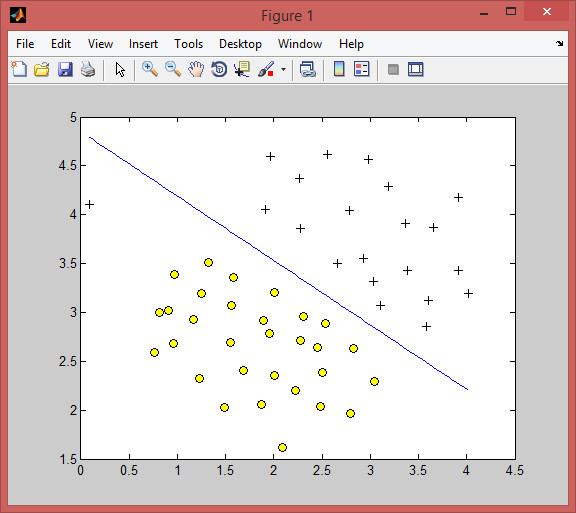
**Machine Learning Programming Assignment 3**

**Comp540 Spring 2015 due 20 February 2015 at 8 pm**

**Miao Wang & Haoyue Zhang (mw56&hz32)**

**2 Support vector machines (15 points)**



**Gaussian kernel (5 points)**

To find non-linear decision boundaries with the SVM, we need to first implement a Gaussian kernel. You can think of the Gaussian kernel as a similarity function that measures the distance between a pair of examples, (*x*(*i*)*, x*(*j*)). The Gaussian kernel is also parameterized by a bandwidth parameter, *σ*, which determines how fast the similarity metric decreases (to 0) as the examples are further apart. You should now complete the code in gaussianKernel.m to compute the Gaussian kernel between two examples. The Gaussian kernel function is defined as:

*k*(*x*(*i*)*, x*(*j*)) = *exp*

( *||x*

*−*

(*i*)

*− x*

2*σ*2

(*j*) 2 \

*||*

5 4

4.5

4

3.5

3

2.5

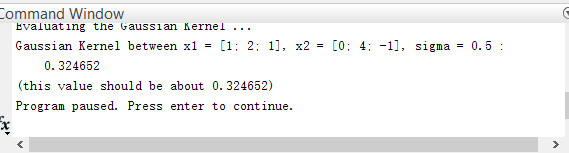
2

1.5

0 0.5 1 1.5 2 2.5 3 3.5 4 4.5

Figure 3: SVM decision boundary with C = 100 for Example dataset 1

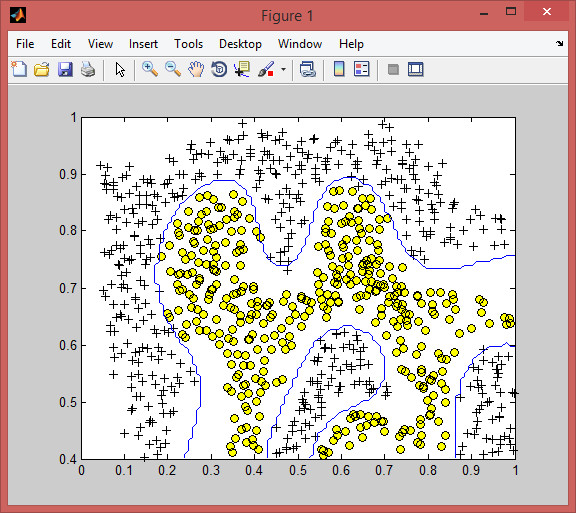
When you have completed the function gaussianKernel.m, the script ex3.m will test your kernel function on two provided examples and you should expect to see a value of 0.324652.



**Example dataset 2**

The next part in ex3.m will load and plot dataset 2 (Figure 4). From the figure, you can observe that there is no linear decision boundary that separates the positive and negative examples for this dataset. However, by using the Gaussian kernel with the SVM, you will be able to learn a non- linear decision boundary that can perform reasonably well for the dataset. If you have correctly implemented the Gaussian kernel function, ex3.m will proceed to train the SVM with the Gaussian kernel on this dataset.

Figure 5 shows the decision boundary found by the SVM with a Gaussian kernel. The decision boundary is able to separate most of the positive and negative examples correctly and follows the contours of the dataset well.



**Example dataset 3 (10 points)**

In this part of the exercise, you will gain more practical skills on how to use a SVM with a Gaussian kernel. The next part of ex3.m will load and display a third dataset (Figure 6). You will be using the SVM with the Gaussian kernel with this dataset. In the provided dataset, ex3data3.mat, you are given the variables X, y, Xval, yval. The provided code in ex3.m trains the SVM classifier using the training set (X, y) using parameters loaded from dataset3Params.m. Your task is to use the validation set Xval, yval to determine the best C and *σ* parameter to use. You should

1 5



0.9

0.8

0.7

0.6

0.5

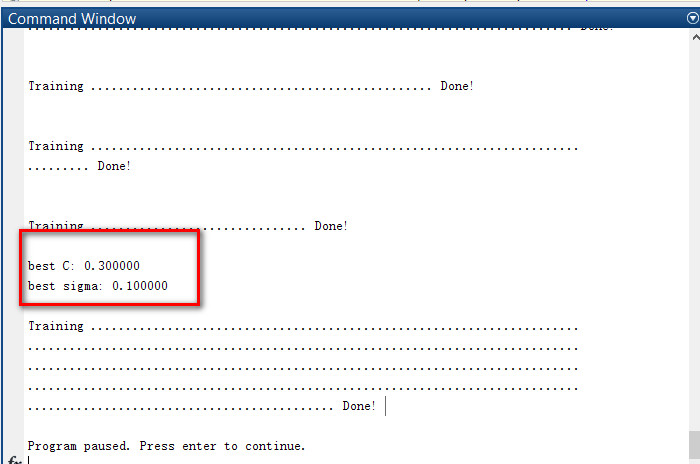
0.4

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Figure 4: Example dataset 2

write any additional code necessary to help you search over the parameters C and *σ*. For both C and *σ*, we suggest trying values in multiplicative steps (e.g., 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30). Note that you should try all possible pairs of values for C and *σ* (e.g., C = 0.3 and *σ* = 0.1). For example, if you try each of the 8 values listed above for C and for *σ*, you would end up training and evaluating (on the validation set) a total of 82 = 64 different models. After you have determined the best C and *σ* parameters to use, you should modify the code in dataset3Params.m, filling in the best parameters you found. For our best parameters, the SVM returned a decision boundary shown in Figure 7.

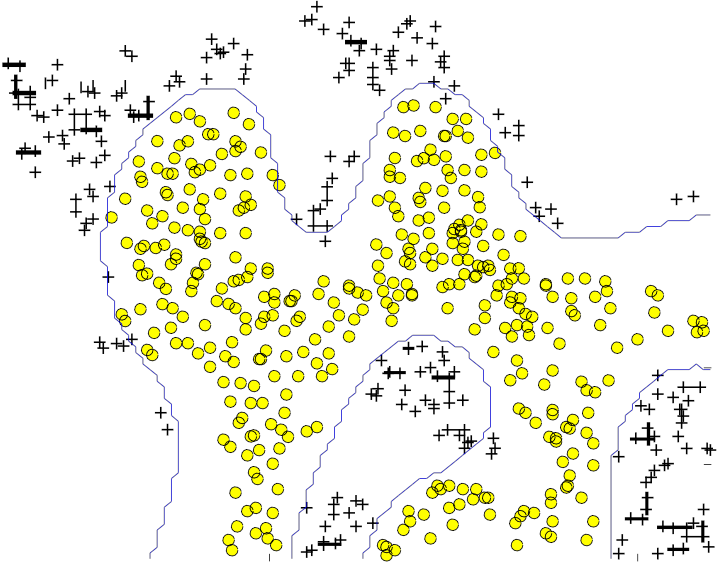
When selecting the best C and *σ* parameter to use, you train on X,y with a given C and *σ*, and then evaluate the error of the model on the validation set. Recall that for classification, the error is defined as the fraction of the validation examples that were classified incorrectly. You can use the svmPredict function to generate the predictions for the validation set.



**3 Spam Classification (10 points)**

Many email services today provide spam filters that are able to classify emails into spam and non- spam email with high accuracy. In this part of the exercise, you will use SVMs to build your own spam filter. You will be training a classifier to classify whether a given email, *x*, is spam or non- spam. In particular, you need to convert each email into a feature vector *x ∈ !Rd*. The following parts of the exercise will walk you through how such a feature vector can be constructed from an email. Throughout the rest of this exercise, you will be using the the script ex3 spam.m. The dataset included for this exercise is based on a subset of the SpamAssassin Public Corpus. For the purpose of this exercise, you will only be using the body of the email (excluding the email headers).

1 6



0.9

0.8

0.7

0.6

0.5

0.4

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Figure 5: SVM gaussian kernel decision boundary for Example dataset 2

**Preprocessing email**

Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset. Figure 8 shows a sample email that contains a URL, an email address (at the end), numbers, and dollar amounts. While many emails would contain similar types of entities (e.g., numbers, other URLs, or other email addresses), the specific entities (e.g., the specific URL or specific dollar amount) will be different in almost every email. Therefore, one method often employed in processing emails is to normalize these values, so that all URLs are treated the same, all numbers are treated the same, etc. For example, we could replace each URL in the email with the unique string ”httpaddr” to indicate that a URL was present.

This has the effect of letting the spam classifier make a classification decision based on whether any URL was present, rather than whether a specific URL was present. This typically improves the performance of a spam classifier, since spammers often randomize the URLs, and thus the odds of seeing any particular URL again in a new piece of spam is very small.

In processEmail.m, we have implemented the following email preprocessing and normalization steps:

*•* **Lower-casing:** The entire email is converted into lower case, so that capitalization is ignored

(e.g., IndIcaTE is treated the same as Indicate).

*•* **Stripping HTML:** All HTML tags are removed from the emails. Many emails often come with HTML formatting; we remove all the HTML tags, so that only the content remains.

*•* **Normalizing URLs:** All URLs are replaced with the text httpaddr.

0.6 7

0.4

0.2

0

−0.2

−0.4

−0.6

−0.8

−0.6 −0.5 −0.4 −0.3 −0.2 −0.1 0 0.1 0.2 0.3

Figure 6: Example dataset 3

*•* **Normalizing Email addresses:** All email addresses are replaced with the text emailaddr.

*•* **Normalizing numbers:** All numbers are replaced with the text number.

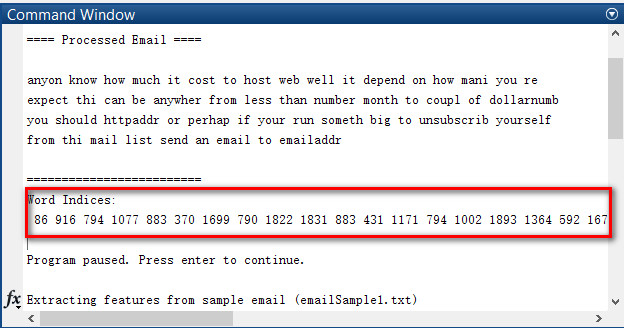
*•* **Normalizing dollars:** All dollar signs ($) are replaced with the text dollar.

*•* **Word stemming:** Words are reduced to their stemmed form. For example, discount, dis- counts, discounted and discounting are all replaced with discount. Sometimes, the stemmer actually strips off additional characters from the end, so include, includes, included, and including are all replaced with includ.

*•* **Removal of non-words:** Non-words and punctuation have been removed. All white spaces

(tabs, newlines, spaces) have all been trimmed to a single space character.

The result of these preprocessing steps is shown in Figure 9. While preprocessing has left word fragments and non-words, this form turns out to be much easier to work with for performing feature extraction.



**Vocabulary list (5 points)**

After preprocessing the emails, we have a list of words (e.g., Figure 9) for each email. The next step is to choose which words we would like to use in our classifier and which we would want to leave out.

For this exercise, we have chosen only the most frequently occuring words as our set of words considered (the vocabulary list). Since words that occur rarely in the training set are only in a few

0.6 8

0.4

0.2

0

−0.2

−0.4

−0.6

−0.8

−0.6 −0.5 −0.4 −0.3 −0.2 −0.1 0 0.1 0.2 0.3

Figure 7: SVM gaussian kernel decision boundary for Example dataset 3

> Anyone knows how much it costs to host a web portal ?

> Well, it depends on how many visitors youre expecting. This can be anywhere from less than 10 bucks a month to a couple of $100. You should checkout <http://www.rackspace.com/>or perhaps Amazon EC2 if youre running something big..

To unsubscribe yourself from this mailing list, send an email to:

[groupname-unsubscribe@egroups.com](mailto:groupname-unsubscribe@egroups.com)

Figure 8: Sample email

emails, they might cause the model to overfit our training set. The complete vocabulary list is in the file vocab.txt. Our vocabulary list was selected by choosing all words which occur at least a

100 times in the spam corpus, resulting in a list of 1899 words. In practice, a vocabulary list with about 10,000 to 50,000 words is often used.

Given the vocabulary list, we can now map each word in the preprocessed emails (e.g., Figure 9) into a list of word indices that contains the index of the word in the vocabulary list. Figure 10 shows the mapping for the sample email. Specifically, in the sample email, the word anyone was first normalized to anyon and then mapped onto the index 86 in the vocabulary list.

Your task now is to complete the code in processEmail.m to perform this mapping. In the code, you are given a string str which is a single word from the processed email. You should look up the word in the vocabulary list vocabList and find if the word exists in the vocabulary list. If the word exists, you should add the index of the word into the word indices variable. If the word does not exist, and is therefore not in the vocabulary, you can skip the word. Once you have implemented

anyon know how much it cost to host a web portal well it depend on how 9

mani visitor your expect thi can be anywher from less than number buck

a month to a coupl of dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth big to unsubscrib yourself from thi mail list send an email to emailaddr

Figure 9: Sample email

86 916 794 1077 883 370 1699 790 1822

1831 883 431 1171 794 1002 1893 1364

592 1676 238 162 89 688 945 1663 1120

1062 1699 375 1162 479 1893 1510 799

1182 1237 810 1895 1440 1547 181 1699

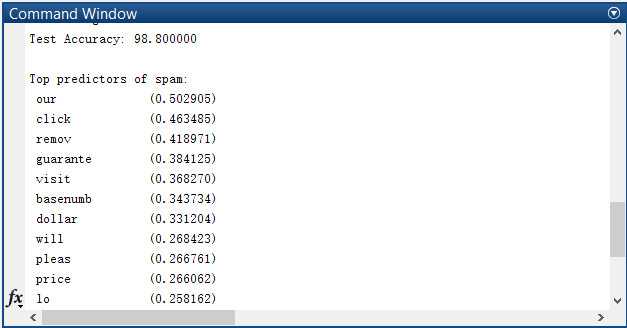
1758 1896 688 1676 992 961 1477 71 530

1699 531

Figure 10: Word indices for Sample email

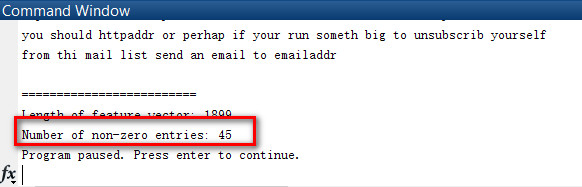
processEmail.m, the script ex3 spam.m will run your code on the email sample and you should see an output similar to Figures 9 and 10.

In the provided code, vocabList is a Matlab cellarray containing the words in the vocabulary. In Matlab, a cellarray is just like a normal array (i.e., a vector), except that its elements can also be strings (which they cannot be in a normal Matlab matrix/vector), and you index into them using curly braces instead of square brackets. Specifically, to get the word at index i, you can use vocabList*{*i*}*. You can also use length(vocabList) to get the number of words in the vocabulary.



**Extracting features from emails (5 points)**

You will now implement the feature extraction that converts each email into a vector in *!Rd*. For this exercise, you will be using *d* = number of words in the vocabulary list. Specifically, the feature *xj ∈ {*0*,* 1*}* for an email corresponds to whether the *jth* word in the dictionary occurs in the email. That is, *xj* = 1 if the *jth* word is in the email and *xj* = 0 if the *jth* word is not present in the email. You should now complete the code in emailFeatures.m to generate a feature vector for an email, given the word indices. Once you have implemented emailFeatures.m, the next part of ex3 spam.m will run your code on the email sample. You should see that the feature vector has length 1899 and 45 non-zero entries.



**Training SVM for spam classification**

After you have completed the feature extraction functions, the next step of ex3 spam.m will load a preprocessed training dataset that will be used to train a SVM classifier. spamTrain.mat contains

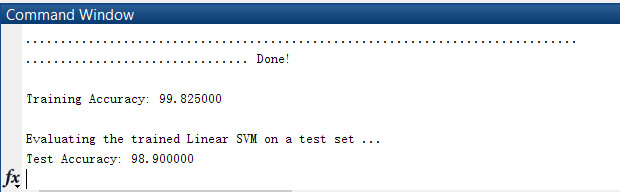
4000 training examples of spam and non-spam email, while spamTest.mat contains 1000 test ex- amples. Each original email was processed using the processEmail and emailFeatures functions

and converted into a vector *x ∈ !R*

1899

. After loading the dataset, ex3 spam.m will proceed to tra1in0

a SVM to classify between spam and non-spam emails. Once the training completes, you should see that the classifier gets a training accuracy of about 99.8% and a test accuracy of about 98.5%.



**Top predictors of spam**

To better understand how the spam classifier works, we can inspect the parameters to see which words the classifier thinks are the most predictive of spam. The next step of ex3 spam.m finds the parameters with the largest positive values in the classifier and displays the corresponding words. In my implementation, I find that this list of top words includes guarantee, remove, dollar, and price. Include the list produced by the script in your writeup for this exercise.

**What to turn in**

Please zip up all the files in the archive (including files that you did not modify) and submit it as pa3 netid.zip on Owlspace before the deadline. Include a PDF file in the archive that presents your plots and your discussion of results from the problems above.

**Acknowledgment**

This exercise was designed by Andrew Ng.