CS5691 Assignment 3: SPAM or HAM?

Debargha Bhattacharjee[1] and Akshay Gupta[2]

[1] Indian Institute of Technology Madras, Chennai, Tamil Nadu, 600036, India
cs19s028@smail.iitm.ac.in
[2] Indian Institute of Technology Madras, Chennai, Tamil Nadu, 600036, India
cs19s011@smail.iitm.ac.in

Abstract. Spam emails (or simply, spams) are unwanted emails which the users are bombarded with. These emails can be of different types. For example, unwanted advertisements sent by marketing firms, suspicious emails sent by fraudsters to dupe unassuming users, emails containing links to malicious web pages that steal users' private information, etc. In this assignment, we try to build a spam filter or spam classifier from scratch using two popular machine learning algorithms- SVM and Naive-Bayes. Besides providing details related to the implementation of the mentioned algorithms, such as hyperparameter tuning, we also provide information about the dataset chosen, the features extracted, etc.

1 Introduction

There are many situations where users are bombarded with unwanted emails. For example, many marketing firms spam people with unwanted advertisements, fraudsters try to dupe people by sending suspicious emails, and attackers also try to steal users' private information by directing them to malicious web pages. These emails (often loaded with misspellings, grammatically incorrect sentences, etc.) are broadly referred to as spam emails or spam. Nowadays, many email service providers automatically filter spam and non-spam emails (also referred to as 'hams'). Additionally, they also allow the users to tag emails as spam to route similar emails to the spam folder in the future.

2 Dataset

We have used the Enron Email Dataset to generate the training set and the cross-validation set. As mentioned in [1], this dataset was collected and prepared by the CALO project. Initially, the dataset contains about 0.5M messages, collected from nearly 150 Enron employees. However, in this assignment, we have randomly selected a subset containing a total of 5172 emails. We have divided this subset [3] in a ratio of 80: 20 such that the training set contains 4137 emails and the cross-validation set contains 1035 emails. While creating the training and cross-validation set, we have also ensured that these sets have nearly the

[3] The training set and the cross-validation set can be found here.

same distribution of hams and spams. Specifically, there are 2937 hams and 1200 spams in the training set, leading to a distribution of 70.99 %: 29.01 %. Similarly, the cross-validation set has 735 hams and 300 spams, leading to a distribution of 71.01 %: 28.99 %. This distribution of spam and ham in the training set is shown in Figure 1 and Figure 2 respectively.

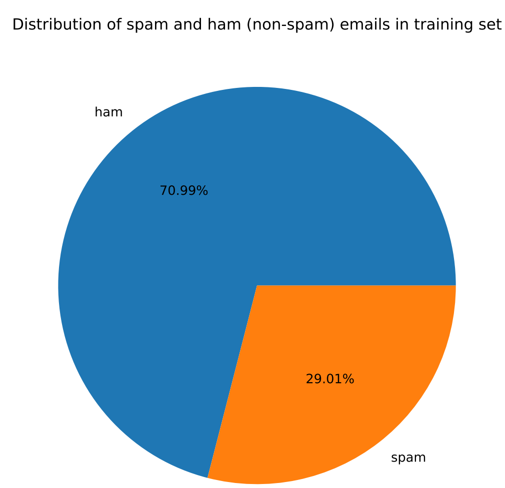

Fig. 1. Distribution of spam and ham in the training set.

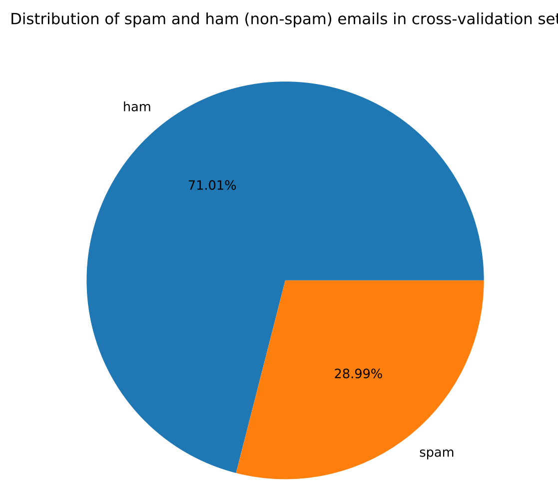

Fig. 2. Distribution of spam and ham in the cross-validation set.

3 Feature Extraction

The spam and ham emails are present in the form of texts of varying lengths. The spam texts often contain misspelled words, target-specific words, etc., which can help distinguish them from non-spam email (ham) texts. Therefore, words become useful features. However, we need to convert these words (present in text format) to integers before using them as features.

3.1 Text Preprocessing and Vocabulary

The emails contain various unwanted words and characters which do not provide any new information which can be used to distinguish the spams from the hams. Some of these redundant words and characters are punctuation symbols, numerical characters, stopwords (functional words like a, an, the, of, and). The process of removing such redundant words and characters from the text is called **text preprocessing**. We have also preprocessed the text to generate the **collection frequency** related to the different unique words present in the text. A word's collection frequency refers to the total number of occurrences of that word in the entire training corpus (i.e., spam and non-spam emails in the training set). These words will be treated as our classifier's vocabulary, i.e., our classifier will only consider these words for filtering the hams and spams. The various stages involved in the text-preprocessing stage are as follows-

- **Removal of punctuation symbols-** Involves removing the punctuation symbols or non-alphanumeric characters from the emails.
- **Removal of numerical characters-** Involves removing numerical characters from the emails.
- **Removal of stopwords-** Involves removing the stopwords (such as functional words like a, an, the, etc) from the emails.
- **Lower-casing the words-** Involves changing the upper-case alphabets to their lower-case counterparts.
- **Removing words having length less than 2-** Involves removing several single-length words generated from the previous steps in the text-preprocessing stage.
- **Word lemmatization-** Involves converting the resulting tokens (or words present in the text) to their root lemmas.

In addition to preprocessing the emails in the training set, we also filter out words whose total frequency of occurrence in the entire training corpus is less than some **threshold** number before creating the vocabulary. The intuition behind using a threshold for filtering out words is that the emails in the training set might contain various sporadic words that might not appear in any other emails, be it spam or ham. Therefore, such sporadic words need not impart any particular information that may help us distinguish the spams from the hams and be removed. Additionally, this also helps in reducing the feature space significantly. However, this threshold is a hyper-parameter, which we tune can on the cross-validation set.

The vocabulary contains unique words extracted from the emails in the training set. Each such word in the vocabulary is assigned a unique integer ID.

3.2 Feature Matrix

After creating the vocabulary, we create the feature matrix for the emails in the training and cross-validation set. We have used **tf-idf scores** for creating the feature matrix. Here the frequency counts are weighted according to the rarity of each token in the corpus. The tf-idf score corresponding to the word w in an email e is calculated as follows-

$$s_{w,e} = \log_{10}(count(w,e) + 1) \times \log_{10}(\frac{N}{N_w}) \tag{1}$$

Here, $s_{w,e}$ is the tf-idf weighted value for word w in document e, $count(w,e)$ is the raw count of the number of times word w appears in email e, N is the total number of emails in the collection, and N_w is the number of emails in which word w occurs.

The feature matrix for the training set is a $\mathcal{R}^{n_t \times d}$ matrix , where, n_t is the number of emails in the training set, and d is the number of words in the vocabulary. Similarly, the feature matrix for the cross-validation set is a $\mathcal{R}^{n_v \times d}$ matrix , where, n_v is the number of emails in the cross-validation set.

4 Classifiers

We have built two different classifiers based on two popular machine learning algorithms for classifying the emails. The two different algorithms are-

- Support Vector Machines (SVM)
- Naive Bayes

We briefly discuss the two algorithms in this section.

4.1 Support Vector Machine (SVM)

SVM classifiers can map the emails to points in space and then maximize the margin around the hyperplane separating the classes (spam and ham). All the emails in the training set are represented as data points x_i such that-

$$(w^T x_i) \geq 1 \text{ if } y_i = +1 \text{ (i.e., spam)} \tag{2}$$

$$(w^T x_i) \leq -1 \text{ if } y_i = -1 \text{ (i.e., ham)} \tag{3}$$

Here, y_i is the true label of the i^{th} email (x_i) in the training set, and $w \in \mathcal{R}^d$ represents the weight vector.

It is, however, often seen that the data is not fully linearly separable. In such a scenario, we can extend the SVM algorithm by relaxing the naive SVM constraints (given in eq (2) and eq (3)) such that some of the points are allowed to be misclassified. The also helps in accounting for the outliers that might have crept in the training set. This is done by introducing non-negative slack variables $\xi_i \ \forall i \in \{1, 2, \ldots, n_t\}$ such that-

$$(w^T x_i) y_i + \xi_i \geq 1 \text{ where } \xi_i \geq 0 \ \forall i \tag{4}$$

The slack variable ξ_i is the amount of penalty which x_i has to pay to get to the correct side of the hyperplane. This is a kind of regularization and consequently, the SVM objective function can be written as-

$$min_w \ \frac{1}{2} ||w||^2 + C \Sigma_{i=1}^{n_t} \xi_i \tag{5}$$

subject to the constraint specified in eq (4).

Here, C is the regularization hyperparameter which can be fine-tuned on the cross-validation set.

Effect of regularization hyperparameter (C)- The effect of C on the optimization problem (specified in eq(5)) is as follows-

- C **is very large-** If C is very large, then the optimization problem heavily depends on the second part of eq(5). Therefore, the optimal solution will reduce ξ_i to a very small value, i.e., x_i won't be charged any significant penalty to move to the correct side of the hyperplane.

– C **is very small-** If C is very small, then the optimization problem does not depend much on the second part of eq(5) compared to the first part. Therefore, ξ_i can take a very large value, i.e., x_i will have pay a huge penalty to move to the correct side of the hyperplane. In other words, we are allowing outliers to be present.

Solving Dual Lagrangian of SVM Objective function- Instead of solving the constrained optimization problem in eq (5), we can solve the *Dual Lagrangian* for the SVM objective function, which is as follows-

$$max_\alpha \ [\Sigma_{i=1}^{n_t}\alpha_i - \frac{1}{2}\Sigma_{i=1}^{n_t}\Sigma_{j=1}^{n_t}\alpha_i\alpha_j y_i y_j x_i^T x_j \tag{6}$$

$$\text{subject to } 0 \leq \alpha_i \leq C \ \forall i \in \{1,2,3,\ldots,n_t\} \ \& \ \Sigma_{i=1}^{n_t} y_i\alpha_i = 0 \tag{7}$$

Here α_i is the Lagrange multiplier constant for the i_{th} data point (x_i).

4.2 Naive Bayes

We also tried classification using the Naive Bayes approach. However, the accuracy was low (67.4%) than the SVM model, so our final classifier uses only the SVM model. The implementation for the Naive Bayes can be found in the code file named 'naive_bayes.py'. The details of the algorithm used are as follows:

$$P(spam) = \frac{\text{No. of words in spam email}}{\text{Total no. of words}} \tag{8}$$

$$P(ham) = \frac{\text{Number of words in ham email}}{\text{Total Number of words}} \tag{9}$$

$$P(w) = \frac{TF(w) \times IDF(w)}{\Sigma_{\forall \text{words x}\in\text{training set}} TF(x) \times IDF(x)} \tag{10}$$

$$P(w|spam) = \frac{TF(w|spam) \times IDF(w)}{\Sigma_{\forall \text{words x}\in\text{training set}} TF(x|spam) \times IDF(x)} \tag{11}$$

$$P(w|ham) = \frac{TF(w|ham) \times IDF(w)}{\Sigma_{\forall \text{words x}\in\text{training set}} TF(x|ham) \times IDF(x)} \tag{12}$$

$$Probability(spam|email) = \frac{\prod_{w\in\text{words in email}} Prob(w_i|spam) * Prob(Spam)}{\sum_{w\in\text{words in email}} Prob(w)} \tag{13}$$

$$Probability(ham|email) = \frac{\prod_{w\in\text{words in email}} Prob(w_i|ham) * Prob(ham)}{\sum_{w\in\text{words in email}} Prob(w)} \tag{14}$$

If Prob(spam|email)$\geq$ Prob(ham|email) then we classify the email as spam, ham otherwise.

5 Performance and Results

We have used the F_1**-score** as the evualtion metric to deteremine the performance of the classifiers. The F_1 score is calculated as follows-

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{15}$$

The reason behind using F_1 score as the evaluation metric instead of a simple *accuracy* score is the fact that both the training set and the cross-validation set are skewed datasets with majority of the emails being hams. In case of such skewed datasets, it is often seen that a classifier whose accuracy is very high needn't always perform best. For example, suppose we have a dataset which contains 100 emails, out of which only 5 emails are spams. In such a dataset, a classifier can easily give a high accuracy of 95 % by classifying all the emails as ham. However, we can see that such a classifier is actually a very poor classifier even though it gives a very high accuracy.

Therefore, we need to take into account other metrics like *precision* and *recall*. Eq (16) gives the formula for calculating *precision* and eq (17) gives the formula for calculating *recall*.

$$precision = \frac{\text{No. of emails classified as spam which are actually spam}}{\text{Total no. of emails classified as spam by the classifier}} \tag{16}$$

$$recall = \frac{\text{No. of spam emails classified as spam by the classifier}}{\text{Total no. of spam emails}} \tag{17}$$

F_1 score is actually the harmonic mean of *precision* and *recall* and can serve as an ideal trade-off for these metrics.

5.1 SVM Results

We have used the *scikit-learn* library to implement the SVM algorithm in this project. As mention in Section 3.1, the threshold for collection frequency (or simply, threshold frequency) that is used to create the vocabulary is a hyperparameter which requires fine-tuning to give best results. We tested our SVM model for 5 different threshold frequencies- $\{5, 10, 30, 50, 100\}$. Additionally, for each threshold frequency, we also tested the model with 10 different regularization parameter (C) values- $\{1, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$. Table 1 shows the results obtained after performing the hyperparameter tests.

Threshold Frequency	Vocabulary size	Best C	F_1 score	Training time (in min.)
5	**6690**	**1**	**0.93**	**34:57**
10	4124	1	0.89	18:47
30[4]	1846	60	0.87	07:35
50[4]	1223	20	0.88	04:45
100	619	10	0.86	02:03

Table 1. Size of vocabulary obtained, value of regularization parameter (C) giving best results, associated F1-score obtained on the cross-validation set, and training time to perform the hyperparameter tuning, for different values of threshold frequency.

From Table 1, we can see that best combination of values for threshold frequency and C, and the associated vocabulary size, F_1 score, and training time, are as follows-

Threshold Frequency	5
C	1
Vocabulary size	6690
F_1 **score on Cross-Validation set**	0.93
Training time (in min.)	34:57

Table 2. Size of vocabulary obtained, value of regularization parameter (C) giving best results, associated F1-score obtained on the cross-validation set, and training time to perform the hyperparameter tuning, for different values of threshold frequency.

6 Conclusion

In this experiment, we built two spam classifiers based on the SVM algorithm and the Naive Bayes algorithm respectively. The majority of the tests were done for the SVM classifier only.

In case of the SVM classifier, we found that the linear SVM classifier with regularization parameter $C = 1$ gave the best results, resulting in an F_1 score of 0.93 and *accuracy* of 95.56 % on the cross-validation set . We also saw that the vocabulary plays an important role in determining the efficacy of the classification. For example, a threshold collection frequency of 5 used to filter the words before vocabulary creation lead to the best results. Compared to the SVM classifier, the Naive Bayes classifier performed rather poorly giving an *accuracy* of only 67 %. As the accuracy was very low compared to that obtained for the linear SVM classifier, we did not calculate the scores for other evaluation metrics (F_1 score, *precison*, *recall*) for the Naive Bayes classifier.

[4] The best classifier obtained for these values of threshold frequency incorrectly classified *email1.txt* as spam.

Based on the results obtained, we decided to submit the linear SVM classifier (trained with threshold frequency of 5, and C = 1) as the model to be used for classifying the emails in the test set.

References

1. [William W. Cohen, MLD, CMU, 2015] *Enron Email Dataset.* Machine Learning Department, Carnegie Mellon University. `https://www.cs.cmu.edu/~./enron/`

A Hyperparameter Tuning Test Results for SVM

This section contains the results obtained after performing the different hyperparameter tuning tests. We show the scores obtained for the different evaluation metrics for different values of the regularization parameter (C), corresponding to each threshold frequency. The different threshold frequencies and C values are listed below-

Hyperparameter	**No. of test values**	**Values**
Threshold Frequency	5	5, 10, 30, 50, 100
C	10	1, 10, 20, 30, 40, 50, 60, 70, 80, 90

Table 3. Test value details for different hyperparameters.

A.1 Hyperparameter tuning test results

```
=============================================
HYPERPARAMETER TUNING TEST RESULT:
---------------------------------------------
   Reg. Parameter (c)  Train Accuracy  Cross-Validation Accuracy  Cross-Validation Precision  Cross-Validation Recall  Cross-Validation F1-Score
0                 1.0        0.997341                   0.955556                    0.884848                 0.973333                   0.926984
1                10.0        0.999033                   0.937198                    0.840580                 0.966667                   0.899225
2                20.0        1.000000                   0.933333                    0.829060                 0.970000                   0.894009
3                30.0        1.000000                   0.932367                    0.826705                 0.970000                   0.892638
4                40.0        1.000000                   0.926570                    0.821839                 0.953333                   0.882716
5                50.0        1.000000                   0.929469                    0.828986                 0.953333                   0.886822
6                60.0        1.000000                   0.924638                    0.824561                 0.940000                   0.878505
7                70.0        1.000000                   0.924638                    0.828402                 0.933333                   0.877743
8                80.0        1.000000                   0.923671                    0.825959                 0.933333                   0.876369
9                90.0        1.000000                   0.923671                    0.825959                 0.933333                   0.876369
---------------------------------------------
SELECTED MODEL STATS:
---------------------------------------------
Classifier type: Linear SVM Classifier
Regularization parameter value (c): 1
---------------------------------------------
Train accuracy: 1.0
Cross-valididation accuracy: 0.9555555555555556
Cross-valididation precision: 0.8848484848484849
Cross-valididation recall: 0.9733333333333334
Cross-valididation f1-Score: 0.926984126984127
=============================================
```

Fig. 3. Hyperparameter tuning test result for *threshold frequency = 5*.

```
=============================================
HYPERPARAMETER TUNING TEST RESULT:
---------------------------------------------
   Reg. Parameter (c)  Train Accuracy  Cross-Validation Accuracy  Cross-Validation Precision  Cross-Validation Recall  Cross-Validation F1-Score
0                 1.0        0.994440                   0.934300                    0.889262                 0.883333                   0.886288
1                10.0        0.998550                   0.927536                    0.876254                 0.873333                   0.874791
2                20.0        0.999275                   0.930435                    0.880000                 0.880000                   0.880000
3                30.0        0.999517                   0.927536                    0.876254                 0.873333                   0.874791
4                40.0        0.999758                   0.923671                    0.872054                 0.863333                   0.867672
5                50.0        0.999758                   0.925604                    0.875421                 0.866667                   0.871022
6                60.0        0.999758                   0.922705                    0.869128                 0.863333                   0.866221
7                70.0        0.999758                   0.922705                    0.869128                 0.863333                   0.866221
8                80.0        0.999758                   0.921739                    0.868687                 0.860000                   0.864322
9                90.0        0.999758                   0.921739                    0.868687                 0.860000                   0.864322
---------------------------------------------
SELECTED MODEL STATS:
---------------------------------------------
Classifier type: Linear SVM Classifier
Regularization parameter value (c): 1
---------------------------------------------
Train accuracy: 0.9997582789460963
Cross-valididation accuracy: 0.9342995169082126
Cross-valididation precision: 0.889261744966443
Cross-valididation recall: 0.8833333333333333
Cross-valididation f1-Score: 0.8862876254180602
=============================================
```

Fig. 4. Hyperparameter tuning test result for *threshold frequency = 10*.

```
=============================================
HYPERPARAMETER TUNING TEST RESULT:
---------------------------------------------
   Reg. Parameter (c)  Train Accuracy  Cross-Validation Accuracy  Cross-Validation Precision  Cross-Validation Recall  Cross-Validation F1-Score
0                 1.0        0.992265                   0.922705                    0.864238                 0.870000                   0.867110
1                10.0        0.997099                   0.919807                    0.865320                 0.856667                   0.860972
2                20.0        0.998066                   0.922705                    0.876712                 0.853333                   0.864865
3                30.0        0.998791                   0.925604                    0.883162                 0.856667                   0.869712
4                40.0        0.999033                   0.925604                    0.888502                 0.850000                   0.868825
5                50.0        0.999033                   0.927536                    0.894737                 0.850000                   0.871795
6                60.0        0.999033                   0.928502                    0.895105                 0.853333                   0.873720
7                70.0        0.999033                   0.925604                    0.885813                 0.853333                   0.869270
8                80.0        0.999033                   0.921739                    0.881533                 0.843333                   0.862010
9                90.0        0.999033                   0.921739                    0.881533                 0.843333                   0.862010
---------------------------------------------
SELECTED MODEL STATS:
---------------------------------------------
Classifier type: Linear SVM Classifier
Regularization parameter value (c): 60
---------------------------------------------
Train accuracy: 0.9990331157843848
Cross-valididation accuracy: 0.9285024154589372
Cross-valididation precision: 0.8951048951048951
Cross-valididation recall: 0.8533333333333334
Cross-valididation f1-Score: 0.8737201365187713
=============================================
```

Fig. 5. Hyperparameter tuning test result for *threshold frequency = 30*.

```
=============================================
HYPERPARAMETER TUNING TEST RESULT:
---------------------------------------------
   Reg. Parameter (c)  Train Accuracy  Cross-Validation Accuracy  Cross-Validation Precision  Cross-Validation Recall  Cross-Validation F1-Score
0                 1.0        0.988881                   0.916908                    0.847403                 0.870000                   0.858553
1                10.0        0.996616                   0.926570                    0.873333                 0.873333                   0.873333
2                20.0        0.998308                   0.930435                    0.885135                 0.873333                   0.879195
3                30.0        0.999033                   0.925604                    0.880546                 0.860000                   0.870152
4                40.0        0.999033                   0.926570                    0.888889                 0.853333                   0.870748
5                50.0        0.999033                   0.928502                    0.892361                 0.856667                   0.874150
6                60.0        0.999033                   0.926570                    0.891608                 0.850000                   0.870307
7                70.0        0.999033                   0.924638                    0.885417                 0.850000                   0.867347
8                80.0        0.999033                   0.924638                    0.885417                 0.850000                   0.867347
9                90.0        0.999033                   0.926570                    0.886207                 0.856667                   0.871186
---------------------------------------------
SELECTED MODEL STATS:
---------------------------------------------
Classifier type: Linear SVM Classifier
Regularization parameter value (c): 20
---------------------------------------------
Train accuracy: 0.9990331157843848
Cross-valididation accuracy: 0.9304347826086956
Cross-valididation precision: 0.8851351351351351
Cross-valididation recall: 0.8733333333333333
Cross-valididation f1-Score: 0.8791946308724832
=============================================
```

Fig. 6. Hyperparameter tuning test result for *threshold frequency = 50*.

```
============================================
HYPERPARAMETER TUNING TEST RESULT:
--------------------------------------------
   Reg. Parameter (c)  Train Accuracy  Cross-Validation Accuracy  Cross-Validation Precision  Cross-Validation Recall  Cross-Validation F1-Score
0                 1.0        0.984772                   0.919807                    0.846645                 0.883333                   0.864600
1                10.0        0.993474                   0.921739                    0.866221                 0.863333                   0.864775
2                20.0        0.996374                   0.914010                    0.855219                 0.846667                   0.850921
3                30.0        0.996616                   0.910145                    0.843854                 0.846667                   0.845258
4                40.0        0.997099                   0.905314                    0.838926                 0.833333                   0.836120
5                50.0        0.997099                   0.897585                    0.825503                 0.820000                   0.822742
6                60.0        0.997099                   0.898551                    0.832765                 0.813333                   0.822934
7                70.0        0.997099                   0.899517                    0.833333                 0.816667                   0.824916
8                80.0        0.997099                   0.898551                    0.830508                 0.816667                   0.823529
9                90.0        0.997099                   0.902415                    0.832776                 0.830000                   0.831386
--------------------------------------------
SELECTED MODEL STATS:
--------------------------------------------
Classifier type: Linear SVM Classifier
Regularization parameter value (c): 10
--------------------------------------------
Train accuracy: 0.9970993473531544
Cross-valididation accuracy: 0.9217391304347826
Cross-valididation precision: 0.8662207357859532
Cross-valididation recall: 0.8633333333333333
Cross-valididation f1-Score: 0.8647746243739565
============================================
```

Fig. 7. Hyperparameter tuning test result for *threshold frequency = 100*.