Lab#8: Support Vector Machines

Prepared by: Teeradaj Racharak (r.teeradaj@gmail.com)

In this exercise, we will practice Support Vector Machines (SVMs) using scikit-learn only. Noted that SVM was marked 'deprecated' in TensorFlow (*cf.* https://github.com/tensorflow/tensorflow/blob/r1.12/tensorflow/contrib/learn/README.md); therefore, we don't need to practice using it.

2D Example Dataset 1[1]

We will begin with a 2D dataset, which can be naturally separated by a linear boundary. To convince ourselves, let us plot the training data from a given dataset ('lab8data1.mat'). Figure 1 shows the plot. In this dataset, the positions of the positive examples (indicated by '+') and the negative examples (indicated by '-') suggest an obvious separation indicated by the gap.

It's worth noticing 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 the outlier affects the SVM decision boundary. Furthermore, you will try to use different values of parameter C to control the penalty of misclassified training examples. Recall that the large value of C tells the SVM to try to classify all examples correctly. C plays a role similar to ($1/\lambda$), where λ is the regularization parameter in logistic regression.

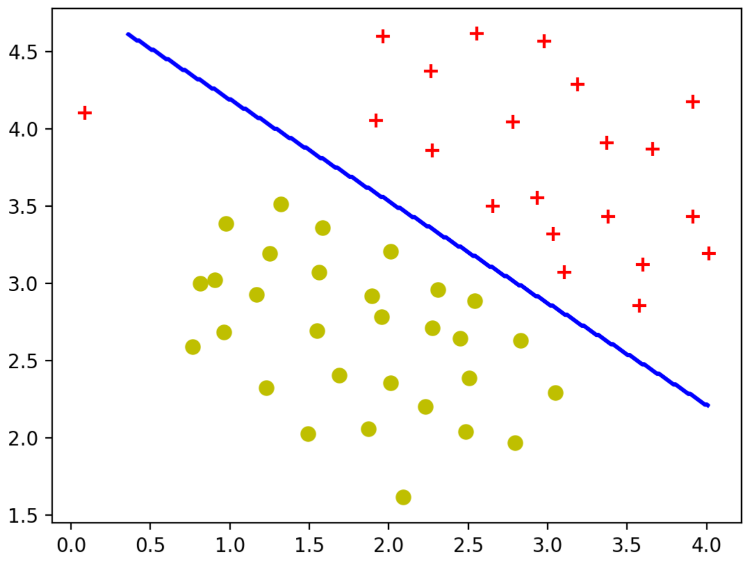


Figure 1 SVM decision boundary with $C = 1$

[1] Taken and revised from Andrew Ng's programming assignments on Coursera

Problem 1.1 [Python with scikit-learn]. Your first task is to try running SVM classifier provided by scikit-learn on this dataset. When $C = 1$, you should find that SVM puts the decision boundary in the gap between the two datasets and 'misclassifies' the data point on the far left (*cf.* Figure 1).

On the other hand, setting $C = 100$ will try to classify every example correctly. But, this classifier may give a decision boundary that does not look like a natural fit for the data (*cf.* Figure 2).

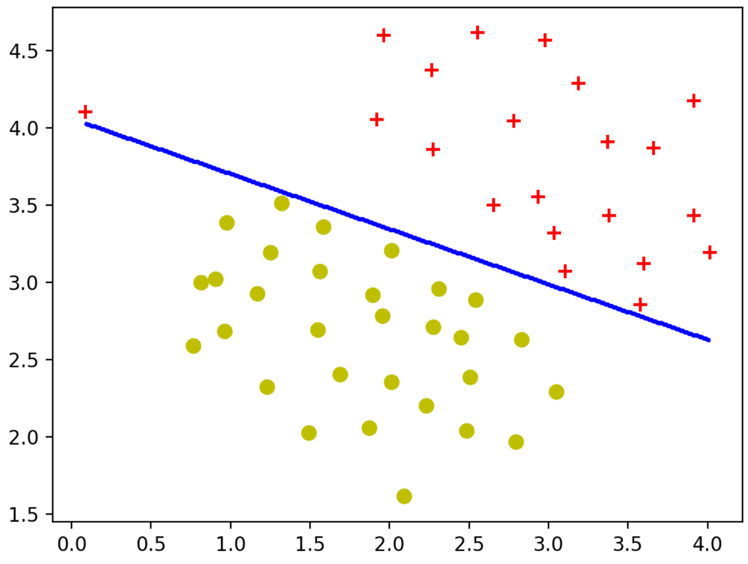


Figure 1 SVM decision boundary with $C = 100$

2D Example Dataset 2

Now, you will practice SVM to do non-linear classification. In particular, you will use SVM with Gaussian kernels on datasets that are not linearly separable.

To find non-linear decision boundaries, ones may use the Gaussian kernel. That is, a Gaussian kernel is 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. This kernel function is defined as follows:

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

Problem 2.1 [Python with scikit-learn]. Write a Python script to load and plot the dataset 2. After that, convince yourself that there is no 'linear' decision boundary that separates the positive and negative examples of this dataset. Then, construct an SVM classifier to learn a non-linear decision boundary on the dataset.

Noted that there is a 'ready-to-use' Gaussian kernel in scikit-learn called 'radial basis function (RBF)' (*cf.* **https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html**). Read the document and try to figure out how we can use it to learn the boundary (hint: use 'gamma' in the RBF for σ).

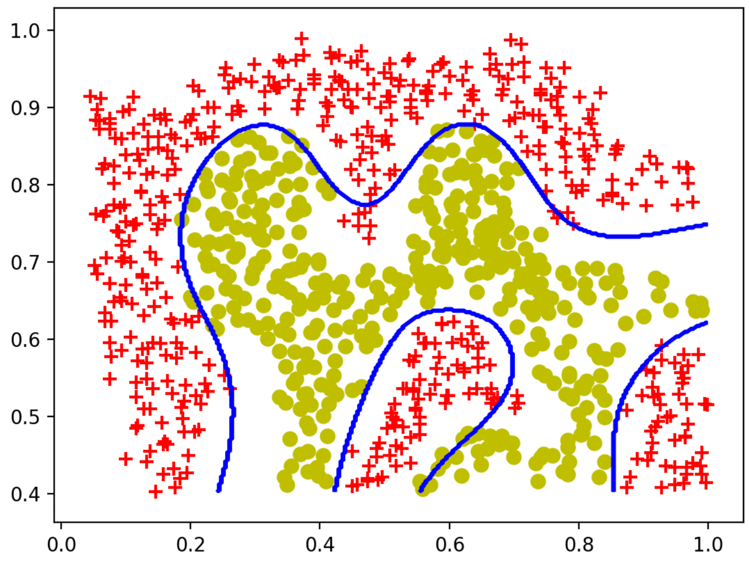


Figure 3 SVM Boundary using Gaussian Kernel with $\sigma = 3$

2D Example Dataset 3

Let's try another dataset (called 'dataset3') to practice a SVM with a Gaussian kernel.

Problem 3.1 [Python with scikit-learn]. As usual, the dataset provides us the information of X, y, Xval, and yval. Your task now is to use the cross validation set Xval, yval to determine the best C and σ to use. To select these values, we suggest trying value in multiplicative steps (*e.g.* 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30). You should try all possible pairs of values for C and σ (*i.e.* 64 pairs). After that, plot a decision boundary w.r.t. the best C and σ you have obtained. You should obtain a plot similar to Figure 4.

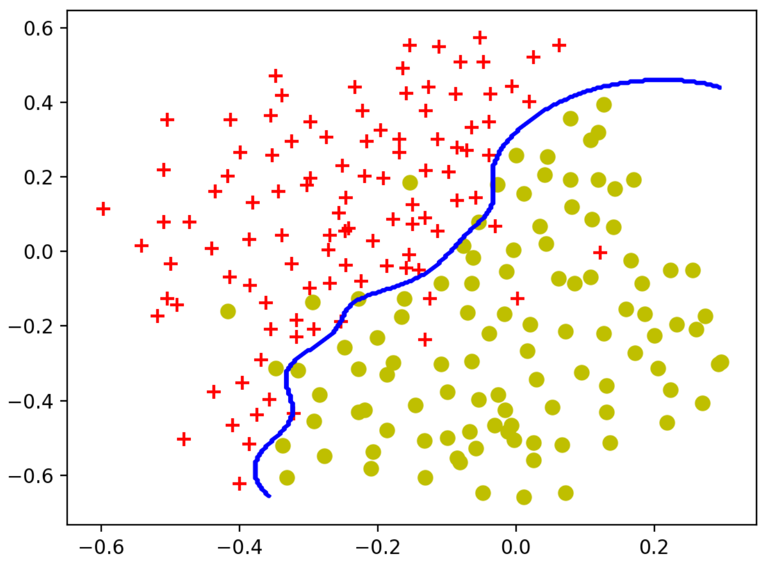


Figure 4 SVM Boundary using Gaussian Kernel the best C and σ

Spam Classification: Bags of Words

Now, let's try to build your own spam classifier using SVMs. The dataset used for this exercise is based on a subset of the SpamAssassin Public Corpus.[2] Noted that, to classify emails, we need to convert each email into a corresponding one-hot vector $x \in \mathbb{R}^n$ (as we have done in previous exercises).

> 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

Table 1 Sample email

Before starting on any machine learning task, it is a 'good practice' that we take a look into the datasets and get some insights. Table 1 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

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

amount) will be different in almost every email. Therefore, a 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.*. This has the effect of letting the spam classifier makes a classification problem based on any URLs since there are a few chances of seeing particular URLs as spammed emails.

In the function preprocessEmail, we have implemented the following normalizing steps.

- Lower-casing: The entire email is converted into lower case. So, IndIcaTE is treated the same as Indicate, for example.
- Stripping all special characters: All special characters are removed. 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'.
- Word stemming: Words are reduced to their stemmed form. For example, 'discount', 'discounts', 'discounted', and 'discounting' are replaced with 'discount'. In some implementations of Stemmer, they just strips off additional characters from the end *e.g.* ''include', 'includes', 'included', and 'including' are replaced with 'includ'.
- Removal of non-words: Non-words and punctuation have been removed. All whitespaces (*e.g.* tabs, newlines, spaces) have been trimmed to a 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

Table 2 Preprocessed sample email

Table 2 shows the result of these preprocessing step for the sample email provided in Table 1. The next step is to perform feature extraction and build a corresponding one-hot vector for a particular email.

In practice, we don't need to consider all words that appear in the preprocessed email. That is, we need to choose which words to be trained by our classifiers and which words to be left out. A common practice is to consider only on the most frequently occurring words as words that occur rarely in the training will only appear in a few emails; hence, cause the model to overfit.

In this exercise, we have chosen this bags of words for you (see ‘vocab.txt’). Our vocabulary list was selected by choosing all words which occur at least 100 times in the spam corpus, resulting in a list of 1,899 words. In practice, a vocabulary list containing 10,000 - 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
```

Table 3 Word indices for sample email

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. Table 3 shows the mapping for the sample email. For example, the word ‘anyone’ was first normalized to ‘anyon’ and then mapped onto the index 86.

Problem 4.1 [Python with scikit-learn]. Your task now is to complete the code in function preprocessEmail to perform this mapping. You should look up the word in the vocabulary and find if that word exists in the vocabulary list. If it exists, then you add its index into array word_indices. Otherwise, just ignore that word.

Once you have completed the code, calling the function preprocessEmai should output similar to Table 3.

Building One-hot from Emails

Now, we are ready to implement a method to convert an email to a one-hot vector in $\mathbb{R}^n$, where n is the number of words in vocabulary list. In particular, 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; otherwise, $x_i = 0$.

Problem 5.1 [Python with scikit-learn]. Your task now is to complete the code in function emailFeatures to generate a one-hot vector for an email. Once you have completed the code, you should see that, for the email sample, the length of the one-hot vector is 1,899 and the number of non-zero entries is 43.

Training SVM for Spam Classification

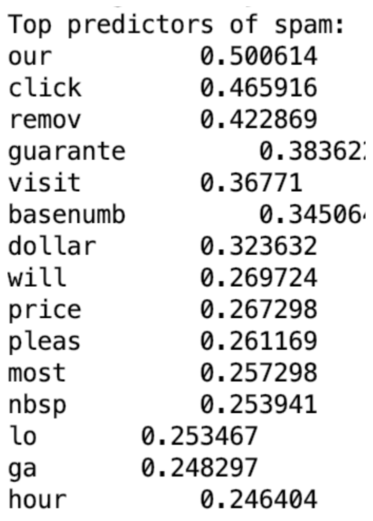
We have already preprocessed the 4,000 training and 1,000 test examples for you. They are kept in `spamTrain.mat` and `spamTest.mat`, respectively.

Problem 6.1 [Python with scikit-learn]. Write a program to load the datasets and train an SVM classifier. Once it is complete, you should see that the classifier gets a training accuracy around 99.825% and a test accuracy around 98.9%.

Top Predictors for Spam

To better understand how the spam classifier works, we should inspect the parameters and see which words the classifier thinks are the most predictive of spam.

Problem 7.1 [Python with scikit-learn]. Complete the program provided by the guide code to print out the top predictors of spam. Once you have completed the implementation, you should obtain a result similar to Figure 5.



```
Top predictors of spam:
our           0.500614
click         0.465916
remov         0.422869
guarante          0.38362
visit         0.36771
basenumb          0.34506
dollar        0.323632
will          0.269724
price         0.267298
pleas         0.261169
most          0.257298
nbsp          0.253941
lo        0.253467
ga        0.248297
hour          0.246404
```

Figure 5 Top predictors of spammed emails