Machine Learning	Assignment 4
Comp540 Spring 2016	**due 18 March 2016 at 8 pm**

This homework is due March 18 at 8 pm on Owlspace. The code base `hw4.zip` *for the assignment is an attachment to* Assignment 4 *on Owlspace. You will add your code at the indicated spots in the files there. Place your answers to Problems 1 and 2 (typeset) in a file called* `writeup.pdf` *and add it to the zip archive. Upload the entire archive back to Owlspace before the due date and time.*

1 Kernelizing the perceptron (10 points)

Consider a binary classification problem with $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid 1 \leq i \leq m, y^{(i)} \in \{-1, 1\}\}$. The perceptron uses hypotheses of the form $h_\theta(x) = sign(\theta^T x)$. In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameter vector θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for the perceptron algorithm for example $(x^{(i)}, y^{(i)})$ in the raw feature space is:

$$\theta^{(i)} \leftarrow \theta^{(i-1)} - [h_{\theta^{(i-1)}}(x^{(i)}) - y^{(i)}]x^{(i)}$$

where $\theta^{(i-1)}$ is the value of the parameter θ after the algorithm has seen the first $i-1$ training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to the all zeros vector. Note that when the prediction $h_{\theta^{(i-1)}}(x^{(i)})$ matches the label $y^{(i)}$, no change is made to θ.

Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. That is, $K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$. Suppose ϕ is so high-dimensional (say, ∞-dimensional) that it is infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the kernel trick to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(x)$.To handle the intercept term, you can think of ϕ as having the property that $\phi_0(x) = 1$.

- (2 points) How can you implicitly represent the high-dimensional parameter vector $\theta^{(i)}$? Note that θ is a vector whose dimension is the same as that of the feature vectors $\phi(x)$.
- (3 points) How can you efficiently make a prediction on a new input $x^{(i+1)}$ i.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = sign({\theta^{(i)}}^T x^{(i+1)})$, using your representation of $\theta^{(i)}$?
- (5 points) How will you modify the update rule corresponding to the basis function ϕ to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$?

$$\theta^{(i)} \leftarrow \theta^{(i-1)} + [y^{(i)} - h_{\theta^{(i-1)}}(\phi(x^{(i-1)}))]\phi(x^{(i)})$$

2 Fitting an SVM classifier by hand (15 points)

Consider a dataset with two points $\mathcal{D} = \{(0, -1), (\sqrt{2}, +1)\}$. We will map each of these points to three dimensions using the feature vector $\phi(x) = (1, \sqrt{2}x, x^2)$. Recall that the maximum margin classifier has the form:

$$min \ \frac{1}{2}||\theta||^2 \text{ such that}$$

$$\begin{aligned} y^{(1)}(\theta^T\phi(x^{(1)}) + \theta_0) &\geq 1 \\ y^{(2)}(\theta^T\phi(x^{(2)}) + \theta_0) &\geq 1 \end{aligned}$$

for the points $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$ in $\mathcal{D}$.

- (5 points) Write down a vector that is parallel to the optimal vector θ. Hint: θ is perpendicular to the decision boundary between the two points in the three-dimensional space.
- (3 points) What is the value of the margin that is achieved by this θ? Hint: the margin is the distance from each support vector to the decision boundary.
- (3 points) Solve for the θ given that the margin is equal to $\frac{1}{||\theta||}$.
- (3 points) Solve for the intercept θ_0 using your value for θ and the inequalities above. Since both points are support vectors, the inequalities will be tight.
- (3 points) Write down the equation for the decision boundary in terms of θ, θ_0 and x.

3 Support vector machines for binary classification (45 points)

In this exercise, you will be building support vector machines for binary and multi-class classification problems. To get started, please download the code base `pa4.zip` from Owlspace. When you unzip the archive, you will see the following files.

Name	Edit?	Read?	Description
ex4.py	Yes	Yes	Python script that will run your functions for this homework
ex4_spam.py	Yes	Yes	Python script that will run your functions for SVM spam classification
linear_svm.py	Yes	Yes	loss functions for the SVM
linear_classifier.py	No	Yes	SVM training and prediction functions
utils.py	Yes	Yes	plot utility functions, kernels
data/ex4data1.mat	No	No	Example dataset 1
data/ex4data2.mat	No	No	Example dataset 2
data/ex4data3.mat	No	No	Example dataset 3
data/spamTrain.mat	No	No	Spam training set
data/spamTest.mat	No	No	Spam test set
data/vocab.txt	No	Yes	vocabulary list
hw4.pdf	No	Yes	this document

3.1: Support vector machines

In the first part of this exercise, you will build support vector machines (SVMs) for solving binary classification problems. You will experiment with your classifier on three example 2D datasets. Experimenting with these datasets will help you gain intuition into how SVMs work and how to use a Gaussian kernel with SVMs. The provided script `ex4.py`, will help you step through these parts of the exercise. In the final part of the exercise, you will be using your SVM implementation to build a spam classifier. You will complete the script `ex4_spam.py` and use your SVM classifier to classify a given spam data set.

3.1A: The hinge loss function and gradient (10 points)

Now you will implement the hinge loss cost function and its gradient for support vector machines. Complete the `svm_loss_twoclass` function in `linear_svm.py` to return the cost and gradient for the hinge loss function. Recall that the hinge loss function is

$$J(\theta) = \frac{1}{2m}\sum_{j=0}^{d} {\theta_j}^2 + \frac{C}{m}\sum_{i=1}^{m} max(0, 1 - y^{(i)}h_\theta(x^{(i)}))$$

where $h_\theta(x) = \theta^T x$ with $x_0 = 1$. C is the penalty factor which measures how much misclassifications are penalized. If $y^{(i)}h_\theta(x^{(i)})) > 1$, then $x^{(i)}$ is correctly classified and the loss associated with that example is zero. If $y^{(i)}h_\theta(x^{(i)})) < 1$, then $x^{(i)}$ is not within the appropriate margin (positive or negative) and the loss associated with that example is greater than zero. The gradient of the hinge loss function is a vector of the same length as θ where the j^{th} element, $j = 0, 1, \ldots, d$ is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \begin{cases} \frac{1}{m}\theta_j + \frac{C}{m}\sum_{i=1}^{m} -y^{(i)}x^{(i)} & \text{if } y^{(i)}h_\theta(x^{(i)})) < 1 \\ \frac{1}{m}\theta_j & \text{if } y^{(i)}h_\theta(x^{(i)})) >= 1 \end{cases}$$

Once you are done, `ex4.py` will call your `svm_loss_twoclass` vector with a zero vector θ. You should see that the cost J is 1.0. The gradient of the loss function with respect to an all-zeros θ vector is also computed and should be $[-1.19917069 - 1.56709412]^T$.

3.1B Example dataset 1: impact of varying C (2 points)

We will begin with a 2D example dataset which can be separated by a linear boundary. The script `ex4.py` will plot the training data (Figure 1). In this dataset, the positions of the positive examples (green circles) and the negative examples (indicated with red circles) 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.

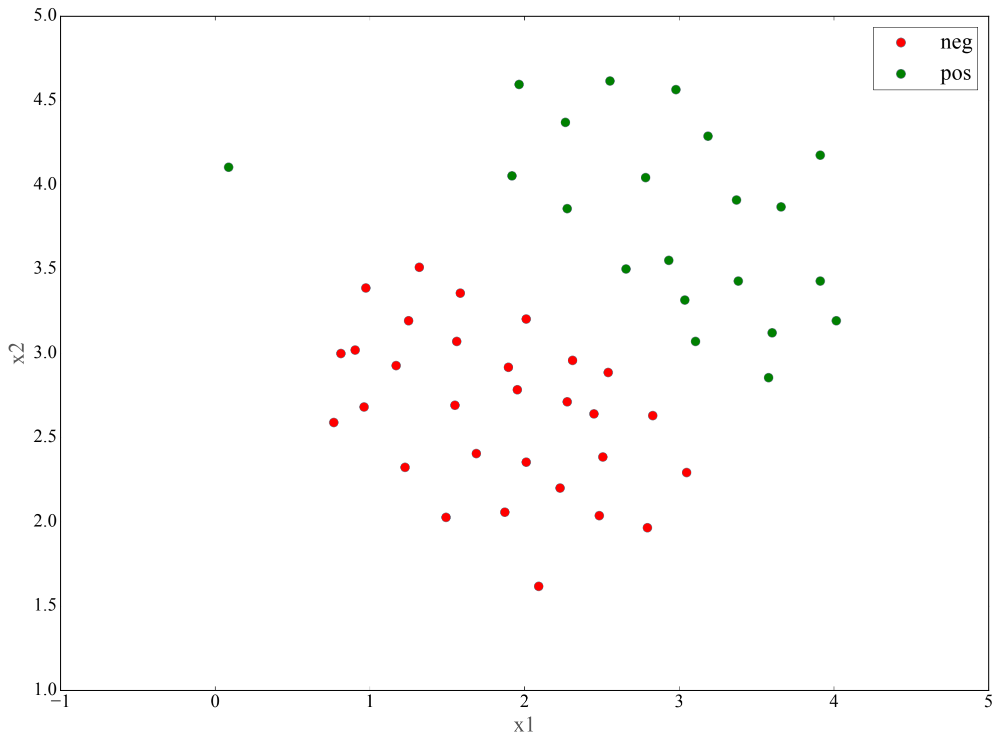

Figure 1: Example dataset 1

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.

The SVM training function is in `linear_classifier.py` – this is a gradient descent algorithm that uses your loss and gradient functions. The next part in `ex4.py` will train an SVM on the example data set 1 with C = 1. It first scales the data to have zero mean and unit variance, and adds the intercept term to the data matrix. 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).

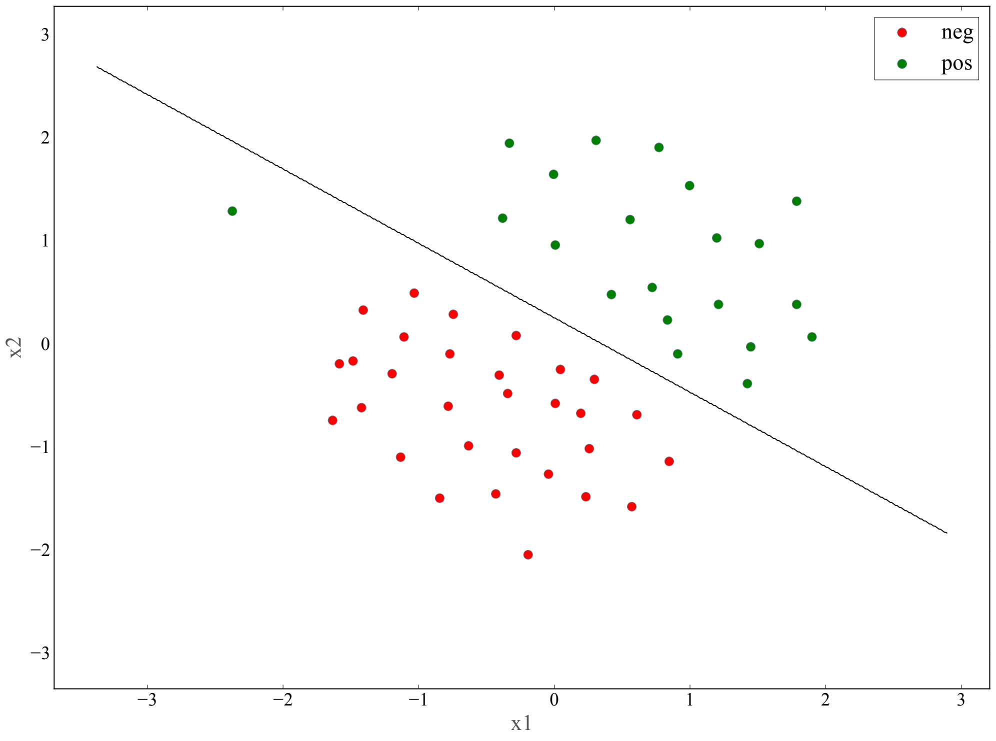

Figure 2: SVM decision boundary with C = 1 for Example dataset 1

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).

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.

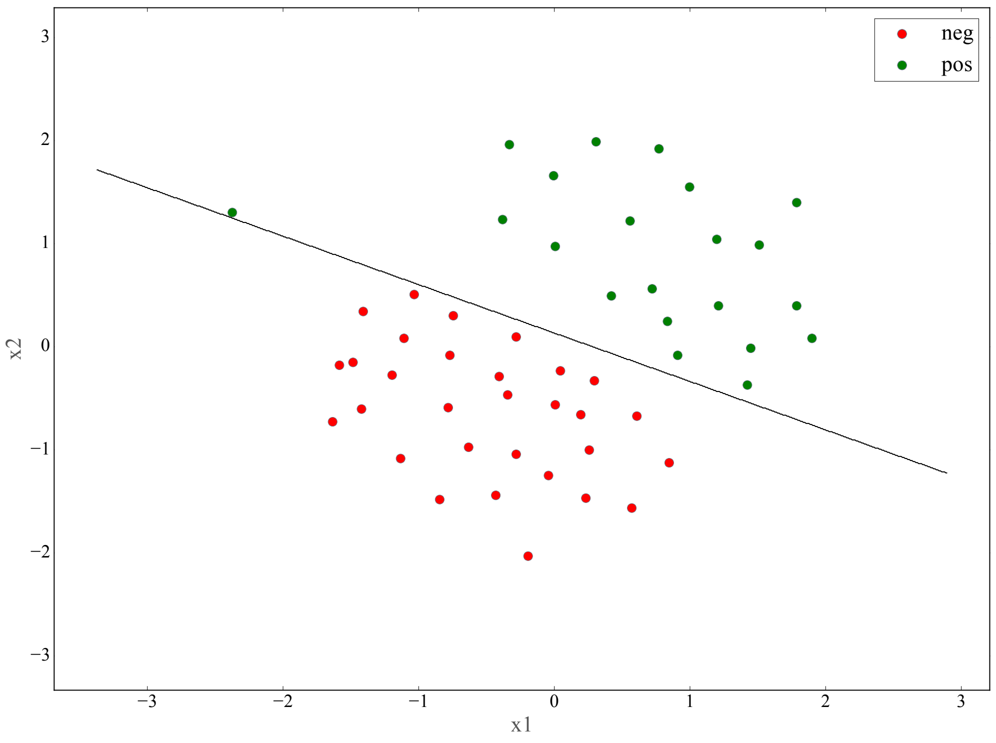

Figure 3: SVM decision boundary with C = 100 for Example dataset 1

3.1C Gaussian kernel (3 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 function `gaussian_kernel` in `utils.py` to compute the Gaussian kernel between two examples. The Gaussian kernel function is defined as:

$$k(x^{(i)}, x^{(j)}) = exp\left(-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}\right)$$

When you have completed the function, the script `ex4.py` will test your kernel function on two provided examples and you should expect to see a value of 0.324652.

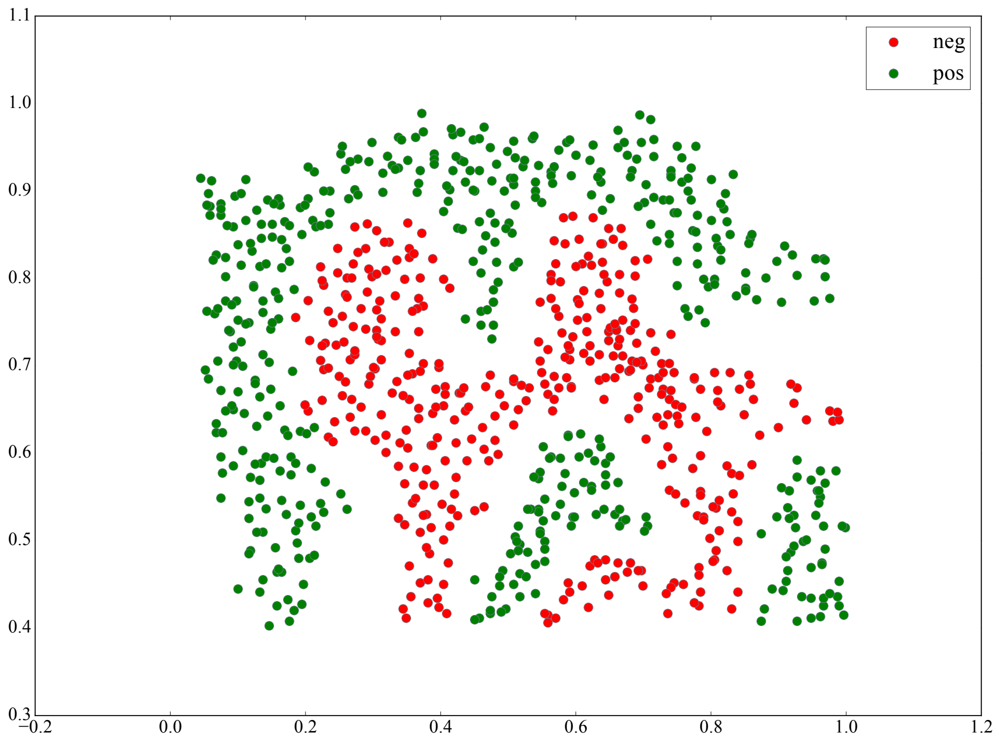

Figure 4: Example dataset 2

Example dataset 2: learning non-linear boundaries

The next part in `ex4.py` 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, `ex4.py` will proceed to train the SVM with the Gaussian kernel on this dataset.

Figure 5 shows the decision boundary found by the SVM with C = 1 and a Gaussian kernel with $\sigma = 0.01$. The decision boundary is able to separate most of the positive and negative examples correctly and follows the contours of the dataset well.

3.2 Example dataset 3: selecting hyper parameters for SVMs (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 `ex4.py` will load and display a third dataset (Figure 6). In the provided dataset, `ex4data3.mat`, you are given the variables `X`, `y`, `Xval`, `yval`. You will be using the SVM with the Gaussian kernel with this dataset. Your task is to use the validation set `Xval, yval` to determine the best C and σ parameter to use. You should

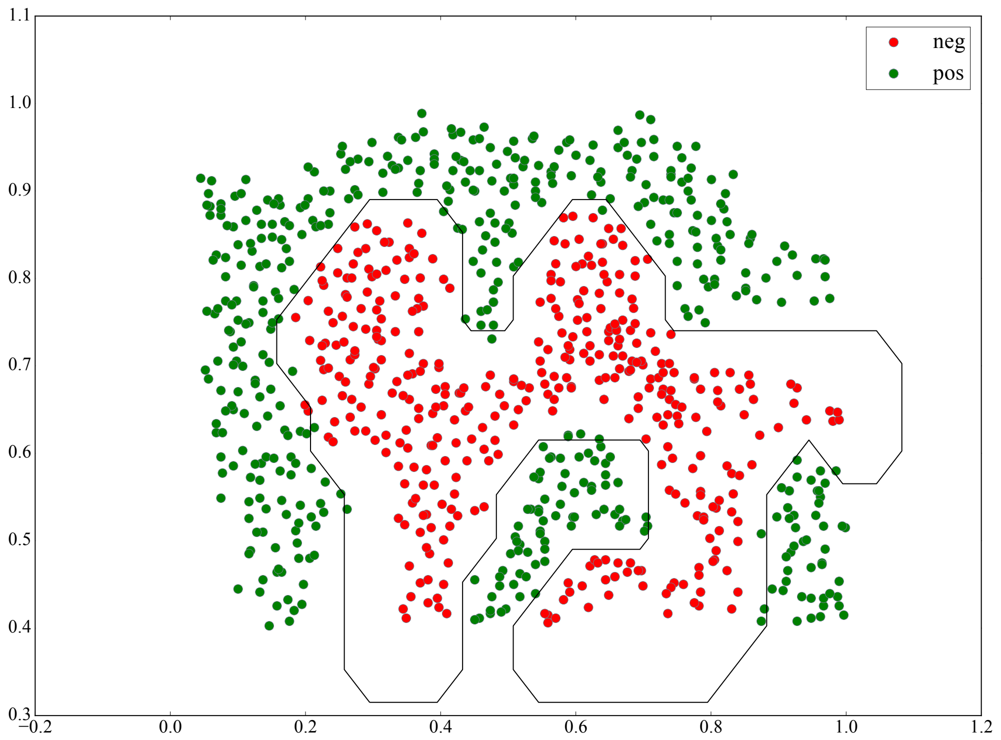

Figure 5: SVM gaussian kernel decision boundary for Example dataset 2, C = 1 and σ = 0.02

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 $8^2 = 64$ different models.

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 `predict` method of the SVM classifier to generate the predictions for the validation set.

After you have determined the best C and σ parameters to use, you should comment out the assignments to `best_C` and `best_sigma` in `ex4.py`. For our best parameters, the SVM returned a decision boundary shown in Figure 7.

3.3 Spam Classification with SVMs (20 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

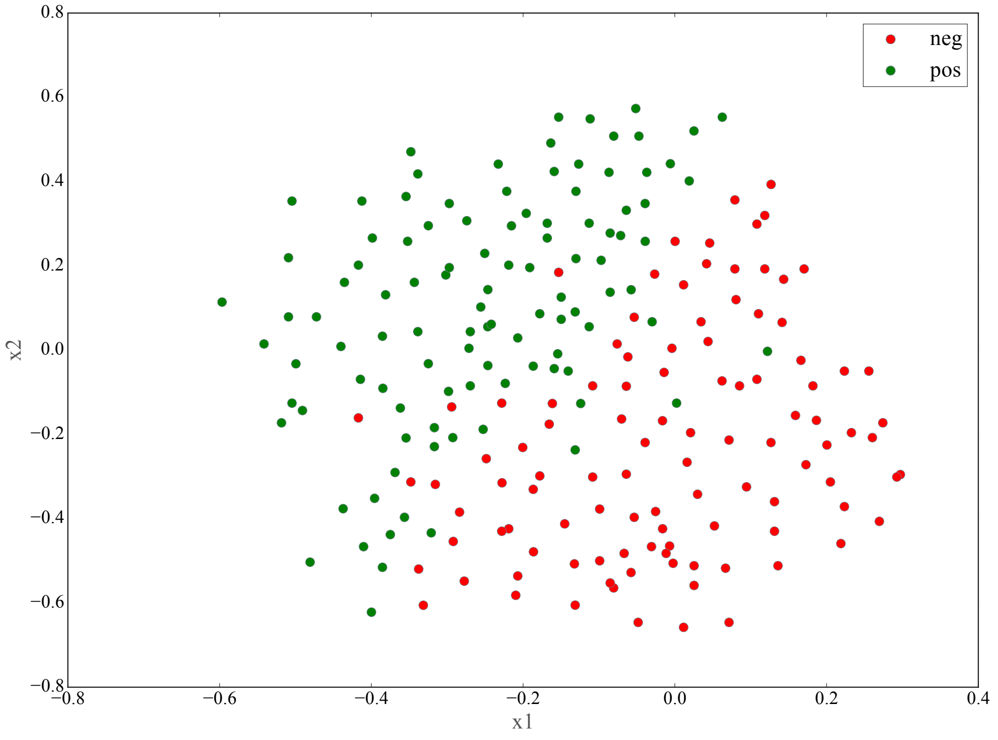

Figure 6: Example dataset 3

to build your own spam filter. You will be training a classifier to classify whether a given email, x, is spam or non-spam. Throughout the rest of this exercise, you will be using the the script `ex4_spam.py`. 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).

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

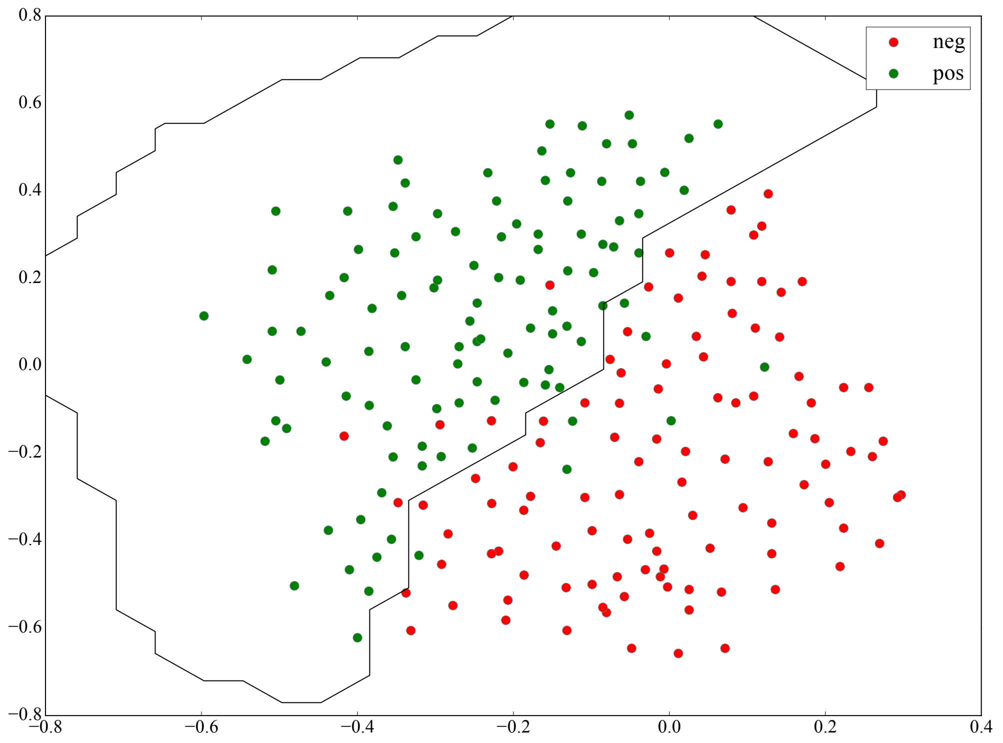

Figure 7: SVM gaussian kernel decision boundary for Example dataset 3 with the best hyper parameters.

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.

Here are typical 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`.
- **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`.

```
> 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

```
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: Sample email - preprocessed

- **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.

Feature representation

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 occurring 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`. 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.

```
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. This email will be represented by a feature vector of length 1899 where the values at indices in this list are set to 1 and the rest are 0.

Given the vocabulary list, we then 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.

Training SVMs for spam classification (20 points)

`ex4_spam.py` will load a training dataset `spamTrain.mat` which has been pre-processed according to the scheme shown above. `spamTrain.mat` contains 4000 training examples of spam and non-spam email, while `spamTest.mat` contains 1000 test examples. That is, each original email was processed and converted into a vector $x \in \Re^{1899}$. Your task is to design a protocol for training an SVM classifier for this data set. Issues you need to consider are: should you scale the data matrix and if so how, how do you set the learning rate, the number of iterations and the penalty parameter C? What kind of kernel is best for this problem, and how do you select the corresponding kernel hyper parameters? Use the SVM with the loss function `svm_loss_twoclass` and the training algorithm in `linear_classifier.py`. In `writeup.pdf` explain how you chose the parameters for training the SVM, providing graphs and tables to support your choices. Give us your best values for all of the chosen parameters and hyper parameters. Make sure that you do not use the test set to choose these parameters. Finally, evaluate the accuracy of your model on the test set and report it.

Our best classifier gets a training accuracy of about 99.8% and a test accuracy of about 98.5%.