NLP_Detecting_Spam

November 1, 2024

[5]:
```
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/
↪NLP_Detecting_Spam.ipynb" --output "/content/NLP_Detecting_Spam.pdf"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/NLP_Detecting_Spam.ipynb to pdf
[NbConvertApp] Support files will be in /content/NLP_Detecting_Spam_files/
[NbConvertApp] Writing 61218 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 575634 bytes to /content/NLP_Detecting_Spam.pdf
```

[ ]:
```
import numpy as np
import pandas as pd
import spacy
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import RandomOverSampler
nltk.download('punkt')
nltk.download('stopwords')
```

[ ]:
```
#Loading datasets
SMS_test = pd.read_csv('SMS_test.csv')
SMS_train = pd.read_csv('SMS_train.csv')
```

EDA Process

```
[ ]: SMS_test
```

```
[ ]:      S. No.                                       Message_body     Label
     0         1  UpgrdCentre Orange customer, you may now claim…      Spam
     1         2  Loan for any purpose £500 - £75,000. Homeowner…      Spam
     2         3  Congrats! Nokia 3650 video camera phone is you…      Spam
     3         4  URGENT! Your Mobile number has been awarded wi…      Spam
     4         5  Someone has contacted our dating service and e…      Spam
     ..      …                                                …        …
     120     121  7 wonders in My WORLD 7th You 6th Ur style 5th…  Non-Spam
     121     122  Try to do something dear. You read something f…  Non-Spam
     122     123  Sun ah… Thk mayb can if dun have anythin on…  Non-Spam
     123     124  SYMPTOMS when U are in love: "1.U like listeni…  Non-Spam
     124     125  Great. Have a safe trip. Dont panic surrender …  Non-Spam

     [125 rows x 3 columns]
```

```
[ ]: SMS_train
```

```
[ ]:      S. No.                                       Message_body     Label
     0         1                        Rofl. Its true to its name  Non-Spam
     1         2  The guy did some bitching but I acted like i'd…  Non-Spam
     2         3  Pity, * was in mood for that. So…any other s…  Non-Spam
     3         4                 Will ü b going to esplanade fr home?  Non-Spam
     4         5  This is the 2nd time we have tried 2 contact u…      Spam
     ..      …                                                …        …
     952     953  hows my favourite person today? r u workin har…  Non-Spam
     953     954                       How much you got for cleaning  Non-Spam
     954     955  Sorry da. I gone mad so many pending works wha…  Non-Spam
     955     956                                  Wat time ü finish?  Non-Spam
     956     957                        Just glad to be talking to you.  Non-Spam

     [957 rows x 3 columns]
```

```
[ ]: #Checking for any missing values
     SMS_train.isnull().sum()
```

```
[ ]: S. No.          0
     Message_body    0
     Label           0
     dtype: int64
```

```
[ ]: SMS_train['Label'].value_counts()
```

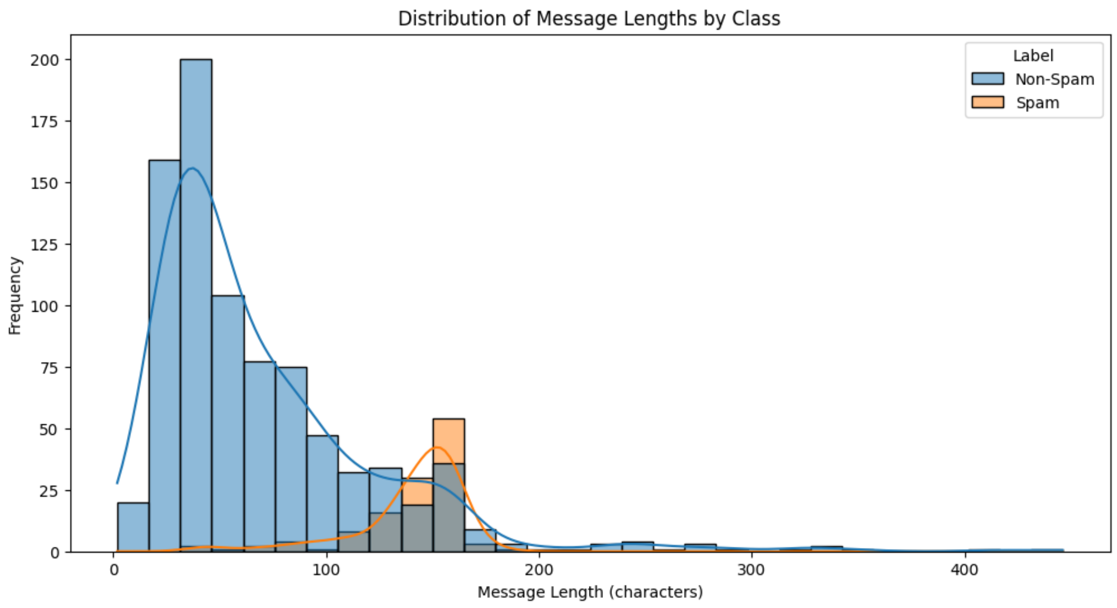
```
[ ]: Label
     Non-Spam    835
```

```
Spam          122
Name: count, dtype: int64
```

Based on this quick count we can see that there are less Spam emails than there are “non- spam” emails. Nothing that interesting yet to take a note of.

[ ]:
```
SMS_train['length'] = SMS_train['Message_body'].apply(len)
```

[ ]:
```
# Plotting the distribution of message lengths
plt.figure(figsize=(12, 6))
sns.histplot(data=SMS_train, x='length', hue='Label', bins=30, kde=True)
plt.title('Distribution of Message Lengths by Class')
plt.xlabel('Message Length (characters)')
plt.ylabel('Frequency')
plt.show()
```



The distribution of message lengths reveals that spam messages tend to be longer than non-spam messages. This difference in length can be a useful feature in spam detection. However, there is some overlap in lengths, highlighting the need for additional features to improve classification accuracy. While length is an indicator, it should not be used as the sole criterion for spam identification.

Data Processing

[ ]:
```
# Convert to lowercase
SMS_train['Message_body'] = SMS_train['Message_body'].str.lower()
SMS_train['Label'] = SMS_train['Label'].str.lower()

# Remove punctuation and special characters
```

```
SMS_train['Message_body'] = SMS_train['Message_body'].apply(lambda x: re.
↪sub(r'\W+', ' ', str(x)))

# Tokenize, clean, and join back to a single string
SMS_train['Message_body'] = SMS_train['Message_body'].apply(lambda x: ' '.
↪join(word_tokenize(x)))

# Check the cleaned data
print(SMS_train[['Message_body', 'Label']].head())
```

```
                                        Message_body     Label
0                          rofl its true to its name  non-spam
1  the guy did some bitching but i acted like i d…  non-spam
2  pity was in mood for that so any other suggest…  non-spam
3                will ü b going to esplanade fr home  non-spam
4  this is the 2nd time we have tried 2 contact u…      spam
```

[ ]:
```
SMS_train
```

[ ]:
```
     S. No.                                     Message_body     Label  \
0         1                        rofl its true to its name  non-spam
1         2  the guy did some bitching but i acted like i d…  non-spam
2         3  pity was in mood for that so any other suggest…  non-spam
3         4              will ü b going to esplanade fr home  non-spam
4         5  this is the 2nd time we have tried 2 contact u…      spam
..      …                                              …       …
952     953  hows my favourite person today r u workin hard…  non-spam
953     954                    how much you got for cleaning  non-spam
954     955  sorry da i gone mad so many pending works what…  non-spam
955     956                                wat time ü finish  non-spam
956     957                     just glad to be talking to you  non-spam

     length
0        26
1       125
2        57
3        36
4       160
..      …
952     101
953      29
954      54
955      18
956      31

[957 rows x 4 columns]
```

Oversampling with RandomOverSampler Addressing class imbalance in the SMS spam detection dataset, where there are significantly more 'ham' messages than 'spam' messages.

Technique: Using RandomOverSampler to randomly duplicate samples from the minority class (spam) to achieve a balanced class distribution.

```
[ ]: # Separate features and target
X = SMS_train['Message_body']
y = SMS_train['Label']
```

```
[ ]: # Encodeing 'Label' to numerical (0 and 1) for modeling later
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

```
[ ]: # Resample the data, this is pretty much adding more Spams into that label␣
 ↪colum to take care of that imbalance that we have going on.
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X.values.reshape(-1, 1),␣
 ↪y_encoded)
```

```
[ ]: # Create a new DataFrame with the resampled data
SMS_train_resampled = pd.DataFrame({'Message_body': X_resampled.flatten(),␣
 ↪'Label': y_resampled})
```

Checking if Imbalance has been fixed

```
[ ]: #Old dataframe count
print(SMS_train['Label'].value_counts())
```

```
Label
non-spam    835
spam        122
Name: count, dtype: int64
```

```
[ ]: # y_resampled is the resampled target variable
print(SMS_train_resampled['Label'].value_counts())
```

```
Label
0    835
1    835
Name: count, dtype: int64
```

BACK TO EDA

Here we will create some word cloud to see what words stick out the most!

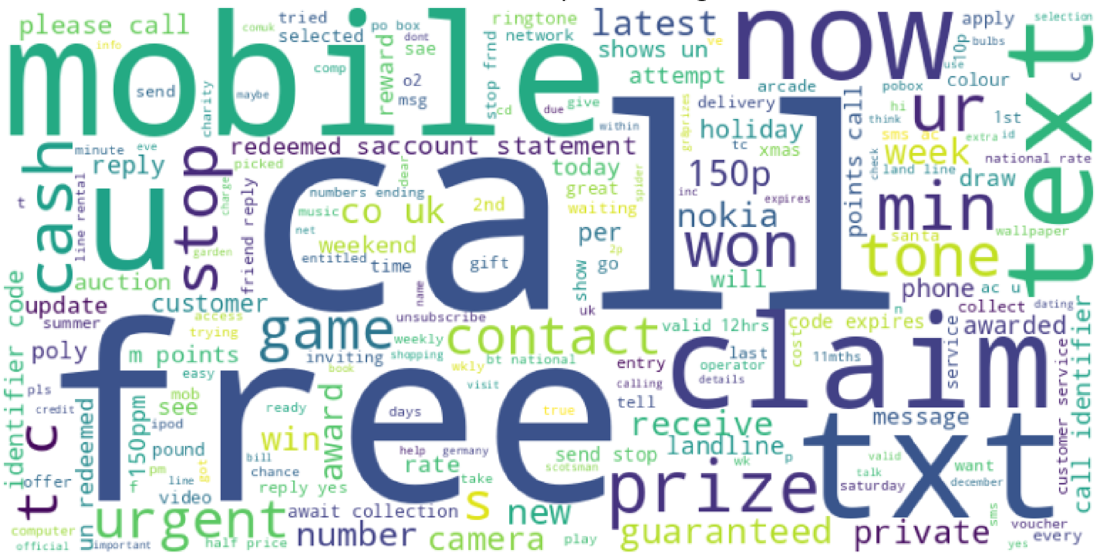
```
[ ]: !pip install wordcloud
from wordcloud import WordCloud, STOPWORDS
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.10/dist-
packages (1.9.3)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.10/dist-
packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from wordcloud) (10.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
```

[ ]:
```
spam_words = ' '.join(SMS_train[SMS_train['Label'] == 'spam']['Message_body'].
  ↪astype(str))
non_spam_words = ' '.join(SMS_train[SMS_train['Label'] ==␣
  ↪'non-spam']['Message_body'].astype(str))
```

[ ]:
```
# Create and display the word cloud for spam messages
spam_wordcloud = WordCloud(width=800, height=400, background_color='white',␣
  ↪stopwords=STOPWORDS).generate(spam_words)
plt.figure(figsize=(10, 5))
plt.imshow(spam_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Spam Messages')
plt.show()
```

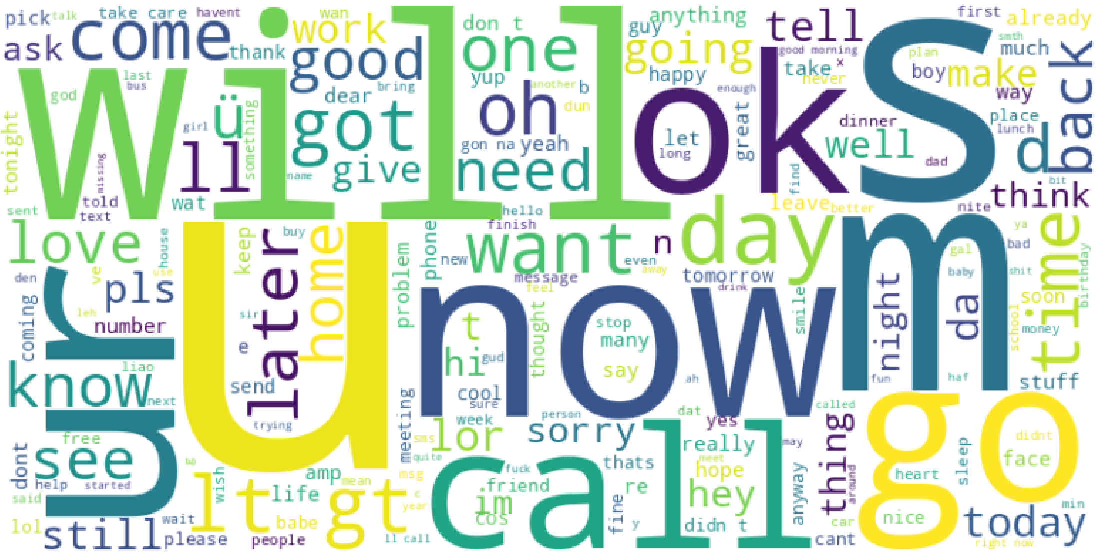
Word Cloud for Spam Messages



Some highlights i can see that give away a spam msg is words like “claim, free, call, prize”

```
[ ]: non_spam_wordcloud = WordCloud(width=800, height=400, background_color='white',␣
       ↪stopwords=STOPWORDS).generate(non_spam_words)
     plt.figure(figsize=(10, 5))
     plt.imshow(non_spam_wordcloud, interpolation='bilinear')
     plt.axis('off')
     plt.title('Word Cloud for Non-Spam Messages')
     plt.show()
```

Word Cloud for Non-Spam Messages



Inaddition I also don't see spam type of words within the non-spam word cloud. Instead i am seeing workds normal msg words like " now, hey, home, tonight "

Model 1: Naive Bayes

```
[ ]: # Split Data
     X_train, X_test, y_train, y_test =␣
      ↪train_test_split(SMS_train_resampled['Message_body'],
     SMS_train_resampled['Label'], test_size=0.25, random_state=42)
```

```
[ ]: #Feature Extraction (TF-IDF)
     # Create a TF-IDF vectorizer to convert text messages into numerical features
     vectorizer = TfidfVectorizer()
     # Fit the vectorizer to the training data and transform it
     X_train_tfidf = vectorizer.fit_transform(X_train)
     # Transform the testing data using the fitted vectorizer
     X_test_tfidf = vectorizer.transform(X_test)
```

We used TF-IDF to convert the text messages into numerical features that our machine learning model could understand. TF-IDF highlights important words that are unique to each message while reducing the overall number of features. This technique is widely used in text classification tasks like spam detection because it effectively captures the essence of the text data, making it easier for the model to learn patterns and classify messages accurately.

```
[ ]: #Model Training (Multinomial Naive Bayes)
     model = MultinomialNB()
     # Train the model using the TF-IDF features and training labels
     model.fit(X_train_tfidf, y_train)
```

```
[ ]: MultinomialNB()
```

```
[ ]: #Model Evaluation
     # Make predictions on the testing data
     y_pred = model.predict(X_test_tfidf)
     # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     # Calculate precision
     precision = precision_score(y_test, y_pred)
     # Calculate recall
     recall = recall_score(y_test, y_pred)
     # Calculate F1-score
     f1 = f1_score(y_test, y_pred)
```

Model Evaluation Method We used accuracy, precision, recall, and F1-score to get a complete picture of how well our model works. Accuracy shows overall correctness, precision focuses on correctly identifying spam, recall ensures we catch most spam, and F1-score balances the two.

These metrics help us ensure the model is accurate and minimizes misclassifications.

```
[ ]: #Print the evaluation metrics
     print(f"Accuracy: {accuracy}")
     print(f"Precision: {precision}")
     print(f"Recall: {recall}")
     print(f"F1-Score: {f1}")
```

```
Accuracy: 0.9856459330143541
Precision: 0.9724770642201835
Recall: 1.0
F1-Score: 0.986046511627907
```

Now I want to test this model on unseen data being the test dataset

```
[ ]: original_test = pd.read_csv('SMS_test.csv')
```

```
[ ]: # Convert to lowercase
     original_test['Message_body'] = original_test['Message_body'].str.lower()
     original_test['Label'] = original_test['Label'].str.lower()

     # Remove punctuation and special characters
     original_test['Message_body'] = original_test['Message_body'].apply(lambda x:␣
      ↪re.sub(r'\W+', ' ', str(x)))

     # Tokenize, clean, and join back to a single string
     original_test['Message_body'] = original_test['Message_body'].apply(lambda x: '␣
      ↪'.join(word_tokenize(x)))

     # Check the cleaned data
     print(original_test[['Message_body', 'Label']].head())
```

```
                                         Message_body Label
0  upgrdcentre orange customer you may now claim …  spam
1  loan for any purpose 500 75 000 homeowners ten…  spam
2  congrats nokia 3650 video camera phone is your…  spam
3  urgent your mobile number has been awarded wit…  spam
4  someone has contacted our dating service and e…  spam
```

```
[ ]: #Extract Features and Target:
     X_original_test = original_test['Message_body']
     y_original_test = original_test['Label']
```

```
[ ]: #Applying TF-IDF Vectorization
     X_original_test_tfidf = vectorizer.transform(X_original_test)
```

```
[ ]: y_original_pred = model.predict(X_original_test_tfidf)
```

[ ]:
```
## Convert numerical predictions to string labels
y_original_pred_labels = np.where(y_original_pred == 1, 'spam', 'non-spam')
```

[ ]:
```
# Calculate metrics with converted predictions
accuracy = accuracy_score(y_original_test, y_original_pred_labels)
precision = precision_score(y_original_test, y_original_pred_labels,␣
 ↪pos_label='spam')
recall = recall_score(y_original_test, y_original_pred_labels, pos_label='spam')
f1 = f1_score(y_original_test, y_original_pred_labels, pos_label='spam')
```

[ ]:
```
# Print the evaluation metrics
print("Performance on Original Test Data:")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
Performance on Original Test Data:
Accuracy: 0.928
Precision: 0.971830985915493
Recall: 0.9078947368421053
F1-Score: 0.9387755102040817
```

Findings: These results indicate that the model performs very well on both the initial test set and the original, unseen test data. While there's a slight drop in performance on the original test data, the model still maintains high accuracy, precision, recall, and F1-score. This slight decrease is expected when evaluating a model on unseen data, as it may encounter new patterns and challenges. The fact that the drop in performance is not significant suggests that the model has generalized well and can be used for real-world applications.

Potential Improvements Even though I performed well with the Multinomial Naive Bayes model, there's always room for improvement. Here's what I think we could explore further:

- Catching More Spam (Recall):

Adjusting My Sensitivity: I could be made more sensitive to spam by adjusting my decision threshold. This means I might classify more messages as spam, but it would also reduce the chances of missing any actual spam. Creating More Examples: We could create variations of existing spam messages using techniques like synonym replacement or back-translation. This would help me learn to identify spam in different forms and improve my ability to generalize.

- Understanding the Context:

Looking at Word Groups: I could be trained to look at groups of words (n-grams) instead of just individual words. This would help me understand the context of a message better and identify spam patterns more effectively. Exploring Word Meanings: We could use word embeddings to represent words as vectors, capturing their semantic relationships. This would allow me to understand the meaning behind words and identify spam based on the overall message intent.

- Fine-tuning My Settings:

Adjusting My Parameters: We could carefully fine-tune my settings (hyperparameters) to optimize my performance for this specific dataset. This would involve experimenting with different settings and finding the combination that works best for spam detection.

- Trying Other Approaches:

Support Vector Machines (SVM): We could try using SVMs, which are known for their ability to handle complex relationships between features. They might be able to identify subtle patterns in spam messages that I currently miss. Logistic Regression: Logistic regression is another model that could be explored. It's simple but effective and might offer a different perspective on spam detection.

By implementing these improvements, focusing on catching more spam, and exploring alternative approaches, I believe we can further enhance my performance and make me an even more effective spam detection tool.