

# CS 229, Autumn 2009

## Problem Set #2 Solutions: Naive Bayes, SVMs, and Theory

---

**Due in class (9:30am) on Wednesday, October 28.**

**Notes:** (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) When sending questions to [cs229-qa@stanford.edu](mailto:cs229-qa@stanford.edu), please make sure to write the homework number and the question number in the subject line, such as **Hwk1 Q4**, and send a separate email per question. (3) If you missed the first lecture or are unfamiliar with the class' collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figure that you are asked to plot.

**SCPD students:** Please fax your solutions to Prof. Ng at (650) 725-1449, and write "ATTN: CS229 (Machine Learning)" on the cover sheet. If you are writing your solutions out by hand, please write clearly and in a reasonably large font using a dark pen to improve legibility.

### 1. [15 points] Constructing kernels

In class, we saw that by choosing a kernel  $K(x, z) = \phi(x)^T \phi(z)$ , we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping  $\phi$  to a higher dimensional space, and then work out the corresponding  $K$ .

However in this question we are interested in direct construction of kernels. I.e., suppose we have a function  $K(x, z)$  that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging  $K$  into the SVM as the kernel function. However for  $K(x, z)$  to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping  $\phi$ . Mercer's theorem tells us that  $K(x, z)$  is a (Mercer) kernel if and only if for any finite set  $\{x^{(1)}, \dots, x^{(m)}\}$ , the matrix  $K$  is symmetric and positive semidefinite, where the square matrix  $K \in \mathbb{R}^{m \times m}$  is given by  $K_{ij} = K(x^{(i)}, x^{(j)})$ .

Now here comes the question: Let  $K_1, K_2$  be kernels over  $\mathbb{R}^n \times \mathbb{R}^n$ , let  $a \in \mathbb{R}^+$  be a positive real number, let  $f : \mathbb{R}^n \mapsto \mathbb{R}$  be a real-valued function, let  $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d$  be a function mapping from  $\mathbb{R}^n$  to  $\mathbb{R}^d$ , let  $K_3$  be a kernel over  $\mathbb{R}^d \times \mathbb{R}^d$ , and let  $p(x)$  a polynomial over  $x$  with *positive* coefficients.

For each of the functions  $K$  below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

- (a)  $K(x, z) = K_1(x, z) + K_2(x, z)$
- (b)  $K(x, z) = K_1(x, z) - K_2(x, z)$
- (c)  $K(x, z) = aK_1(x, z)$
- (d)  $K(x, z) = -aK_1(x, z)$
- (e)  $K(x, z) = K_1(x, z)K_2(x, z)$

- (f)  $K(x, z) = f(x)f(z)$   
 (g)  $K(x, z) = K_3(\phi(x), \phi(z))$   
 (h)  $K(x, z) = p(K_1(x, z))$

[Hint: For part (e), the answer is that the  $K$  there *is* indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

**Answer:** All 8 cases of proposed kernels  $K$  are trivially symmetric because  $K_1, K_2, K_3$  are symmetric; and because the product of 2 real numbers is commutative (for (1f)). Thanks to Mercer's theorem, it is sufficient to prove the corresponding properties for positive semidefinite matrices. To differentiate between matrix and kernel function, we'll use  $G_i$  to denote a kernel matrix (Gram matrix) corresponding to a kernel function  $K_i$ .

- (a) Kernel. The sum of 2 positive semidefinite matrices is a positive semidefinite matrix:  $\forall z z^T G_1 z \geq 0, z^T G_2 z \geq 0$  since  $K_1, K_2$  are kernels. This implies  $\forall z z^T G z = z^T G_1 z + z^T G_2 z \geq 0$ .
- (b) Not a kernel. Counterexample: let  $K_2 = 2K_1$  (we are using (1c) here to claim  $K_2$  is a kernel). Then we have  $\forall z z^T G z = z^T (G_1 - 2G_1) z = -z^T G_1 z \leq 0$ .
- (c) Kernel.  $\forall z z^T G_1 z \geq 0$ , which implies  $\forall z a z^T G_1 z \geq 0$ .
- (d) Not a kernel. Counterexample:  $a = 1$ . Then we have  $\forall z -z^T G_1 z \leq 0$ .
- (e) Kernel.  $K_1$  is a kernel, thus  $\exists \phi^{(1)} K_1(x, z) = \phi^{(1)}(x)^T \phi^{(1)}(z) = \sum_i \phi_i^{(1)}(x) \phi_i^{(1)}(z)$ . Similarly,  $K_2$  is a kernel, thus  $\exists \phi^{(2)} K_2(x, z) = \phi^{(2)}(x)^T \phi^{(2)}(z) = \sum_j \phi_j^{(2)}(x) \phi_j^{(2)}(z)$ .

$$K(x, z) = K_1(x, z)K_2(x, z) \quad (1)$$

$$= \sum_i \phi_i^{(1)}(x) \phi_i^{(1)}(z) \sum_i \phi_i^{(2)}(x) \phi_i^{(2)}(z) \quad (2)$$

$$= \sum_i \sum_j \phi_i^{(1)}(x) \phi_i^{(1)}(z) \phi_j^{(2)}(x) \phi_j^{(2)}(z) \quad (3)$$

$$= \sum_i \sum_j (\phi_i^{(1)}(x) \phi_j^{(2)}(x)) (\phi_i^{(1)}(z) \phi_j^{(2)}(z)) \quad (4)$$

$$= \sum_{(i,j)} \psi_{i,j}(x) \psi_{i,j}(z) \quad (5)$$

Where the last equality holds because that's how we define  $\psi$ . We see  $K$  can be written in the form  $K(x, z) = \psi(x)^T \psi(z)$  so it is a kernel.

- (f) Kernel. Just let  $\psi(x) = f(x)$ , and since  $f(x)$  is a scalar, we have  $K(x, z) = \phi(x)^T \phi(z)$  and we are done.
- (g) Kernel. Since  $K_3$  is a kernel, the matrix  $G_3$  obtained for *any* finite set  $\{x^{(1)}, \dots, x^{(m)}\}$  is positive semidefinite, and so it is also positive semidefinite for the sets  $\{\phi(x^{(1)}), \dots, \phi(x^{(m)})\}$ .
- (h) Kernel. By combining (1a) sum, (1c) scalar product, (1e) powers, (1f) constant term, we see that any polynomial of a kernel  $K_1$  will again be a kernel.

## 2. [15 points] Kernelizing the Perceptron

Let there be a binary classification problem with  $y \in \{0, 1\}$ . The perceptron uses hypotheses of the form  $h_\theta(x) = g(\theta^T x)$ , where  $g(z) = \mathbf{1}\{z \geq 0\}$ . In this problem we will consider

a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters  $\theta$  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 this version of the perceptron algorithm is given by

$$\theta^{(i+1)} := \theta^{(i)} + \alpha[y^{(i+1)} - h_{\theta^{(i)}}(x^{(i+1)})]x^{(i+1)}$$

where  $\theta^{(i)}$  is the value of the parameters after the algorithm has seen the first  $i$  training examples. Prior to seeing any training examples,  $\theta^{(0)}$  is initialized to  $\vec{0}$ .

Let  $K$  be a Mercer kernel corresponding to some very high-dimensional feature mapping  $\phi$ . Suppose  $\phi$  is so high-dimensional (say,  $\infty$ -dimensional) that it's 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  $\phi$ , but without ever explicitly computing  $\phi(x)$ . [Note: You don't have to worry about the intercept term. If you like, think of  $\phi$  as having the property that  $\phi_0(x) = 1$  so that this is taken care of.] Your description should specify

- How you will (implicitly) represent the high-dimensional parameter vector  $\theta^{(i)}$ , including how the initial value  $\theta^{(0)} = \vec{0}$  is represented (note that  $\theta^{(i)}$  is now a vector whose dimension is the same as the feature vectors  $\phi(x)$ );
- How you will efficiently make a prediction on a new input  $x^{(i+1)}$ . I.e., how you will compute  $h_{\theta^{(i)}}(x^{(i+1)}) = g(\theta^{(i)T} \phi(x^{(i+1)}))$ , using your representation of  $\theta^{(i)}$ ; and
- How you will modify the update rule given above to perform an update to  $\theta$  on a new training example  $(x^{(i+1)}, y^{(i+1)})$ ; i.e., using the update rule corresponding to the feature mapping  $\phi$ :

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

[Note: If you prefer, you are also welcome to do this problem using the convention of labels  $y \in \{-1, 1\}$ , and  $g(z) = \text{sign}(z) = 1$  if  $z \geq 0$ ,  $-1$  otherwise.]

**Answer:**

In the high-dimensional space we update  $\theta$  as follows:

$$\theta := \theta + \alpha(y^{(i)} - h_{\theta}(\phi(x^{(i)})))\phi(x^{(i)})$$

So (assuming we initialize  $\theta^{(0)} = \vec{0}$ )  $\theta$  will always be a linear combination of the  $\phi(x^{(i)})$ , i.e.,  $\exists \beta_l$  such that  $\theta^{(i)} = \sum_{l=1}^i \beta_l \phi(x^{(l)})$  after having incorporated  $i$  training points. Thus  $\theta^{(i)}$  can be compactly represented by the coefficients  $\beta_l$  of this linear combination, i.e.,  $i$  real numbers after having incorporated  $i$  training points  $x^{(i)}$ . The initial value  $\theta^{(0)}$  simply corresponds to the case where the summation has no terms (i.e., an empty list of coefficients  $\beta_l$ ).

We do not work explicitly in the high-dimensional space, but use the fact that  $g(\theta^{(i)T} \phi(x^{(i+1)})) = g(\sum_{l=1}^i \beta_l \cdot \phi(x^{(l)})^T \phi(x^{(i+1)})) = g(\sum_{l=1}^i \beta_l K(x^{(l)}, x^{(i+1)}))$ , which can be computed efficiently.

We can efficiently update  $\theta$ . We just need to compute  $\beta_i = \alpha(y^{(i)} - g(\theta^{(i-1)T} \phi(x^{(i)})))$  at iteration  $i$ . This can be computed efficiently, if we compute  $\theta^{(i-1)T} \phi(x^{(i)})$  efficiently as described above.

In an alternative approach, one can observe that, unless a sample  $\phi(x^{(i)})$  is misclassified,  $y^{(i)} - h_{\theta^{(i)}}(\phi(x^{(i)}))$  will be zero; otherwise, it will be  $\pm 1$  (or  $\pm 2$ , if the convention  $y, h \in \{-1, 1\}$

is taken). The vector  $\theta$ , then, can be represented as the sum  $\sum_{\{i: y^{(i)} \neq h_{\theta}(x^{(i)})\}} \alpha(2y^{(i)} - 1)\phi(x^{(i)})$  under the  $y, h \in \{0, 1\}$  convention, and containing  $(2y^{(i)})$  under the other convention. This can then be expressed as  $\theta^{(i)} = \sum_{i \in \text{Misclassified}} \beta_i \phi(x^{(i)})$  to be in more obvious congruence with the above. The efficient representation can now be said to be a list which stores only those indices that were misclassified, as the  $\beta_i$ s can be recomputed from the  $y^{(i)}$ s and  $\alpha$  on demand. The derivation for (b) is then only cosmetically different, and in (c) the update rule is to add  $(i + 1)$  to the list if  $\phi(x^{(i+1)})$  is misclassified.

### 3. [30 points] Spam classification

In this problem, we will use the naive Bayes algorithm and an SVM to build a spam classifier.

In recent years, spam on public electronic newsgroups has been an increasing problem. Here, we'll build a classifier to distinguish between "real" newsgroup messages, and spam messages. For this experiment, we obtained a set of spam emails, and a set of genuine newsgroup messages.<sup>1</sup> Using only the subject line and body of each message, we'll learn to distinguish between the spam and non-spam.

All the files for the problem are in `/afs/ir/class/cs229/ps/ps2/`. **Note: Please do not circulate this data outside this class.** In order to get the text emails into a form usable by naive Bayes, we've already done some preprocessing on the messages. You can look at two sample spam emails in the files `spam_sample_original*`, and their preprocessed forms in the files `spam_sample_preprocessed*`. The first line in the preprocessed format is just the label and is not part of the message. The preprocessing ensures that only the message body and subject remain in the dataset; email addresses (EMAILADDR), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens" rather than "words," since some of the features will correspond to special values like EMAILADDR. You don't have to worry about the distinction.) The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam mail.

The work to extract feature vectors out of the documents has also been done for you, so you can just load in the design matrices (called document-word matrices in text classification) containing all the data. In a document-word matrix, the  $i^{th}$  row represents the  $i^{th}$  document/email, and the  $j^{th}$  column represents the  $j^{th}$  distinct token. Thus, the  $(i, j)$ -entry of this matrix represents the number of occurrences of the  $j^{th}$  token in the  $i^{th}$  document.

For this problem, we've chosen as our set of tokens considered (that is, as our vocabulary) only the medium frequency tokens. The intuition is that tokens that occur too often or too rarely do not have much classification value. (Examples tokens that occur very often are words like "the," "and," and "of," which occur in so many emails and are sufficiently content-free that they aren't worth modeling.) Also, words were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same word. For a list of the tokens used, see the file `TOKENS_LIST`.

Since the document-word matrix is extremely sparse (has lots of zero entries), we have stored it in our own efficient format to save space. You don't have to worry about this

<sup>1</sup>Thanks to Christian Shelton for providing the spam email. The non-spam messages are from the 20 newsgroups data at <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html>.

format.<sup>2</sup> The file `readMatrix.m` provides the `readMatrix` function that reads in the document-word matrix and the correct class labels for the various documents. Code in `nb_train.m` and `nb_test.m` shows how `readMatrix` should be called. The documentation at the top of these two files will tell you all you need to know about the setup.

- (a) Implement a naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing.

You should use the code outline provided in `nb_train.m` to train your parameters, and then use these parameters to classify the test set data by filling in the code in `nb_test.m`. You may assume that any parameters computed in `nb_train.m` are in memory when `nb_test.m` is executed, and do not need to be recomputed (i.e., that `nb_test.m` is executed immediately after `nb_train.m`)<sup>3</sup>.

Train your parameters using the document-word matrix in `MATRIX.TRAIN`, and then report the test set error on `MATRIX.TEST`.

**Remark.** If you implement naive Bayes the straightforward way, you'll find that the computed  $p(x|y) = \prod_i p(x_i|y)$  often equals zero. This is because  $p(x|y)$ , which is the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called “underflow.”) You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as  $p(x|y)$ . [Hint: Think about using logarithms.]

- (b) Intuitively, some tokens may be particularly indicative of an email being in a particular class. We can try to get an informal sense of how indicative token  $i$  is for the SPAM class by looking at:

$$\log \frac{p(x_j = i|y = 1)}{p(x_j = i|y = 0)} = \log \left( \frac{P(\text{token } i|\text{email is SPAM})}{P(\text{token } i|\text{email is NOTSPAM})} \right).$$

Using the parameters fit in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., have the highest positive value on the measure above). The numbered list of tokens in the file `TOKENS.LIST` should be useful for identifying the words/tokens.

- (c) Repeat part (a), but with training sets of size ranging from 50, 100, 200, ..., up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error vs. training set size). You may need to change the call to `readMatrix` in `nb_train.m` to read the correct file each time. Which training-set size gives the best test set error?
- (d) Implement the simplified version of the SMO algorithm (see separate handout on Course Materials webpage) to train an SVM on this dataset, using the linear kernel  $\phi(x) = x$ . For this problem, use  $C = 1$ ,  $max\_passes = 10$  and a stopping criterion with  $tol$  no larger than 0.01 (smaller, such as 0.001, would be even better). State in your submission what value of  $tol$  you used.

The SMO algorithm may take a while to train in Matlab, so you only need to train on `MATRIX.TRAIN.50` and `MATRIX.TRAIN.100`. (If you can run it on the other sizes as well, that's even better.) Report the test errors on `MATRIX.TEST`. Similar to the

<sup>2</sup>Unless you're not using Matlab/Octave, in which case feel free to ask us about it.

<sup>3</sup>Matlab note: If a `.m` file doesn't begin with a function declaration, the file is a script. Variables in a script are put into the global namespace, unlike with functions.

Naive Bayes implementation, an outline for your code is provided in `svm_train.m` and `svm_test.m`. We've provided a file `smo_train.m`, in which you should implement SMO for an arbitrary training set. The file `smo_verify.m` calls `smo_train` with a simple matrix and plot the separating hyperplane. This should be helpful for debugging.

- (e) In our experiments, training an SVM on the full training set (`MATRIX.TRAIN.1400`) achieves about 99.5% accuracy on the test-set.

Given this information, and your results from your previous experiments, how do you think naive Bayes and Support Vector Machines compare (in terms of generalization error) as a function of the training set size?

**Answer:**

- (a) The test error when training on the full training set was 1.63%. If you got a different error (or if you got the words `website` and `lowest` for part b), you most probably implemented the wrong Naive Bayes model.
- (b) The five most indicative words for the spam class were: `httpaddr`, `spam`, `unsubscribe`, `ebai` and `valet`.
- (c) The test set error for different training set sizes was:
- i. Training set size 50: Test set error = 4.5%
  - ii. Training set size 100: Test set error = 3.12%
  - iii. Training set size 200: Test set error = 2.38%
  - iv. Training set size  $\geq 400$ : Test set error = 1.63%
- (d) The test set error from the SVM for different training set sizes was:
- i. Training set size 50: Test set error = 6.50%
  - ii. Training set size 100: Test set error = 2.88%
  - iii. Training set size 200: Test set error = 1.12%
- (e) The deduction that can be drawn is that Naive Bayes learns quickly with less data, but has higher asymptotic error. On the other hand, the SVM classifier has relatively higher error on very small training sets, but is asymptotically much better than Naive Bayes. Note that this is consistent with the observation discussed in class that generative learning algorithms (such as Naive Bayes) have smaller sample complexity than discriminative algorithms (such as SVMs), but may also have higher asymptotic error.

The Matlab code for the problem:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nb_train.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[spmatrix, tokenlist, trainCategory] = readMatrix('MATRIX.TRAIN');

trainMatrix = full(spmatrix);
numTrainDocs = size(trainMatrix, 1);
numTokens = size(trainMatrix, 2);
```

```

% ...
% YOUR CODE HERE

V = size(trainMatrix, 2);
neg = trainMatrix(find(trainCategory == 0), :);
pos = trainMatrix(find(trainCategory == 1), :);

neg_words = sum(sum(neg));
pos_words = sum(sum(pos));

neg_log_prior = log(size(neg,1) / numTrainDocs);
pos_log_prior = log(size(pos,1) / numTrainDocs);

for k=1:V,
    neg_log_phi(k) = log((sum(neg(:,k)) + 1) / (neg_words + V));
    pos_log_phi(k) = log((sum(pos(:,k)) + 1) / (pos_words + V));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nb_test.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[spmatrix, tokenlist, category] = readMatrix('MATRIX.TEST');

testMatrix = full(spmatrix);
numTestDocs = size(testMatrix, 1);
numTokens = size(testMatrix, 2);

% ...
output = zeros(numTestDocs, 1);

%-----
% YOUR CODE HERE

for k=1:numTestDocs,
    [i,j,v] = find(testMatrix(k,:));
    neg_posterior = sum(v .* neg_log_phi(j)) + neg_log_prior;
    pos_posterior = sum(v .* pos_log_phi(j)) + pos_log_prior;

    if (neg_posterior > pos_posterior)
        output(k) = 0;
    else
        output(k) = 1;
    end
end

%-----

```

```

% Compute the error on the test set
error=0;
for i=1:numTestDocs
    if (category(i) ~= output(i))
        error=error+1;
    end
end

%Print out the classification error on the test set
error/numTestDocs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% svm_train.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[spmatrix, tokenlist, trainCategory] = readMatrix('MATRIX.TRAIN.200');

trainMatrix = full(spmatrix);
numTrainDocs = size(trainMatrix, 1);
numTokens = size(trainMatrix, 2);

% ...

% YOUR CODE HERE

for i=1:numTrainDocs,
    if (trainCategory(i) == 0)
        trainCategory(i) = -1;
    end
end

[b, alphas] = smo_train(trainMatrix, trainCategory', 1, 0.01, 10);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% svm_test.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[spmatrix, tokenlist, category] = readMatrix('MATRIX.TEST');

testMatrix = full(spmatrix);
numTestDocs = size(testMatrix, 1);
numTokens = size(testMatrix, 2);

% ...

```

```

output = zeros(numTestDocs, 1);

%-----
% YOUR CODE HERE

for i=1:numTestDocs,
    if (sum(alphas .* trainCategory' .* (trainMatrix * testMatrix(i,:))) - b > 0)
        output(i) = 1;
    else
        output(i) = 0;
    end
end

%-----

% Compute the error on the test set
error=0;
for i=1:numTestDocs
    if (category(i) ~= output(i))
        error=error+1;
    end
end

%Print out the classification error on the test set
error/numTestDocs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% smo_train.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [b, alphas] = smo_train(X, Y, C, tol, max_passes)

% ...
% YOUR CODE HERE

% initialize parameters
alphas = zeros(size(X, 1), 1);
b = 0;
passes = 0;

while(passes < max_passes)
    num_changed_alphas = 0;
    checked = 0;
    for i=1:size(X,1),

        % Compute E_i
        E_i = sum(alphas .* Y .* (X * X(i,:))) - b - Y(i);

```

```

if((Y(i)*E_i < -tol && alphas(i) < C) || Y(i)*E_i > tol && alphas(i) > 0)

    % choose j != i randomly
    j = floor(rand*size(X,1))+1;
    while (j == i)
        j = floor(rand*size(X,1))+1;
    end

    % calculate E_j and save old alphas
    E_j = sum(alphas .* Y .* (X * X(j,:)')) - b - Y(j);
    alpha_i_old = alphas(i);
    alpha_j_old = alphas(j);

    % compute L and H
    if (Y(i) ~= Y(j))
        L = max([0, alphas(j) - alphas(i)]);
        H = min([C, C + alphas(j) - alphas(i)]);
    else
        L = max([0, alphas(i) + alphas(j) - C]);
        H = min([C, alphas(i) + alphas(j)]);
    end
    if (L == H)
        continue;
    end

    % compute and compute and clip new alpha_j
    eta = 2*X(i,:)*X(j,:)' - X(i,:)*X(i,:)' - X(j,:)*X(j,:);
    if (eta >= 0)
        continue;
    end
    alphas(j) = alphas(j) - Y(j)*(E_i - E_j)/eta;
    if (alphas(j) > H)
        alphas(j) = H;
    elseif (alphas(j) < L)
        alphas(j) = L;
    end
    if (abs(alphas(j) - alpha_j_old) < 10^-5)
        continue;
    end
    alphas(i) = alphas(i) + Y(i)*Y(j)*(alpha_j_old - alphas(j));

    % compute b
    b1 = b + E_i + Y(i)*(alphas(i) - alpha_i_old) * X(i,:)*X(i,:)' + ...
        Y(j)*(alphas(j) - alpha_j_old) * X(i,:)*X(j,:);
    b2 = b + E_j + Y(i)*(alphas(i) - alpha_i_old) * X(i,:)*X(j,:)' + ...
        Y(j)*(alphas(j) - alpha_j_old) * X(j,:)*X(j,:);
    if (alphas(i) > 0 && alphas(i) < C)
        b = b1;
    elseif (alphas(j) > 0 && alphas(j) < C)

```

```

        b = b2;
    else
        b = (b1 + b2)/2;
    end

    num_changed_alphas = num_changed_alphas + 1;
end
end

if (num_changed_alphas == 0)
    passes = passes + 1;
else
    passes = 0;
end
end
end

```

#### 4. [20 points] Properties of VC dimension

In this problem, we investigate a few properties of the Vapnik-Chervonenkis dimension, mostly relating to how  $VC(H)$  increases as the set  $H$  increases. For each part of this problem, you should state whether the given statement is true, and justify your answer with either a formal proof or a counter-example.

- Let two hypothesis classes  $H_1$  and  $H_2$  satisfy  $H_1 \subseteq H_2$ . Prove or disprove:  $VC(H_1) \leq VC(H_2)$ .
- Let  $H_1 = H_2 \cup \{h_1, \dots, h_k\}$ . (I.e.,  $H_1$  is the union of  $H_2$  and some set of  $k$  additional hypotheses.) Prove or disprove:  $VC(H_1) \leq VC(H_2) + k$ . [Hint: You might want to start by considering the case of  $k = 1$ .]
- Let  $H_1 = H_2 \cup H_3$ . Prove or disprove:  $VC(H_1) \leq VC(H_2) + VC(H_3)$ .

**Answer:**

- True. Suppose that  $VC(H_1) = d$ . Then there exists a set of  $d$  points that is shattered by  $H_1$  (i.e., for each possible labeling of the  $d$  points, there exists a hypothesis  $h \in H_1$  which realizes that labeling). Now, since  $H_2$  contains all hypotheses in  $H_1$ , then  $H_2$  shatters the same set, and thus we have  $VC(H_2) \geq d = VC(H_1)$ .
- True. If we can prove the result for  $k = 1$ , then the result stated in the problem set follows immediately by applying the same logic inductively, one hypothesis at a time. So, let us prove that if  $H_1 = H_2 \cup \{h\}$ , then  $VC(H_1) \leq VC(H_2) + 1$ . Suppose that  $VC(H_1) = d$ , and let  $S_1$  be a set of  $d$  points that is shattered by  $H_1$ . Now, pick an arbitrary  $x \in S_1$ . Since  $H_1$  shatters  $S_1$ , there must be some  $\bar{h} \in H_1$  such that  $h$  and  $\bar{h}$  agree on labelings for all points in  $S_1$  except  $x$ . This means that  $H' := H_1 \setminus \{h\}$  achieves all possible labelings on  $S' := S_1 \setminus \{x\}$  (i.e.  $H'$  shatters  $S'$ ), so  $VC(H') \geq |S'| = d - 1$ . But  $H' \subseteq H_2$ , so from part (a),  $VC(H') \leq VC(H_2)$ . It follows that  $VC(H_2) \geq d - 1$ , or equivalently,  $VC(H_1) \leq VC(H_2) + 1$ , as desired.

For this problem, there were a number of possible correct proof methods; generally, to get full credit, you needed to argue formally that there exists no set of  $(VC(H_2) + 2)$  points shattered by  $H_1$ , or equivalently, that there always exists a set of  $(VC(H_1) - 1)$  points shattered by  $H_2$ . Here are a couple of the more common errors:

- Some submitted solutions stated that adding a single hypothesis to  $H_2$  increases the VC dimension by at most one, since the new hypothesis can only realize a single labeling. While this statement is vaguely true, it is neither sufficiently precise, nor is its correctness immediately obvious.
  - Some solutions made arguments relating to the cardinality of the sets  $H_1$  and  $H_2$ . However, generally when we speak about VC dimension, the sets  $H_1$  and  $H_2$  often have infinite cardinality (e.g., the set of all linear classifiers in  $\mathbb{R}^2$ ).
- (c) False. Counterexample: let  $H_1 = \{h_1\}$ ,  $H_2 = \{h_2\}$ , and  $\forall x$ ,  $h_1(x) = 0, h_2(x) = 1$ . Then we have  $VC(H_1) = VC(H_2) = 0$ , but  $VC(H_1 \cup H_2) = 1$ .

### 5. [20 points] Training and testing on different distributions

In the discussion in class about learning theory, a key assumption was that we trained and tested our learning algorithms on the same distribution  $\mathcal{D}$ . In this problem, we'll investigate one special case of training and testing on different distributions. Specifically, we will consider what happens when the training labels are *noisy*, but the test labels are not.

Consider a binary classification problem with labels  $y \in \{0, 1\}$ , and let  $\mathcal{D}$  be a distribution over  $(x, y)$ , that we'll think of as the original, "clean" or "uncorrupted" distribution. Define  $\mathcal{D}_\tau$  to be a "corrupted" distribution over  $(x, y)$  which is the same as  $\mathcal{D}$ , except that the labels  $y$  have some probability  $0 \leq \tau < 0.5$  of being flipped. Thus, to sample from  $\mathcal{D}_\tau$ , we would first sample  $(x, y)$  from  $\mathcal{D}$ , and then with probability  $\tau$  (independently of the observed  $x$  and  $y$ ) replace  $y$  with  $1 - y$ . Note that  $\mathcal{D}_0 = \mathcal{D}$ .

The distribution  $\mathcal{D}_\tau$  models a setting in which an unreliable human (or other source) is labeling your training data for you, and on each example he/she has a probability  $\tau$  of mislabeling it. Even though our training data is corrupted, we are still interested in evaluating our hypotheses with respect to the original, uncorrupted distribution  $\mathcal{D}$ .

We define the generalization error *with respect to*  $\mathcal{D}_\tau$  to be

$$\varepsilon_\tau(h) = P_{(x,y) \sim \mathcal{D}_\tau}[h(x) \neq y].$$

Note that  $\varepsilon_0(h)$  is the generalization error with respect to the "clean" distribution; it is with respect to  $\varepsilon_0$  that we wish to evaluate our hypotheses.

- (a) For any hypothesis  $h$ , the quantity  $\varepsilon_0(h)$  can be calculated as a function of  $\varepsilon_\tau(h)$  and  $\tau$ . Write down a formula for  $\varepsilon_0(h)$  in terms of  $\varepsilon_\tau(h)$  and  $\tau$ , and justify your answer.
- (b) Let  $|H|$  be finite, and suppose our training set  $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$  is obtained by drawing  $m$  examples IID from the corrupted distribution  $\mathcal{D}_\tau$ . Suppose we pick  $h \in H$  using empirical risk minimization:  $\hat{h} = \arg \min_{h \in H} \hat{\varepsilon}_S(h)$ . Also, let  $h^* = \arg \min_{h \in H} \varepsilon_0(h)$ .

Let any  $\delta, \gamma > 0$  be given. Prove that for

$$\varepsilon_0(\hat{h}) \leq \varepsilon_0(h^*) + 2\gamma$$

to hold with probability  $1 - \delta$ , it suffices that

$$m \geq \frac{1}{2(1-2\tau)^2\gamma^2} \log \frac{2|H|}{\delta}.$$

**Remark.** This result suggests that, roughly,  $m$  examples that have been corrupted at noise level  $\tau$  are worth about as much as  $(1 - 2\tau)^2 m$  uncorrupted training examples. This is a useful rule-of-thumb to know if you ever need to decide whether/how much to pay for a more reliable source of training data. (If you've taken a class in information theory, you may also have heard that  $(1 - \mathcal{H}(\tau))m$  is a good estimate of the information in the  $m$  corrupted examples, where  $\mathcal{H}(\tau) = -(\tau \log_2 \tau + (1 - \tau) \log_2 (1 - \tau))$  is the "binary entropy" function. And indeed, the functions  $(1 - 2\tau)^2$  and  $1 - \mathcal{H}(\tau)$  are quite close to each other.)

(c) Comment **briefly** on what happens as  $\tau$  approaches 0.5.

**Answer:**

(a) We compute  $\varepsilon_\tau$  as a function of  $\varepsilon_0$  and then invert the obtained expression. An error occurs on the corrupted distribution, if and only if, an error occurred for the original distribution and the point that was not corrupted, or no error occurred for the original distribution but the point was corrupted. So we have

$$\varepsilon_\tau = \varepsilon_0(1 - \tau) + (1 - \varepsilon_0)\tau$$

Solving for  $\varepsilon_0$  gives

$$\varepsilon_0 = \frac{\varepsilon_\tau - \tau}{1 - 2\tau}$$

(b) We will need to apply the following (in the right order):

$$\forall h \in H, |\varepsilon_\tau(h) - \hat{\varepsilon}_\tau(h)| \leq \bar{\gamma} \quad \text{w.p.}(1 - \delta), \quad \delta = 2K \exp(-2\bar{\gamma}^2 m) \quad (6)$$

$$\varepsilon_\tau = (1 - 2\tau)\varepsilon + \tau, \quad \varepsilon_0 = \frac{\varepsilon_\tau - \tau}{1 - 2\tau} \quad (7)$$

$$\forall h \in H, \hat{\varepsilon}_\tau(\hat{h}) \leq \hat{\varepsilon}_\tau(h), \quad \text{in particular for } h^* \quad (8)$$

Here is the derivation:

$$\varepsilon_0(\hat{h}) = \frac{\varepsilon_\tau(\hat{h}) - \tau}{1 - 2\tau} \quad (9)$$

$$\leq \frac{\hat{\varepsilon}_\tau(\hat{h}) + \bar{\gamma} - \tau}{1 - 2\tau} \quad \text{w.p.}(1 - \delta) \quad (10)$$

$$\leq \frac{\hat{\varepsilon}_\tau(h^*) + \bar{\gamma} - \tau}{1 - 2\tau} \quad \text{w.p.}(1 - \delta) \quad (11)$$

$$\leq \frac{\varepsilon_\tau(h^*) + 2\bar{\gamma} - \tau}{1 - 2\tau} \quad \text{w.p.}(1 - \delta) \quad (12)$$

$$= \frac{(1 - 2\tau)\varepsilon_0(h^*) + \tau + 2\bar{\gamma} - \tau}{1 - 2\tau} \quad \text{w.p.}(1 - \delta) \quad (13)$$

$$= \varepsilon_0(h^*) + \frac{2\bar{\gamma}}{1 - 2\tau} \quad \text{w.p.}(1 - \delta) \quad (14)$$

$$= \varepsilon_0(h^*) + 2\bar{\gamma} \quad \text{w.p.}(1 - \delta) \quad (15)$$

Where we used in the following order: (7)(6)(8)(6)(7), and the last 2 steps are algebraic simplifications, and defining  $\gamma$  as a function of  $\bar{\gamma}$ . Now we can fill out  $\bar{\gamma} = \gamma(1 - 2\tau)$  into  $\delta$  of (6), solve for  $m$  and we are done.

Note: one could shorten the above derivation and go straight from (9) to (12) by using that result from class.

- (c) The closer  $\tau$  is to 0.5, the more samples are needed to get the same generalization error bound. For  $\tau$  approaching 0.5, the training data becomes more and more random; having no information at all about the underlying distribution for  $\tau = 0.5$ .