of the rest of the columns.

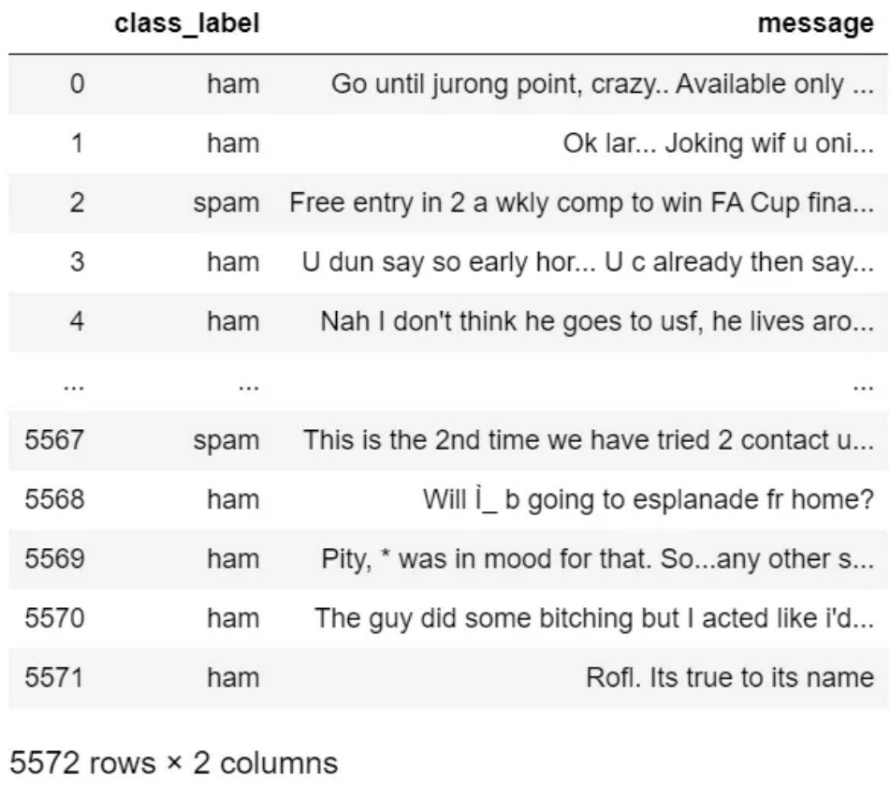
Import pandas as pd

Df = pd.read\_csv(r’spam.csv’,encoding=’ISO-8859-1’)

Df.rename(columns = {‘v1’:’class\_label’, ‘v2’:’message’}, inplace = True)

Df.drop([‘Unnamed: 2’, ‘Unnamed: 3’, ‘Unnamed: 4’], axis = 1, inplace = True)

df



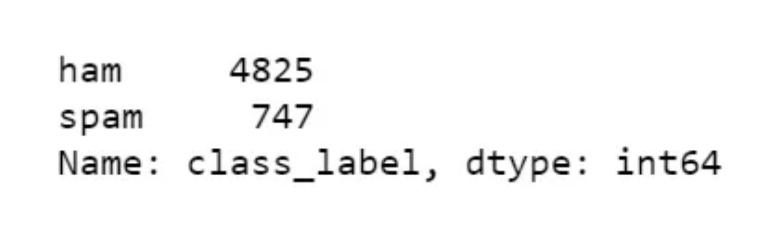
Check out the fact that ‘5572 rows x 2 columns’ means that our dataset has 5572 text messages!

Explore the dataset: Bar Chart

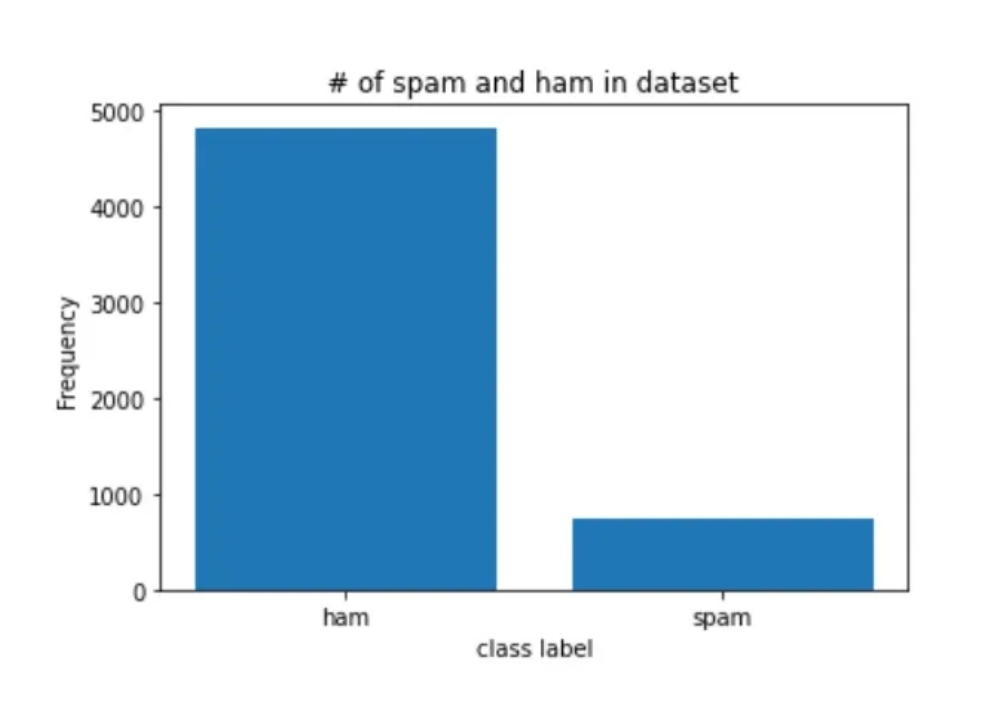
It’s a good idea to carry out some Exploratory Data Analysis (EDA) in a classification problem to visualize, get some information out of, or find any issues with your data before you start working with it. We’ll look at how many spam/ham messages we have and create a bar chart for it.

#exploring the dataset

Df[‘class\_label’].value\_counts()



Our dataset has 4825 ham messages and 747 spam messages. This is an imbalanced dataset; the number of ham messages is much higher than those of spam! This can potentially cause our model to be biased. To fix this, we could resample our data to get an equal number of spam/ham messages.

To generate our bar chart, we use NumPy and pyplot from Matplotlib.

Explore the dataset: Word Clouds

For my project, I generated word clouds of the most frequently occurring words in my spam messages.

First, we’ll filter out all the spam messages from our dataset. Df\_spam is a DataFrame that contains only spam messages.

df\_spam = df[df.class\_label==’spam’]

df\_spam



Next, we’ll convert our DataFrame to a list, where every element of that list will be a spam message. Then, we’ll join each element of our list into one big string of spam messages. The lowercase form of that string is the required format needed for our word cloud creation.

Spam\_list= df\_spam[‘message’].tolist()

Filtered\_spam = filtered\_spam.lower()

Finally, we’ll import the relevant libraries and pass in our string as a parameter:

Import os

From wordcloud import WordCloud

From PIL import Image

Comment\_mask = np.array(Image.open(“comment.png”))

#create and generate a word cloud image

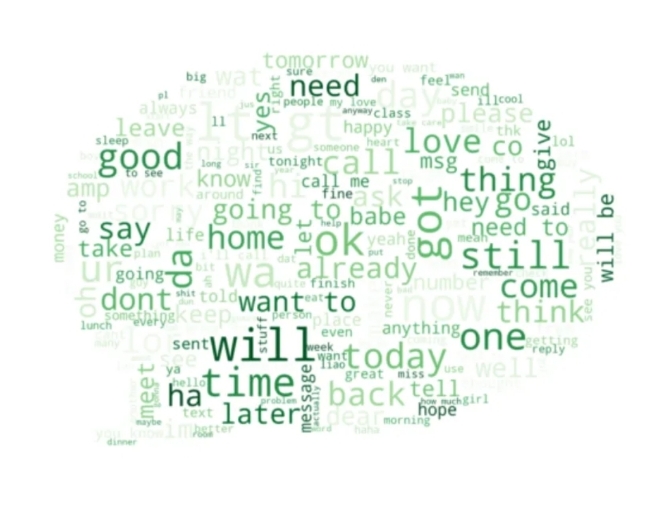
Wordcloud = WordCloud(max\_font\_size = 160, margin=0, mask = comment\_mask, background\_color = “white”, colormap=”Reds”).generate(filtered\_spam)

**After displaying it:**

Pretty cool, huh? The most common words in spam messages in our dataset are ‘free,’ ‘call now,’ ‘to claim,’ ‘have won,’ etc.

For this word cloud, we needed the Pillow library only because I’ve used masking to create that nice speech bubble shape. If you want it in square form, omit the mask parameter.

Similarly, for ham messages:



**4. Handle imbalanced ddatasets**

To handle imbalanced data, you have a variety of options. I got a pretty good f-measure in my project even with unsampled data, but if you want to resample, see this.

**5**.**Split the dataset**

First, let’s convert our class labels from string to numeric form:

Df[‘class\_label’] = df[‘class\_label’].apply(lambda x: 1 if x == ‘spam’ else 0)

In Machine Learning, we usually split our data into two subsets — train and test. We feed the train set along with the known output values for it (in this case, 0 or 1 corresponding to spam or ham) to our model so that it learns the patterns in our data. Then we use the test set to get the model’s predicted labels on this subset. Let’s see how to split our data.

First, we import the relevant module from the sklearn library:

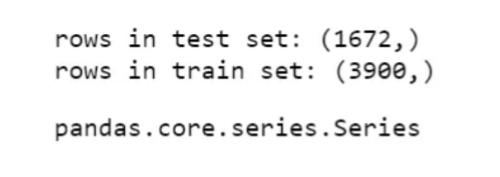
From sklearn.model\_selection import train\_test\_split

And then we make the split:

X\_train, x\_test, y\_train, y\_test = train\_test\_split(df[‘message’], df[‘class\_label’], test\_size = 0.3, random\_state = 0)

Let’s now see how many messages we have for our test and train subsets:

Print(‘rows in test set: ‘ + str(x\_test.shape))

Print(‘rows in train set: ‘ + str(x\_train.shape))

So we have 1672 messages for testing, and 3900 messages for training!

**6. Apply Tf-IDF Vectorizer for feature extraction**

Our Naïve Bayes model requires data to be in either Tf-IDF vectors or word vector count. The latter is achieved using Count Vectorizer, but we’ll obtain the former through using Tf-IDF Vectorizer.

TF-IDF Vectorizer creates Tf-IDF values for every word in our text messages. Tf-IDF values are computed in a manner that gives a higher value to words appearing less frequently so that words appearing many times due to English syntax don’t overshadow the less frequent yet more meaningful and interesting terms.

Lst = x\_train.tolist()

Vectorizer = TfidfVectorizer(

Input= lst , # input is the actual text

Lowercase=True, # convert to lowercase before tokenizing

Stop\_words=’english’ # remove stop words

)

Features\_train\_transformed = vectorizer.fit\_transform(list) #gives tf idf vector for x\_train

Features\_test\_transformed = vectorizer.transform(x\_test) #gives tf idf vector for x\_test

**7. Train our Naïve Bayes Model**

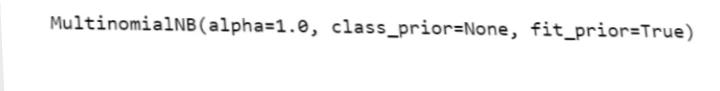
We fit our Naïve Bayes model, aka MultinomialNB, to our Tf-IDF vector version of x\_train, and the true output labels stored in y\_train.

From sklearn.naive\_bayes import MultinomialNB

# train the model

Classifier = MultinomialNB()

Classifier.fit(features\_train\_transformed, y\_train)

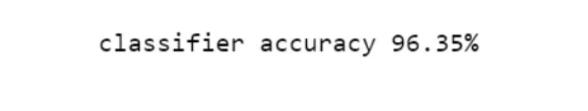


**8. Check out the accuracy, and f-measure**

It’s time to pass in our Tf-IDF matrix corresponding to x\_test, along with the true output labels (y\_test), to find out how well our model did!

First, let’s see the model accuracy:

Print(“classifier accuracy {:.2f}%”.format(classifier.score(features\_test\_transformed, y\_test) \* 100))



Our accuracy is great! However, it’s not a great indicator if our model becomes biased. Hence we perform the next step.

**9. View the confusion matrix and classification report**

Let’s now look at our confusion matrix and f-measure scores to confirm if our model is doing OK or not:

Labels = classifier.predict(features\_test\_transformed)

From sklearn.metrics import f1\_score

From sklearn.metrics import confusion\_matrix

From sklearn.metrics import accuracy\_score

From sklearn.metrics import classification\_report

Actual = y\_test.tolist()

Predicted = labels

Results = confusion\_matrix(actual, predicted)

Print(‘Confusion Matrix :’)

Print(results)

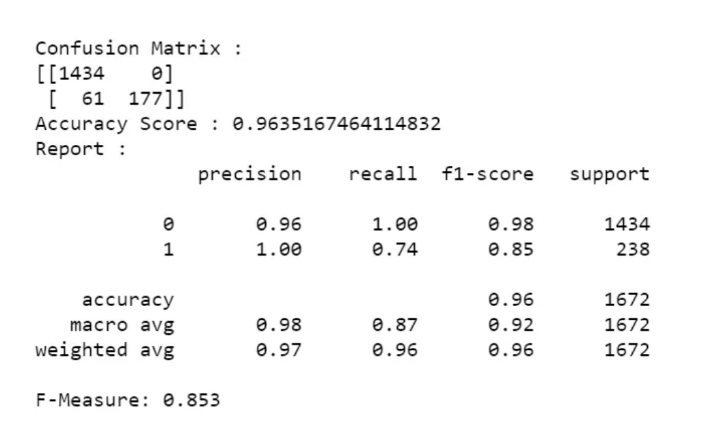
Print (‘Accuracy Score :’,accuracy\_score(actual, predicted))

Print (‘Report : ‘)

Print (classification\_report(actual, predicted) )

Score\_2 = f1\_score(actual, predicted, average = ‘binary’)

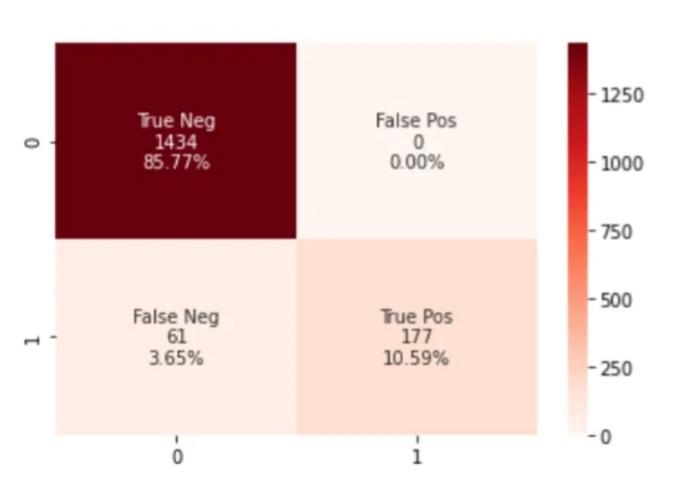
Print(‘F-Measure: %.3f’ % score\_2)



We have an f-measure score of 0.853, and our confusion matrix shows that our model is making only 61 incorrect classifications. Looks pretty good to me

**10. Heatmap for our Confusion Matrix (Optional)**

You can create a heatmap using the seaborn library to visualize your confusion matrix. The code below does just that.



And that’s it to make your very own spam classifier! To summarize, we imported the dataset and visualized it. Then we split it into train/test and converted it into Tf-IDF vectors. Finally, we trained our Naïve Bayes model, and saw the results! You could take this a step further and deploy it as a web app if you like.