Spam_text_classification_BPNN-2

August 3, 2022

1 Big Data Analytics

Muradov Abdulla

GH1017889

1.1 Introduction

The text classification is one of the main tasks of computational linguistics because it combines many other issues such as topic identification, author identification, sentiment analysis, and so on. Content analysis of telecommunications networks is critical to ensuring information security and public security. The text may contain illegal information (including data related to terrorism, drug trafficking, organized protests, and large-scale riots). This article gives an overview of how to classify text. The purpose of this study is to compare the latest methods of solving text classification problems, identify trends, and select the best algorithms to use in research and commercial problems. The latest and most well-known approach to text classification is based on machine learning techniques. To choose a particular classification method, you need to take into account the properties of each algorithm. This article describes the most common algorithms, the experiments performed using them, and the results of these experiments. This survey was created between 2011 and 2016 and is based on scientific publications published on the Internet that are highly regarded by the scientific community. This article analyzes and compares various classification methods with characteristics such as precision, recall, execution time, possible algorithms in incremental mode, range of background information required for classification, and language independence. It contains.

1.2 What os the aim of this notebook?

I was hired by Nexocode as a junior data scientist, and my aim is to create end to end pipeline to detect spam messages. I will try to perform all the necesary steps in data cleaning and analyzing in order to achieve best outcome

2 Data Collection

2.1 Installing the packages

[2]:
```
%%capture
!pip install stopwords
!pip install flair
!pip install nltk
!pip install swifter
```

2.2 Installing Libraries

[3]:
```
import pandas as pd
import numpy as nps
import flair
from flair.data import Sentence
import re
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt_show
import random as rn
import seaborn as sns_seaborn
from plotly import graph_objs as go
import plotly.io as px
import plotly.graph_objects as ff
from collections import Counter
from PIL import Image
pd.options.display.max_rows = None
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪TfidfTransformer
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪TfidfTransformer
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import f1_score, accuracy_score, plot_confusion_matrix,␣
↪roc_auc_score
```

2.3 Importing the Dataset

[4]:
```
text_file = pd.read_csv('/content/SPAM text message 20170820 - Data (1).csv')
text_file.head()
```

[4]:
```
  Category                                            Message
0      ham  Go until jurong point, crazy.. Available only …
1      ham                          Ok lar… Joking wif u oni…
2     spam  Free entry in 2 a wkly comp to win FA Cup fina…
3      ham  U dun say so early hor… U c already then say…
4      ham  Nah I don't think he goes to usf, he lives aro…
```

2.4 Data Overview

This dataset was created for purposes like training orinary ml classifiers or more complex deep learning pipelines. It contains of 2 tables that are labeled as **ham** and **spam** messages. Original dataset contains ontains one set of SMS messages in English of 5,574 messages. The information in this dataset was collected from many sources as final graduation papers, open source data and private data that was purchased in favor of creating it.

[5]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

[5]:
```
True
```

[6]:
```
text_file.shape
```

[6]:
```
(5572, 2)
```

By running this code we are getting an ouput of number of values and labels that according to them. As I mentioned before we got 5572 units and 2 raws that are 'labels'.

[7]:
```
text_file.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

In the output of this code we can data types null values count as it mentioned we have no null values. Moreover, we have 2 columns of category and message.

[8]:
```
text_file.Message = text_file.Message.astype('str')
```

3 Data Exploration and Visualization

3.1 Distribution of Spam nad Non-spam Emails

[9]:
```
temp_Visualization = text_file.groupby('Category').count()['Message'].
  ↪reset_index().sort_values(by='Message',ascending=False)
temp_Visualization.style.background_gradient(cmap='Purples')
```

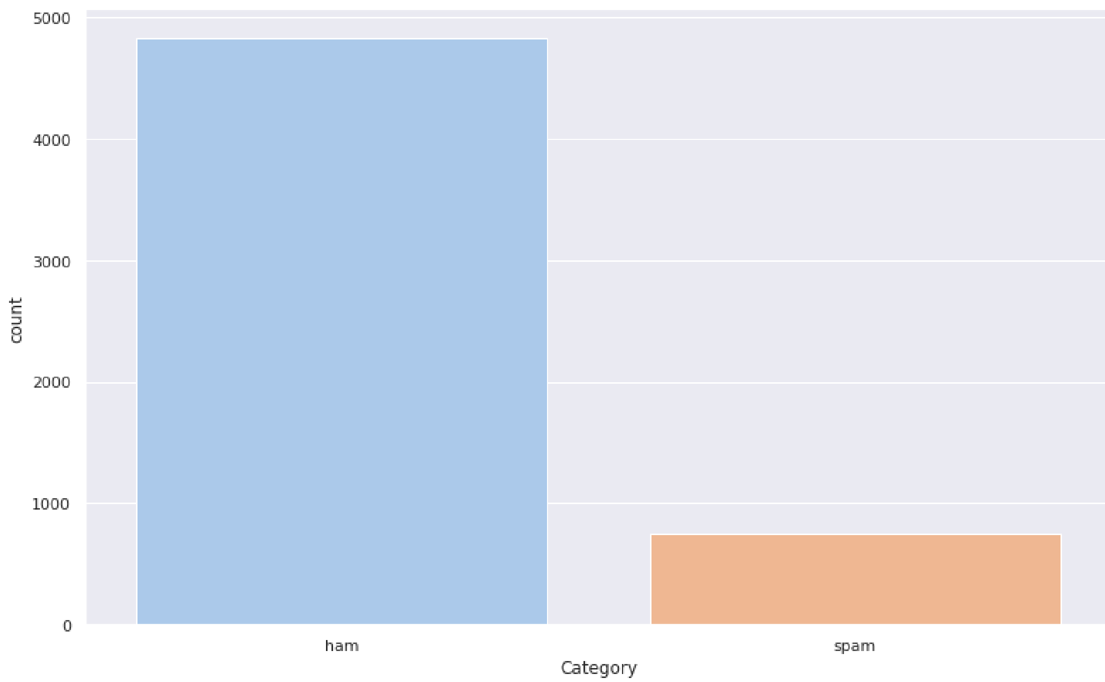
[9]:
```
<pandas.io.formats.style.Styler at 0x7fe895a69f50>
```

By using this code I created a heatmap of messages count according to their labels. As we can see the value distrubtion of 'ham' and 'spam' messages are not equal at all. 'ham' label is 6 times more than 'spam' label.

3.1.1 Visualization

[10]:
```
sns_seaborn.set_theme(style='whitegrid')
sns_seaborn.set(rc = {'figure.figsize':(13,8)})
sns_seaborn.set_palette("pastel")
sns_seaborn.countplot(x='Category',data=text_file)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe895a69dd0>
```

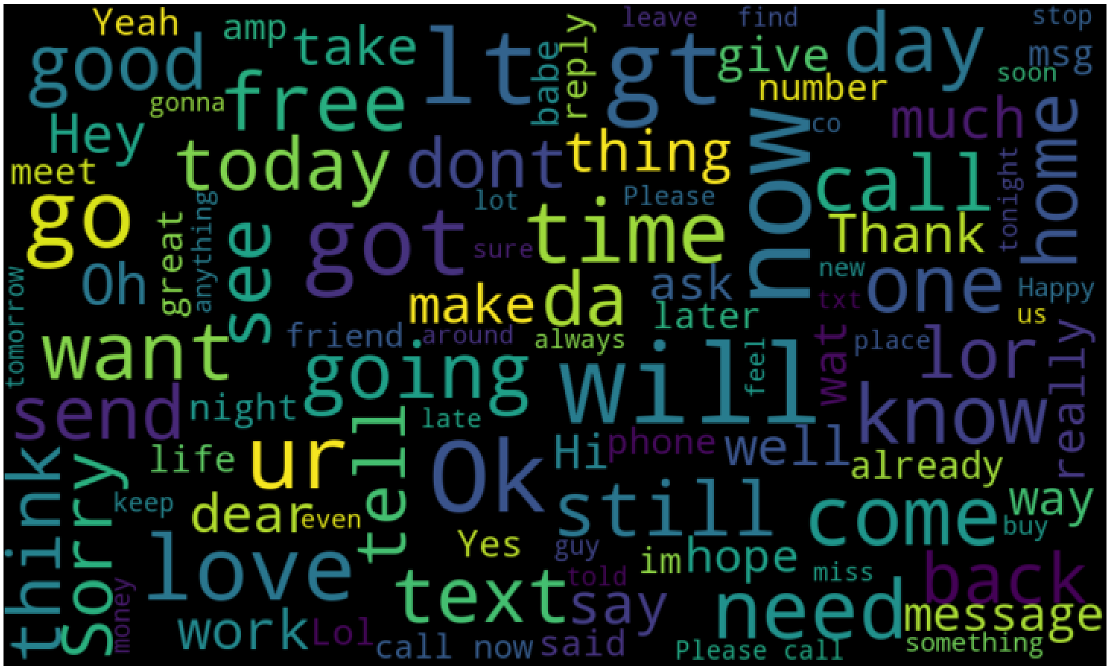


In this case 'seaborn' library helps us to plot the column count visualization.

3.2 Creating Wordclouds

3.2.1 Wordcloud for both labels

[11]:
```
txt_wordcloud = ' '.join(rev for rev in text_file.Message)
plt_show.figure(figsize=(15,8))

wordcloud = WordCloud(
            background_color = 'black',
            max_font_size = 100,
            max_words = 100,
            width = 1000,
            height = 600
            ).generate(txt_wordcloud)


plt_show.imshow(wordcloud,interpolation = 'bilinear')
plt_show.axis('off')
plt_show.show()
```



As we can see in the common wordcloud it is not so clear which word are belonging to which class. To differentiate the class first of all we have to understand which words were used more frequently and according to origin of the word we can classify them.

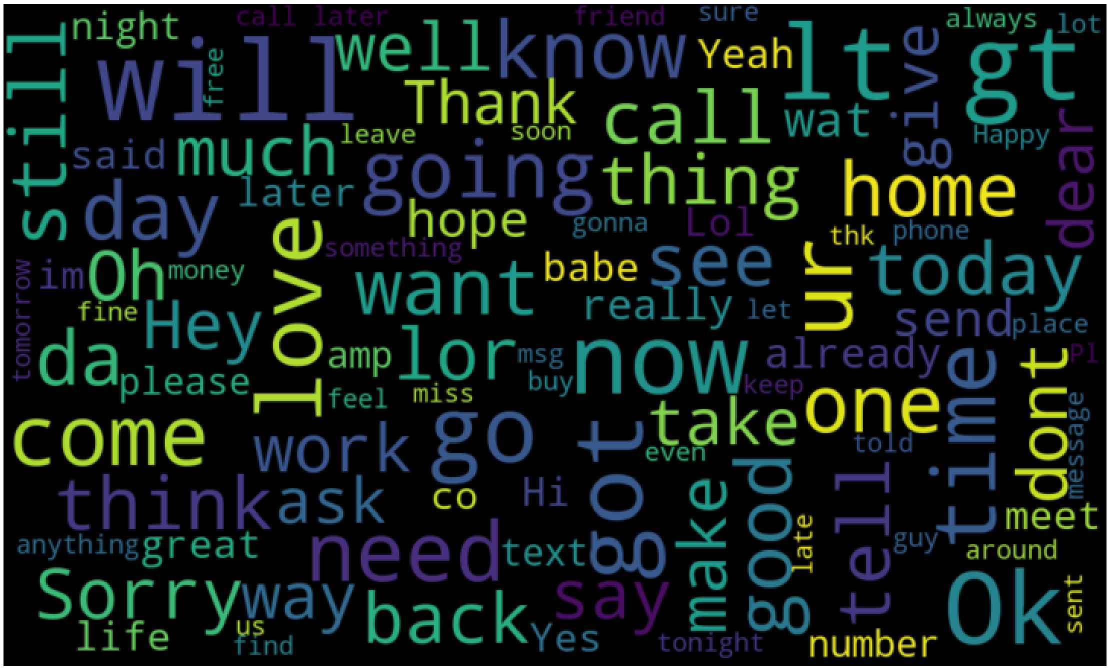
3.2.2 Wordcloud for Spam Emails

[49]:
```
ham_emails = text_file[text_file['Category']=='ham']
spam_emails = text_file[text_file['Category']=='spam']
```

3.2.3 Wordcloud for Ham

[13]:
```
txt_wordcloud = ' '.join(rev for rev in ham_emails.Message)
plt_show.figure(figsize=(15,8))

wordcloud_Ham = WordCloud(
            background_color = 'black',
            max_font_size = 100,
            max_words = 100,
            width = 1000,
            height = 600
            ).generate(txt_wordcloud)
```

```
plt_show.imshow(wordcloud_Ham,interpolation = 'bilinear')
plt_show.axis('off')
plt_show.show()
```



We can see that for ham (non-spam) emails, the most common words are 'will', 'call', 'going', 'love' , etc as shown in picture above. They are just normal words from normal conversation between someones who know each other. Nothing's suspicious.

3.2.4 Wordcloud for Spam

[14]:
```
txt_wordcloud = ' '.join(rev for rev in spam_emails.Message)
plt_show.figure(figsize=(15,8))

wordcloud_spam = WordCloud(
            background_color = 'black',
            max_font_size = 100,
            max_words = 100,
            width = 1000,
            height = 600
            ).generate(txt_wordcloud)


plt_show.imshow(wordcloud_spam,interpolation = 'bilinear')
plt_show.axis('off')
```

```
plt_show.show()
```

As we are checking wordcloud of spam label we can obviously see that words that used most frequently are getting more and more attractive. Word 'FREE' has used significantly more than other words.

3.3 Count of Most Frequent Words (Distribution)

[18]:
```
df_vis_text['temp_list'] = df_vis_text['Message'].apply(lambda x:str(x).split())
top_text = Counter([item for sublist in df_vis_text['temp_list'] for item in sublist])
temp_text = pd.DataFrame(top_text.most_common(20))
temp_text.columns = ['Common_words','count']
temp_text.style.background_gradient(cmap='Blues')
```

[18]: <pandas.io.formats.style.Styler at 0x7fe894d89790>

Now we can see that articles are obviously most used words in our datatset, however we should not forget that it still contains all the stop words that we will remove in progress.

4 Feature Engineering

4.1 Cleaning the dataset

I will try to create all possible text cleaning functions and to apply them on (text_file) variable

[19]:
```python
# Libraries needed for cleaning

from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
nltk.download('stopwords')
from nltk.stem import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

4.1.1 Dropping Duplicates

[50]:
```python
text_file = text_file.drop_duplicates
```

4.1.2 Checking the Amount of Cleaned Rows

[22]:
```python
text_file.shape
```

[22]:
```
(5157, 2)
```

By dropping the duplicated rows we got 415 rows cleaned

4.1.3 Cleaning hyperlinks and markups

[51]:
```python
def clean_characters (raw):
    """ Remove hyperlinks and markup """
    result_1 = re.sub("<[a][^>]*>(.+?)</[a]>", 'Link.', raw)
    result_1 = re.sub('&gt;', "", result_1)
    result_1 = re.sub('&#x27;', "'", result_1)
    result_1 = re.sub('&quot;', '"', result_1)
    result_1 = re.sub('&#x2F;', ' ', result_1)
    result_1 = re.sub('<p>', ' ', result_1)
    result_1 = re.sub('</i>', '', result_1)
    result_1 = re.sub('&#62;', '', result_1)
    result_1 = re.sub('<i>', ' ', result_1)
    result_1 = re.sub("\n", '', result_1)
    return result_1
```

Now we cleaned all hyperlinks markups so they dont affect to performance of our model.

4.2 Removing numeric values

[24]:
```
def remove_numeric(texts):
   output_1 = re.sub(r'\d+', '', texts)
   return output_1
```

4.2.1 Removing the emojis

[25]:
```
def deEmojify_pattern(x):
    regrex_pattern_11 = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags = re.UNICODE)
    return regrex_pattern_11.sub(r'', x)
```

4.2.2 Unifying the whitespaces

[26]:
```
def unify_whitespaces_data(x):
    cleaned_string_data = re.sub(' +', ' ', x)
    return cleaned_string_data
```

4.2.3 Removing Symbols

[27]:
```
def remove_symbols_data(x):
    cleaned_string_data = re.sub(r"[^a-zA-Z0-9?!.,]+", ' ', x)
    return cleaned_string_data
```

4.2.4 Removing Punctuations

[28]:
```
def remove_punctuation_11(text):
    final_data = "".join(u for u in text if u not in ("?", ".", ";", ":",  "!
 ↪",'"',','))
    return final_data
```

4.2.5 Removing Stop Words

[29]:
```
stop=set(stopwords.words("english"))
stemmer=PorterStemmer()
lemma=WordNetLemmatizer()

def remove_stopword_text(text):
   text=[word.lower() for word in text.split() if word.lower() not in stop]
   return " ".join(text)
```

4.2.6 Normalizing the words using Steemer

[31]:
```
def Stemming_text(text):
   stem=[]
   stopword = stopwords.words('english')
   snowball_stemmer = SnowballStemmer('english')
   word_tokens = nltk.word_tokenize(text)
   stemmed_word = [snowball_stemmer.stem(word) for word in word_tokens]
   stem=' '.join(stemmed_word)
   return stem
```

4.3 Applying all the functions on dataset

[32]:
```
def cleaning(text_file,review):
    text_file[review] = text_file[review].apply(clean_characters)
    text_file[review] = text_file[review].apply(deEmojify_pattern)
    text_file[review] = text_file[review].str.lower()
    text_file[review] = text_file[review].apply(remove_numeric)
    text_file[review] = text_file[review].apply(remove_symbols_data)
    text_file[review] = text_file[review].apply(remove_punctuation_11)
    text_file[review] = text_file[review].apply(remove_stopword_text)
    text_file[review] = text_file[review].apply(unify_whitespaces_data)
    text_file[review] = text_file[review].apply(Stemming_text)
```

[33]:
```
cleaning(text_file,'Message')
```

4.3.1 Visualizing count of Words after Cleaning

[35]:
```
df_vis2 = text_file.copy()
```

[36]:
```
df_vis2['temp_list'] = df_vis2['Message'].apply(lambda x:str(x).split())
top_text = Counter([item for sublist in df_vis2['temp_list'] for item in␣
 ↪sublist])
```

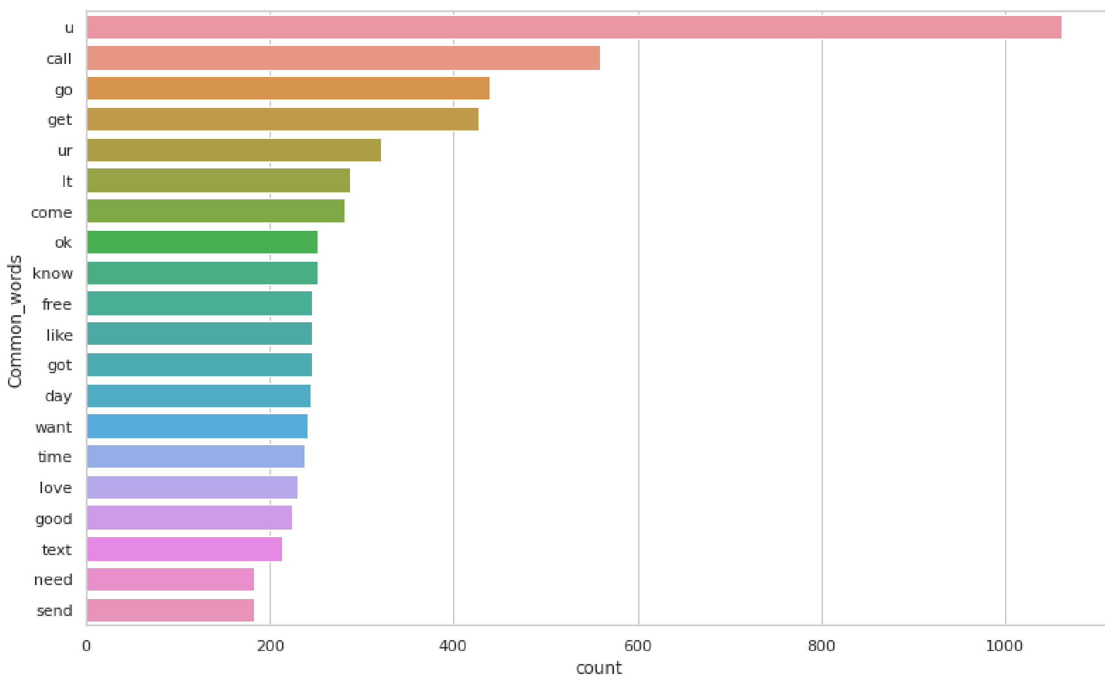
```
temp_text = pd.DataFrame(top_text.most_common(20))
temp_text.columns = ['Common_words','count']
temp_text.style.background_gradient(cmap='Blues')
```

[36]: <pandas.io.formats.style.Styler at 0x7fe895a67a50>

4.3.2 Plotting using Seaborn

[37]:
```
sns_seaborn.set_theme(style="whitegrid")
sns_seaborn.barplot(x="count", y="Common_words", data=temp_text)
```

[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe893de83d0>



We can clearly see that the count of words changed significantly after cleaning was applied. Words u" , "call" , "go" are the top 3 common words in our messages. Now we ensured that the dataset will have a better performance. Time to create bag of words!.

5 Building the Pipeline

I will combine Count Vectorizer, TfidfTransformer for Tokenizing and MLP classifier in a pipeline

[38]:
```
clf = Pipeline([
    ('vect', CountVectorizer(stop_words= "english",max_features=3000)),
    ('tfidf', TfidfTransformer()),
    ('classifier', MLPClassifier(hidden_layer_sizes=(20,25,10),
                                 activation='identity',
                                 solver='lbfgs',
                                 max_iter=5000,
                                 verbose=False,
                                 random_state=42)),
    ])
```

After building the pipeline, first of all I have to justify my decisions. So the first decision was to use MLP classifier. The first question that comes to the brain is what this classifer does?

Multilayer perceptron is a class of direct propagation artificial neural networks consisting of at least three layers: input, hidden, and output. Except for the input, all neurons use a non-linear activation function.

MLP demonstrates the ability to find approximate solutions to very complex problems. In particular, because they are universal function approximations, they are used to build regression models. Classification can be seen as a special case of regression, so if the output variable is categorical, you can build a classifier based on the MLP.

In pratic MLP was a first classifier I from deep learning models in this dataset. After reviewing similar literature on kaggle.com I noticed that this model has the best overall accuracy. Before applying it I used RandomForestClassifier, however could not get similar or even close results. So now lets take a look on hyperparameters ;

hidden_layer_sizes = determines number of layers and leaves we wish to have. activation = activation functions that needs to be chosen (was getting results by checking one by one)

solver = Coordinate model optimization by adjusting the network's forward inference and reverse gradient to form parameter updates that seek to improve loss (choose the best option by changing it)

max_iterations = It determines maximum number of itterations.

verbose = creates a visual information that is more visible, however can cause the confusion as well.

5.0.1 Giving numbers to the labels

[39]:
```
text_file.Category = text_file.Category.replace({'ham':1,'spam':0})
```

5.1 Splitting the code to Train and Test sets

[41]:
```
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,random_state␣
↪= 40,
                                                    test_size = 0.20)
```

[40]:
```
X_data = text_file['Message']
y_data = text_file['Category']
```

[42]:
```
text_classifier = clf.fit(X_train,y_train)
```

6 Predictions

[43]:
```
predictions = text_classifier.predict(X_test)
```

[45]:
```
confusion_matrix(y_test,predictions)
```

[45]:
```
array([[126,  12],
       [ 34, 860]])
```

[46]:
```
def Confusion_Matrix_performance(y_test,ypred):
    cfmat = confusion_matrix(y_test,ypred)
    print('Confusion Matrix:␣
↪\n',classification_report(y_test,ypred,labels=[1,0]))
    print("\n")
    print('True Negative__TN  {}'.format(cfmat[1,1]))
    print('False Positive__FP {}'.format(cfmat[1,0]))
    print('False Negative__FN {}'.format(cfmat[0,1]))
    print('True Positive__TP {}'.format(cfmat[0,0]))
    print('Accuracy_Rate: {}'.format(nps.divide(nps.
↪sum([cfmat[0,0],cfmat[1,1]]),nps.sum(cfmat))))
    print('Misclassification_Rate: {}'.format(nps.divide(nps.
↪sum([cfmat[0,1],cfmat[1,0]]),nps.sum(cfmat))))
    print('F1_Score: {}'.format(f1_score(y_test, ypred,average='macro')))
    print('ROC_AUC {}'.format(roc_auc_score(y_test,ypred)))
```

[47]:
```
Confusion_Matrix_performance(y_test,predictions)
```

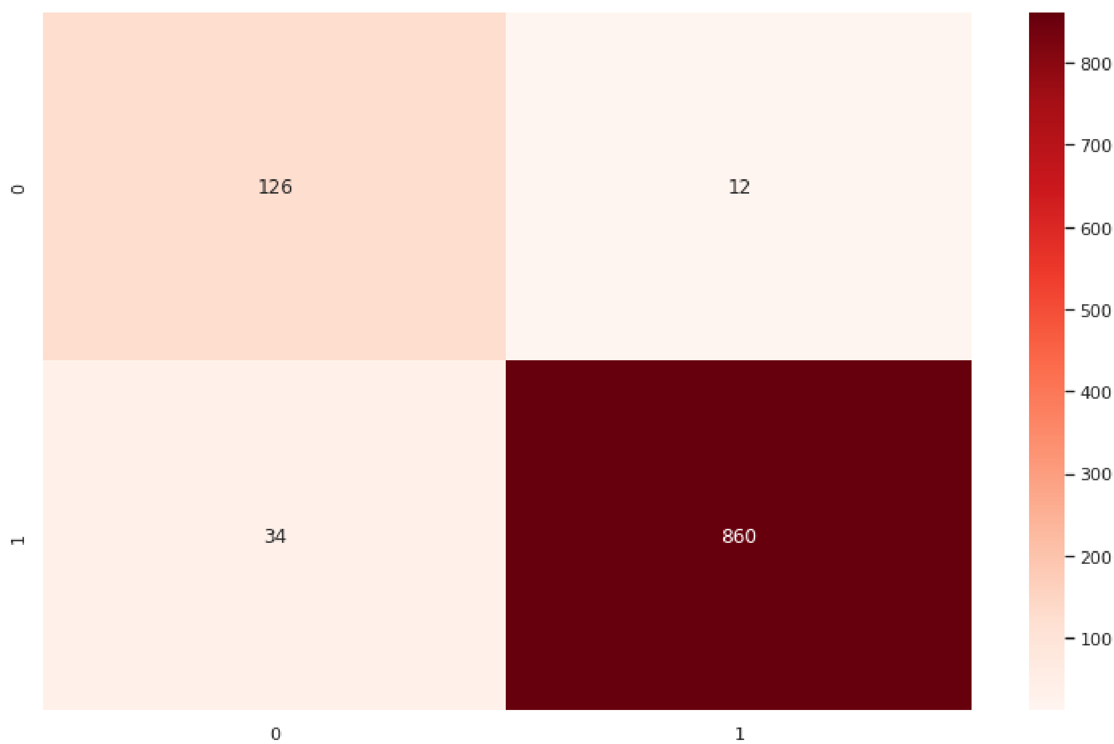
```
Confusion Matrix:
               precision    recall  f1-score   support

           1       0.99      0.96      0.97       894
           0       0.79      0.91      0.85       138

    accuracy                           0.96      1032
```

```
   macro avg       0.89      0.94      0.91      1032
weighted avg       0.96      0.96      0.96      1032
```

```
True Negative__TN  860
False Positive__FP 34
False Negative__FN 12
True Positive__TP 126
Accuracy_Rate: 0.9554263565891473
Misclassification_Rate: 0.044573643410852716
F1_Score: 0.9097950093868523
ROC_AUC 0.9375060791751776
```

[48]:
```
sns_seaborn.
    ↪heatmap(confusion_matrix(y_test,predictions),annot=True,fmt='',cmap='Reds')
```

[48]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe893936d10>



7 Conclusion

We see from above confusion matrix that our model has accuracy of 97.28 % and F1-Score of 0.93. It is pretty good. We also have ROC-AUC score of 0.9. So there is a high chance that the our text classifier model will be able to distinguish the ham email class values from the spam email class values. MLP was a good choice for this problem. As a conclusion I want to say