Tashrif Billah

tb2699@columbia.edu

Columbia University

Department of Electrical Engineering

EEOR 4650: Convex Optimization Course Project

Spam Email Identification Using Support Vector Machine Classifier

Abstract

SVM is an effective tool in data classification and machine learning. Applying different forms of SVM technique, we have tried to come up with a model that effectively classifies Spam and Non-Spam emails. Linear SVM with soft margin modification has been found to perform the best achieving ~99% classification accuracy on both training and test data. Its performance is also compared with other methods in terms of accuracy and running time. The following table demonstrates the whole work.

Table-1: Comparision among Different Classification Methods

Learning Method	#Obtained Percentage Accuracy $\mu \pm \sigma$	Comparative Running Time
SVM with Soft Margin [1]	98.75 ± 0.34	Quite fast
*SVM with Gaussian Kernel [1]	89.57 ± 8.50	Slow
SVM with Polynomial Kernel [1]	84.77 ± 6.35	Slow
*SVM with SMO [2]	82.42 ± 10.50	Moderate
MATLAB Built-in SVM	98.10 ± 0.90	Very fast
Gradient Descent Method with Newton Step	[$]Future Work	Unexplored

[#]*The percentage accuracy is averaged over at least six runs of each model with best parameters learned through cross validation.*

[$]*Gradient descent methodology has been elaborated in this report. However, due to time constraint and complexity in coding, it could not be tested. We present this algorithm here because we believe that this method can increase speed of learning through vectorization in the coding.*

**The Gaussian Kernel method and Sequential Optimal Minimization may perform better than what is presented here. Their performance is contingent upon right parameter choice. We tried our best to pick up the right parameter. But due to slow convergence time, a wide range of parameter cross validation could not be done.*

Table of Contents

List of Code Files

Table-1

***** ReadMe file (provided with codes) contains specific instruction about running the code**

Detailed instruction and functionality should be apparent from the table below.

Type	Name of the file (.m)		Input	Output
Main file, calls all other sub routines	Main_Soft_Margin		Train ratio, C, Sigma, and p parameters	Percentage classification accuracy
	Main_Gauss_Kernel			
	Main_Poly_Kernel			
	Main_SMO			
Performs K-fold cross validation	Cross_Validation_Poly_Kernel		Fold, range of C, range of p, and range of Sigma	Minimum error Parameters
	Cross_Validation_Gauss_Kernel			
	Cross_Validation_Soft_Margin			
Feature extraction	email_representation		Database	Feature vector of all emails
Auxiliary file	matlab_svm_compare		Train ratio	A model and its peformance
	SVM_Model_Test		File name that contains learned parameters	Average performance of the model
	renaming_txt		Database	Renames all the files as .txt
Sub routines	Calling function	Sub routines		
	All main files	email_dataset	Train ratio	Train set and Test set
	email_representation	processEmail	Emails	Word indices
		readFile	File name	Reads the contents
		feature_extraction	Word indices	$[0,1]^{1899x1}$
	processEmail	getVocabList	-	Reads dictionary
		porterStemmer	Emails	Normalization
	Main_SMO	examineExample	Index	Flag
		takeStep	Index	Flag

Table-2: List of Output Files

Name of the file	Type	Content
Gauss_Kernel_Cross_Val	.txt	Errors over different C and Sigma for 3 fold
Poly_Kernel_Cross_Val		Errors over different C and p for 3 fold
Soft_Margin_Cross_Val		Errors over different C for 5 fold

Table-3: List of Raw Database Files

Name of the file	Type	Content
easy_ham	Folder	2,500 non spam emails
easy_ham_2		1,400 non spam emails
hard_ham		250 non spam emails
spam		500 spam emails
spam_2		1,400 spam emails
Total 6,050 emails- 4,150 non spam emails, 1,900 spam emails with ~31% spam ratio		
vocab	.txt	Dictionary

Table-4: List of Processed Data Files

Name of the file	Type	Content	
		Feature Vector X	Feature Label y
easy_ham	.mat	2500x1899	2500x1
easy_ham_2		1400x1899	1400x1
hard_ham		250x1899	250x1
spam		500x1899	500x1
spam_2		1400x1899	1400x1
All_email_features		6050x1899	6050x1
spamTest		Professor's Data	
spamTrain			

Problem Statement

There is a database of both spam and non-spam emails. We are required to find a classification model that can distinguish spam emails from the non-spam ones. Specifically, we want to come up with a model that achieve over 95% classification accuracy on both train and test data set in the classification process.

Task-1

Feature Vector Construction for All Emails

We are given a database of 6,050 emails as described in Table-3. Firstly, we renamed all the emails there as .txt file using rename.m code. After all the files are accessible, we constructed a feature vector for each email while the feature label is 1 for spam emails and 0 for non-spam emails.

For this task, processEmail.m is called with each email. It then stems the words in that email calling porterStemmer.m following the normalization procedure given in the problem description. It then compares each stemmed word with dictionary words in vocabList.txt file. There are 1899 words in the dictionary. The initial feature vector is a column of zeros. If a word from the dictionary appears in the email, the corresponding feature vector is set to 1 and 0 otherwise. Thereby, each email is represented by $[0,1]^{1899x1}$ feature vector.

The following output shows elapsed time for the above feature extraction of each folder in the database. The total time required to represent all the 6,050 emails by their corresponding feature vectors was about one hour.

```
The total time taken for this step is 58.50 minutes.
ans =

easy_ham

Elapsed time is 1194.960666 seconds.

ans =

easy_ham_2

Elapsed time is 805.191621 seconds.

ans =

hard_ham

Elapsed time is 325.904113 seconds.

ans =

spam

Elapsed time is 322.929495 seconds.
```

```
ans =

spam_2

Unable to open spam_2\00489.txt
Unable to open spam_2\00529.txt
Unable to open spam_2\00563.txt
Unable to open spam_2\00601.txt

Elapsed time is 861.016929 seconds.
```

Total elapsed time is 3509.986054 seconds.

As the output shows, some emails did not exist. Later, the four vacant emails were filled up by other same tag emails.

Task-2

Method-1: Soft Margin SVM[1]

When data are linearly non-separable, we relax the constraints of classification by introducing slack variables ζ_i and cost for this introduction. The procedure is elaborately explained in [1]. The scenario of slack variable introduction is depicted in the following figure.

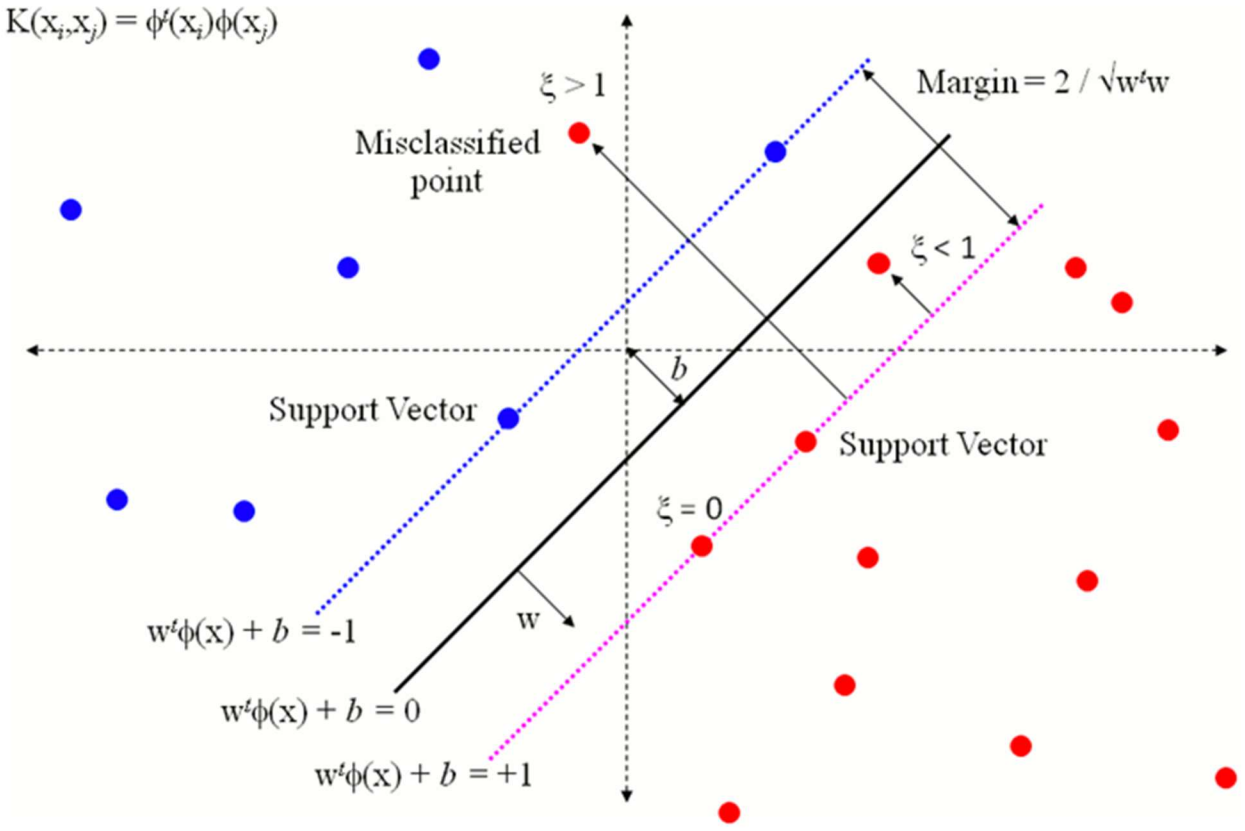


Image Courtesy: Google

We then solve the following optimization problem for vector **w** that is perpendicular to the optimal hyperplane where x_i is the feature vector for each training example and y_i is its label.

minimize

$$\|\mathbf{w}\|^2/2 \text{ to } \|\mathbf{w}\|^2/2 + C\left(\sum_i \xi_i\right)^k$$

Subject to

$$\begin{aligned} \mathbf{x}_i \cdot \mathbf{w} + b &\geq +1 - \xi_i \quad \text{for } y_i = +1 \\ \mathbf{x}_i \cdot \mathbf{w} + b &\leq -1 + \xi_i \quad \text{for } y_i = -1 \\ \xi_i &\geq 0 \; \forall i. \end{aligned}$$

Once we have the optimal **w,** we find the bias b by solving the equation for all training data points and then averaging over all of them.

$$\alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} = 0$$

We then find the label of each test element as the $sgn(w.x + b)$.

Method-2: Gaussian Kernel SVM[1]

The heart of this idea is to map the data to a higher dimensional space and then look for linear separator in that space.

$$\Phi : \mathbf{R}^d \mapsto \mathcal{H}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j).$$

Then the training algorithm depends on the data through dot products in *H.* Again, if there were a "kernel function" *K* defined as above, we would only need to use *K* in the training algorithm, and would never need to explicitly even know what ϕ is. This approach will find out the support vectors in the same time as it would take for the un-mapped data. On the other hand, we have a better optimal hyperplane that separates the data more accurately. The Kernel function and the method is outlined below which is explained at length in [1].

Kernel function

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}.$$

Convex optimization problem

Maximize:

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \, \boldsymbol{K(x_i, x_j)}$$

subject to:

$$0 \leq \alpha_i \leq C,$$

$$\sum_i \alpha_i y_i = 0.$$

The $K(x_i,x_j)$ is shown in bold to point out the distinction of Gaussian Kernel method with un-mapped data classification where we work on only the dot product $x_i.x_j$. The rest of the procedure is same as the latter one.

Finding the b parameter

$$\alpha_i \{ y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \} = 0$$

Determining the label

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b$$

We check the sign of f(x) to determine the label of each test example.

Method-3: Polynomial Kernel SVM[1]

The procedure is same as that of Gaussian Kernel case. However, the Kernel function is replaced with the following one where p is the degree of the polynomial.

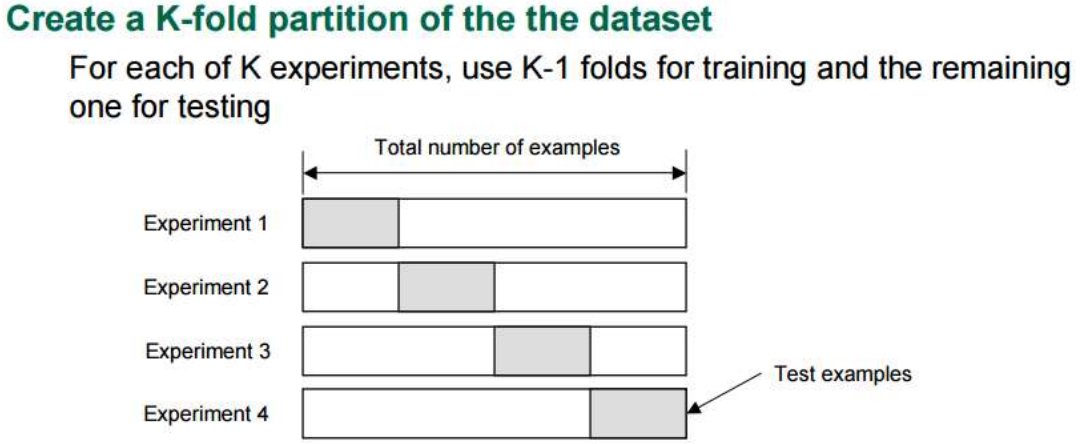
$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$

Method-4: Sequential Minimal Optimization [2]

This method is elaborated in details in [2]. Even, the pseudocode for this algorithm is given in that paper. We followed the pseudocode and wrote exact code for our implementation. To explain briefly, Sequential Minimal Optimization (SMO) is a simple algorithm that can quickly solve the SVM QP problem utilizing space. It does not have to solve numerical QP optimization steps as that of SVM. It decomposes the overall QP problem into QP sub-problems. At every step, SMO chooses two Lagrange multipliers to jointly optimize, finds the optimal values for these multipliers, and updates the SVM to reflect the new optimal values. The advantage of SMO lies in the fact that solving for two Lagrange multipliers can be done analytically. Thus, numerical QP optimization is avoided entirely and running time is improved. Since this is a well-known and at length algorithm, we refer the reader to [2] and skip any further derivation here.

Cross Validation

The variable parameters in Method-1 and Method-4 is the penalty C. Meanwhile, in Method-2 and Method-3, they are C, Sigma, and p. Cross validation is a well-known technique that thrives the choose the best parameters to that maximum likelihood estimation can be done. We have implemented K-fold cross validation where K=3 or 5 for specific methods. The following diagram[4] explains the method we followed. We considered entire data, sequentially partitioned it into disjoint training and test sets so that all examples are considered as test set one by one. The error obtained over this K-folding was averaged to get error of specific parameters. We varied the parameters until the lowest error was obtained. The learned best parameters are then used to train and test the given data set.



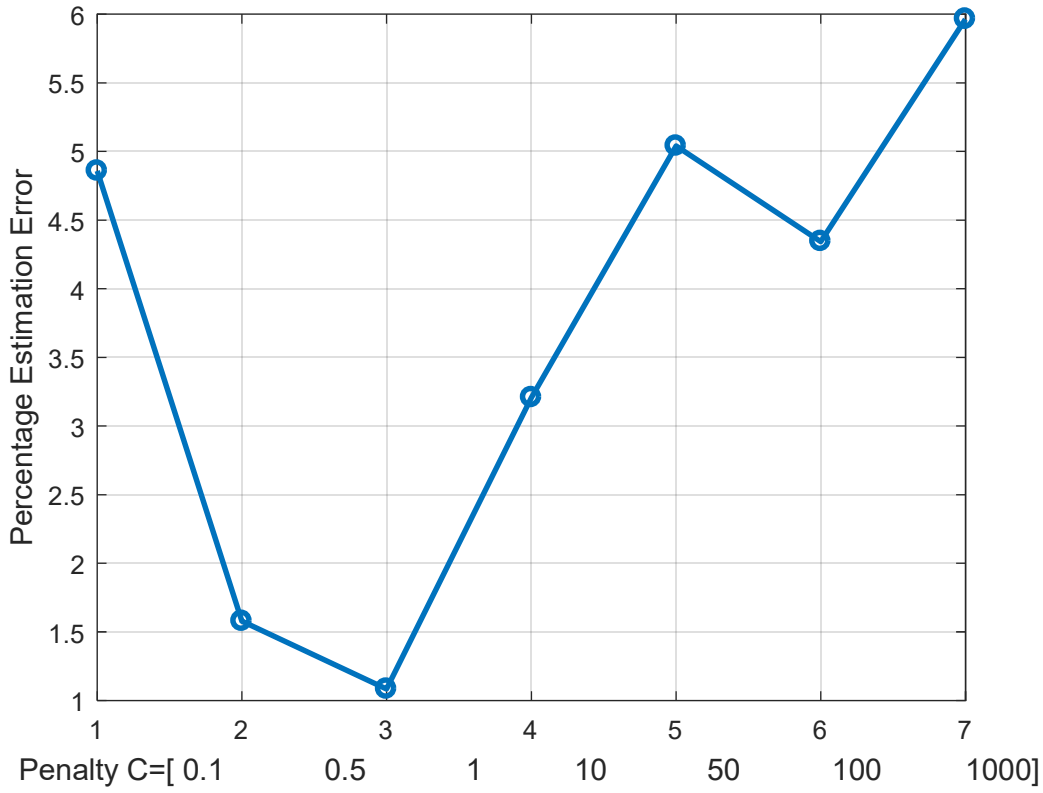
Results

1. Cross Validation

We are trying to establish Method-1 as the best. So, we shall explain cross validation implementation for this method in detail. However, for the other methods, we shall just mention the best parameters we found and refer the reader to the output files and codes we have.

1.1 Cross_Validation_Soft_Margin.m, Soft_Margin_Cross_Val.txt

This code performs 5 fold validation over the entire data set of 6,050 emails. It finds out the average error for the parameters C= =[0.1 0.5 1 10 50 100 1000]. We obtained least error at C= 0.5 and 1 and therefore we have used this penalty function to test the performance of our proposed classification method. The following graph illustrates the learning curve of this validation technique. As it is seen in the graph, the estimation error is least in C= 0.5 and C= 1. In simulation, we found both of them giving good performance.



1.2 Cross_Validation_Gauss_Kernel.m, Gauss_Kernel_Cross_Val.txt

Three fold cross validation was done here over C= [50 100] and Sigma= [50 100]. The least ML error was obtained for C= 50, Sigma= 100. We understand that the obtained parameter still does not give good classification result. Due to slow convergence of the Method-2, we could not investigate any further. Also, our proposed Method-1 has over 98% accuracy and so we leave it for future work.

1.3 Cross_Validation_Poly_Kernel.m, Poly_Kernel_Cross_Val.txt

We did three fold cross validation over C= [10 50 100] and p= [2 3]. The least error was found to be at C= 10, p=2. For the same reason as noted in 1.2, it was not investigated any further.

2. Classification Accuracy

2.1 Main_Soft_Margin.m

We chose 80% of all data as our training set and 20% mutually exclusive data as our test set. Then, we found the optimal **w** i.e. support vectors to separate spam and non-spam emails applying the C= 0.5 that we learnt through cross validation technique. The code first generates the train and test sets calling other sub routines as explained in Table-1. It then solves the optimization problem as explained for Method-1 and fits the model to find the label of each element in the test set. To validate our performance, we have run the model 10 times on arbitrarily chosen train and test data each time. Our model obtains 98.75% overall accuracy in test data classification while more than 99.5% accuracy in train data classification. Performance of Method-1 is concisely noted in Table-5 below. **The running time of this algorithm on the large training and test set that we have chosen- is only 5 minutes.**

Table- 5

Serial #	Percentage Class Identification Accuracy with C= 0.5 and 1 on Test Set			
	Spam	**Non Spam**	**Combined**	**Average $\mu \pm \sigma$**
1	97.37	98.80	98.35	98.75 ± 0.34
2	99.47	98.80	99.01	
3	98.16	98.31	98.26	
4	98.95	98.80	98.84	
5	97.37	98.92	98.43	
6	98.68	99.28	99.09	
7	97.89	98.92	98.60	
8	97.89	99.04	98.68	
9	98.68	99.52	99.26	
10	98.95	98.92	98.93	

Sample Output

==========Test Data==========
True classification rate of spam emails

Tr =

0.9868

False classification rate: given a non-spam email, it is classified as spam w.p.

Fl =

0.0072

True classification rate of non-spam emails

Tr =

0.9928

False classification rate: given a spam email, it is classified as non-spam w.p.

Fl =

0

True classification rate of all emails

Tr =

0.9909

False classification rate of all emails

ans =

0.0091

==========Train Data==========
True classification rate of spam emails

Tr =

0.9993

False classification rate: given a non-spam email, it is classified as spam w.p.

Fl =

0

True classification rate of non-spam emails

Tr =

1

False classification rate: given a spam email, it is classified as non-spam w.p.

Fl =

0

True classification rate of all emails

Tr =

0.9998

False classification rate of all emails

ans =

2.0661e-04

2.2 Main_Gauss_Kernel.m

This method took 3.5 minutes for each iteration and around 2.5 hours for about 55 iterations on a train data length of 4,840 and test data length of 1,210. We tried our best to improve the running time. We calculated the Kernel only once even using vectorization. The only for loop we used in our code is to identify the test set. Yet, this method did not show any sign of improvement in terms of running time. We used C= 50 and Sigma= 100, as learnt through cross validation, yet obtained only around 90% accuracy on our test data.

2.3 Main_Poly_Kernel.m

This method also had the artifact of slow convergence as of Method-3. We used C=10, and p= 3 to obtain our best result of around 85% accuracy.

2.4 Main_SMO.m

The pseudocode we wrote ourselves is exactly taken from [2]. We understand that this is an iterative algorithm. Therefore, we could not do much vectorization. Due to use of a large number of while and for loops, we believe this algorithm suffered slow convergence like Method-2 and Method-3. We used tolerance of 10^{-4} and epsilon of 10^{-3}. Such moderate values could not improve the running time although we did only 5 iterations. We were bound to reduce our sample space to only 2,000 data for training set and 700 data for test set. Yet, we have a poor accuracy of around 83%

Discussion

I. Comparison with MATLAB built-in SVM function

```
SVMModel =
fitcsvm(X,y,'KernelFunction','rbf','KernelScale','auto','Standardize',true);
y_svm= predict(SVMModel, Xtest);
```

$$98.10 \pm 0.9\ \%$$

If we take this algorithm as a ground truth, by comparison with our obtained accuracy given by Method-1 of 98.75%, we can fairly conclude that we have successfully identified spam and non-spam emails.

II. Vectorization

To improve the running time of our algorithms, we have used vectorization as much as possible. We could not use vectorization in Method-4 though because that is an iterative algorithm. However, we used it to evaluate our Kernel in just 1.5 seconds that takes 13 seconds for the for loop to evaluate in the following way.

```
I= ones(m,1);
M= sum_square(X,2);
D= exp(-(M*I'+I*M'-2*(X*X'))/(2*Sigma^2));
```

Also, we used vectorization and CVX tool to evaluate our bias parameter in 1.4 seconds which takes the for loop 6.5 seconds to do.

```
w= (alp.*y)'*X;
Tempb= E*w;

Mb= exp(-sum_square(X-Tempb,2)/(2*Sigma^2));

cvx_begin

    variables bb(m)

    minimize 0

        alp.*(y.*(Mb+bb)-1)==0;

cvx_end

b= mean(bb);
```

III. Separate error calculation

To make our model robust, we have separately calculated spam email identification accuracy and also non-spam email identification accuracy. As a consequence, we can identify both type of emails with more than 98.5% confidence on average. Considering only overall identification rate would not achieve this robustness.

IV. Mistake in the provided resource

There was a mistake in readFile.m. It was discovered during feature vector construction of spam_2 dataset. It does not have 1400 emails in it, rather the following four are missing-

```
Unable to open spam_2\00489.txt
Unable to open spam_2\00529.txt
Unable to open spam_2\00563.txt
Unable to open spam_2\00601.txt
```

The variable fid takes on a value -1 when a file cannot be opened and takes a value greater than 3 vice versa. Putting only fid after the **If** condition generates a MATLAB error when the file does not exist. Hence, the **If** condition was corrected as-

```
if fid>=0
    file_contents = fscanf(fid, '%c', inf);
    fclose(fid);
else
    file_contents = '';
    fprintf('Unable to open %s\n', filename);
end
```

V. Scope of Future Work

Due to the poor convergence time of Method-2,3, and 4 we considered implementation of "Inequality constrained Gradient Descent Algorithm" using Newton's Step and unsmooth Barrier function[3]. Our motivating was based on the scope of massive vectorization possible in this method given vectorization drastically improves running time in machine learning. Although this method is discussed in detail in [3], for the completeness of this report, we note down the skeletal equations of this method which are taken from class lecture.

Inequality constrained minimization

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array} \qquad (1)$$

Logarithmic barrier

reformulation of (1) via indicator function:

$$\begin{array}{ll} \text{minimize} & f_0(x) + \sum_{i=1}^{m} I_-(f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

where $I_-(u) = 0$ if $u \leq 0$, $I_-(u) = \infty$ otherwise (indicator function of $\mathbf{R}_-$)

approximation via logarithmic barrier

$$\begin{array}{ll} \text{minimize} & f_0(x) - (1/t) \sum_{i=1}^{m} \log(-f_i(x)) \\ \text{subject to} & Ax = b \end{array}$$

logarithmic barrier function

$$\phi(x) = -\sum_{i=1}^{m} \log(-f_i(x)), \quad \mathbf{dom}\, \phi = \{x \mid f_1(x) < 0, \ldots, f_m(x) < 0\}$$

Newton decrement

$$\lambda(x) = \left(\Delta x_{\mathrm{nt}}^T \nabla^2 f(x) \Delta x_{\mathrm{nt}}\right)^{1/2} = \left(-\nabla f(x)^T \Delta x_{\mathrm{nt}}\right)^{1/2}$$

Newton's method with equality constraints

---

given starting point $x \in \mathbf{dom}\, f$ with $Ax = b$, tolerance $\epsilon > 0$.
repeat

1. Compute the Newton step and decrement Δx_{nt}, $\lambda(x)$.
2. *Stopping criterion.* **quit** if $\lambda^2/2 \leq \epsilon$.
3. *Line search.* Choose step size t by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$.

Barrier method

---

given strictly feasible x, $t := t^{(0)} > 0$, $\mu > 1$, tolerance $\epsilon > 0$.
repeat

1. *Centering step.* Compute $x^\star(t)$ by minimizing $tf_0 + \phi$, subject to $Ax = b$.
2. *Update.* $x := x^\star(t)$.
3. *Stopping criterion.* **quit** if $m/t < \epsilon$.
4. *Increase* t. $t := \mu t$.

Conclusion

In this work, we presented Soft Margin SVM as the best classification method for spam and non-spam email identification on the provided data set. We learnt the parameters of our method through 5 fold cross validation. Applying the learnt parameter, we obtained a robust accuracy of over 98.75% for both spam and non-spam emails. To look at the result other way, given a spam or non-spam email, it will be falsely identified as the other one with only 1.25% chance. We also compared our method with MATLAB built in algorithm and found the results harmonious with each other. We also self-coded three other classification methods found in previous literatures. The simulation result substantiates our proposed method as the best one for the particular problem.

Acknowledgement

The database was obtained from http://spamassasin.apache.org. We acknowledge the effort they have put forth to develop this data in order to facilitate further research on spam filter making.

Reference

1. Christopher J.C. Burges, "*A Tutorial on Support Vector Machines for Pattern Recognition*", Data Mining and Knowledge Discovery, 2, 121–167 (1998)

2. John C. Platt, "*Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines",* Technical Report MSR-TR-98-14 (1998)

3. Chong Li, *Convex Optimization Course Lecture*, Spring 2017, Columbia University

4. Ricardo Gutierrez-Osuna, *Validation Lecture Slide*, Intelligent Sensor Systems, Wright State University