Programming Exercise 6
Support Vector Machines

Machine Learning (Stanford University)

September 2020

This exercise shows the implementation of support vector machines (SVMs) to build a spam classifier.

1 Support Vector Machines

Objectives:

- Use support vector machines with various example 2D datasets
- Experimenting with these datasets will help you gain an intuition on
  - how SVMs work
  - how to use a Gaussian kernel with SVMs

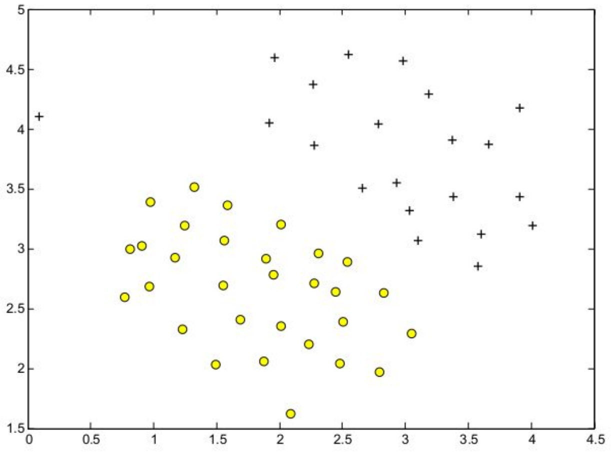
1.1 Example Dataset 1

Dataset

- 2D example dataset that can be separated by a linear boundary
- Indicated with **+** are the positive examples; **o** for negative examples

Remark:

- notice that there is an outlier positive example + on the far left at about (0.1, 4.1)
  - We will also see how this outlier affects the SVM decision boundary

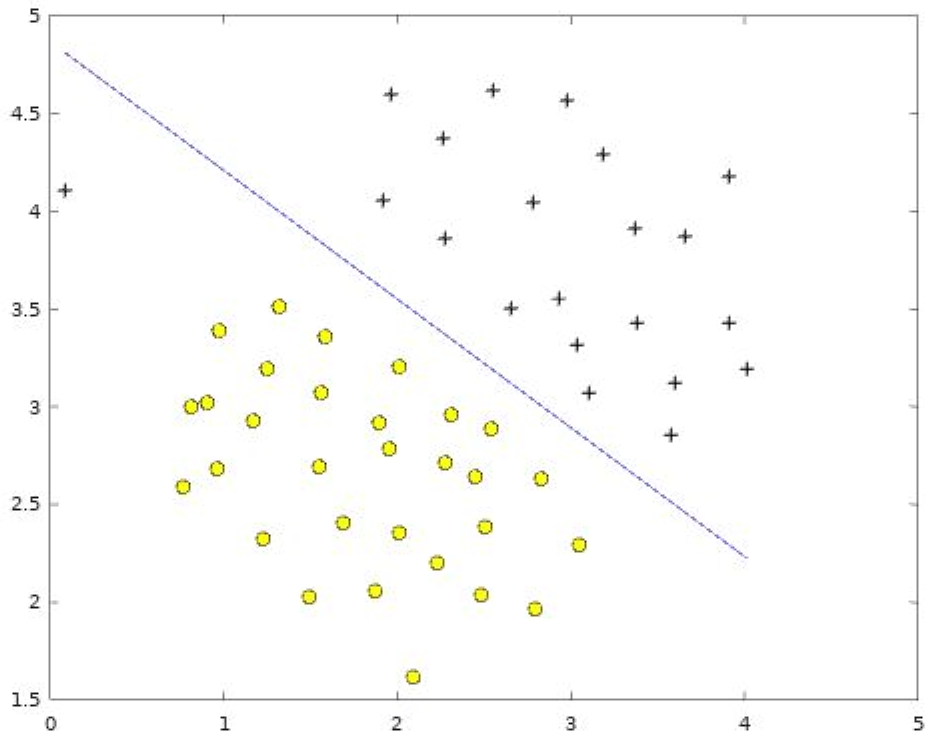


Example Dataset 1

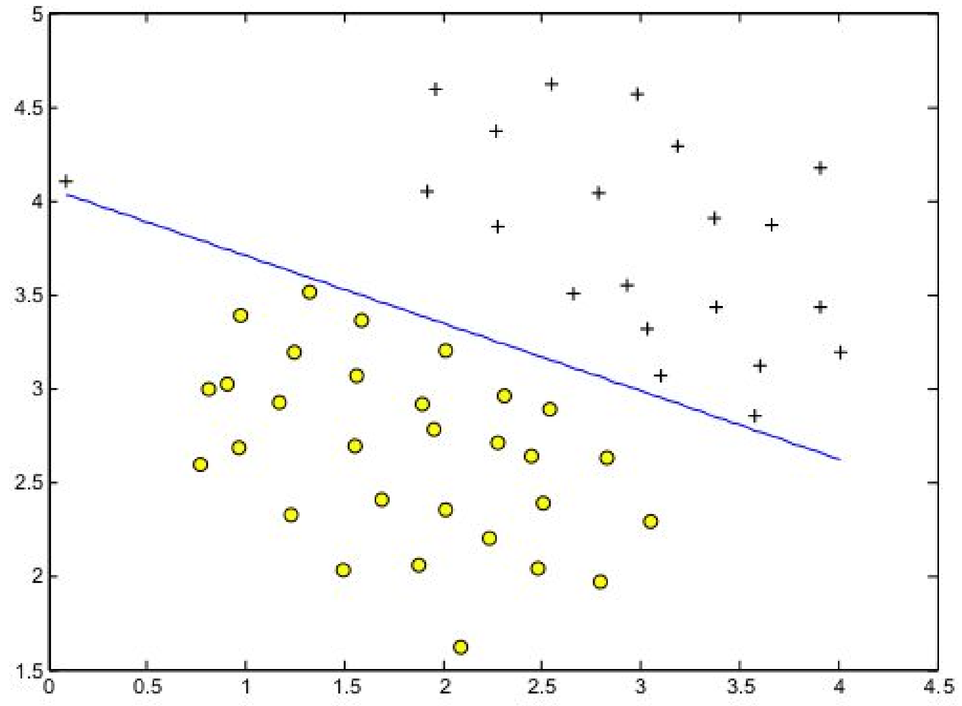
Instruction:
In this part of the exercise, you will try using different values of the C parameter with SVMs. Informally,

C parameter

- A positive value that controls penalty for misclassified training examples
- A large C parameter tells the SVM to try to classify all the examples correctly
- Recall the lecture for its specific behaviors



C =1, No penalty at all



C = 100

Notice how the outlier was considered than when C=1. Also the decision boundary does not appear to be a natural fit for the data.

1.2 SVM with Gaussian Kernels

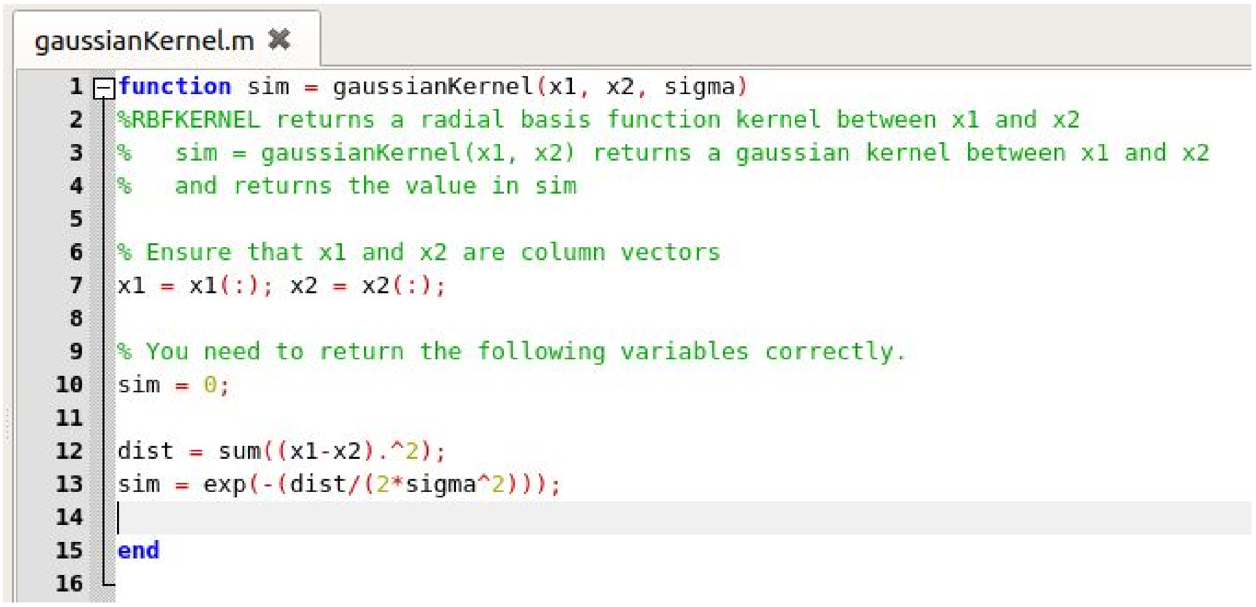
Instruction:
In this part of the exercise, you will be using SVMs to do nonlinear classification. In particular, you will be using SVMs with Gaussian kernels on datasets that are not linearly separable.

1.2.1 Gaussian Kernel

Recall:
Gaussian kernel function is defined as:

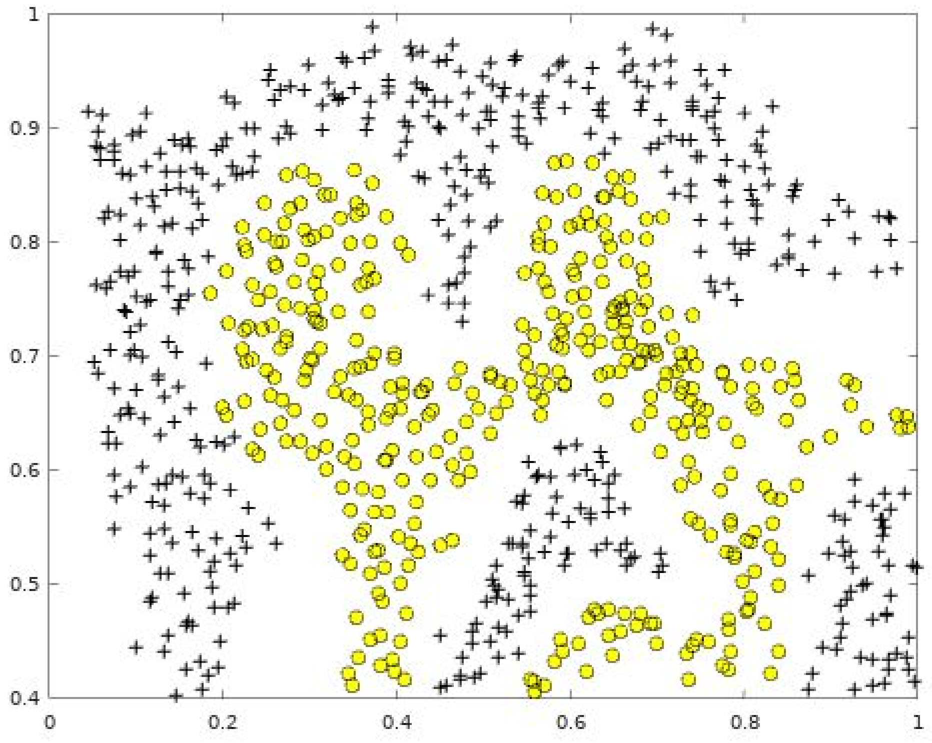
$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$

Implementation:

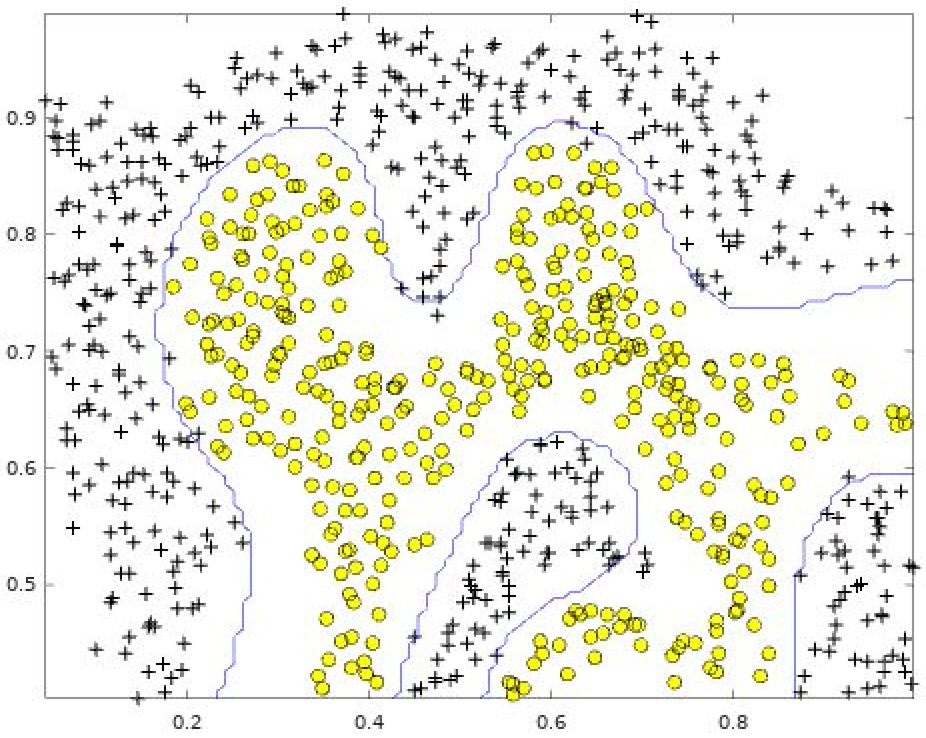


```
function sim = gaussianKernel(x1, x2, sigma)
%RBFKERNEL returns a radial basis function kernel between x1 and x2
%   sim = gaussianKernel(x1, x2) returns a gaussian kernel between x1 and x2
%   and returns the value in sim

% Ensure that x1 and x2 are column vectors
x1 = x1(:); x2 = x2(:);

% You need to return the following variables correctly.
sim = 0;

dist = sum((x1-x2).^2);
sim = exp(-(dist/(2*sigma^2)));

end
```

1.2.2 Example Dataset 2

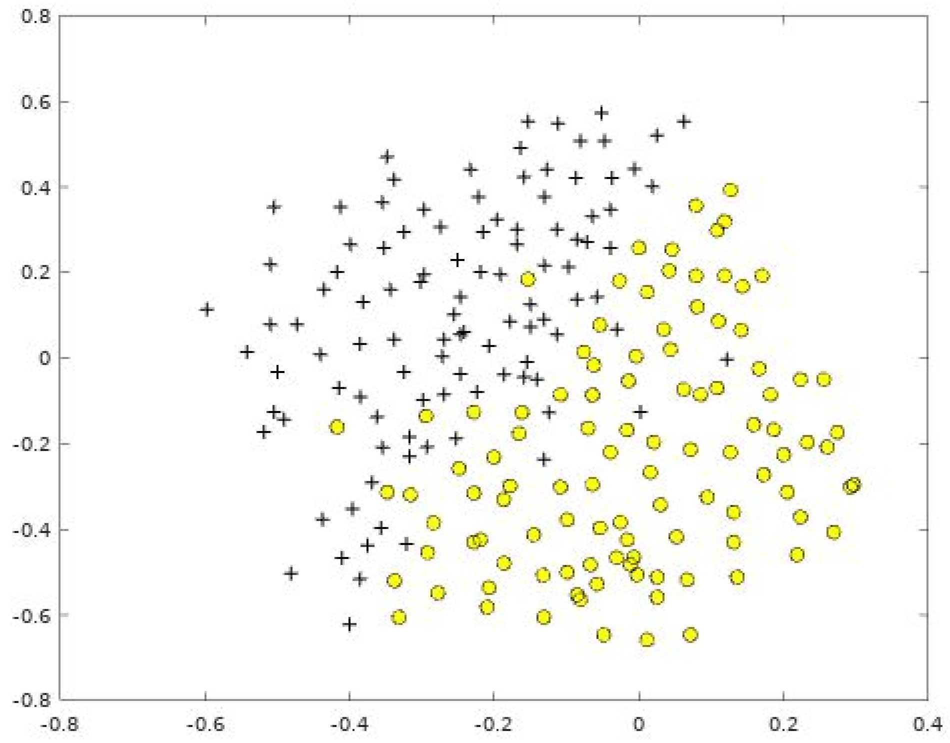


Example Dataset 2



Example Dataset 2. SVM (Gaussian Kernel) Decision Boundary

1.2.3 Example Dataset 3



Example Dataset 3

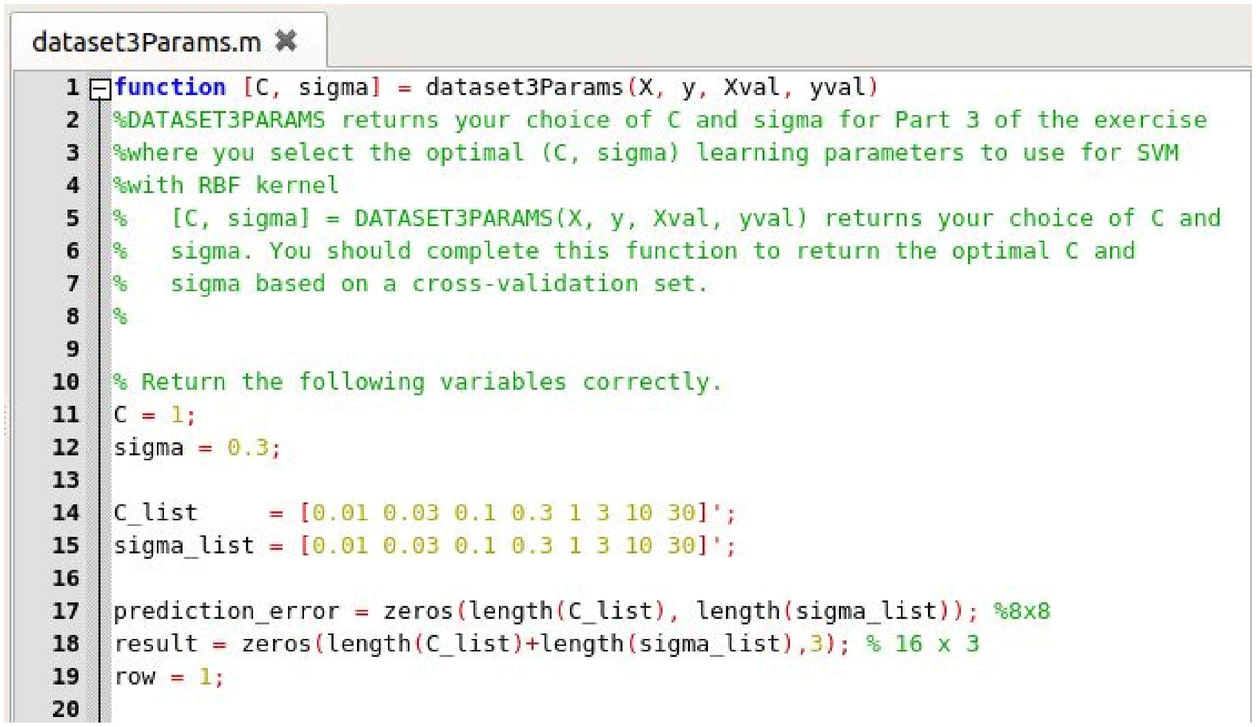
Dataset

- X, Y, Xval, Yval will be provided
- The provided code trains the SVM classifier using the training set (X, y) using loaded parameters

Instruction:

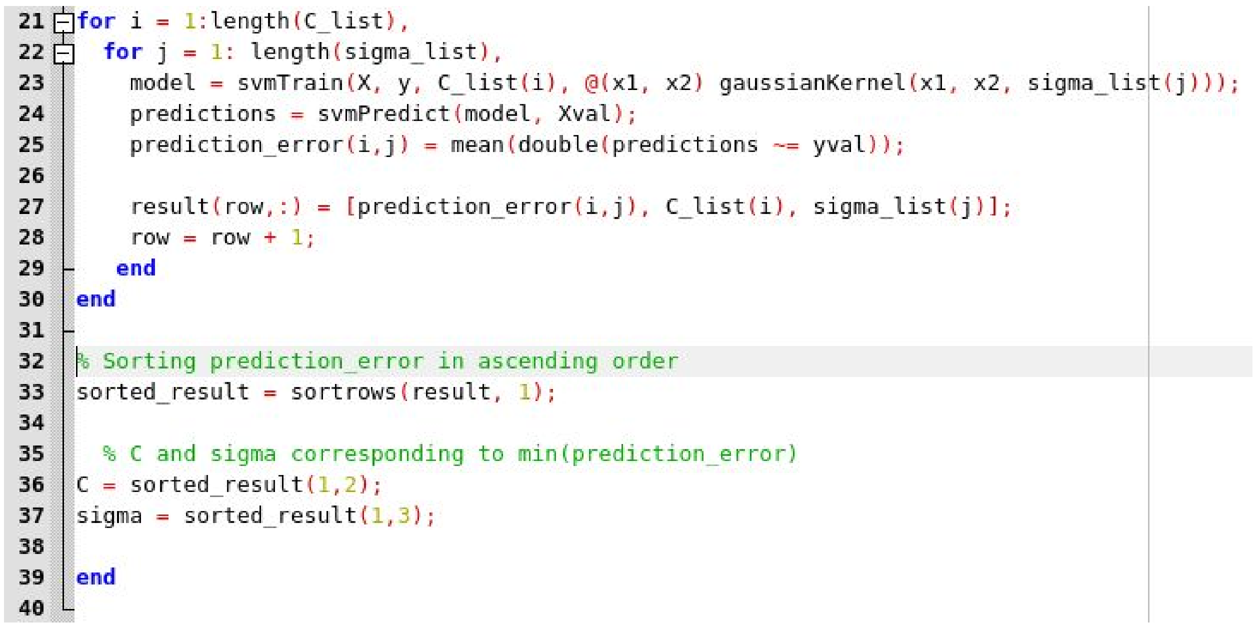
Your task is to use the cross validation set Xval, yval to determine the best C and σ parameter to use. You should write any additional code necessary to help you search over the parameters C and σ. For both C and σ, we suggest trying values in multiplicative steps (e.g., 0.01, 0.03, 0.1, 0.3, 1, 3, 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 2 , you would end up training and evaluating (on the cross validation set) a total of 8 2 = 64 different models.

Implementation:

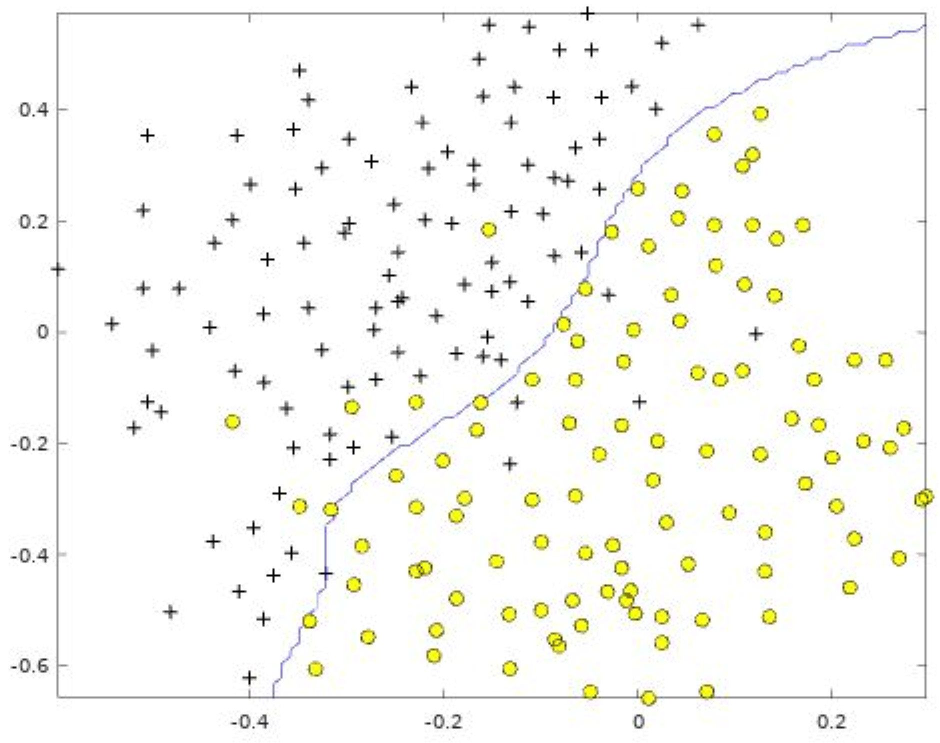


dataset3Params.m

```
function [C, sigma] = dataset3Params(X, y, Xval, yval)
%DATASET3PARAMS returns your choice of C and sigma for Part 3 of the exercise
%where you select the optimal (C, sigma) learning parameters to use for SVM
%with RBF kernel
%   [C, sigma] = DATASET3PARAMS(X, y, Xval, yval) returns your choice of C and
%   sigma. You should complete this function to return the optimal C and
%   sigma based on a cross-validation set.
%

% Return the following variables correctly.
C = 1;
sigma = 0.3;

C_list     = [0.01 0.03 0.1 0.3 1 3 10 30]';
sigma_list = [0.01 0.03 0.1 0.3 1 3 10 30]';

prediction_error = zeros(length(C_list), length(sigma_list)); %8x8
result = zeros(length(C_list)+length(sigma_list),3); % 16 x 3
row = 1;

```



```
for i = 1:length(C_list),
  for j = 1: length(sigma_list),
    model = svmTrain(X, y, C_list(i), @(x1, x2) gaussianKernel(x1, x2, sigma_list(j)));
    predictions = svmPredict(model, Xval);
    prediction_error(i,j) = mean(double(predictions ~= yval));

    result(row,:) = [prediction_error(i,j), C_list(i), sigma_list(j)];
    row = row + 1;
   end
end

% Sorting prediction_error in ascending order
sorted_result = sortrows(result, 1);

  % C and sigma corresponding to min(prediction_error)
C = sorted_result(1,2);
sigma = sorted_result(1,3);

end

```



Example Dataset 3. SVM with C and sigma selected.

2 Spam Classification

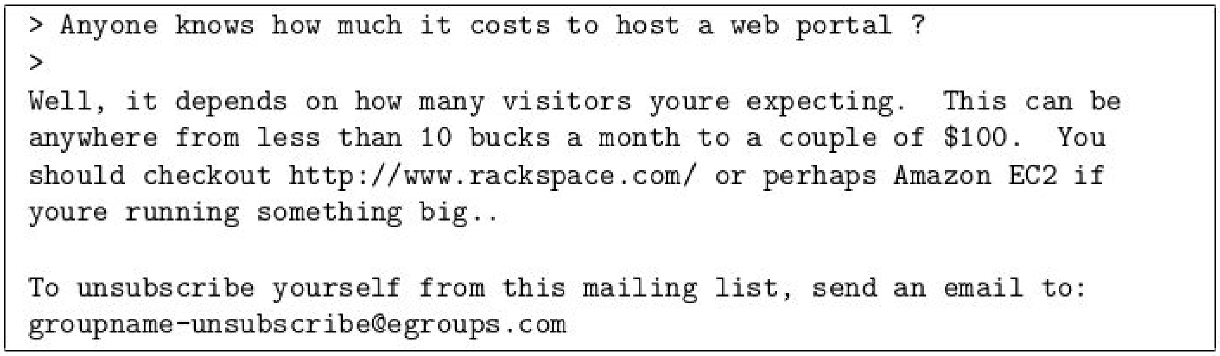
Instruction:
Many email services today provide spam filters that are able to classify emails into spam and non-spam email with high accuracy. In this part of the exercise, you will use SVMs to build your own spam filter.

You will be training a classifier to classify whether a given email, x, is spam (y = 1) or non-spam (y = 0). In particular, you need to convert each email into a feature vector x ∈ R n . The following parts of the exercise will walk you through how such a feature vector can be constructed from an email.

Dataset
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).

2.1 Preprocessing Emails

1. **Look at some examples**
   Advice:
   Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset.



```
> 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
```

Sample Email

2. **The need for normalization**
   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.

- 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
- 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

What’s implemented?

- **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”

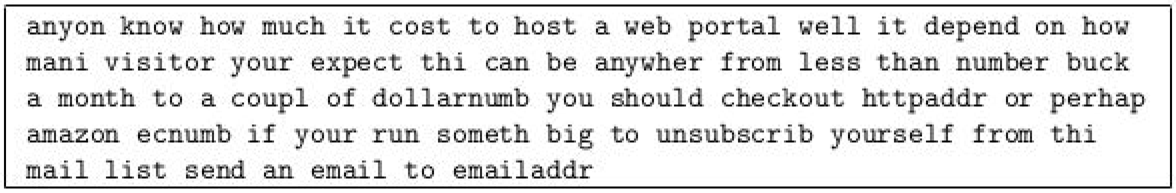
- **Normalizing Email Address**
  - All email addresses are replaced with the text “emailaddr”.

- **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”, “discounts”, “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.



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

Result of these preprocessing steps

Remark:

- While preprocessing has left word fragments and non-words, this form turns out to be much easier to work with for performing feature extraction.

3. **Vocabulary List**

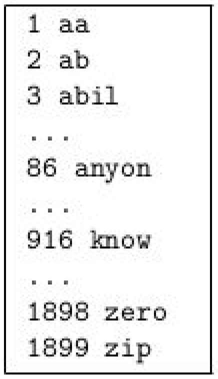
Instruction:
After preprocessing the emails, we have a list of words (e.g., Figure 9) for each email.

A. The next step is to choose which words we would like to use in our classifier and which we would want to leave out.

We have chosen only the most frequently occuring words as our set of words considered (the vocabulary list).

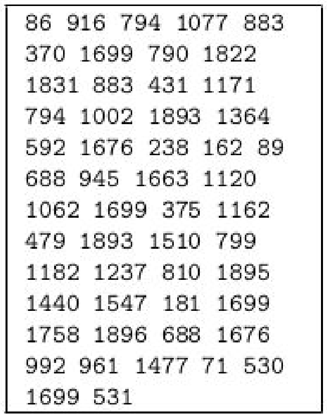
- we have chosen only the most frequently occuring words as our set of words considered (the vocabulary list)
- Our vocabulary list was selected by choosing all words which occur at least a 100 times in the spam corpus, resulting in a list of 1899 words. In practice, a vocabulary list with about 10,000 to 50,000 words is often used.

B. Given the vocabulary list, we can now map each word in the preprocessed emails into a list of word indices that contains the index of the word in the vocabulary list.



```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```

Vocabulary List



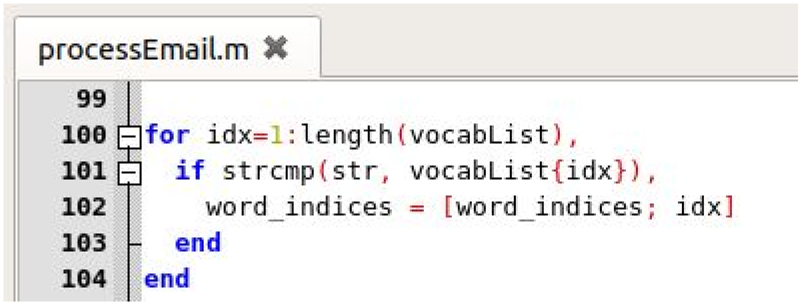
```
86 916 794 1077 883
370 1699 790 1822
1831 883 431 1171
794 1002 1893 1364
592 1676 238 162 89
688 945 1663 1120
1062 1699 375 1162
479 1893 1510 799
1182 1237 810 1895
1440 1547 181 1699
1758 1896 688 1676
992 961 1477 71 530
1699 531
```

Word indices for sample email

Your task now is to complete the code 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.

Implementation:



processEmail.m

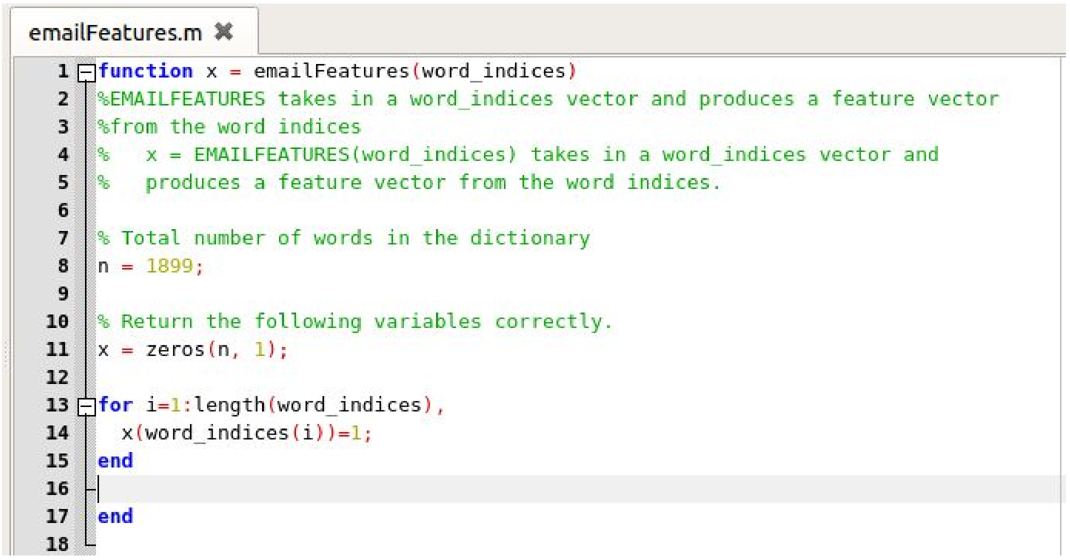
```

for idx=1:length(vocabList),
  if strcmp(str, vocabList{idx}),
    word_indices = [word_indices; idx]
  end
end
```

2.2 Extracting Features from Emails

Instruction:
You will now implement the feature extraction that converts each email into a vector in R n . For this exercise, you will be using n = # words in vocabulary list. Specifically, the feature x i ∈ {0, 1} for an email corresponds to whether the i-th word in the dictionary occurs in the email. That is, x i = 1 if the i-th word is in the email and x i = 0 if the i-th 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.

Implementation:



emailFeatures.m

```
function x = emailFeatures(word_indices)
%EMAILFEATURES takes in a word_indices vector and produces a feature vector
%from the word indices
%   x = EMAILFEATURES(word_indices) takes in a word_indices vector and
%   produces a feature vector from the word indices.

% Total number of words in the dictionary
n = 1899;

% Return the following variables correctly.
x = zeros(n, 1);

for i=1:length(word_indices),
  x(word_indices(i))=1;
end

end

```

2.3 Training SVM for Spam Classification

1. After you have completed the feature extraction functions, we load a preprocessed training dataset that will be used to train a SVM classifier.

2. **Dataset**
    - spamTrain.mat contains 4000 training examples of spam and non-spam email
    - spamTest.mat contains 1000 test examples

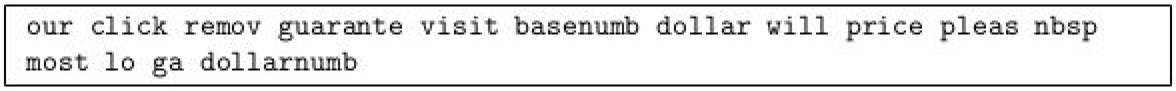
3. Each original email was processed using the processEmail and emailFeatures functions and converted into a vector x (i) ∈ R^{1899}

4. After loading the dataset, we train the SVM to classify between spam (y = 1) and non-spam (y = 0) 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%.

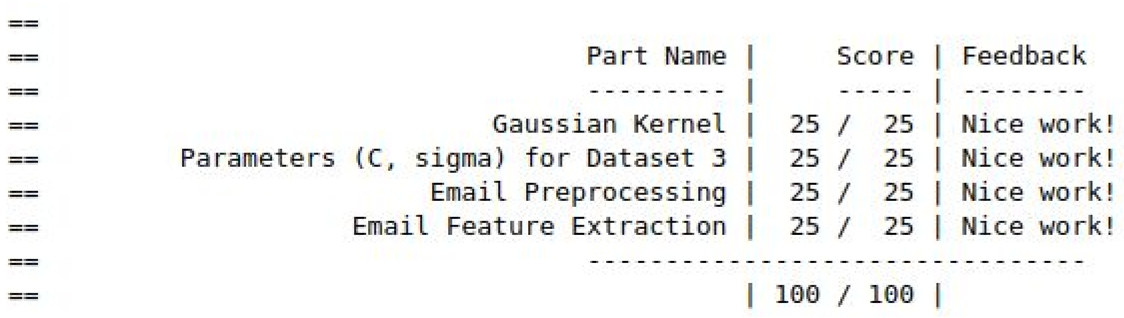
2.4 Top Predictors for Spam

Looking at the words which the classifier thinks are the most predictive of spam, we look at the largest positive values in the classifier. Seeing the result, we know that if an email contains words such as “guarantee”, “remove”, “dollar”, and “price” , it is likely to be classified as spam.



```
our click remov guarante visit basenumb dollar will price pleas nbsp
most lo ga dollarnumb
```

3 Scores



```
==
==                                 Part Name |     Score | Feedback
==                                 --------- |     ----- | --------
==                           Gaussian Kernel |  25 /  25 | Nice work!
==        Parameters (C, sigma) for Dataset 3 |  25 /  25 | Nice work!
==                       Email Preprocessing |  25 /  25 | Nice work!
==                  Email Feature Extraction |  25 /  25 | Nice work!
==                                 --------------------------------
==                                           | 100 / 100 |
```