Email Classification Using NLP

MD Jisan Uddin Chowdhury (26100)
Rhine-Waal University of Applied Science
Md-Jisan-Uddin.Chowdhury@hsrw.org
June 28, 2021

Abstract

The increased number of unwanted emails called “spam” has led to the intense need for more reliable and robust spam detection filters. Email filtering is an important feature for service providers. Machine learning techniques are widely used by service providers such as Gmail, Outlook to successfully filter and classify them. There are several email spam filtering techniques available. This paper will illustrate some of the specific Machine learning techniques such as Naive Bayes, Support Vector Machine, and TF-IDF and demonstrate the outcome.

Keywords

Naive bayes, Support Vector Machine or SVM, TF-IDF, Confusion Matrix, Wordcloud, Tokenization, Lemmatization, Bag of Words.

Introduction:

Spam message filtering is a tough challenge. The path between spam and non-spam emails is thin, and the metrics used to identify it evolve with time. Natural Language Processing (NLP) is a field of artificial intelligence that allows machines to understand and interpret human natural languages.Once the preprocessing is done and the dataset is transformed into a standard format, various machine learning techniques can be applied to classify the dataset. In general, its tasks break down language into smaller, essential components, try to understand links between the pieces, and explore how the pieces interact to form meaningful links. Spam email comes in a wide range of forms. Many of the messages are simply unpleasant attempts to bring attention to a cause or propagate inaccurate information. They all have one thing in common: they are unrelated to the recipient's needs. A spam-detector algorithm must figure out how to filter out spam while avoiding marking legitimate items that people wish to receive in their mailbox. Spam filtering has become increasingly important and relevant as email has increasingly been used to exploit users and their data. Organizations

must implement a spam filter to limit the danger of people accidentally clicking on something they shouldn't, thereby protecting their internal data from a cyber attack.

1.Text Preprocessing:

Text preprocessing has long been a crucial step in natural language processing (NLP). It translates text into a usable format, making machine learning algorithms more effective. Email data is full of unstructured mess, so before extraction and modeling, it is important to preprocess the data. It's now quite simple to conduct this preprocessing with just a few lines of python code thanks to the nltk package. Since we are working with spam classification, before applying our algorithms such as SVM and Naive Bayes, and machine learning techniques such as TF-IDF, BagOfWord, we have to clean our raw data. Removing stopwords, punctuation, special characters, email subjects, converting int data type to string, tokenization, lemmatization, etc. are part of text preprocessing.

Text Analytics:

In this part, we'll try to focus more on the importance of text preprocessing just by simply visualizing the word frequency in spam and non-spam emails. For visualizing we will use a bar plot to demonstrate the more frequently used words in spam and non-spam emails.

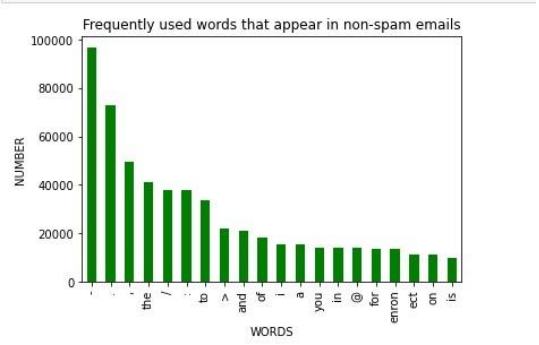


Fig. 1: Bar Plot of word frequency Non-Spam Emails.

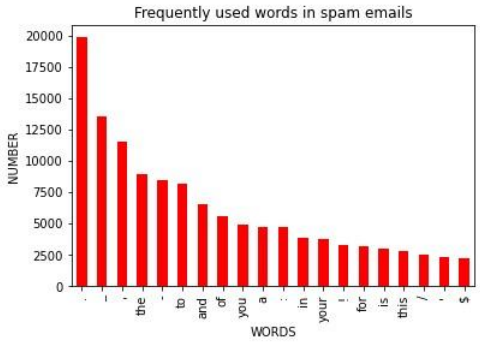


Fig. 2: Bar Plot of word frequency Spam Emails.

It is clearly visible that in both kinds of emails there are a frequent number of punctuations available such as comma, full stop, slash, etc. They actually have no practical impact or relation with our approach. So removing them from our dataset will make our process much easier and smooth. That is why text preprocessing has a significant impact on the journey of spam classification.

1.1. Tokenization:

The technique of tokenization entails breaking down a huge block of text into smaller tokens. In this scenario, tokens can be words, characters, or subwords. Word,

character, and subword (n-gram characters) tokenization are the three types of tokenization. For example, if we consider the sentence: “Math is not difficult”.The most popular method for creating tokens is to use space. The tokenization of the statement, using space as a delimiter, yields four tokens: math-is-not-difficult. It's a Word tokenization example since each token is a word. Characters or subwords, on the other hand, can be used as tokens. For example, let’s consider “FEWER”:

- Character Tokens: F-E-W-E-R
- Subword Tokens: FEW-ER

Out[4]:

	text	spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1

Fig.3: Dataset before tokenization

```
In [20]: print(dt['tokenized'])

0       [Subject, :, naturally, irresistible, your, co...
1       [Subject, :, the, stock, trading, gunslinger, ...
2       [Subject, :, unbelievable, new, homes, made, e...
3       [Subject, :, 4, color, printing, special, requ...
4       [Subject, :, do, not, have, money, ,, get, sof...
                              ...
5723    [Subject, :, re, :, research, and, development...
5724    [Subject, :, re, :, receipts, from, visit, jim...
5725    [Subject, :, re, :, enron, case, study, update...
5726    [Subject, :, re, :, interest, david, ,, please...
5727    [Subject, :, news, :, aurora, 5, ., 2, update,...
Name: tokenized, Length: 5728, dtype: object
```

Fig.4: Dataset after tokenization

For our project we have used a python library named “NLTK” which is short for Natural Language Toolkit. NLTK contains a module called *tokenize()* which further classified into 2 sub categories:

- Word Tokenize: use the *word_tokenize()* method to split a sentence into tokens
- Sentence Tokenize: use the *sent_tokenize()* method to split a paragraph into sentences.

1.2. Removing Stopwords & Punctuation:

In Machine learning when we work with text data, we get them as raw sentences. A raw sentence may include punctuation such as full stop, semi-colon, dash, etc., special characters, and so on. They actually don’t add any distinctive value to our dataset instead of differentiating from other texts. The same idea also goes for stopwords. A stop word is a widely used word (such as "the," "a," "an," or "in") that a search engine has been configured to reject both while indexing and retrieving entries as the result of a search query.We don't want these terms to consume processing time or database space. We can easily get rid of them if we keep a list of keywords we consider stop words. In Python, the NLTK (Natural Language Toolkit) has a list of stopwords in 16 different languages. Here is a demo of how it works:

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Fig.5: Stopwords demo

This process will make our further tasks more efficient.

```
In [46]: dt['filtered_text']

Out[46]: 0       naturally irresistible corporate identity lt r...
         1       stock trading gunslinger fanny merrill muzo co...
         2       unbelievable new homes made easy im wanting sh...
         3       4 color printing special request additional in...
         4       money get software cds software compatibility ...
                                      ...
         5723     research development charges gpg forwarded sh...
         5724     receipts visit jim thanks invitation visit ls...
         5725     enron case study update wow day super thank m...
         5726     interest david please call shirley crenshaw a...
         5727    news aurora 5 2 update aurora version 5 2 fast...
         Name: filtered_text, Length: 5728, dtype: object
```

Fig.6: Our text column after removing stopwords & punctuation

```
In [47]: dt.head(5)

Out[47]:
```

	text	spam	length	tokenized	filtered_text
0	Subject: naturally irresistible your corporate...	1	1484	[Subject, :, naturally, irresistible, your, co...	naturally irresistible corporate identity lt r...
1	Subject: the stock trading gunslinger fanny i...	1	598	[Subject, :, the, stock, trading, gunslinger, ...	stock trading gunslinger fanny merrill muzo co...
2	Subject: unbelievable new homes made easy im ...	1	448	[Subject, :, unbelievable, new, homes, made, e...	unbelievable new homes made easy im wanting sh...
3	Subject: 4 color printing special request add...	1	500	[Subject, :, 4, color, printing, special, requ...	4 color printing special request additional in...
4	Subject: do not have money , get software cds ...	1	235	[Subject, :, do, not, have, money, ,, get, sof...	money get software cds software compatibility ...

Fig.7: Dataset after being filtered

The figures shown above demonstrate how simple our dataset looks, after removing unnecessary things.

1.3. Lemmatization:

The process of lemmatization is the reduction of a word to its base form. Lemmatization takes the context into account when converting a word to its meaningful base form. For example, lemmatization would correctly identify the base form of “loving” to “love”.
“Loving” >> lemmatization >> “Love”.
Additionally, the single word can have many ‘lemma's at times. As a result, we should determine the ‘part-of-speech' (POS) tag for the word in that context and extract the relevant lemma depending on the context.
For our project we have used a NLTK interface named “*WordNetLemmatizer*”. To lemmatize, we must first build a *WordNetLemmatizer()* instance and then call the *lemmatize()* function on a single word.

LEMMATIZATION

```
: lema = nltk.WordNetLemmatizer()
  #nltk.download()

: dt['filtered_text'] = dt['filtered_text'].map(lambda text: lema.lemmatize(text))

: dt['filtered_text'][9]

: 'save money buy getting thing tried cialis yet even imagine like real man bed thing great errrectlon provided exactly want cial
  is lot advantages viagra effect lasts 36 hours ready start within 10 minutes mix alcohol ship country get right '
```

Fig.8: Lemmatization.

A simple 2 line python code is enough to solve this complex task.

Wordcloud:

Word clouds are graphical portrayals of word frequency that lend more prominence to words that appear frequently in a source text. The more frequently the image's words occur in the document, the larger they are (s). Such type of representation can help assessors in interpretative textual analysis by emphasizing words that appear frequently in a collection of interviews, documents, or other text. With our email dataset, we have tried to get a look at which kind of words appeared more in our spam and non-spam emails.

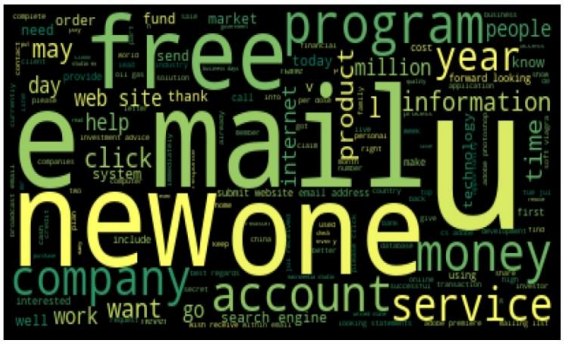


Fig.9: WordCloud of SPAM emails

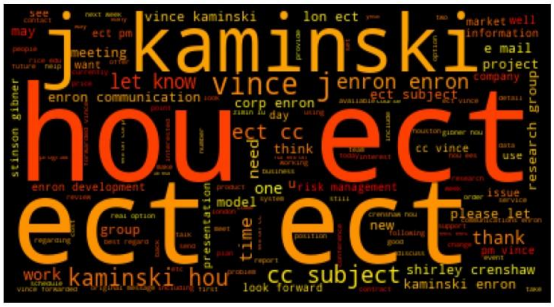


Fig.10: WordCloud of non SPAM emails

This simple visualization technique gives us a quick overview about the abundance of words in a dataset.

2. Bag of Words:

Bag Of Words is a technique for extracting features from text sources. Machine learning algorithms can be trained using these features. It develops a vocabulary of all the unique words found in the training set's papers. In a nutshell, it's a bunch of words that are used to represent a sentence, with no respect for the order in which they appear. Among the most major challenges with text is that it is unstructured and untidy; machine learning algorithms prefer structured, well-defined fixed-length inputs, and we can use the Bag-of-Words method to convert variable-length texts into fixed-length vectors.

Fig.10: BOW steps

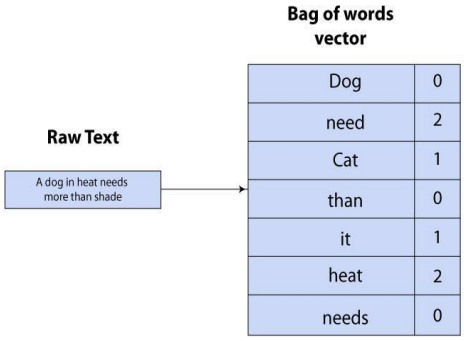


Fig.11: BOW Working.

Bag of words, as the names indicate, is simply a count of all relevant words in a document.

Fig.12: BOW in our project

3. TF-IDF:

The TF-IDF(Term Frequency-Inverse Document Frequency) is a statistical measure that assesses the relevance of a word to a document in a set of documents. This is performed by multiplying two metrics: the number of times a word appears in a document and the word's inverse document frequency over a collection of documents.

Fig.13: TF-IDF basic idea.

3.1. TF-IDF Mathematically Explained:

TF-IDF weight is composed by two terms: the first computes the normalized Term Frequency (TF), the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF) computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears. To put it another way, the TF-IDF score for the word t in document d from document set D is determined as follows:

TF IDF(t,d,D) = TF(t,d) . IDF(t,D)

Where:

TF(t,d) = log(1+ freq(t,d))

IDF(t,D) = $log\ (N/count(d \in D : t \in d))$

The TF-IDF values of our project follows:

TF-IDF

```
In [30]: tfidf_vectorizer = TfidfTransformer().fit(count)
         tfidf = tfidf_vectorizer.transform(count)

In [31]: classifier = MultinomialNB()
         targets = dt['spam'].values
         classifier.fit(count, targets)

Out[31]: MultinomialNB()

In [35]: print(tfidf)

           (0, 36494)    0.06909969862066065
           (0, 36361)    0.06758050177560837
           (0, 36359)    0.059688219024067 6
           (0, 35954)    0.13195884687352108
           (0, 34965)    0.05234204590356858 5
           (0, 34731)    0.0938569785598016
           (0, 33491)    0.06922230324784197
           (0, 32967)    0.09491736727758139
           (0, 32775)    0.060745800715687 14
           (0, 32549)    0.11154173247342794
           (0, 32534)    0.1196379573681351
           (0, 32252)    0.1196379573681351
           (0, 32196)    0.07897017578675673
           (0, 31901)    0.11852622471547422
           (0, 31892)    0.08500621505740441
           (0, 31481)    0.10455724023138163
           (0, 30158)    0.04608064372595012
           (0, 29798)    0.09335029857939502
           (0, 28818)    0.07692922620869343
           (0, 28205)    0.09214520050901033
           (0, 28049)    0.11016292660431957
           (0, 27925)    0.06592853454480324
           (0, 27166)    0.07329840924511424
           (0, 27165)    0.05762396774659246
           (0, 27063)    0.11083853780061895
```

Fig.14: TF-IDF result in our project

In tf–idf, a high weight is achieved by a high term frequency. SInce the ratio inside the idf's log function is always greater or equal 1, the value of idf (and tf-idf) is greater than or equal 0. As a term appears in more documents, the ratio inside the log approaches 1, bringing the idf and tf-idf closer to 0.

4. Naive Bayes Classifier:

The Bayes Theorem is used to create a Naive Bayes classifier. It calculates membership probabilities for each class, such as the likelihood that a certain record or data point belongs to that class. The most likely class is defined as the one having the highest probability. For our project we've used a Multinomial Naive Bayes Classifier. On multinomialy distributed data, MultiNomial Naive Bayes is the preferred method. It is one of the most

well-known algorithms. Which is used to categorize text (classification). The occurrence of a word in a document is represented by each event in text classification. Here we will train different bayes models by changing the regularization parameter α. After That we'll test the set, assess the model's accuracy, recall and precision.

Out[35]:

	Alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision
0	0.00001	0.999739	0.981491	0.941176	0.987234
1	0.11001	0.998958	0.989952	0.987830	0.974000
2	0.22001	0.998958	0.988895	0.987830	0.970120
3	0.33001	0.998436	0.988366	0.987830	0.968191
4	0.44001	0.998176	0.988366	0.987830	0.968191
5	0.55001	0.997654	0.988366	0.987830	0.968191
6	0.66001	0.997394	0.988366	0.987830	0.968191
7	0.77001	0.997133	0.987837	0.987830	0.966270
8	0.88001	0.996873	0.988366	0.987830	0.968191
9	0.99001	0.996612	0.988366	0.987830	0.968191

Fig.15:Our Model's accuracy with different α values in NBC

4.1. Naive Bayes Mathematically Explained:

Now let's try to focus on the math behind this algorithm with an example:
We assume that we have 2 folders of emails labelled spam(red) and normal(blue). First of all we make a histogram of all the words that occur in the blue.

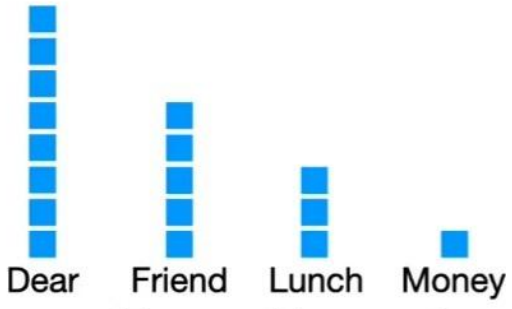
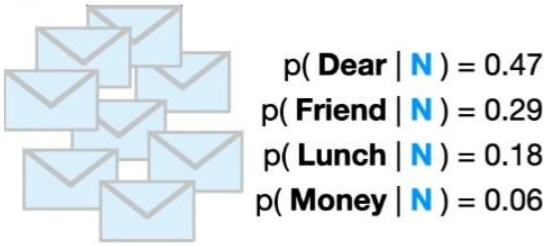


Fig.16: Histogram of our assumed word freq.

Fig.16.1: Histogram of our assumed word freq.

From the histogram then we calculate the probability of each word given that they occur in the normal and spam emails.



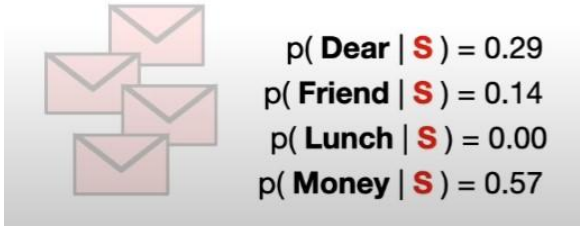


Fig.17: Probability of words in normal & spam emails

Now we assume we have a new message that says: "Dear Friend". Despite what it says we start with an initial guess that it's a normal message. The guess can be any probability we want, but from the training data we can estimate the common guess. For example, since 8 out of 12 messages are normal, so our initial guess will be:

p(N) = 8(N)/(8(N)+4(S)) = 0.67

The value of p(N), the initial guess that we observe a Normal email is called Prior Probability. Now the probability of our incoming message “Dear Friend” occurs in normal email will be:

p(N|Dear Friend)** = p(N) * p(Dear | N) * p(Friend | N)= 0.67*0.47*0.29 = **0.09

Similarly, for spam messages the initial guess will be:

$p(S) = 4/(4+8) = 0.33$

The probability of our incoming message “Dear Friend” occurs in spam email will be:

p(S|Dear Friend)** = p(S) * p(Dear | S) * p(Friend | S)= 0.33*0.29*0.14 = **0.01

Since, 0.09 > 0.01, our algorithm will classify that “Dear Friend” as a normal email.

5. Support Vector Machine (SVM):

SVM stands for Support Vector Machine and is a supervised machine learning technique that can be used for classification and regression. SVMs are more typically utilized in classification applications.SVMs are based on the concept of determining the optimum hyperplane for dividing a dataset into two classes. Before going to the math we need to clarify some definitions like, hyperline, support vector, distance margin etc.

Hyperplane : A hyperplane is a line that divides and categorizes a set of data in a linear manner. Intuitively, the further our data points are from the hyperplane, the more certain we are that they have been classified correctly.

Support Vector: The data points closest to the hyperplane, or the points of a data set that, if removed, would change the position of the dividing hyperplane, are termed support vectors.

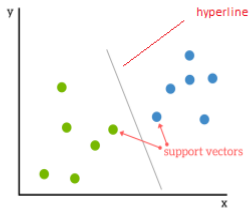


Fig.18 : SVM

Distance Margin: The sum of the shortest distance from the closest positive point and negative point is called Distance Margin.

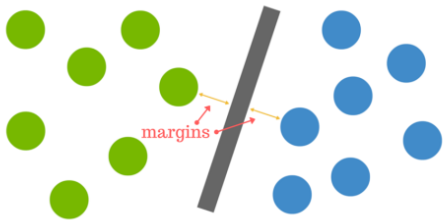


Fig.18.1 : SVM margins

The goal is to select a hyperplane with the greatest possible margin between it and any point in the training set, increasing the likelihood of new data being properly classified. But what happens when the hyperplane is not clear enough to

distinguish or we have a linearly non separable dataset?

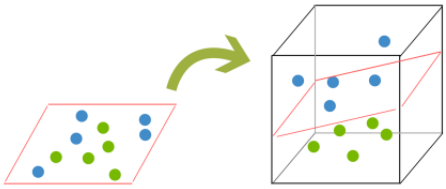


Fig.18.2 : SVM

In that case we have to move away from the 2 dimensional to a 3 dimensional view. In a 3D area our hyperplane can no longer be a line. The goal is to keep mapping the data into higher and higher dimensions until a hyperplane can be created to segregate it.

5.1. SVM Kernel Explained:

For our project we have used SVM with the Gaussian kernel. The Gaussian kernel is given as:

$$K(x, y) = exp\left(\|x - y\|^2 / 2\sigma^2\right)$$

The Gaussian kernel is an exponentially decaying function in the input feature space. It decays uniformly in all directions around the support vector, leading to hyper-spherical contours of the kernel function. The role of a support vector in the classification of a data point is tempered by global prediction usefulness.

```
In [42]: best_index = models['Test Precision'].idxmax()
         models.iloc[best_index, :]

Out[42]: C                 500.000000
         Train Accuracy      1.000000
         Test Accuracy       0.978847
         Test Recall         0.937120
         Test Precision      0.980892
         Name: 0, dtype: float64

In [43]: models[models['Test Precision']==0.980892]

Out[43]:
```

	C	Train Accuracy	Test Accuracy	Test Recall	Test Precision

```
In [ ]:

In [44]: best_index = models['Test Accuracy'].idxmax()
         svc = svm.SVC(C=list_C[best_index])
         svc.fit(x_train, y_train)
         models.iloc[best_index, :]

Out[44]: C                 500.000000
         Train Accuracy      1.000000
         Test Accuracy       0.978847
         Test Recall         0.937120
         Test Precision      0.980892
         Name: 0, dtype: float64
```

Fig.19 : SVM model accuracy

6. Confusion Matrix:

An N x N matrix is used to evaluate the performance of a classification model, where N is the number of target classes. The matrix compares the actual goal values to the machine learning model's predictions. This provides us with a comprehensive picture of how well our classification model is working and the types of errors it makes. LEt's decipher the matrix:

- The target variable has two values: Positive or Negative
- The columns represent the actual value of the target variable
- The rows represent the predicted values of the target variable.

CONFUSION MATRIX with naive bayes

```
In [39]: m_confusion_test = metrics.confusion_matrix(y_test, bayes.predict(x_test))
         pd.DataFrame(data = m_confusion_test, columns = ['Predicted 0', 'Predicted 1'],
                      index = ['Actual 0', 'Actual 1'])

Out[39]:
```

	Predicted 0	Predicted 1
Actual 0	1398	0
Actual 1	245	248

we've classified 245 spam as non spam emails and we haven't misclassify any non spam emails

Fig.20 : Confusion Matrix with Naive Bayes

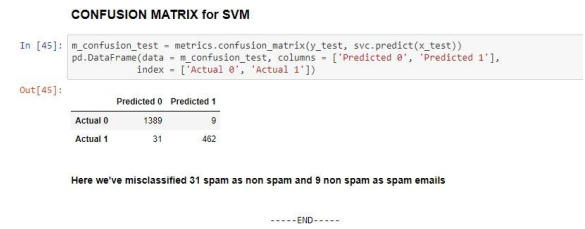


Fig.21 : Confusion Matrix with SVM

Conclusion:

Spam detection is a very small subset of NLP which is also a subfield of artificial intelligence. Being a beginner in the data science field and gaining such experience about how to program computers to process and analyze large amounts of natural language data was challenging. There are so many techniques and algorithms available to solve such modern day problems. Out of them I just got familiar with a few of the many and also the mathematics behind these. I have tried my best to introduce and explain them through this paper.

Link To GitHub: Here is the link to the complete code in my GitHub Repository.
Click Here

References:

Gaussian Kernel

Support Vector Machine

Naive Bayes

TF-IDF

Text Preprocessing method

BOW explained