Uczenie maszynowe 2018/2019

# Programming Exercise 7: Support Vector Machines

## Introduction

In this exercise, you will be using support vector machines (SVMs) to build a spam classifier. Before starting on this programming exercise, we strongly recommend watching the video lectures and reading the lecture notes.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise.

## Files included in this exercise

- [*] `svm_spam.py` – The main script for spam classification
- [*] `get_vocabulary_dict.py` – Load vocabulary list
- [*] `process_email.py` – Email preprocessing
- [*] `email_features.py` – Feature extraction from emails
- `data/spamTrain_X.csv` – Spam training set
- `data/spamTrain_y.csv` – Spam training set
- `data/spamTest_X.csv` – Spam test set
- `data/spamTest_y.csv` – Spam test set
- `data/emailSample1.txt` – Sample email 1
- `data/emailSample2.txt` – Sample email 2
- `data/spamSample1.txt` – Sample spam 1
- `data/spamSample2.txt` – Sample spam 2
- `data/vocab.txt` – Vocabulary list

(* indicates files you will need to complete)

## 1 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.

Throughout the rest of this exercise, you will be using the the script `svm_spam.py`. The dataset included for this exercise is based on a 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).

### 1.1 Preprocessing Emails

Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset. Figure 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 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.

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

```
> 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 1.** Sample Email

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 `process_email.py`, implement the following email preprocessing and normalization steps where necessary (some of the steps were already implemented):

- **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.
- **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 2. 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 2.** Preprocessed Sample Email

### 1.1.1 Vocabulary List

After preprocessing the emails, we have a list of words (e.g., Figure 2) 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

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

**(a)** 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
```

**(b)** Word Indices for Sample Email

**Figure 3.** Word Processing using a Vocabulary List

emails, they might cause the model to overfit our training set. The complete vocabulary list is in the file `data/vocab.txt` and also shown in Figure 3a. 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. Complete the `get_vocabulary_dict()` function in `get_vocabulary_dict.py` which loads the vocabulary list.

Given the vocabulary list, we can now map each word in the preprocessed emails (e.g., Figure 2) into a list of word indices that contains the index of the word in the vocabulary list. Figure 3b 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 `process_email.py` 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 `process_email.py`, the script `svm_spam.m` will run your code on the email sample and you should see an output similar to Figures 2 & 3.

### 1.2 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 $n =$ # 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:

$$x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n$$

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

Once you have implemented `email_features.py`, the next part of `svm_spam.py` will run your code on the email sample. You should see that the feature vector had length 1899 and 44 non-zero entries.

### 1.3 Training SVM for Spam Classification

After you have completed the feature extraction functions, the next step of `svm_spam.py` will load a preprocessed training dataset that will be used to train a SVM classifier. `data/spamTrain_X.csv` and `data/spamTrain_y.csv` contain 4000 training examples (feature vectors and labeling) of spam and non-spam email, while `data/spamTest_X.csv` and `data/spamTest_y.csv` contain 1000 test examples. Each original email was processed using the `process_email` and `email_features` functions and converted into a vector $x^{(i)} \in \mathbb{R}^{1899}$. After loading the dataset, `svm_spam.py` 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 $98.5\,\%$.

### 1.4 Top Predictors for Spam

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

**Figure 4.** Top predictors for spam email

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 `svm_spam.py` finds the parameters with the largest positive values in the classifier and displays the corresponding words (Figure 4). Thus, if an email contains words such as "guarantee", "remove", "dollar", and "price" (the top predictors shown in Figure 4), it is likely to be classified as spam.