Exercise 6 - Support Vector Machines

Azka NA

January 16, 2021

1 Introduction

This is the guide for Andrew Ng's Machine Learning course programming assignment done in Python, adapted from the original guide written for Octave or MATLAB.

In this exercise, you will be using support vector machines (SVMs) to build a spam classifier. Before starting on the programming exercise, we strongly recommend completing the review questions for the associated topics.

For Programming Exercise 6: Support Vector Machines, you will need to download the following files:

`exercise6.ipynb` - Jupyter notebook containing the script
`ex6data1.mat` - Example Dataset 1
`ex6data2.mat` - Example Dataset 2
`ex6data3.mat` - Example Dataset 3
`spamTrain.mat` - Spam training set
`spamTest.mat` - Spam test set
`emailSample1.txt` - Sample email 1
`emailSample2.txt` - Sample email 2
`spamSample1.txt` - Sample spam 1
`spamSample2.txt` - Sample spam 2
`vocab.txt` - Vocabulary list

---

2 Support Vector Machines

In the first half of this exercise, you will be using support vector machines (SVMs) with various example 2D datasets. Experimenting with these datasets will help you gain an intuition of how SVMs work and how to use a Gaussian kernel with SVMs. In the next half of the exercise, you will be using support vector machines to build a spam classifier.

2.1 Example Dataset 1

We will begin by with a 2D example dataset which can be separated by a linear boundary. You will plot the training data (Figure 1). In this dataset, the positions of the positive examples (indicated with +) and the negative examples (indicated wit o) suggest a natural separation indicated by the gap. However, notice that there is an outlier positive example + on the far left at about (0.1,4.1). As part of this exercise, you will also see how this outlier affects the SVM decision boundary.

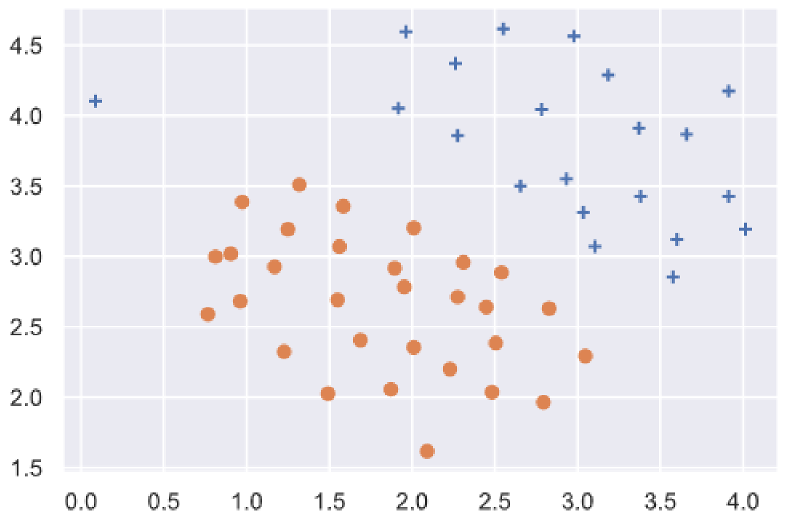

Figure 1: Example Dataset 1

In this part of the exercise, you will try using different values of the C parameter with SVMs. Informally, the C parameter is a positive value that controls the penalty for misclassified training examples. A large C parameter tells the SVM to try to classify all the examples correctly. C plays a role similar to $\frac{1}{\lambda}$, where λ is the regularization parameter that we were using previously for logistic regression.

```python
from sklearn.svm import SVC
svc_cls = SVC(kernel='linear', C=1)
svc_cls.fit(X, y)
svc_cls.score(X, y)

pos = y==1
neg = y==0
plt.scatter(X[pos[:,0],0], X[pos[:,0],1], marker='+')
plt.scatter(X[neg[:,0],0], X[neg[:,0],1], marker='o')

# plotting the decision boundary
X_1,X_2 = np.meshgrid(np.linspace(X[:,0].min(),X[:,1].max(),num=100),
    np.linspace(X[:,1].min(),X[:,1].max(),num=100))

```

```
plt.contour(X_1,X_2, svc_cls.predict(np.array([X_1.ravel(),X_2.ravel()
    ]).T).reshape(X_1.shape),1,colors='b')
```

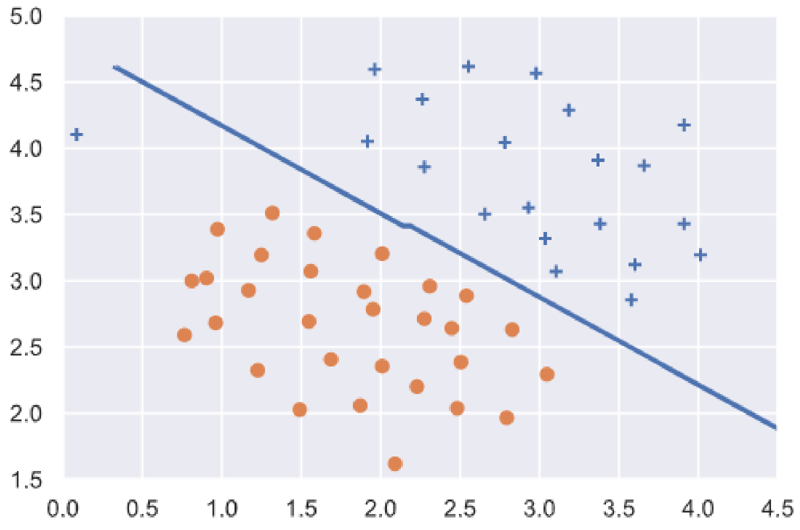

Figure 2: SVM Decision Boundary with $C = 1$ (Example Dataset 1)

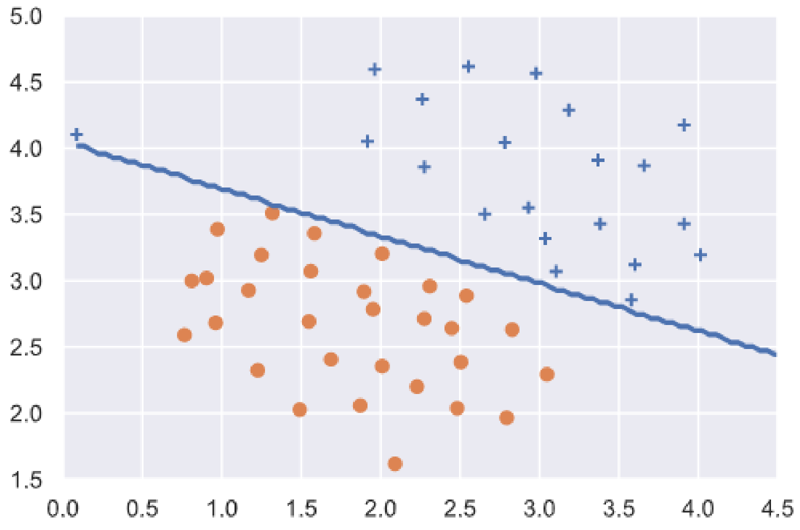

Figure 3: SVM Decision Boundary with $C = 100$ (Example Dataset 1)

The next part you will run the SVM training (with $C = 1$) using SVM module from ‘sklearn‘. When $C = 1$, you should find that the SVM puts the decision boundary in the gap between the two datasets and misclassifies the data point on the far left (Figure 2)

Implementation Note: Most SVM software packages automatically add the extra feature $x_0 = 1$ for you and automatically take care of learning the intercept term 0. So when passing your training data to the SVM software, there is no need to add this extra feature $x_0 = 1$ yourself.

Your task is to try different values of C on this dataset. Specifically, you should change the value of C in the script to $C = 100$ and run the SVM training again. When $C = 100$, you should find that the SVM now classifies every single example correctly, but has a decision boundary that does not appear to be a natural fit for the data (Figure 3).

2.2 SVM with Gaussian Kernels

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

2.2.1 Gaussian Kernel

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 for `gaussianKernel` to compute the Gaussian kernel between two examples, $(x^{(i)}, x^{(j)})$. The Gaussian kernel function is defined as:

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

Once you've completed the function `gaussianKernel`, tou will test your kernel function on two provided examples and you should expect to see a value of 0.324652.

```python
def gaussianKernel(x1, x2, sigma):
    atas = np.sum((x1 - x2)**2)
    bawah = 2*(sigma**2)
    return np.exp(-atas/bawah)

x1 = np.array([1.0, 2.0, 1.0])
x2 = np.array([0.0, 4.0, -1.0])
sigma = 2
gaussianKernel(x1, x2, sigma)
```

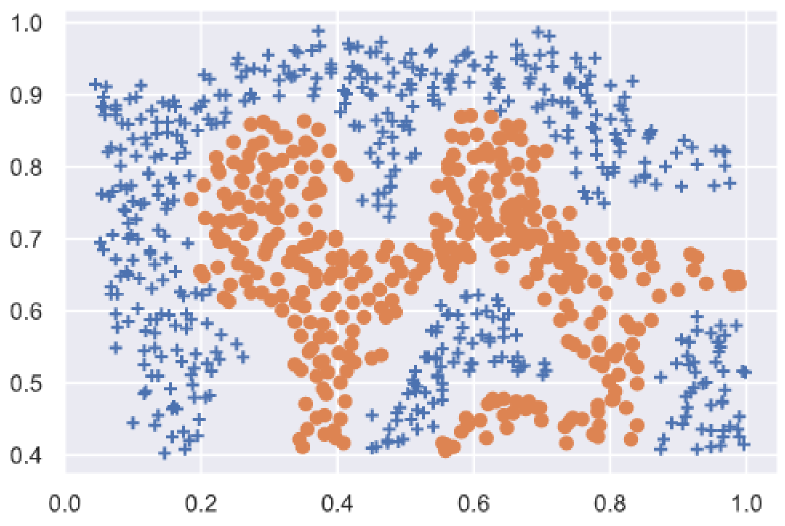

Figure 4: Example Dataset 2

2.2.2 Example Dataset 2

The next part you will load and plot dataset 2 (Figure4). 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.

```
svc_cls3 = SVC(kernel='rbf', gamma=30)
svc_cls3.fit(X2, y2)
svc_cls3.score(X2, y2)
```

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.

2.2.3 Example Dataset 3

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 you 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, `ex6data3.mat`, you are given the variables `X`, `y`, `Xval`, `yval`. The provided code in ex6.m trains the SVM classifier using the training set (`X`, `y`) using parameters loaded from `dataset3Params`.

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 . For both C and σ, we suggest trying values in

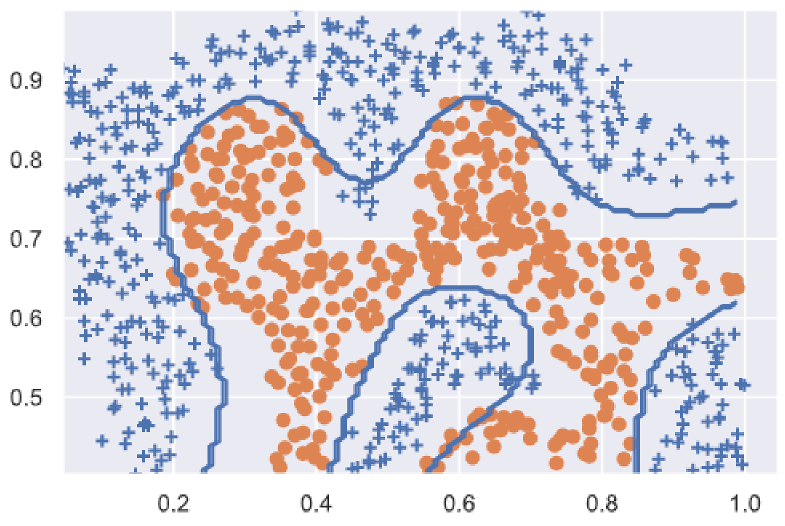

Figure 5: SVM (Gaussian Kernel) Decision Boundary (Example Dataset 2)

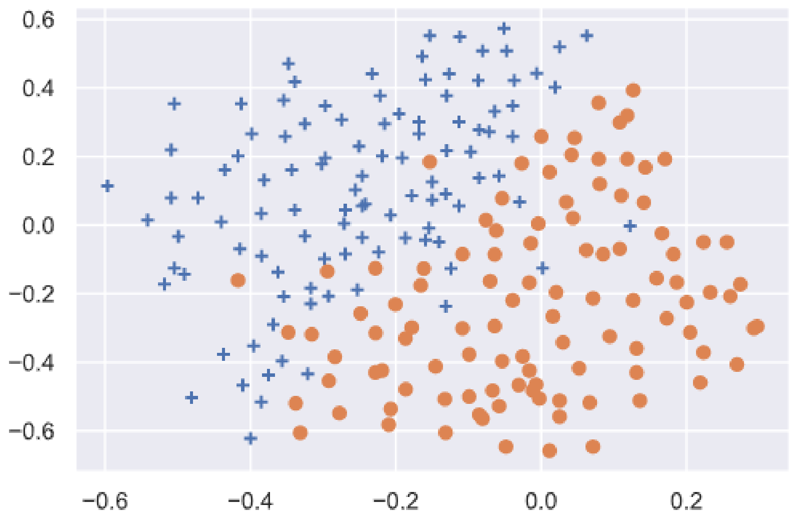

Figure 6: Example Dataset 3

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 $\sigma = 0.1$). 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.

After you have determined the best C and σ parameters to use, you should modify the code in `dataset3Params`, filling in the best parameters you found. For our best parameters, the SVM returned a decision boundary shown in Figure 7.

```
def dataset3Params(X, y, Xval, yval, vals):
```

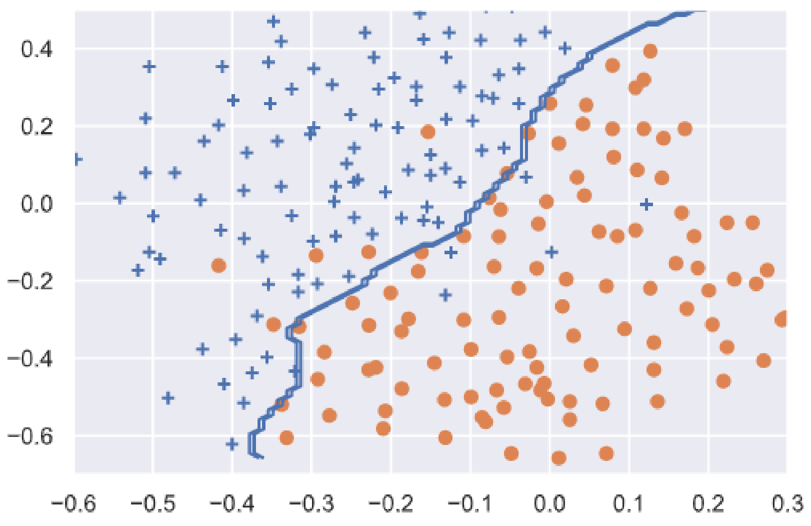

Figure 7: SVM (Gaussian Kernel) Decision Boundary (Example Dataset 3)

```python
    best_score = 0
    best_params = {'C':None, 'gamma':None}

    for i in vals:
        C = i
        for j in vals:
            gamma = 1/j
            svc_cls = SVC(C=C, gamma=gamma)
            svc_cls.fit(X,y)
            prediction = svc_cls.predict(Xval)
            score = svc_cls.score(Xval, yval)
            if score > best_score:
                best_score = score
                best_params['C'] = C
                best_params['gamma'] = gamma
    return best_params['C'], best_params['gamma']
```

---

3 Spam Classification

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 $x \in \mathbb{R}^n$. The following parts of the exercise will walk you through how such a feature vector can be constructed from an email.

The dataset included for this exercise is based on a subset of the SpamAssassin Public Corpus.[1] For the purpose of this exercise, you will only be using the body of the email (excluding the email headers).

3.1 Preprocessing Emails

```
 > 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
```

Figure 8: Sample 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`, we have implemented the following email preprocessing and normalization steps:

- **Lower-casing:** The entire email is converted into lower case, so that captialization 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.

[1]http://spamassassin.apache.org/old/publiccorpus/

- **Normalizing URLs:** All URLs are replaced with the text "`httpaddr`".
- **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", "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.

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.

```
 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
```

Figure 9: Preprocessed Sample Email

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

Figure 10: 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
```

Figure 11: Word Indices for Sample Email

3.1.1 Vocabulary List

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 emails, they might cause the model to overfit our training set. The complete vocabulary list is in the file `vocab.txt` and also shown in Figure 10. 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 11 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` 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 processEmail.m, the script `ex6_spam` will run your code on the email sample and you should see an output similar to Figures 9 11.

3.2 Extracting Features from Emails

You will now implement the feature extraction that converts each email into a vector in $\mathbb{R}^n$. For this exercise, you will be using n = number of words in vocabulary list. Specifically, the feature $x_i \in \{0, 1\}$ 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.

Thus, for a typical email, this feature would look like:

$$\mathbf{x} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n \qquad (2)$$

You should now complete the code in `emailFeatures` to generate a feature vector for an email, given the `word_indices`.

Once you have implemented `emailFeatures`, the next part of `ex6_spam` will run your code on the email sample. You should see that the feature vector had length 1899 and 45 non-zero entries.

3.3 Training SVM for Spam Classification

After you have completed the feature extraction functions, the next step of `ex6_spam` will load a preprocessed training dataset that will be used to train a SVM classifier. `spamTrain` contains 4000 training examples of spam and non-spam email, while `spamTest` contains 1000 test examples. Each original email was processed using the `processEmail` and `emailFeatures` functions and converted into a vector $x^{(i)} \in \mathbb{R}^{1899}$.

After loading the dataset, `ex6_spam` will proceed to train a 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%.

3.4 Top Predictors for 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

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

Figure 12: Top predictors for spam email

ex6 spam.m finds the parameters with the largest positive values in the classifier and displays the corresponding words (Figure 12). Thus, if an email contains words such as "guarantee", "remove", "dollar", and "price" (the top predictors shown in Figure 12), it is likely to be classified as spam.

3.5 Optional (ungraded) exercise: Try your own emails

Now that you have trained a spam classifier, you can start trying it out on your own emails. In the starter code, we have included two email examples (`emailSample1.txt` and `emailSample2.txt`) and two spam examples (`spamSample1.txt` and `spamSample2.txt`). The last part of ex6 spam.m runs the spam classifier over the first spam example and classifies it using the learned SVM. You should now try the other examples we have provided and see if the classifier gets them right. You can also try your own emails by replacing the examples (plain text files) with your own emails.

3.6 Optional (ungraded) exercise: Build your own dataset

In this exercise, we provided a preprocessed training set and test set. These datasets were created using `processEmail` and `emailFeatures` that you now have completed. For this optional (ungraded) exercise, you will build your own dataset using the original emails from the SpamAssassin Public Corpus.

Your task in this optional (ungraded) exercise is to download the original files from the public corpus and extract them. After extracting them, you should run the `processEmail`[2] and `emailFeatures` functions on each email to extract a feature vector from each email. This will allow you to build a dataset `X, y` of examples. You should then randomly divide up the dataset into a training set, a cross-validation set and a test set.

While you are building your own dataset, we also encourage you to try building your own vocabulary list (by selecting the high frequency words that occur in the dataset) and adding any additional features that you think might be useful.

[2]The original emails will have email headers that you might wish to leave out. We have included code in `processEmail` that will help you remove these headers.