Introduction to Machine Learning, Fall 2012

Problem Set 1: Perceptron algorithm & Support vector machines

Due: Tuesday, September 25, 2012 at 11am (*before* class begins)

Important: *See problem set policy on the course web site.*

---

1. (5 points) Consider a (hard margin) support vector machine and the following training data from two classes:

$$\begin{array}{lllll} +1: & (2,2) & (4,4) & (4,0) \\ -1: & (0,0) & (2,0) & (0,2) \end{array}$$

   (a) Plot these six training points, and construct by inspection the weight vector for the optimal hyperplane. In your solution, specify the hyperplane in terms of $\vec{w}$ and b such that $w_1 x_1 + w_2 x_2 + b = 0$. Calculate what the margin is (i.e., 2γ, where γ is the distance from the hyperplane to its closest data point), showing all of your work.

   (b) What are the support vectors? Explain why.

2. (5 points) Show that, irrespective of the dimensionality of the data space, a data set consisting of just two data points (call them $\vec{x}_1$ and $\vec{x}_2$), one from each class, is sufficient to determine the maximum-margin hyperplane. Fully explain your answer, including giving an explicit formula for the maximum-margin hyperplane as a function of $\vec{x}_1$ and $\vec{x}_2$.

3. (5 points) In class we discussed a generalization of the SVM to multi-class classification, where we introduced parameters $\vec{w}^{(k)}$ and $b^{(k)}$ for each class $k = 1, \ldots, K$ (where K is the number of classes), and where prediction for a new data point $\vec{x}$ is done using

$$\hat{y} \leftarrow \arg\max_k \ \vec{w}^{(k)} \cdot \vec{x} + b^{(k)}$$

   For this problem, prove that this is equivalent to the binary prediction rule $\text{sign}(\vec{w} \cdot \vec{x} + b)$ in the case that $K = 2$. That is, given as input $\vec{w}^{(1)}$, $b^{(1)}$, $\vec{w}^{(2)}$ and $b^{(2)}$, demonstrate $\vec{w}$ and b that gives an equivalent decision rule. As always, you must show all of your work to obtain full credit.

4. (10 points) Kernels

   (a) For any two documents x and z, define $k(x, z)$ to equal the number of unique words that occur in both x and z (i.e., the size of the intersection of the sets of words in the two documents). Is this function a kernel? Justify your answer. (Hint: $k(x, z)$ is a kernel if there exists $\phi(x)$ such that $k(x, z) = \phi(x)^T \phi(z)$).

   (b) Assuming that $\vec{x} = [x_1, x_2], \vec{z} = [z_1, z_2]$ (i.e., both vectors are two-dimensional) and $\beta > 0$, show that the following is a kernel:

$$k_\beta(x, z) = (1 + \beta \vec{x} \cdot \vec{z})^2 - 1$$

   (c) One way to construct kernels is to build them from simpler ones. Assuming $k_1(x, z)$ and $k_2(x, z)$ are kernels, then one can show that so are these:

i. (scaling) $f(x)f(z)k_1(x,z)$ for any function $f(x) \in \mathcal{R}$,
ii. (sum) $k(x,z) = k_1(x,z) + k_2(x,z)$,
iii. (product) $k(x,z) = k_1(x,z)k_2(x,z)$.

Using the above rules and the fact that $k(x,z) = x^T z$ is a kernel, show that the following is also a kernel:

$$\left(1 + \left(\frac{x}{||x||_2}\right)^T \left(\frac{z}{||z||_2}\right)\right)^3.$$

5. (15 points) In this question you will implement the Perceptron algorithm and apply it to the problem of e-mail spam classification.

   Instructions. You may use the programming language of your choice. However, you are **not** permitted to use or reference any machine learning code or packages not written by yourself. In addition to your answers to the below questions, hand in a print out of all code that you write for this assignment.

   Data files. We have provided you with three files: `spamTrain.dat`, `spamTest.dat`, and `vocab.txt`. The data is comma deliminited. Each row of the data files corresponds to a single e-mail. The first column gives the label (1=spam, 0=not spam), whereas the subsequent columns give the pre-computed feature vector for each e-mail (described below).

   Pre-processing. The dataset included for this exercise is based on a subset of the SpamAssassin Public Corpus[1]. 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. 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.

[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 e-mail in SpamAssassin corpus before pre-processing.

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: Pre-processed version of the sample e-mail from Figure 1.

We have already implemented the following email preprocessing steps: lower-casing; removal of HTML tags; normalization of URLs, e-mail addresses, and numbers. In addition, words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Finally, we removed all non-words and punctuation. The result of these preprocessing steps is shown in Figure 2.

We ignore any word that occurs fewer than 100 times in the spam corpus, resulting in a vocabulary with 1899 words. Since words that occur rarely in the training set are only in a few emails, they might cause the learning algorithm to overfit our training set. The complete vocabulary list is in the file `vocab.txt` (the ith line corresponds to the ith feature/column in the data file (not counting the first column). The feature vector for an e-mail is given by the presence or absence of each of these 1899 words, and is thus of dimension 1899. That is, the feature $x_i \in \{0, 1\}$ corresponds to whether the ith word in the vocabulary occurs in the e-mail.

(a) Implement the functions `perceptron_train(data_file)` and `perceptron_test(w, data_file)`.

The function `perceptron_train(data_file)` trains a perceptron classifier using the examples in the provided data file, and should return $\vec{w}$ and k, the final classification vector and the number of updates (mistakes) performed, respectively. You may assume that the input data provided to your function is linearly separable (so the stopping criterion should be that all points are correctly classified).

For this exercise, you do not need to add a bias feature to the feature vector (it turns out not to improve classification accuracy, possibly because a frequently occuring word already serves this purpose). Your implementation should cycle through the data points in the order as given in the data file (rather than randomizing), so that results are consistent for grading purposes.

The function `perceptron_test(w, data_file)` should take as input the weight vector $\vec{w}$ (the classification vector to be used) and a data file containing the test examples and their labels. The function should return the test error, i.e. the fraction of test examples which were misclassified by $\vec{w}$.

(b) Train the linear classifier using the provided training data (`spamTrain.dat`). How many mistakes are made before the algorithm terminates? Test your implementation of `perceptron_test` by running it with the learned parameters and the training data, making sure that the training error is zero. Next, classify the test e-mails (`spamTest.dat`). What is the test error?

(c) One should expect that the test error decreases as the amount of training data increases. Using only the first N rows of `spamTrain.dat` as training data, run the perceptron algorithm on this smaller training set and evaluate the corresponding test error (using all of the test data). Do this for $N = 100, 200, 400, 800, 1600$, and 3200, and create a plot of the corresponding test errors as a function of N.

(d) 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. Using `vocab.txt` together with the parameters learned using all of the training data, output the 10 words with the *most positive* weights. What are they? Which 10 words have the most *negative* weights?

(e) Implement the *averaged* perceptron algorithm, which is the same as your current implementation but which, rather than returning the final weight vector, returns the average of all weight vectors considered during the algorithm (including examples where no mistake was made). Averaging reduces the variance between the different vectors, and helps prevent the learning algorithm from overfitting (serving as a type of regularization). What is the test error of this approach when trained on all of `spamTrain.dat`?

Acknowledgement: This question is adapted from an assignment developed by Andrew Ng of Stanford University and Coursera.