Project 1.

Email Spam Classification

Haejong Dong & Marko Haataja

1. Introduction

Today, we are living in a period of human history known as the Information Age. The phenomenon of the Information Age is that the digital industry creates a knowledge-based society surrounded by a high-tech global economy. The world progressed to the Information Age by growing information storage, information transmission and computation capabilities. [5]

One of the most important factors in progressing to the Information Age is the founding of the Internet. The Internet has enabled new forms of social interaction, activities and social associations. Internet users have steadily risen from about 10% in 1997 to about 80% in 2014. [6] One of the earliest uses, and reasons why it was created, of the Internet was email. Email preceded the Internet, reaching back to the early ARPANET. [7]

Email was the early connector of people, and it is still used today very much. However, soon after the email went mainstream unsolicited messages were being received by most email users. This junk mail is called spam. Spam messages can be commercial but unsolicited or malicious that tries to get user's credentials or credit card number or something else that can bring them money.

In 2010, it was estimated that the proportion of spam email was around 80% of all email messages sent, with Microsoft estimating unwanted messages being up to 97% of all messages on the Internet. This amount of unwanted messages cost estimated 21 billion US dollars of lost productivity in 2004. The amount of lost productivity and user inconvenience begged for a solution, and email spam message filtering was one of the tools used. [9]

To combat the problem of spam, email filtering was used to process the received emails and to organize it according to specific criteria, e.g. spam and ham (legitimate messages). In some cases, this technique reduces the amount of spam received significantly. This does not come without problems, though. Often legitimate messages, or ham, are erroneously categorized into spam, and most likely it is never read by the recipient. [10] This is where data classification and Natural Language Processing (NLP) steps in.

In Data and Text Mining course work, our objective was to study Email Spam Classification, and test several implementations and compare the results.

The project work first familiarized us with basic email classification and testing, using the techniques that we learned earlier in the course. After that, several other weighting schemes needed to be tested and results reported. In addition to other weighting schemes, different ways of choosing the vocabulary were tested. After testing all the variations of different classifiers, weighting schemes and ways of choosing vocabulary, the data set was updated. We wrote a Python script to extract spam and ham messages from our email folders to be used as a data set. The earlier experiments were then re-run with the new dataset. An online email spam corpus was used to increase the amount of data used for training and testing.

2. Literature Review

Several papers throughout the years address the same problems as this paper aims to answer. One paper, titled Machine Learning Methods for Spam E-mail Classification by W.A. Awad & S.M. ELseuofi [14], acknowledges the current situation of email spam similarly to this paper. The paper motivates the research by claiming that a major part of all emails on the Internet are spam, and are costing Internet users a lot. The paper suggests machine learning, instead of traditional static filters with strict rules, as an approach for email filtering. The paper introduces the principles of several machine learning classification methods that are used in email filtering. These include Naïve Bayes, K-nearest neighbour, Artificial Neural Networks, Support Vector Machine, Artificial Immune System and Rough sets classifier methods. The paper uses a data set containing 6000 email messages with a spam rate of 37.04%. This data set was obtained from the public online SpamAssassin email spam corpus. The experiments show that all classifying methods have high accuracy of >95%.

A slightly older research called Machine Learning Techniques in Spam Filtering by Konstantin Tretyakov [16] again introduces the setting where email spam is a big and costly nuisance. The paper presents that there are two general approaches to email filtering, which are Knowledge Engineering and Machine Learning. Knowledge Engineering is the older technique with static rules that determine what email is spam and what is not. Machine Learning approach does not require any specific rules, but can be trained with classified training data using a specific algorithm. The paper introduces some popular classifiers and their performance. The paper aims to acquaint readers (and the author) to machine learning techniques.

3. Research Question

Q1. Comparison between classifications (SVM, Naive Bayes, KNN) on email classification purpose

- which classification algorithm yields the highest accuracy on given train and test datasets?
- which classification algorithm scores the highest in overall?

3. Methodology

We have selected SVM, Naive Bayes, and KNN classifications for major comparison in this report. The classification selection was based on precedent researches carried out on email classification that have reported those were producing high accuracy in email classification [12][13][14][15], also these classifications are renowned for approaching many other supervised machine learning application problems.

We apply unified preprocessing to the datasets are then fed into all classifications at target, which includes removing stop words and entity words - i.e. name of institutions, companies, or person -, stemming words, and lowercasing words. We are interested to see how the processing data beforehand in a way that the computer can learn only from meaningful datasets may affect to the result accuracy.

In case of choosing best K features (feature extraction), we would also import 'SelectKBest' and 'chi2' classes from scikit-learn library which will help select K highest feature values from the total word features.

The classification evaluation will be done on the basis of confusion matrix, mean of cross validation scores, accuracy error rate, and the general performance metrics (accuracy, recall, precision, and f1 scores). First we would observe preprocessed results of all classifications followed by comparison between feature weighting schemas. Dimensional reduction methodologies are employed only to a specific algorithms due to applicability and the need. For the essential performance comparison between algorithms, we are going to try different models available depending on the type of classification, for instance, SVM supports number of kernel types such as linear, RBF, and polynomial which will classify given dataset with different cut-off lines.

Finally, we find some of useful evaluation metrics picked from previous experiments to apply on a bigger set of data - preferably are real-world, noisy samples - to generalize outcomes derived from relatively smaller and neatly labeled samples.

Instead of implementing classification algorithms from the scratches, we employed Python libraries such as scikit-learn, nltk, and numpy are broadly used in text analysis and machine learning, particularly scikit-learn is famous for writing machine learning algorithms and is open source, licensed under BSD [11].

3.1. Email Extraction

One requirement for the course work was to implement an application to retrieve emails from various email providers. These emails would then be used in the training and testing of classifiers. The suggested way of doing this was taking advantage of Pythons built-in IMAP libraries, and that was also done. This section presents how the application was developed, how the application works, what is the output of the application and the significance of all this for rest of the work.

Internet Message Access Protocol, or IMAP, is a standard used by email clients to retrieve email messages from mail server. Virtually all modern email clients and servers support IMAP and the older POP3 (Post Office Protocol).

The built-in IMAP library for Python, imaplib, implements the client for communicating with IMAP servers. Most of the IMAP commands defined by the protocol are available as methods when using the Python imaplib. The documentation for this library can be found at https://pymotw.com/2/imaplib/.

The application created for email retrieval was called email_scraper.py. A Python example code was used as the basis for email_scraper application. This code provided an example of connecting to a Gmail email account via IMAP. The example code can be found at http://www.voidynullness.net/blog/2013/07/25/gmail-email-with-python-via-imap/.

The email_scraper application uses Pythons imaplib to connect to the email server. This was tested with two Gmail accounts and one MBNet account (a Finnish email provider / tech magazine).

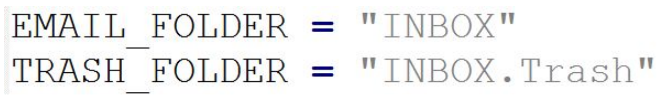
```
INPUT email account name, account password, imap address
TRY login to email server
    IF successful
        PROCEED
    ELSE
        PRINT error
        EXIT
```

Figure 1. Connection code.

In Figure 1 the connection code can be seen. The imaplib login function requires the IMAP address of an email provider, account name and a password. In email_scraper application, getpass is used to not have to have clear text passwords. Getpass prompts the user for a password without echoing, thus preventing password leak even if someone is watching when password is entered.

```
PRINT list of mailboxes
SELECT mailbox to be used
IF mailbox exists
    CALL process_mailbox function
ELSE
    PRINT error
    EXIT
```

Figure 2. Mailbox selection.



```
EMAIL_FOLDER = "INBOX"
TRASH_FOLDER = "INBOX.Trash"
```

Figure 3. Mailbox names.

When an email is sent or received, it goes into some mailbox in your email account. While email accounts might have some default folders or mailboxes for mails, such as Trash, Drafts, etc., most email providers make it possible for the user to make custom mailboxes. For the specific need of extracting emails from a couple of email accounts, the correct mailboxes were manually identified. The mailbox selection process is seen in Figure 2. The list() function from imaplib return all mailboxes from the account. This can also be seen in Figure 7. From these mailboxes, correct ones were identified and selected. In Figure 3 the selected mailbox names are presented. The chosen mailbox is then passed to process_mailbox function.

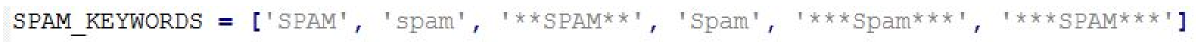
```
INPUT imap address
FOR all emails in box
    FIND UIDs
FOR each UID
    GET email parts
    FOR email part
        GET text payload
```

Figure 4. Process mailbox function part 1.

The chosen mailbox is then processed in the process_mailbox function. In the first part of this function, the mailbox is searched for all unique identifiers (uids). The search has to be for unique identifiers instead of regular identifiers. This is because if messages are deleted, only unique identifiers stay unique, and regular identifiers might have duplicates. After that, every email with a unique identifier is parsed through to get the raw email. Because emails have multiple parts in them, we have to use the walk() method to find the text body of the message. This is presented in Figure 4.

```
IF any email subject line word IN SPAM_KEYWORDS
    MAKE spam text file
    SAVE spam email message to spam text file
ELSE
    MAKE ham text file
    SAVE ham email message to ham text file
```

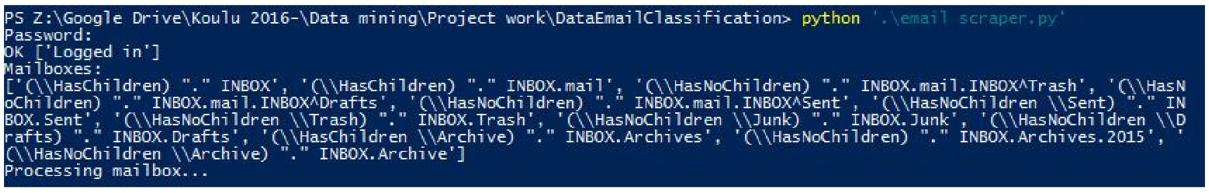
Figure 5. Process mailbox function part 2.



```
SPAM_KEYWORDS = ['SPAM', 'spam', '**SPAM**', 'Spam', '***Spam***', '***SPAM***']
```

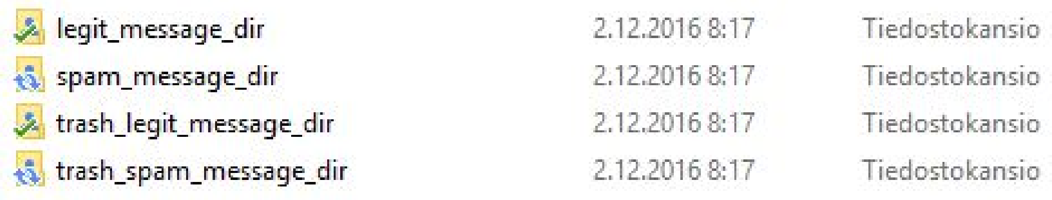
Figure 6. Spam keywords.

In the latter part of the process mailbox function, emails are divided into spam and ham with a crude method, leveraging the spam classification system of the email provider. This can be seen in Figure 5. If the email subject line has the word SPAM in it, the spam classification system of the email provider (in our case) has determined the message to be spam. When the message type is determined, the email content is then saved to a text file in a folder named corresponding to the type of the message, i.e. ham or spam. The folders are presented in Figure 8. The spam keywords used for filtering is seen in Figure 6.



```
PS Z:\Google Drive\Koulu 2016-\Data mining\Project work\DataEmailClassification> python '.\email scraper.py'
Password:
OK ['Logged in']
Mailboxes:
['(\\HasChildren) "." INBOX', '(\\HasChildren) "." INBOX.mail', '(\\HasNoChildren) "." INBOX.mail.INBOX^Trash', '(\\HasN
oChildren) "." INBOX.mail.INBOX^Drafts', '(\\HasNoChildren) "." INBOX.mail.INBOX^Sent', '(\\HasNoChildren \\Sent) "." IN
BOX.Sent', '(\\HasNoChildren \\Trash) "." INBOX.Trash', '(\\HasNoChildren \\Junk) "." INBOX.Junk', '(\\HasNoChildren \\D
rafts) "." INBOX.Drafts', '(\\HasChildren \\Archive) "." INBOX.Archives', '(\\HasNoChildren) "." INBOX.Archives.2015', '
(\\HasNoChildren \\Archive) "." INBOX.Archive']
Processing mailbox...
```

Figure 7. Command line application start-up.



```
legit_message_dir          2.12.2016 8:17     Tiedostokansio
spam_message_dir           2.12.2016 8:17     Tiedostokansio
trash_legit_message_dir    2.12.2016 8:17     Tiedostokansio
trash_spam_message_dir     2.12.2016 8:17     Tiedostokansio
```

Figure 8. Resulting folders from email extraction.

The dataset received from three email accounts contained 3839 ham emails and 1143 spam emails. This dataset was then used for classification and testing, in addition to the original dataset given by the course. To add to this dataset, online database for spam email was used.

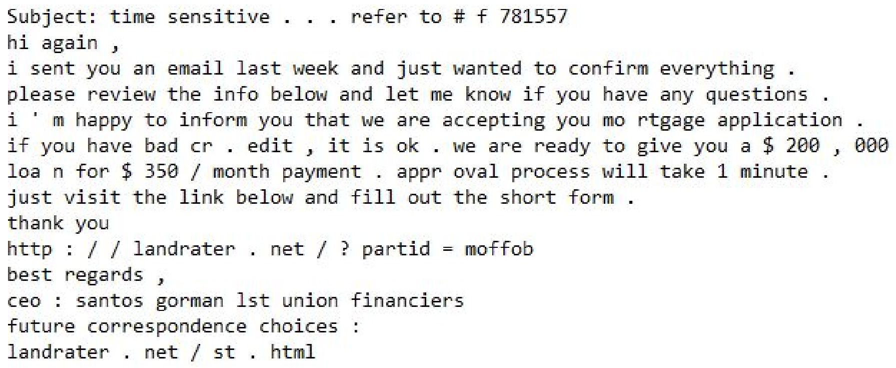
The CSDMC2010 spam corpus is a dataset used for a data mining competition. The dataset is composed of a selection of mail messages, suitable for use in testing spam filtering systems. The site contains multiple datasets, including:

1. Enron-Spam dataset
2. Ling-Spam dataset
3. PU1 and PU123A dataset
4. Spam-Assassin dataset

Both the Ling-Spam dataset and Spam-Assassin dataset were used to obtain more emails for training and testing. The total size of the final dataset was 21571 samples, from which 12952 were ham and 8619 spam.

4. Supervised Email Classifications

The dataset was given 2000 samples—1000 ham emails and 1000 spam emails—as a first start-out. Also, it is worth to highlight that these samples are neatly classified datasets containing ham-/spam-representative content structure; Figure 9 shows one example of spam email amongst 2000 samples.



```
Subject: time sensitive . . . refer to # f 781557
hi again ,
i sent you an email last week and just wanted to confirm everything .
please review the info below and let me know if you have any questions .
i ' m happy to inform you that we are accepting you mo rtgage application .
if you have bad cr . edit , it is ok . we are ready to give you a $ 200 , 000
loa n for $ 350 / month payment . appr oval process will take 1 minute .
just visit the link below and fill out the short form .
thank you
http : / / landrater . net / ? partid = moffob
best regards ,
ceo : santos gorman 1st union financiers
future correspondence choices :
landrater . net / st . html
```

Figure 9. Example of a spam email

Before training the classifier, we needed to process the raw data into the form of which the classification algorithm takes as input. First, we read ham and spam emails from the drive to label them using a method ‘sort_corpus()’ that takes class label and corpus as variables (see below Figure 10).

```
INPUT email label and the number of mails
SET flag to 1 for spam file or 0 for ham file
DECLARE labels array
DECLARE corpus array
FOR each email of total count
```

```
        READ file by path
        ADD flag to labels array
        ADD email content to corpus array
        END READ
    RETURN corpus and labels array
```

Figure 10. Loading email content function

Then, the data returned was passed into ‘data_divider()’ to randomize the orders and divide into training and testing samples (see Figure 11).

```
    INPUT array of emails and array of corresponding labels
    VAR training set ratio
    SHUFFLE emails and labels while keeping the pair
    RETURN first training set ratio of the total emails for training and its labels, reminders for
    testing and its labels
```

Figure 11. A function separating data samples for training and testing by given ratio

The dataset is ready to feed into learning algorithm now, the function seen in figure 11 returns two list items that are divided into training and testing; each list consists of two nested lists are containing content of the email and its label respectively.

4.1. SVM (Support Vector Machine)

4.1.1. Weighting schemas and other basic settings

In order to compare performance of different classifications, we have first started with Support Vector Machine (SVM). Using sklearn.svm.SVC library in Python, we were able to test various algorithms in relation to the SVM classification. To start with it, we need to create a feature matrix of all data samples for which we have a number of options to choose. The most common approaches are integer count model—similar to Boolean weight model is a weighting model setting 1 to all of non-zero values—and vector weight model. There are many other ways to vectorize the word features but, in our experiment, we employ ‘TfidfVectorizer’ class [2], and ‘CountVectorizer’ class [3] which are relatively weak but commonly utilized, found from scikit-learn library. These methods take number of arguments but, to make it simpler, we used the default settings throughout the entire project session except a few things are responsible for further processing of data which will be presented later.

First, we compared performance of weighting models (see Figure 12), as it is shown, using ‘tfidf vectorizer yielded the most decently high accuracy in the result (each accuracy presented are the average scores of 50 iterations with randomized datasets). In addition, also the ‘HashingVectorizer’ [4] showed second highest accuracy in result. Hash vectorizer was only examined on SVM because for other classification algorithms had more strict input requirement in values, the vectorizer produces negative values that was not allowed in some classifications which will be discussed later again. In our experiment of comparing three different weighting schemas, 'tfidf' took the top of the accuracy in SVM (see Figure 14 and 20).

It is also interesting to observe while executing SVM algorithm in different ways, we found implementations have shown discrepancy in the result accuracies throughout approaches of LinearSVC, SVC, and SVC with Pipeline that are supposed to be the same linear SVM classification algorithm though which resulted in varying accuracies (see Figure 13); however, we do not intent to investigate in this report as it is not within our project scope.

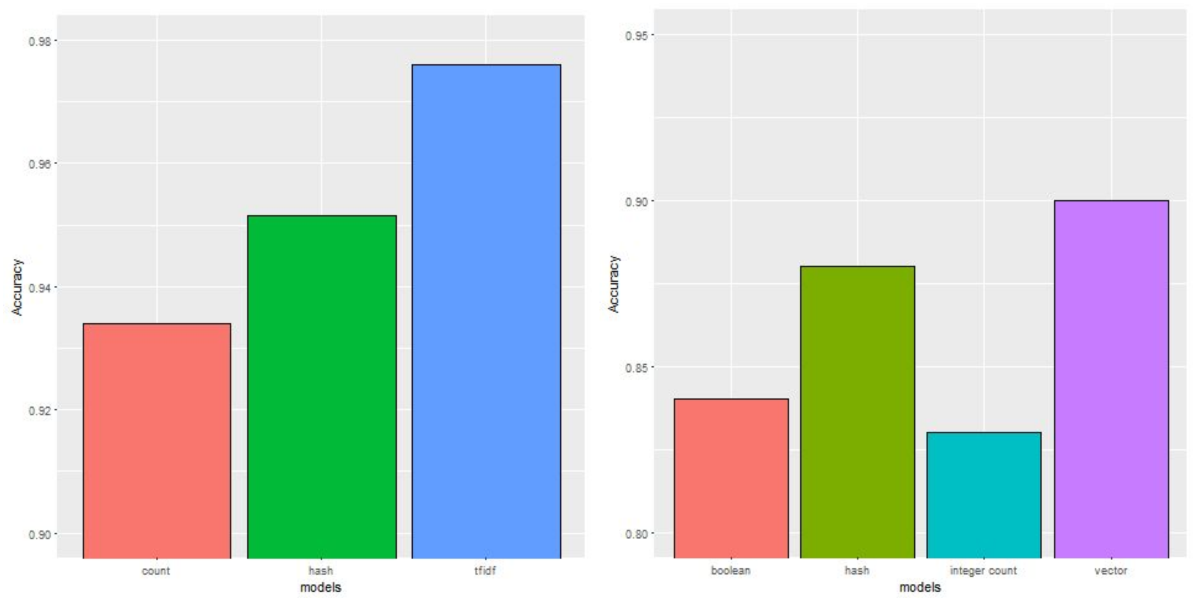


Figure 12. Comparison between different weighting models in SVM (left: 2000 samples, right: 21571 samples)

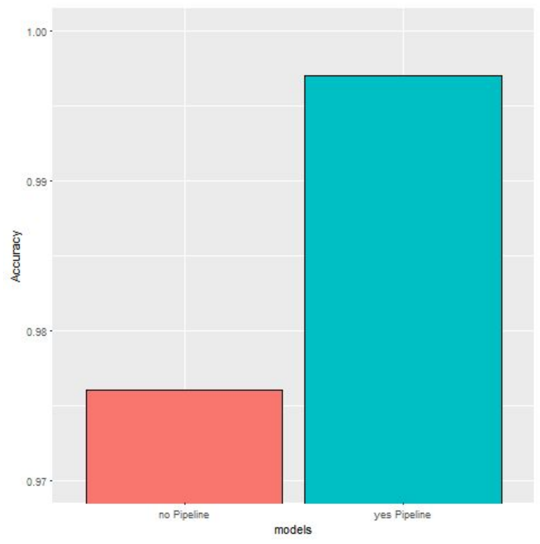


Figure 13. With and without using Pipeline class, accuracy variation on average

4.1.2. Email preprocessing

In general text classification, extracting good features are crucial procedure before putting datasets on training, as part of that, we added preprocessing steps in the form of a function passed to each vectorizer in which it removes stop words and entity words, lowercases words, and stems words. The list of stop words are referred from ‘stop_words.get_stop_words’ and for the stemming and entity removing, we used PorterStemmer’s ‘stem()’ and ‘pos_tag()’ respectively; these are the basic implementation found in nltk library. With pos-tagger, we were given tag for each word to the input sentence, from which we could remove words indicated with tags as ‘NNP’ and ‘NNPS’.

In the following experiment, we preprocessed the dataset and calculated the accuracy of SVM classification (see Figure 14).

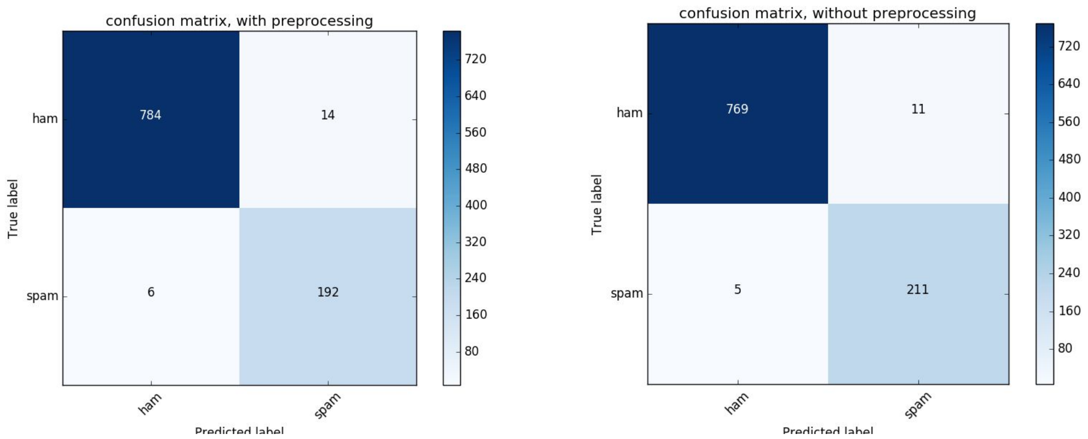


Figure 14. Accuracy with and without preprocessed datasets

Although it took more time operating training algorithm with preprocessing, the result was disappointing. To underpin this, we decided to run 50 iterations of each cases to see exactly how much accuracy difference they would yield.

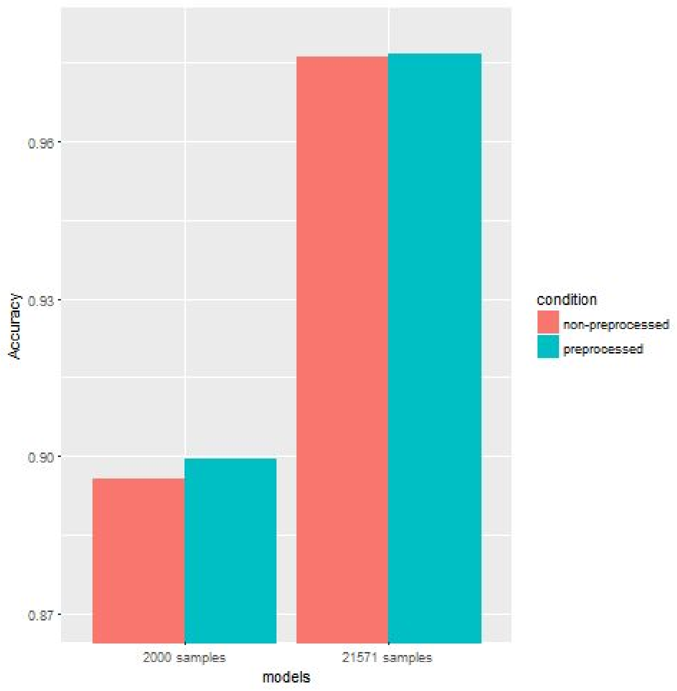


Figure 15. Accuracy comparison for pre-processed data and non-processed data

As shown in Figure 15, it still did not show significant difference performance-wise between two classifications; in which case, taking the time required to train 500 datasets and validate 500 datasets into consideration, without preprocessing is much computationally economic as it takes 7.8 seconds whereas with pre-processing takes 74.5 seconds to complete the training and validating. There was no difference found when using larger size of datasets for training (figure 16).

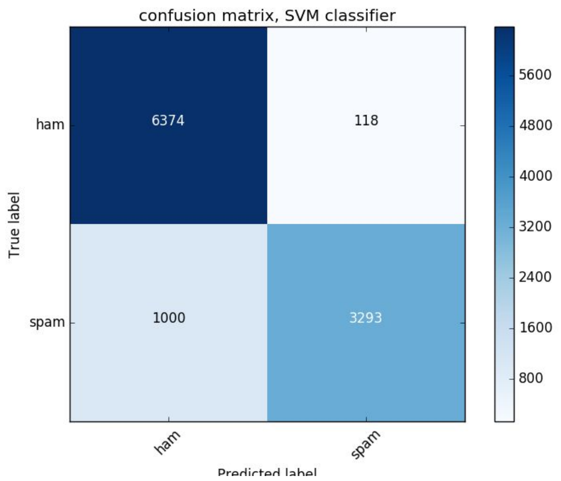


Figure 16. Classification mistakes on 21571 datasets

Within word features given from email input dataset (average 25000 features for 2000 datasets, 50% training), we tested selecting only highest score features from vectorized feature matrix for SVM, the result was disappointing as it appears that training high score values does not affect to the classification performance. The experiment was undertaken on both small size sample and large size sample which resulted in no difference (+/- 0.02 score) in terms of performance scores for both scenarios (figure 17).

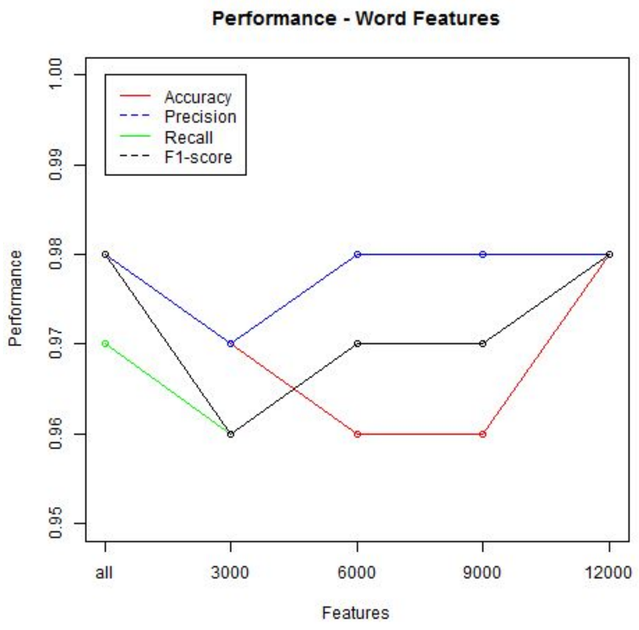


Figure 17. Comparison between number of best word features included and performance results in SVM (2000 samples)

4.1.3. Dimensional reduction algorithms and Kernels

Principal Components Analysis (PCA) is generally used to reduce the dimension of given dataset having multiple dimensions. It helps plot the entire dataset in a way that is human perceivable multi-dimensional dataset at a glance, as well as finding relationship (clusters) between specific components. However, PCA requires dense matrix as an input variable for which our implementation needs converting process from sparse data that is given, and such transformation takes up a huge amount of memory and is slow processing, thus we decided to use SVD equivalent to PCA but taking sparse matrix data as an input as opposed to PCA. With SVD we could save enormous of time preparing to train classification and memory space.

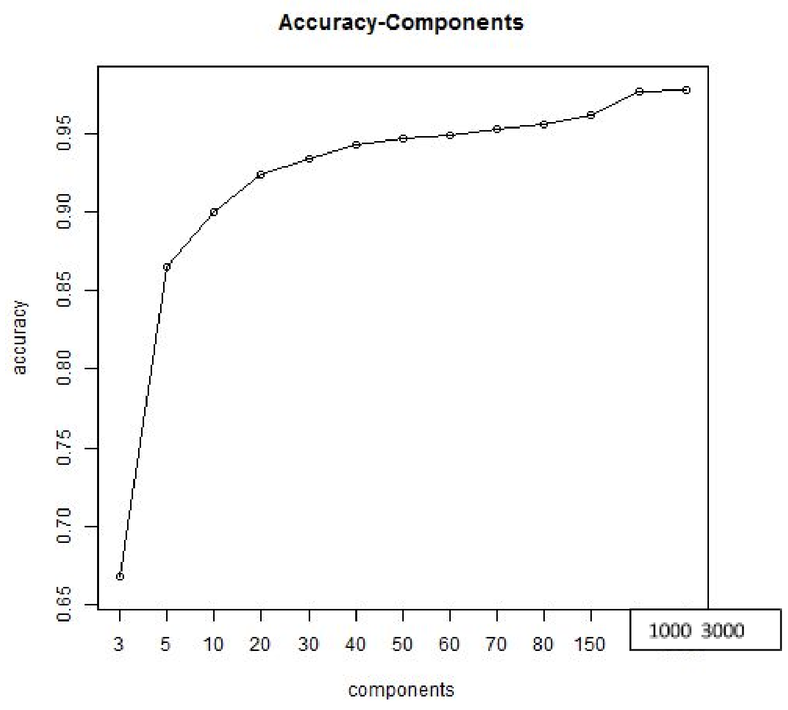


Figure 18. SVD components – SVM accuracy

Above 'Accuracy-Components' plot implies that with 5 dimensions we could explain most of datasets we trained of, the rest of about 2995 dimensions only cover small portion of the training data. Using PCA or SVD, turns the word features of vector matrix into a matrix of given dimension; the parameter deciding number of dimensions to use is given when initiating PCA or SVD class, and the algorithm chooses Principal Components (also referring to eigenvectors) based on the highest variance (measured by eigenvalues corresponded by eigenvectors) in the data plot—meaning that it can explain more datasets than other dimensions.

For instance, giving 5 dimensions as an input for 'TruncatedSVD' in scikit-learn library, we would get result of (num of dataset, num of total word features), i.e. (1000, 24015) fitted into (num of dataset, 5 best largest components/eigenvalues—derived from eigenvectors of the word features matrix), i.e. (1000, 5) with this matrix we can train SVM classification.

Applying PCA or SVD, we could get maximum accuracy of 0.9771 which is slightly higher than without these techniques applied which is 0.9769. In conclusion, handling small set of data, PCA did not contribute terribly much to the classification performance and yet it is computationally more expensive.

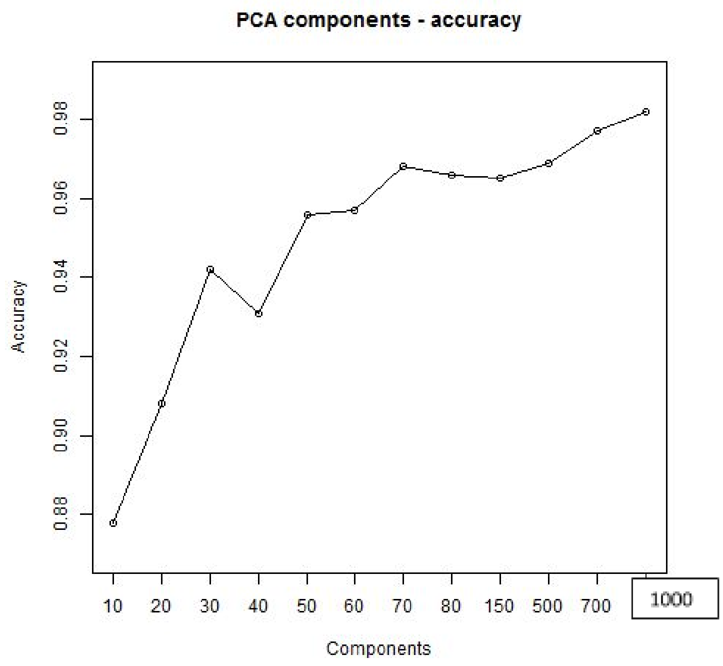


Figure 19. PCA components – SVM accuracy

Having to compare between two SVD (figure 18) and PCA (figure 19) gave the same insight to the result of applying dimensional reduction on the datasets—although above two figures are tested with different conditions (size of samples etc.), the graph tendency stays constant.

In contrast to PCA or SVD, LDA approach showed distinct tendency in accuracy by the number of components (figure 20), instead it drew an arbitrary accuracy graph across the components employed.

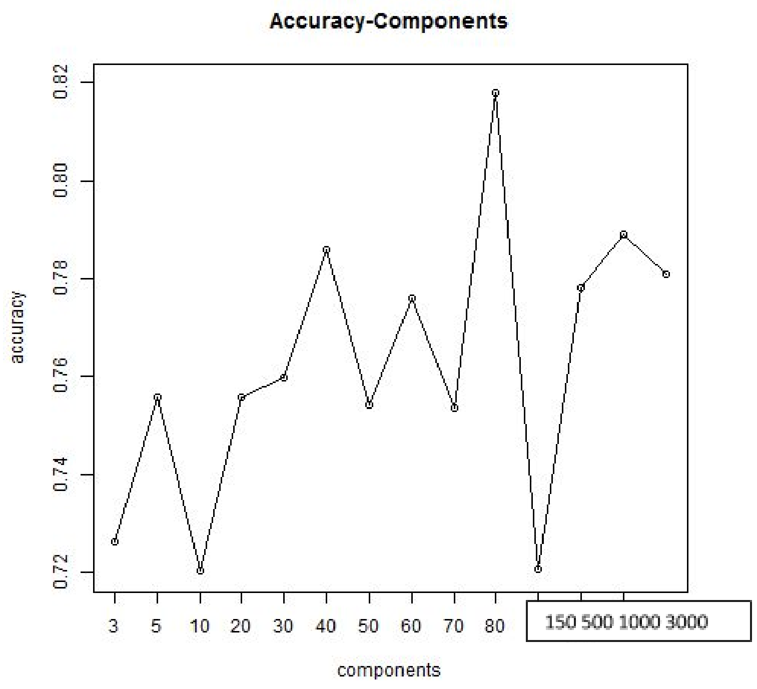


Figure 20. LDA applied SVM and the result accuracies

SVM provides adjustable kernel models with which supports classifying the data more flexibly. With our dataset (2000 samples, 50% for testing), we could run a kernel comparison as shown below.

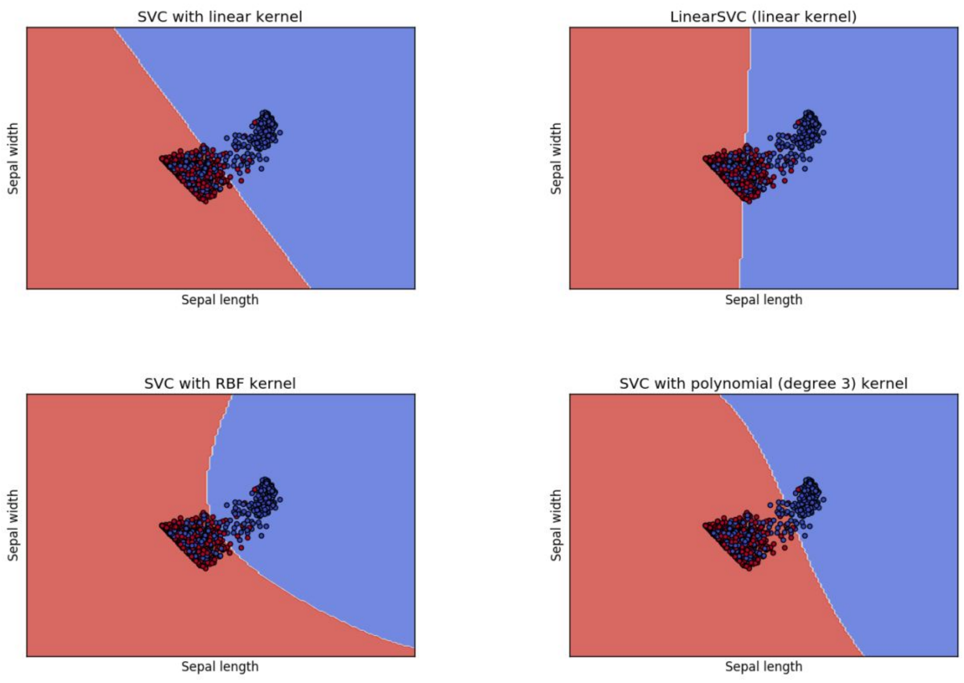


Figure 21. Comparison between different SVM models

In a lower dimensional dataset (SVD, dimension reduced to 2), performances between kernel models seem noticeable (figure 21). However, when using a full dimensional dataset, these models stack up each other about equal except RBF kernel model (figure 22), which we believe RBF model is not suitable for multi-dimensional datasets.

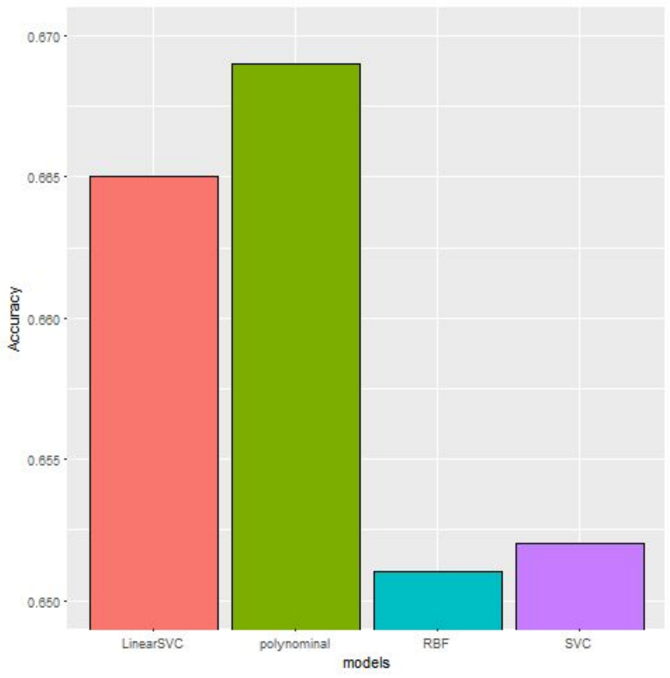


Figure 22. Performance comparison between different SVM models, SVD (c=2) applied

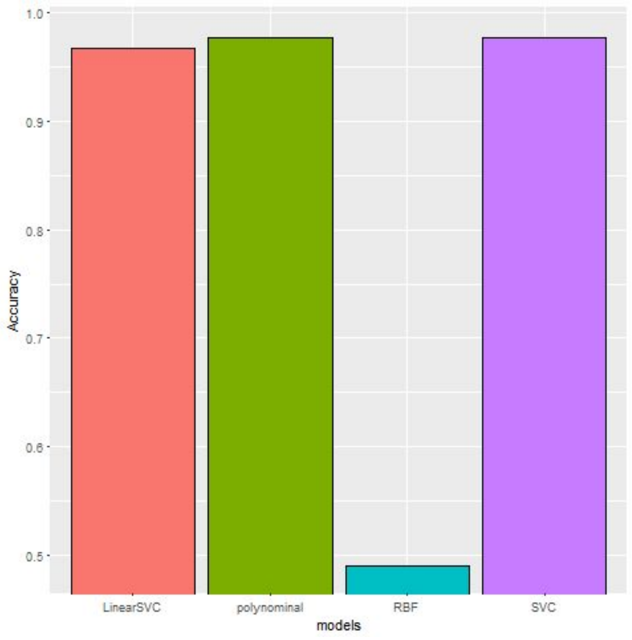


Figure 23. Performance comparison between different SVM models

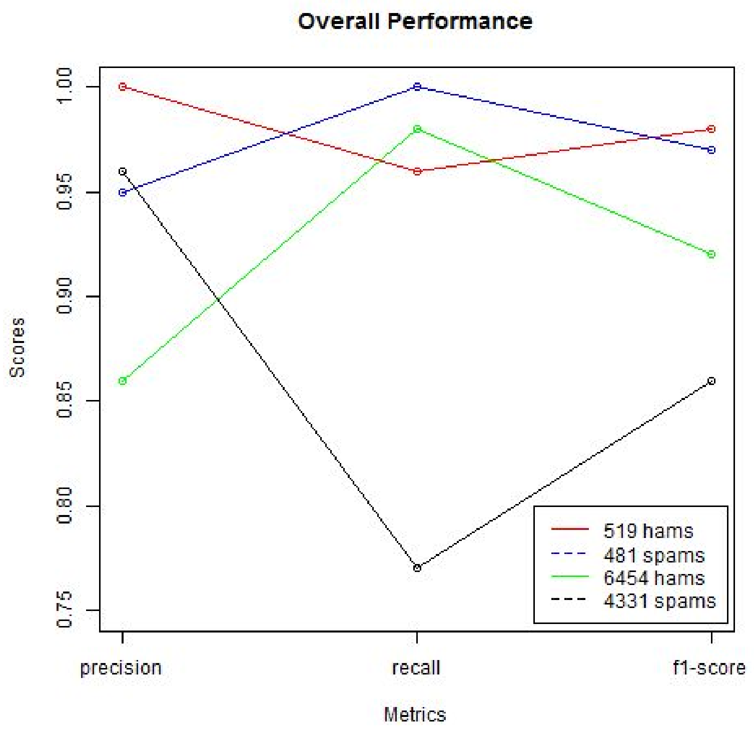


Figure 24. SVM overall performance

We have run number of experiments using SVM classification with small and large dataset (2000, 21571 samples, 50% used for testing). Overall, SVM performs very decent—accuracy of 0.9762 and 0.9769 with and without preprocessing; accuracy of 0.9771 with SVD applied (included large number of components, almost all); accuracy of 0.9762 with all applied. Applying PCA or SVD has resulted in lower performances in our tests, reason is because it is often true how a classifier distinguish one class from another is through the components with low variance—the fewer frequency the word has, the higher the influence it has—therefore, PCA or SVD that select principal components are high variance can miss out important features in the components with lower variance.

Feature selection does not seem to work well with SVM, testing on 2000 and 21571 datasets indicated there is no consistent influence using best feature selection (or highest feature selection).

While the difference between performance metrics (precision, recall, f1-score, and accuracy scores) on small datasets was marginal (generally maximum 0.05 or less variance in each score across all metrics) with larger datasets, the gap score between metrics became radical (figure 24). Given results in figure 24, we can assume that in general the classifier struggles more finding correct spam emails, similarly, it does a good job distinguishing positive spams from non-spams but perform poor finding actual spams as the datasets growing.

4.2. Naïve Bayes

4.2.1. Classification models

The second classification we chose was NaiveBayes classification that can be also found in scikit-learn library in Python. With this classification, we have followed the same reasoning approaches as SVM. First, started off with weight models comparison (see figure 25).

Normally NaiveBayes performs better with integer counts such as boolean weighting schema though, tfidf model was not too bad either in performance. Similarly, it is interesting to observe the results between MultinomialNB and BernoulliNB, both models are suitable for classification with discrete features yet, MultinomialNB is designed for integer counts—however, this is not always true, i.e. figure 28 and 30—whereas BernoulliNB is designed for Boolean counts. Despite, BernoulliNB did improve accuracy using Boolean count type over ordinary integer count type (figure 26), still not working well in comparison to the other models.

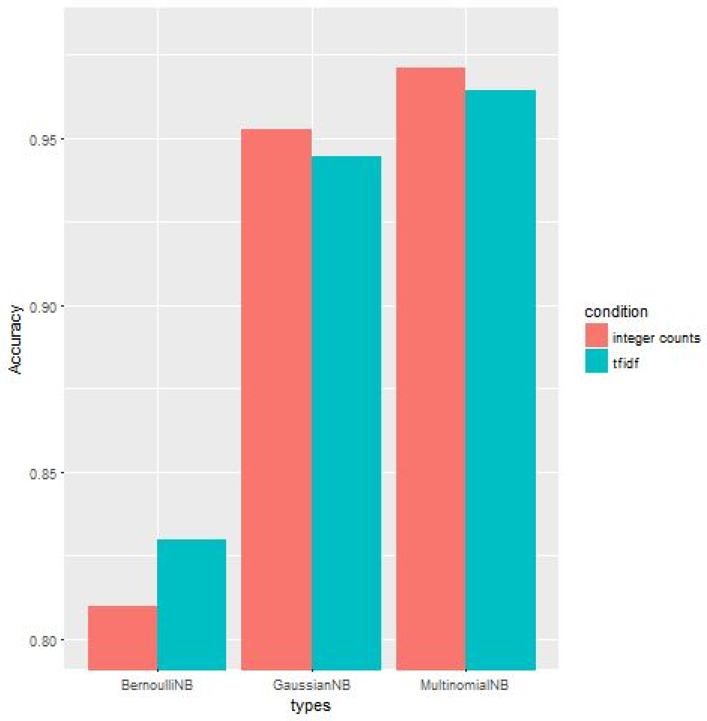


Figure 25. Comparison between weighting schemas and result accuracy with NaiveBayes

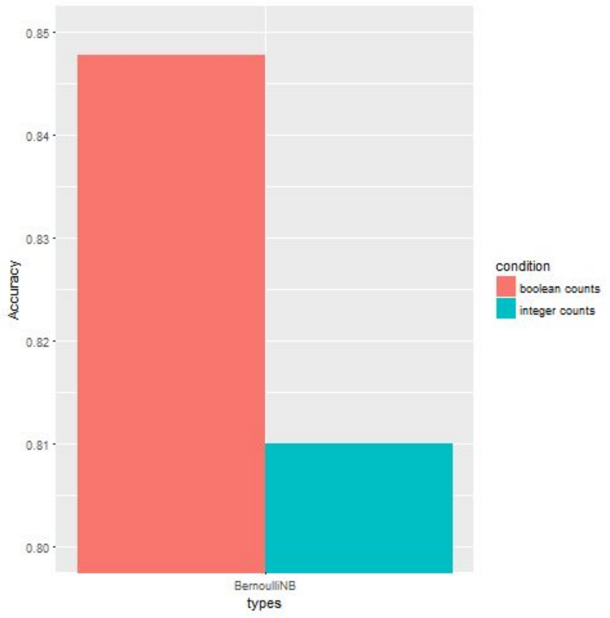


Figure 26. Comparison between Boolean count and integer count types for BernoulliNB model

Using MultinomialNB model designed for text classification showed the best accuracy amongst all models, we have compared two conditions again with and without preprocessing datasets (figure 27).

Weirdly, Naïve Bayes classification showed lower accuracy from training and testing after data preprocessing as opposed to SVM. It may be an indication that the current implementation for processing data in prior to training is not appropriate or too weak.

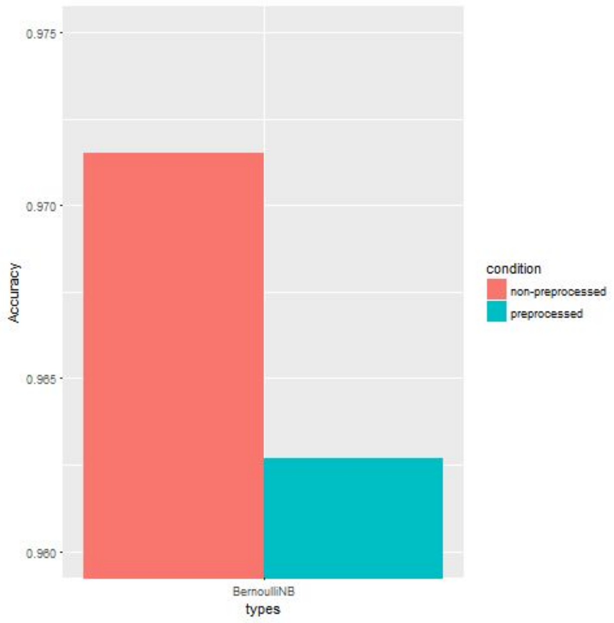


Figure 27. Naïve Bayes accuracy comparison between with pre-processed datasets and without

Comparison-wise with SVM, Naïve Bayes does not generally support dimensional reduction methods like PCA or SVD because they tend to assign values in the (-1, 1) range in which case, it is not allowed in Naïve Bayes classification algorithm thus throwing a ValueError.

4.2.2. Weighting schemas and data preprocessing

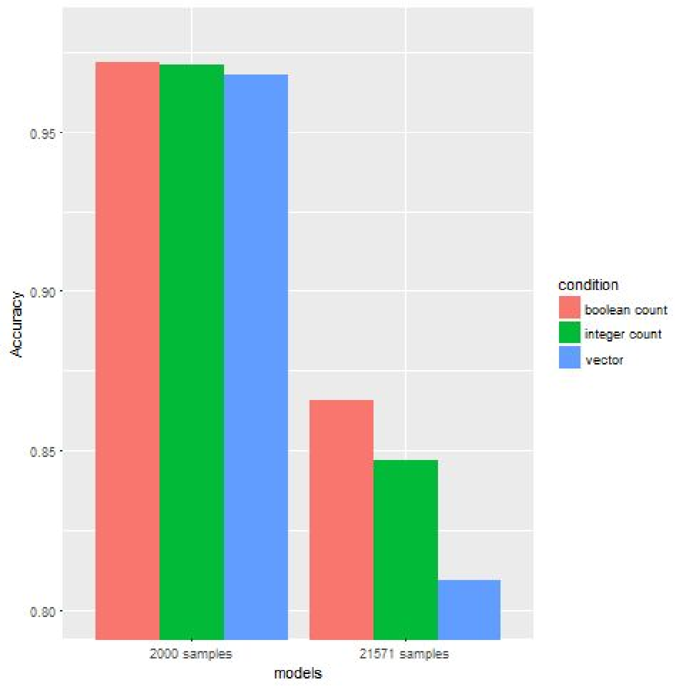


Figure 28. Naïve Bayes accuracy-weight models comparison based on the sample sizes, multi-nominal model

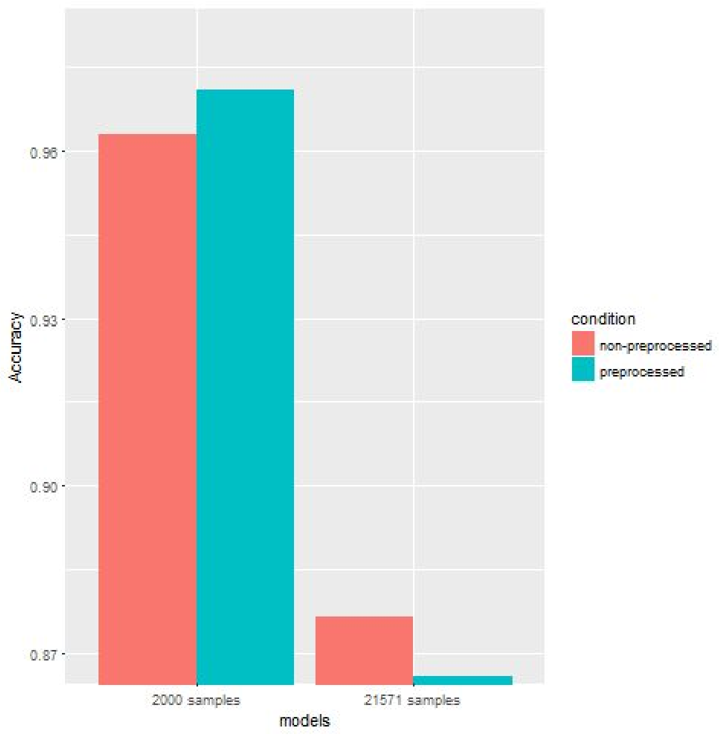


Figure 29. Naïve Bayes accuracy comparison two size samples with regard to preprocessing data

Adding more data to the training and testing sample gave a bit of variance in performance between weighting models employed on Naïve Bayes. Expanding datasets ten times reduced average accuracy significantly (figure 28, 29, and 30). Also, we confirm by the result from figure 29 that applying preprocess to the dataset does not influence classifier performance effectively.

In general, the classifier appears to struggle more categorizing spam messages from ham messages (figure 30, 31 and table 1).

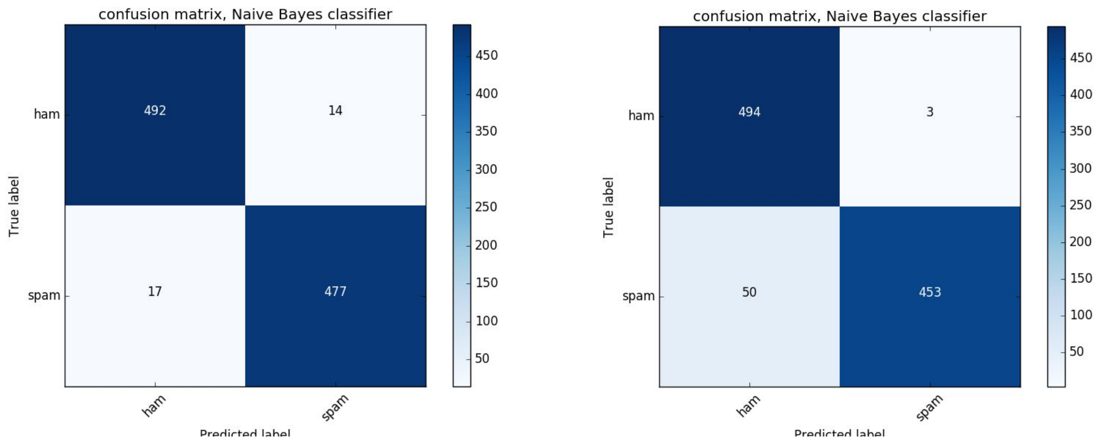


Figure 30. Naïve Bayes error rate using Multi-nominal model on 2000 datasets (left: Integer count weight model, right: Boolean weight model)

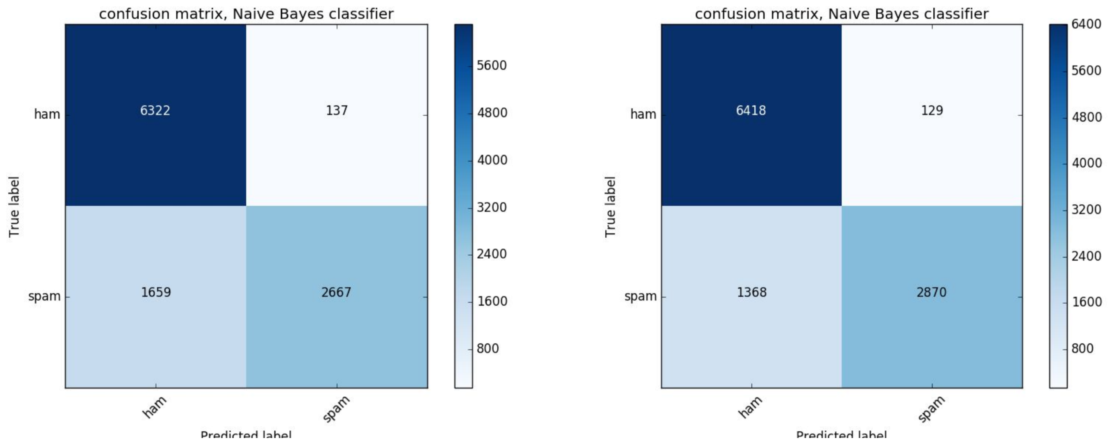


Figure 31. Naïve Bayes error rate using Multi-nominal model on 21571 datasets (left: Integer count weight model, right: Boolean weight model)

Multi-nominal Naïve Bayes model				
	Integer count weight model		Boolean weight model	
	2000 samples	21571 samples	2000 samples	21571 samples
Ham error	2.76%	2.12%	0.6%	1.97%
Spam error	3.44%	38.34%	9.94%	32.27%

Table 1. Naïve Bayes classifier error rates on the table

An extension to the preprocessing, we applied best feature selection on Naive Bayes which showed again no difference in performance. The resulting scores drew about same tendency graph as figure 17 that is +/- 0.02 arbitrary variance on the scores (300000 average number of features, tested on 3000, 6000, 12000, 24000, 48000, 100000, and all).

Applying cross-validation gives accuracy score along with its error rate on average, below figure indicates that by increasing the size of datasets to train on the classifier, we can actually reduce the error rate of accuracy score (figure 32).

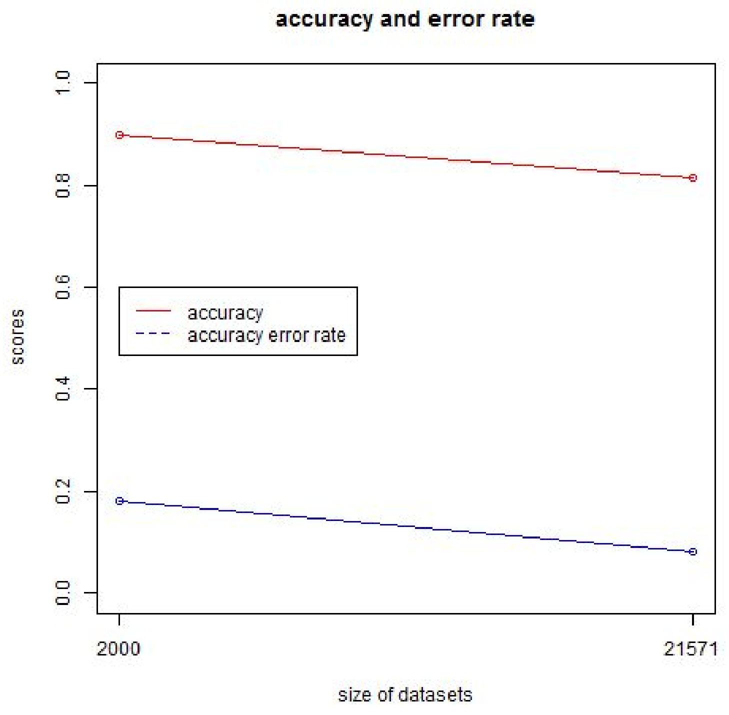


Figure 32. Accuracy and accuracy error rates dropping down by increasing the size of datasets for training

In the overall performance of Naive Bayes, the classifier works stable on smaller datasets whereas on larger datasets it confuses a lot labeling spam emails among spam-like emails (figure 33).

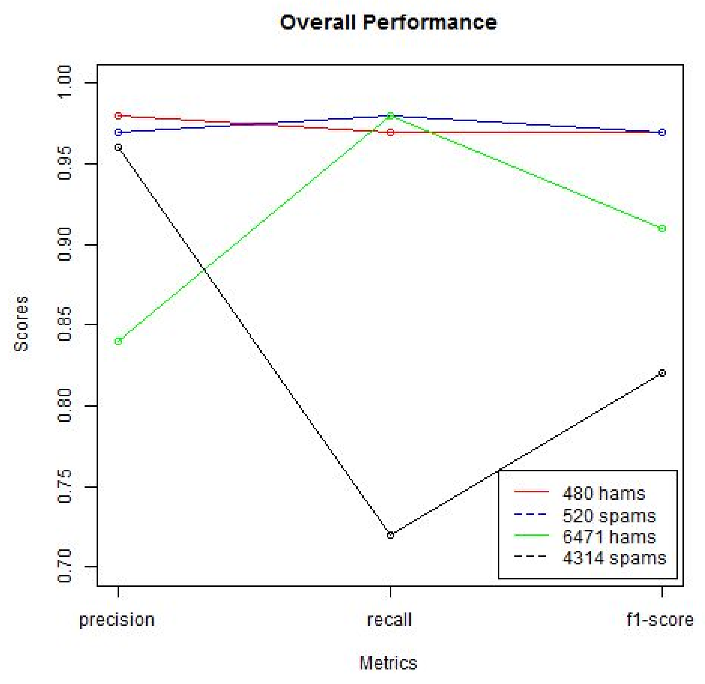


Figure 33. Naive Bayes overall performance

4.3. KNN (K Nearest Neighbors)

The third classifier we tested is K Nearest Neighbors (KNN). KNN is relatively simple machine learning classification that is also known as 'non-generalizing machine learning methods', since they remember all of its training data. Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems [1].

4.3.1. Email preprocessing and choosing K value

Again, preprocessing datasets did not help much with KNN either (figure 32). However, selecting proper K value is known to be much influential in KNN, despite assigning low K value may give higher chance for the classifier overfitted as seen in figure 35 and 36, on the other hand, higher K value may drive possible underfit of the classifier. In figure 38, of the 2000 data sample that used 50% for training drew descending performance results as the K value gets higher than 10, up to 100. In principle, selection of good K is substantially dependent on the data, in this case, k=10 seems working best.

Although we could apply dimension reduction algorithms like SVD or PCA on KNN, it would drag down the accuracy of the classifier as well by including only a partial number of components; by looking at the data plotted on 2-D space (figure 35 or 36), we can easily observe that applying PCA or the likes to the originally high dimensional dataset plotted into lower dimensional space may end up without a clear cut-off line and often overfitted.

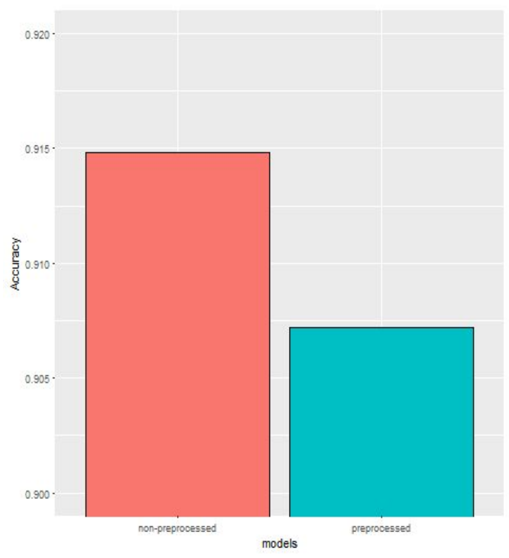


Figure 34. Preprocessing and non-preprocessing comparison in KNN

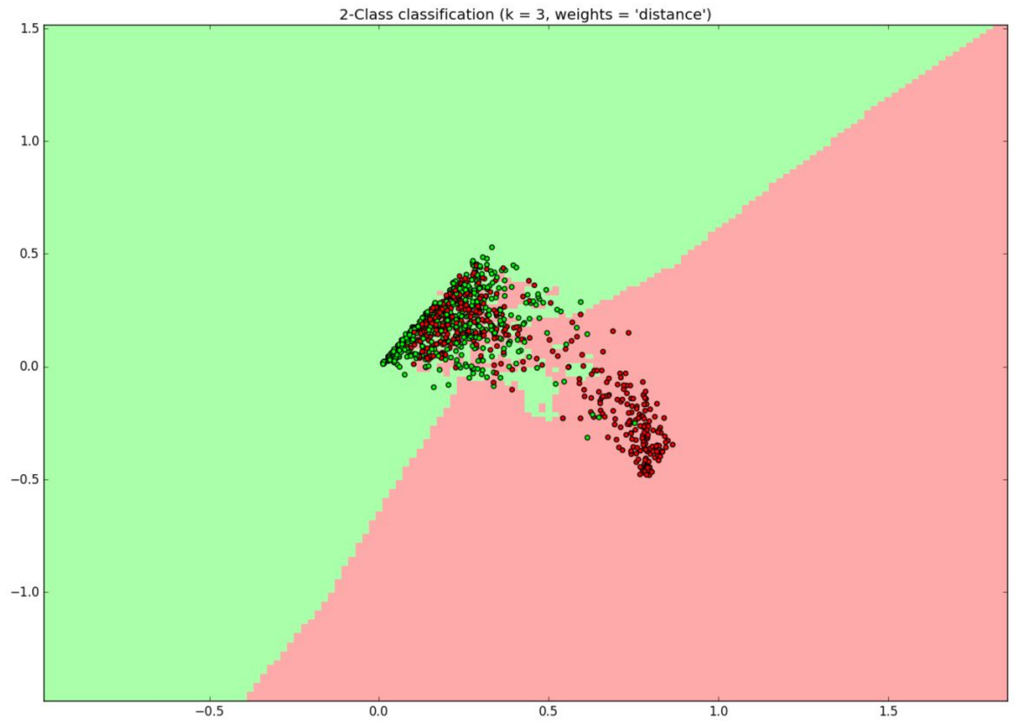


Figure 35. KNN plot, k=3 (2000 samples)

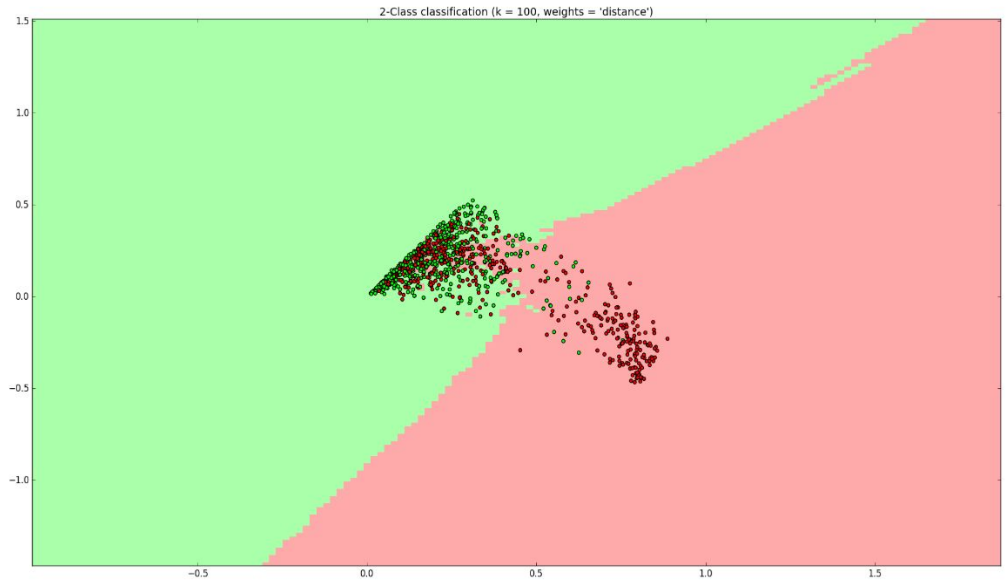


Figure 36. KNN plot, k=100 (2000 samples)

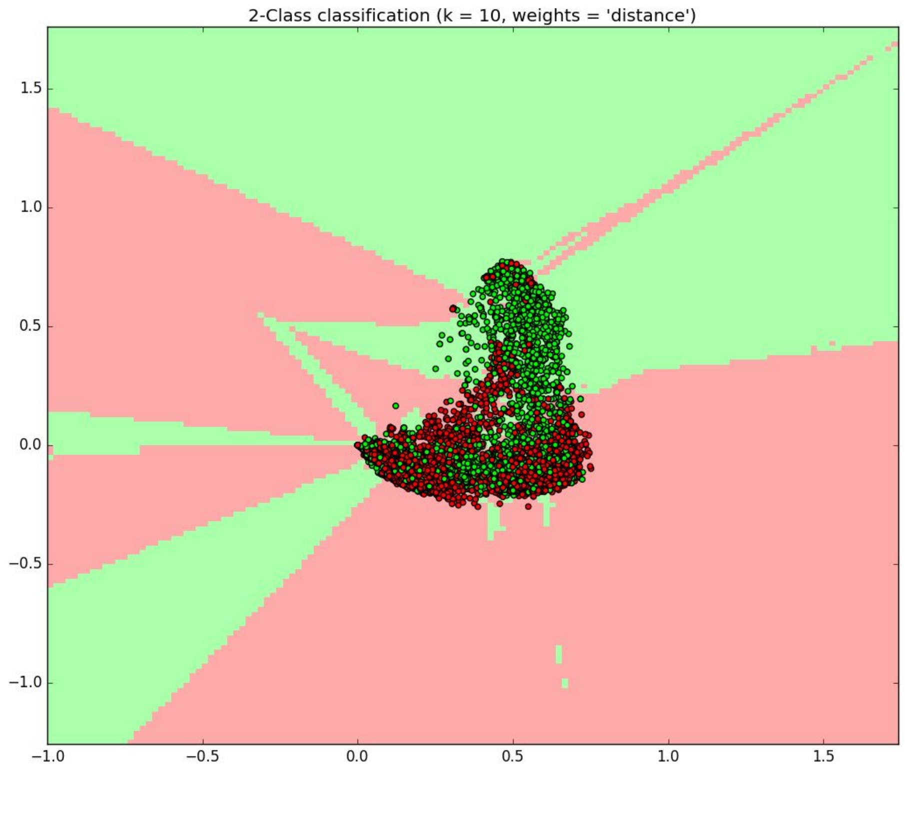


Figure 37. KNN plot, k=10 (21571 samples)

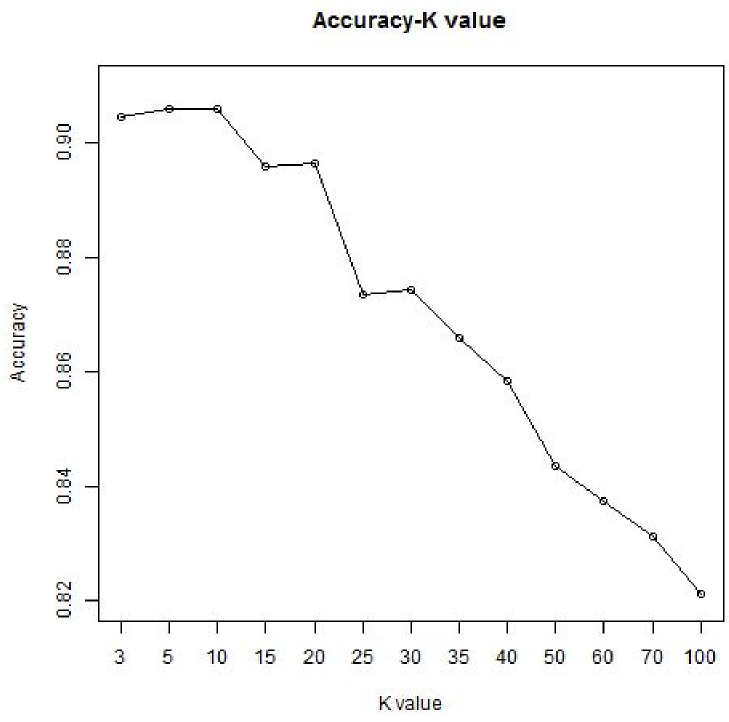


Figure 38. KNN classifier accuracies depending on K values (2000 samples)

We wanted to apply previous reasonings to the larger datasets is up to 21571 (12952 hams, 8619 spams) in comparison to the initial 2000 samples (1000 hams, 1000 spams).

So, the result was against what W.A. Awad and S.M. ELseuofi [14] claimed on the subject of dependency between k value and the classification performance that it appears to be nearly independent though, in our experiment, the performance variation throughout varying k value was definitely noticeable (figure 38 and 39).

Again, feature extraction on KNN did not show difference from other classifications' result, which we applied selecting highest feature values from the feature matrix before training on smaller and larger datasets resulted in similar to what was seen from figure 17.

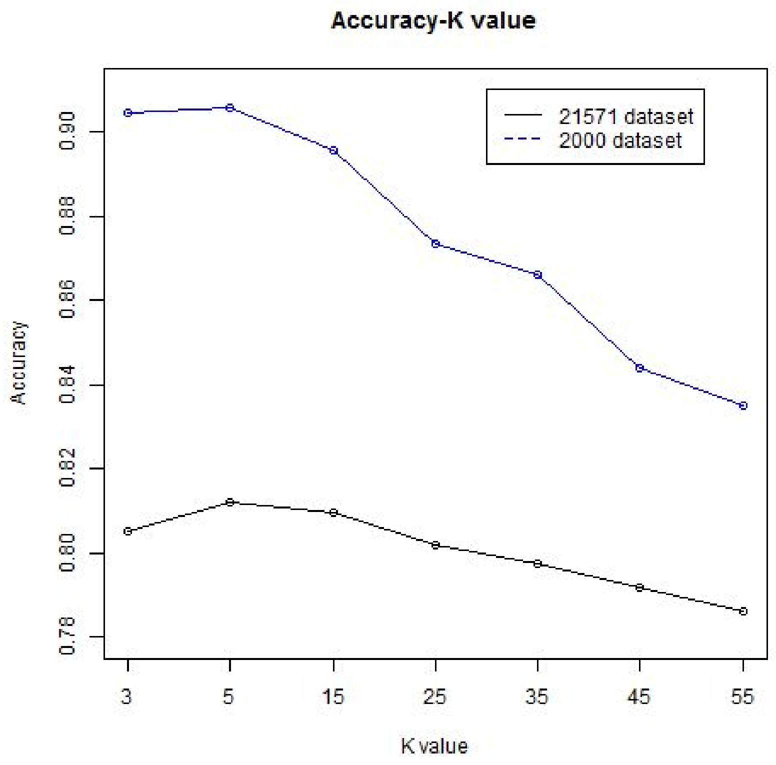


Figure 39. KNN classifier accuracies comparison between two datasets

4.3.2. Weighting schemas

It appears to be k=5 fits best on this larger dataset, also we can observe that the average accuracy has dropped down with larger datasets. Figure 40 down below compares weighting schemas used on different size of samples indicating there is no one master weight model that works for KNN classifier but it depends on the data conditions; it is interesting to observe that how much accuracy went poor with vector model by switching data condition.

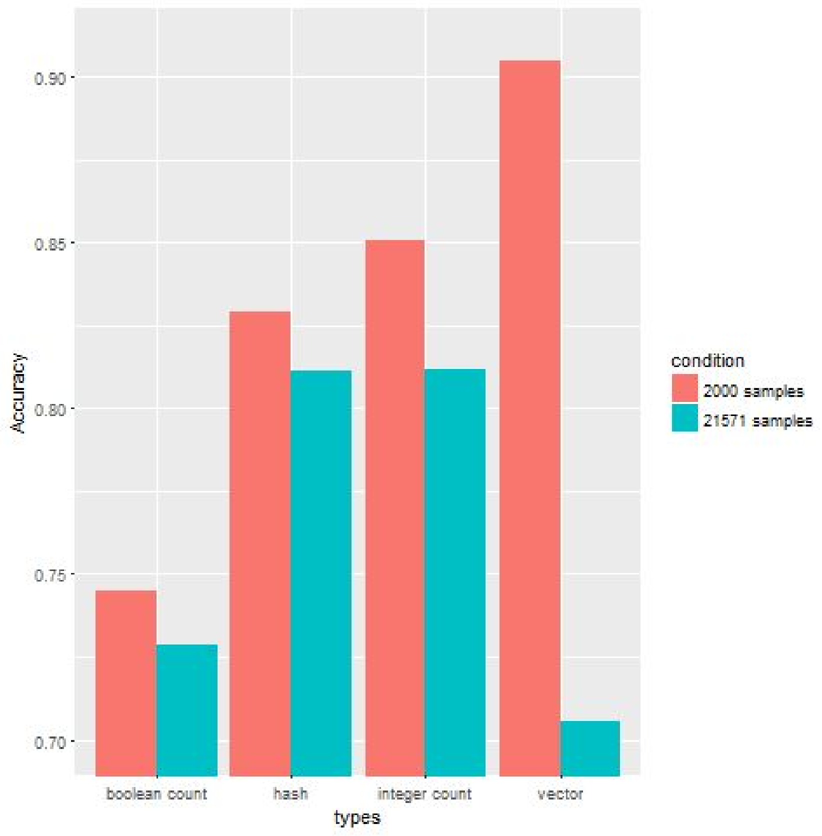


Figure 40. KNN weighting models comparison with regard to the size of datasets

It is clearly seen that the overall performance of KNN is relatively poorer than the other two classifiers however, it has a consistent capacity labeling across ham and spam emails (figure 41).

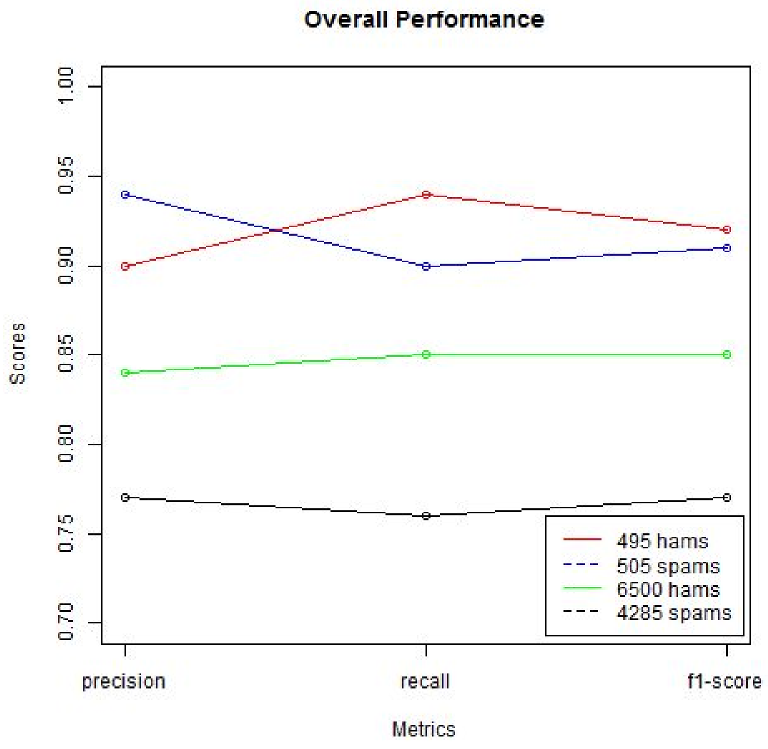


Figure 41. KNN overall performance

5. Unsupervised Classification and Deep Learning (Neural Nets)

5.1. K-means and MeanShift

We have tried K-means classification/clustering algorithm on our initial dataset to see how it works with text classification. As the name implies, this algorithm also takes K value as a parameter that is a variable deciding how many classes/clusters to divide the dataset into, generally the classification is sensitive to outlier and noise, which is not very good for such purpose of text classification that normally includes a lot of variance on the data.

The result was very poor, it gave us around below 0.5 accuracy on average so we dabbled a few more trials and decided not to move further. The same goes to the MeanShift, these algorithms were rather fit for clustering after all.

5.2. Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM)

This time we implement a neural net classification called RNN; it is a general term in the field of deep learning and there usually a lot of variations has been introduced with respect to neural nets classification. LSTM is one of the most common, yet powerful algorithm in RNN which allows to relate sequence of data to the training. For the implementation, we decided to use Keras Python library for deep learning algorithms.

It is not very much different from ordinary machine learning procedure LSTM, first input the data sorted for training and testing by the method presented in figure 12 based on the given ratio, and then tokenize emails before queueing them in sequence using 'keras.preprocessing.text.Tokenizer'. In this phase, we can truncate the total word features to be included by passing 'nb_words' argument.

6. Conclusion

In this report, we have identified a few of renowned machine learning algorithms were successfully proven to work for filtering spam emails to review and provide in-depth examination on their performance, as part of that, we proposed a research question (consists of two sub-questions) to answer throughout the report by running several experiments with various configurations providing an insight to the classification performance from different perspectives.

As for the accuracy of the classification, SVM seems to work better over the other two (table 2 and figure 24, 33, 41, 42), which is slightly different outcome from the work carried out by W.A. Awad and S.M. ELseuofi [14]; while Naive Bayes classifier gives more stability in result score and its accuracy score is nearly close to SVM, KNN performance falls a bit behind (all scores presented here were obtained by independent best configuration for each classifier). Similarly, although we did not include in this report the full comparison with Neural Net classification, it appears that Neural Nets tend to give lower performance comparing to the traditional machine learning algorithms. Our reasoning to this is perhaps because we used 'clean' datasets that were sorted for the purpose of classification study however, presumably if the datasets were 'noisy' and the training data volume was much larger like millions, Neural Nets approach may stand out in resulting performance over ordinary machine learning classification algorithms. Also, in the future study, we would like to investigate deeper into how preprocessing datasets may influence to the result performance of classification as it does not seem to have difference according to our experiments, which might attribute to our processing implementation employed only basic methods.

	SVM		Naïve Bayes		KNN	
	Ham	Spam	Ham	Spam	Ham	Spam
Precision	0.87	0.97	0.84	0.96	0.84	0.77
Recall	0.98	0.77	0.98	0.72	0.85	0.75
F1-score	0.92	0.86	0.91	0.82	0.85	0.76
Accuracy score	0.90		0.88		0.81	
Error rate (+/-)	0.05		0.02		0.08	

Table 2. Classifier overall comparison with their best configuration (SVM: vector weight model, linear kernel; KNN: k=5, 21571 samples, integer count weight model; Naïve Bayes: boolean weight model, MultinomialNB)

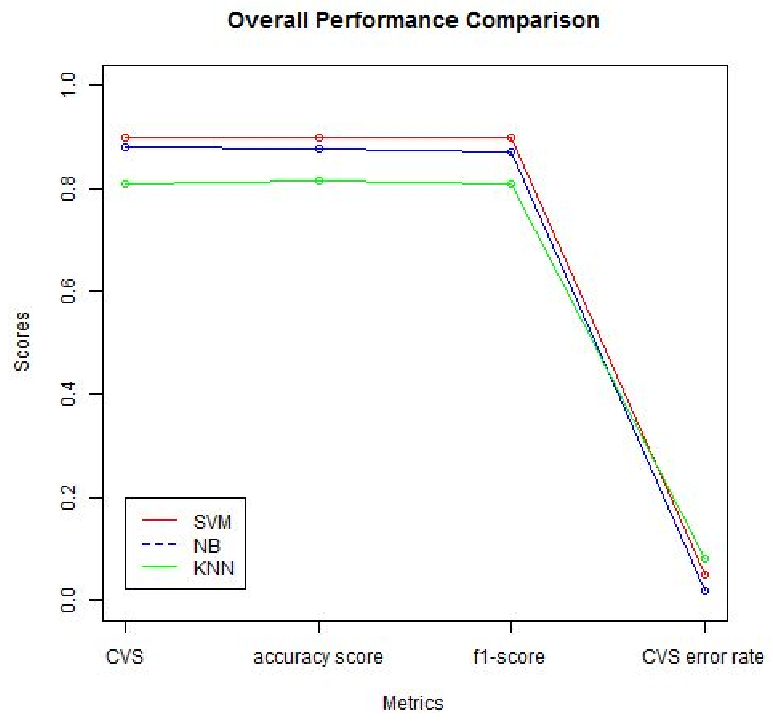


Figure 42. Overall performance comparison between three classifications (CVS : Cross-Validation Score)

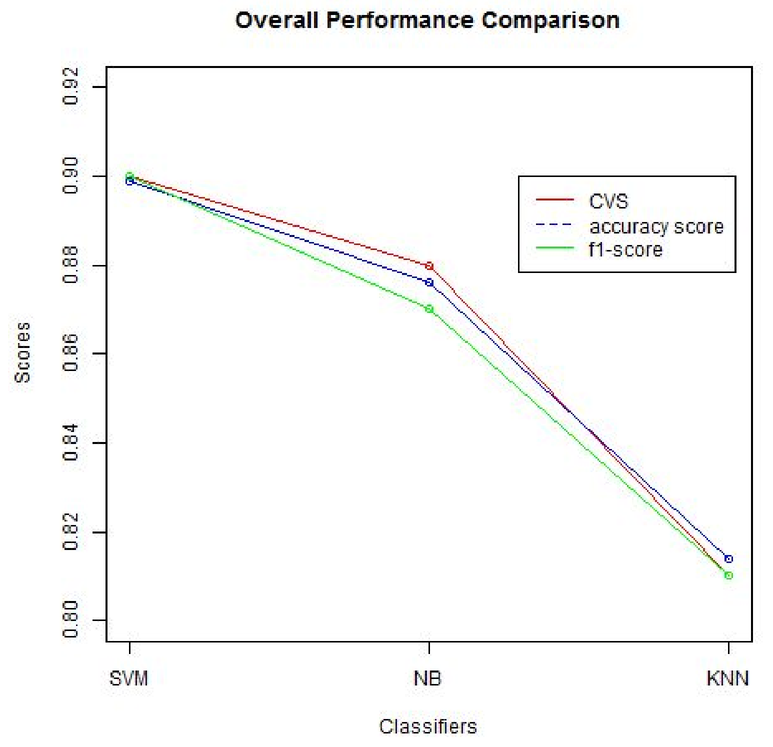


Figure 43. Overall performance comparison between three classifications version 2

7. Reference

[1] http://scikit-learn.org/stable/modules/neighbors.html

[2] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[3] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[4] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html#sklearn.feature_extraction.text.HashingVectorizer

[5] https://en.wikipedia.org/wiki/Information_Age

[6] https://en.wikipedia.org/wiki/International_Telecommunication_Union

[7] https://en.wikipedia.org/wiki/Email

[8] http://www.themonitordaily.com/email-spam-rates-reaching-all-time-low-might-be-the-calm-before-the-malware-storm/24025/ “Spam hits all time low”

[9] https://en.wikipedia.org/wiki/Email_spam

[10] https://en.wikipedia.org/wiki/Email_filtering

[11] http://scikit-learn.org/stable/

[12] Guzella, T. S., & Caminhas, W. M. (2009). A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, *36*(7), 10206-10222.

[13] Tang, Y., Krasser, S., He, Y., Yang, W., & Alperovitch, D. (2008, November). Support vector machines and random forests modeling for spam senders behavior analysis. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference* (pp. 1-5). IEEE.

[14] Awad, W. A., & ELseuofi, S. M. (2011). Machine learning methods for E-mail classification. *International Journal of Computer Applications (0975–8887)*, *16*(1).

[15] Guzella, T. S., & Caminhas, W. M. (2009). A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, *36*(7), 10206-10222.

[16] Tretyakov, Konstantin. "Machine learning techniques in spam filtering." *Data Mining Problem-oriented Seminar, MTAT*. Vol. 3. No. 177. 2004.