Final Project: Spam Classifier

Many email services today provide spam filters that are able to classify emails into spam and non-spam email with high accuracy. In this project, you will use SVMs (convex optimizer) to build your own spam filter. You will be training a classier 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 $x \in R^n$, the input of your classifier. The following procedure gives you one example of how such a feature vector can be constructed from an email and how to build up a classifier. Other approaches are possible. In this project, you are encouraged to explore other approaches.

A. Build your own data set

Your can download the original sample (spam and non-spam) emails from the public corpus,

$$http://spamassassin.apache.org/old/publiccorpus/.$$

For later use, you need to convert all file extensions to .txt.

B. Preprocessing emails

Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset. Typically, an email may contain a URL, an email address, numbers, and dollar amounts, etc. 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. You can follow the following email preprocessing and normalization steps:

1) Lower-casing: The entire email is converted into lower case, so that captialization is ignored (e.g., IndIcaTE is treated the same as Indicate).
2) 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.
3) Normalizing URLs: All URLs are replaced with the text "httpaddr"
4) Normalizing Email Addresses: All email addresses are replaced with the text "emailaddr".
5) Normalizing Numbers: All numbers are replaced with the text "number".
6) Normalizing Dollars: All dollar signs are replaced with the text "dollar".
7) 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".
8) 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.

For email preprocessing, you can use and make changes (if needed) on the sample Matlab code given in "processEmail.m".

After preprocessing the emails, we have a list of words 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". 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 into a list of word indices that contains the index of the word in the vocabulary list. For example, in an email, the word "anyone" can be first normalized to "anyon" and then mapped onto the index 86 in the vocabulary list. 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.

C. Extracting Features from Emails

You will now implement the feature extraction that converts each email into a vector in R^n. 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. For the given vocabulary list in "vocab.txt", each email will induce one vector $X \in R^{1899}$ where each entry is 0 or 1.

Task 1: create your own data set (X, y). Some Matlab files are attached to help process the original emails, feel free to use them if needed.

D. Train SVM for Spam Classification

After you have completed the feature extraction functions, the next step is to train a SVM classifier. First of all, you should divide up the dataset into a training set and a test set. You first need to select a kernel for your SVM, e.g. linear, polynomial, Gaussian, quadratic, etc. Then use the training dataset X, y to train your SVM. You are free to choose your own training algorithm. In particular, you need to formulate a SVM model with an objective function of interest, then implement an algorithm/tool, e.g. CVX, Sequential minimal optimization (SMO), or others, to solve your SVM. Here, you may need to do some research study and program your own algorithm in Matlab. DO NOT use the off-the-shelf toolbox available on-line. Once you have the SVM classifier, the next step is to test your SVM with the test dataset. What is the failure or error rate of your classifier? Is your training accuracy the same with or close to the test accuracy? If both accuracies are larger than 95%, your SVM performs well. Otherwise, you need to change your objective function, tune the parameters in your kernel or select another kernel.

Task 2: Model you SVM mathematically. Program the algorithm to train and test SVM by using YOUR OWN dataset. PS: there are two datasets (spamTrain and spamTest) attached to

help you design and test your SVM model and algorithms.

E. Submission

You need to submit a report, your dataset and Matlab files to CourseWorks. Your report should written in the style of an academic paper, which includes all necessary details for review, e.g. before-and-after of preprocessing emails, SVM mathematical model, and algorithms for training your SVM classifier. An instruction to run your submitted codes should be included in the report as well. Credits will be given to each aforementioned step. If you prefer to use other programming language (e.g. python, C++), this will be fine for this project.