Problem Statement:

Building a spam detector for messages. Given a set of messages and labels associated with them (spam or ham), the goal is to build a classifier that will classify new messages as spam or ham correctly. It is important to note what the end goal of this classifier is. Since this model is used to classify spam emails, we need to make sure that the model correctly classifies the maximum number of Spam emails. The goal is to minimize the number of spam emails entering the inbox of a given user. We can afford to have some error of misclassifying a few ham emails as spam as we are interested in capturing all the Spam emails. This is the main aim of the classifier that I am building.

Data set used:

I have used the SMS Spam Collection dataset from Kaggle. (Link: https://www.kaggle.com/uciml/sms-spam-collection-dataset). The dataset has a total of 5572 messages that have been labeled as either Spam or Ham. It has 5 columns with the first 2 columns being the label and the content of the messages.

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

The same dataset can be extended to emails as well. At the beginning of the assignment, I tried to extract my own emails for analysis. I used the IMAP protocol to communicate with Googles email server and pull emails from my account. Below are the reasons why I chose to proceed with the SMS dataset:

- My email had very few spam messages compared to Ham messages. (1000+ Ham vs around 20 spams messages)
- Out of these 20 messages, only 4 of them had bodies that could be extracted
- After extracting the bodies the body of all my emails, I was finding it very difficult to get rid of the extra stuff like html tags, hyperlinks, non-unicode characters, etc. I wasn't able to figure out how to automate the cleaning the data. I was also not aware of how to automate the normalization of data (i.e. remove unwanted characters, convert all hyperlinks to "link", convert all $-signs to "dollar" etc.)

Also, I wanted to focus most of my time in learning the techniques of NLP (bag of words, vectorization of words, creating tf-idf vector matrices, etc.). I have given more emphasis to that rather than spending too much time in pre-processing my own emails

Models for classification:

Naïve Bayes Classifier

Used Multinomial Naïve Bayes Classifier to classify emails as spam or ham. Steps followed to classify:

- Use CountVectorizer to apply bag-of-words model to get all the words and counts in the corpus
- Convert these counts into document-text matrix with TF-IDF scores for words in each message of the corpus
- Apply Multinomial Naive Bayes model to fit the training data and predict labels for the test data

```
''' Use Pipeline to sequentially apply an intermediate list of
    transforms and fits to obtain a final estimator. The final estimator should
    only implement fit with the training data to train the model.
'''

pipe_MNB = Pipeline([('bow'   , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                     ('tfidf'   , TfidfTransformer()),
                     ('clf_MNB' , MultinomialNB()),
                    ])
```

K-Nearest Neighbors Classifier

Steps followed to classify using K-NN:

- Use CountVectorizer to apply bag-of-words model to get all the words and counts in the corpus
- Convert these counts into document-text matrix with TF-IDF scores for words in each message of the corpus
- Use KNN on this document term matrix to classify
- Used GridSearchCV to perform five-fold cross validation

```
# k = 5
pipe_KNN = Pipeline([ ('bow'   , CountVectorizer(analyzer = remove_punctuation_and_stopwords) ),
                      ('tfidf'   , TfidfTransformer()),
                      ('clf_KNN' , KNeighborsClassifier() )
                       ]) #default k=5
parameters_KNN = {'clf_KNN__n_neighbors': (5,), }

grid_KNN = GridSearchCV(pipe_KNN, parameters_KNN, cv=5,
                        n_jobs=-1, verbose=1)
```

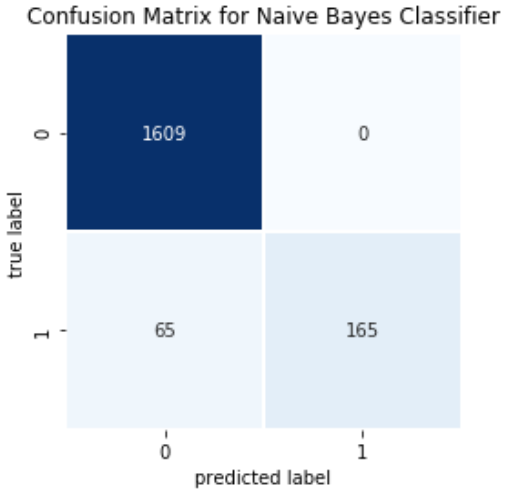
Model Performance

As discussed above, the goal is to minimize the number of spam emails entering the inbox of a user. We can afford to have some error of misclassifying a few ham emails as spam as we are interested in capturing all the Spam emails. So, the true negative rate (specificity) must be maximum. I have split the dataset into test and train sets with 67% of the data used as test set and the remaining 33% used as the training set. The rows were randomly chosen.

```
Size of training set = 3733
Size of test set = 1839
```

Multinomial Naïve Bayes

Confusion Matrix



Label 0 – Ham Label 1 -Spam

For the given confusion matrix:

1. True Positives = 1609
2. True Negatives = 165
3. False Positives = 0
4. False Negatives = 65

```
Sensitivity = 1.0
Specificity = 0.7173
```

Here we can also see that out of 230 Spam messages the test dataset, the model correctly identified 165 of them (around 71.73%). Also, all Ham messages are correctly getting classified.

Accuracy

```
MNB predicted spam mails with an accuracy of 96.46547036432844 %
```

Accuracy is the percentage of correct classifications (both spam and ham)

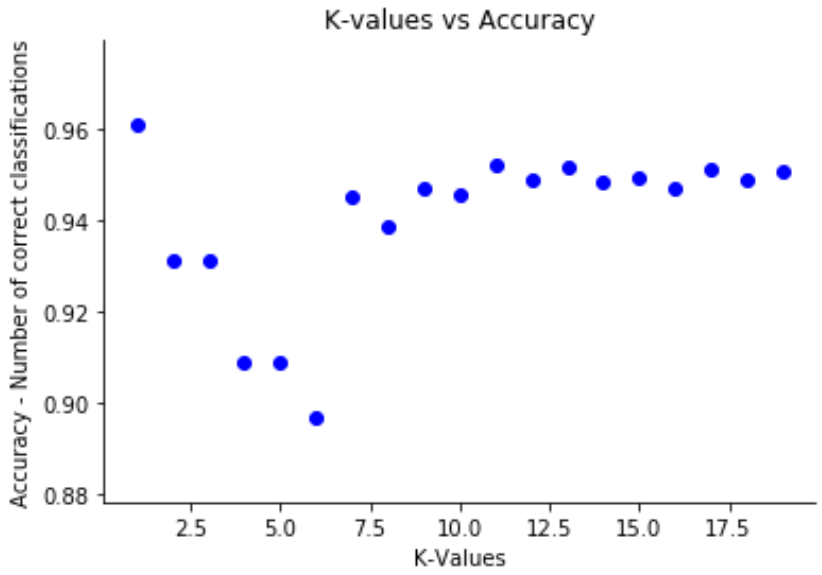
```
# Predict the values for the test data set
y_pred_MNB = pipe_MNB.predict(X_test)

# Calculate the accuracy of the classification
accuracy_ratio = accuracy_score(y_test,y_pred_MNB) # proportion of correct predictions
print("MNB predicted spam mails with an accuracy of",accuracy_ratio*100,"%")
```

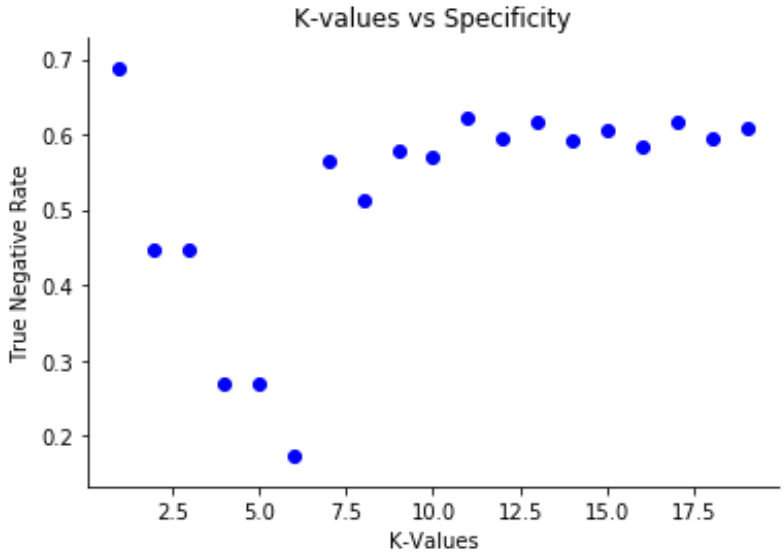
```
MNB predicted spam mails with an accuracy of 96.46547036432844 %
```

K-Nearest Neighbors Classifier

To find the optimum value of K, I have plotted accuracy and specificity values for different values of K to check which K gives the maximum for both these values. As discussed above, we will focus more on Specificity as I want the model to correctly classify maximum number of Spam emails.

Below are the graphs.

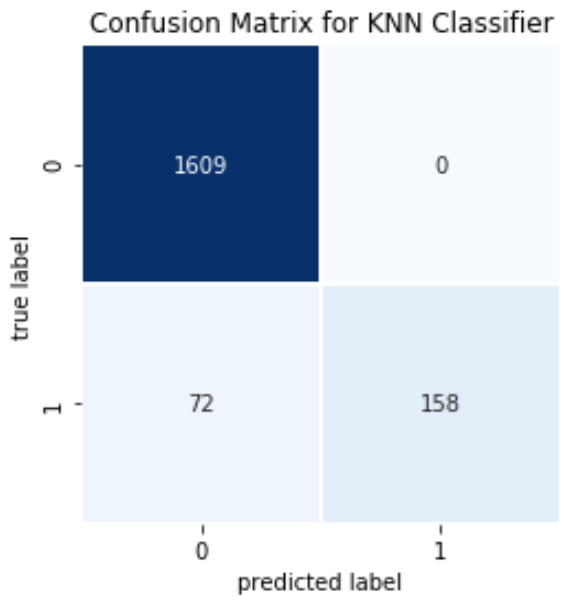




We can see that the above two graphs look identical. The maximum specificity and accuracy are obtained when K=1. This is probably because the data has way too many features (9431). In so many dimensions, the distance between all points in that space become identical.

Confusion Matrix

Below is the confusion matrix for K=1 obtained above.



Label 0 – Ham Label 1 -Spam

For the given confusion matrix:

5. True Positives = 1609
6. True Negatives = 158
7. False Positives = 0
8. False Negatives = 72

```
Sensitivity = 1.0
Specificity = 0.6869565217
```

Here we can see that out of 230 spam messages, 158 are correctly classified as Spam (about 68%). Also, all the Ham messages are getting correctly classified.

Accuracy

```
KNN was able to predict Spam emails with and accuracy of 96.08482871125 %
```

Accuracy is the percentage of correct classifications (both spam and ham)

```
# Predict using the test data set
y_pred_KNN = grid_KNN.predict(X_test)

# Calculate the accuracy
acc_KNN = accuracy_score(y_test, y_pred_KNN)
print("KNN was able to predict Spam emails with and accuracy of ",acc_KNN*100,"%")
```

```
KNN was able to predict Spam emails with and accuracy of  96.08482871125612 %
```

Final Verdict

	Naïve Bayes	KNN (K=1)
Specificity	0.7173	0.68695
Accuracy	0.96465	0.96084
Sensitivity	1	1

We can see that in term of accuracy and specificity, Naïve Bayes does better at classifying spam messages better than K-NN. So, we will choose Naïve Bayes for this