10-601 Machine Learning, Fall 2009: Homework 5

Due: December,2nd, 10:30 am

Instructions There are 4 questions on this assignment worth a total of 160 points. Please hand in a hard copy at the beginning of the class. Please print your code and attach it to the write-up. Refer to the webpage for policies regarding collaboration, due dates, and extensions.

1 Support Vector Machines and Kernels [Points: 40 pts]

You are given a data set D in Figure 1 with data from a single feature X in $\mathbb{R}^1$ and corresponding label $Y \in \{+, -\}$. The data set contains three positive examples at $\{-3, -2, 3\}$ and three negative examples at $\{-1, 0, 1\}$.

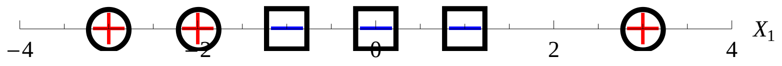

Figure 1: Dataset for SVM feature map task in Question 1.

1.1 Finite Features and SVMs

1. **[Points: 1 pts]** Can this data set (in its current feature space) be perfectly separated using a linear separator? Why or why not?

2. **[Points: 4 pts]** Define the simple feature map $\varphi(u) = (u, u^2)$, which transforms points in $\mathbb{R}^1$ to points in $\mathbb{R}^2$. Apply φ to the data and plot the points in the new $\mathbb{R}^2$ feature space.

3. **[Points: 1 pts]** Can a linear separator perfectly separate the points in the new $\mathbb{R}^2$ feature space induced by φ? Why or why not?

4. **[Points: 2 pts]** Give an equation for the kernel that corresponds to the feature map φ. That is, write $K(X, X')$ in terms of only X and X'.

5. **[Points: 8 pts]** Construct a maximum-margin separating hyperplane. This hyperplane will be a line in $\mathbb{R}^2$, which can be parametrized by its normal equation, i.e. $w_1V_1 + w_2V_2 + c = 0$ for appropriate choices of w_1, w_2, c. Here, $(V_1, V_2) = \varphi(X)$ is the result of applying the feature map φ to the original feature X. Give the values for w_1, w_2, c. Also, explicitly compute and report the margin for your hyperplane. *Hint: you do not need to solve a quadratic program to find the maximum margin hyperplane. Instead, let your geometric intuition guide you.*

6. **[Points: 5 pts]** On the plot of the transformed points (from part 2), plot the separating hyperplane and the margin, and circle the support vectors.

7. **[Points: 2 pts]** Draw the decision boundary of the separating hyperplane, in the original $\mathbb{R}^1$ feature space.

8. **[Points: 5 pts]** Compute the coefficients α and the constant b in Equation 1.1 for the kernel k and the support vectors $SV = \{u_1, u_2\}$ you chose in part 6. Be sure to explain how you obtained these coefficients.

$$y(x) = \mathsf{sign}\left(\sum_{n=1}^{|SV|} \alpha_n y_n K(x, u_n) + b\right) \tag{1.1}$$

   Think about the dual form of the quadratic program and the constraints placed on the α values.

9. **[Points: 2 pts]** If we add another positive ($Y = +$) point to the training set at $X = 5$ would the hyperplane or margin change? Why or why not?

1.2 Infinite Features Spaces and the Kernel Trick

Let's define a new more complicated feature transformation $\varphi_n : \mathbb{R}^1 \to \mathbb{R}^n$ given in Equation 1.2.

$$\varphi_n(x) = \left\{ e^{-x^2/2}, e^{-x^2/2}x, \frac{e^{-x^2/2}x^2}{\sqrt{2}}, \ldots, \frac{e^{-x^2/2}x^n}{\sqrt{n!}} \right\} \tag{1.2}$$

Suppose we let $n \to \infty$ and define new feature transformation in Equation 1.3. You can think of this feature transformation as taking some finite feature vector and producing an infinite dimensional feature vector rather than the simple two dimensional feature vector used in the earlier part of this problem.

$$\varphi_\infty(x) = \left\{ e^{-x^2/2}, e^{-x^2/2}x, \frac{e^{-x^2/2}x^2}{\sqrt{2}}, \ldots, \right\} \tag{1.3}$$

1. **[Points: 1 pts]** Can we directly apply this feature transformation to the data? Put another way, can we explicity construct $\varphi_\infty(x)$? *Hint: this is nearly rhetorical and not intended as a trick question.*

2. **[Points: 2 pts]** Is there a finite set of points that cannot be linearly separated in this feature space? Explain why or why not.

3. **[Points: 5 pts]** We know that we can express a linear classifier using only inner products of support vectors in the transformed feature space as seen in Equation 1.1. It would be great if we could somehow use the feature space obtained by the feature transformation φ_∞. However, to do this we must be able to compute the inner product of examples in this infinite vector space. The inner product between two infinite vectors $a = \langle a_1, a_2, \ldots \rangle$ and $b = \langle b_1, b_2, \ldots \rangle$ is defined as the infinite sum given in Equation 1.4 (assuming the series converges).

$$K(a, b) = a \cdot b = \sum_{k=0}^{\infty} a_k b_k \tag{1.4}$$

   Can we explicity compute $K(a, b)$? What is the explicit form of $K(a, b)$? *Hint: you may want to use the Taylor series expansion of e^x which is given in Equation 1.5.*

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \tag{1.5}$$

4. **[Points: 2 pts]** With such a high dimensional feature space should we be concerned about overfitting? Why or why not?

2 VC Dimension and PAC Bounds [Points: 20 pts]

Suppose you are given training data $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathbb{R}^2$ and $y_i \in \{0, 1\}$. You are told that there exists some $h^* \in \mathcal{H}_{\text{triangle}}$ such that $y = h^*(x)$, where the hypothesis space $\mathcal{H}_{\text{triangle}}$ contains hypotheses that correspond to triangular decision rules. More precisely, for every triangle $T = \{t_1, t_2, t_3\}$ (where t_i are vertices, i.e., $t_i \in \mathbb{R}^2$), there is a corresponding $h \in \mathcal{H}_{\text{triangle}}$ such that $h(x) = 1$ if and only x is interior to the triangle T. This hypothesis space is often called the positive triangle hypothesis space. You find out that each training instance costs \$1.00 and you want to know how much it will cost to obtain 95% accuracy with probability 0.75.

2.1 Complexity of the Hypothesis Space

The number of training examples we will need for the desired accuracy and the desired probability depends on the complexity of the hypothesis space. The first step to this problem is to understand the "complexity" of the hypothesis space, $\mathcal{H}_{\text{triangle}}$. Unfortunately, direct measures of the size of $\mathcal{H}_{\text{triangle}}$ are insufficient. Instead, we will adopt the VC dimension as a measure of the complexity of this hypothesis space. Recall that the VC dimension of a hypothesis space is the maximum number of points that can be shattered. Here we will show that the VC dimension of $\mathcal{H}_{\text{triangle}}$ is $\text{VC}(\mathcal{H}_{\text{triangle}}) = 7$.

1. **[Points: 2 pts]** Why would directly measuring the size of the hypothesis space, $|\mathcal{H}_{\text{triangle}}|$, be insufficient for the triangle hypothesis space?
2. **[Points: 5 pts]** Show that there exists a configuration of 7 points that can be shattered.
3. **[Points: 5 pts]** Argue that there is no configuration of 8 points that can be shattered.

2.2 Consistent Training Estimate [Points: 3 pts]

Suppose that there exists an $h \in \mathcal{H}_{\text{triangle}}$ such that $h(x_i) = y_i$ for all $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ in the training set. Briefly describe a simple training algorithm which achieves zero training error.

2.3 Price of Learning [Points: 5 pts]

1. Using the following PAC bound for the consistent setting (i.e., $h^* \in \mathcal{H}_{\text{triangle}}$),

$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8\,\text{VC}(\mathcal{H})\log_2\left(\frac{13}{\epsilon}\right)\right)$$

   where m is the number of examples and δ is the probability of achieving ϵ test error, compute the cost to obtain 95% accuracy with probability 0.75.

2. If instead of $\mathcal{H}_{\text{triangle}}$ we used $\mathcal{H}_{\text{half}}$, the hypothesis space consisting of all half space hypotheses of the form "$y = 1$ **iff** $w \cdot x + b \geq 0$" and we assumed $h^* \in \mathcal{H}_{\text{half}}$ (the true hypothesis is also a halfspace), what would be the cost of obtaining 95% accuracy with probability 0.75?

3 Online to Batch Learning [Points: 20 pts]

In this problem we will derive a batch learning algorithm from an online learning algorithm. Consider the noiseless classification setting in which $x \in \mathbb{R}^2$ and $y \in \{0, 1\}$ and there exists a deterministic function ϕ such that $\phi(x) = y$. You are given an online learning algorithm A that, given its hypothesis H_t calculated in the step t, and a new point (x_{t+1}, y_{t+1}), constructs a new hypothesis $H_{t+1} = A(H_t, (x_{t+1}, y_{t+1}))$; we assume that in the first step the algorithm uses some fixed hypothesis H_0 (possibly randomized). Recall that in the online setting for each new x_{t+1} we predict $H_t(x_{t+1})$ and then obtain y_{t+1} and train $H_{t+1} = A(H_t, (x_{t+1}, y_{t+1}))$. The only guarantee we have about A is that the error rate is bounded by:

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{I}\left[H_{t-1}(x_t) \neq y_t\right] \leq b(T)$$

where T is the number of examples presented to the online learning algorithm (T is upper-case, but not a random variable), $\mathbb{I}[\cdot]$ is the indicator function, and $b(T)$ is a bound on the error rate which depends on the number of examples. Typically $b(T)$ goes to zeros as T goes to infinity. For example in class we covered the halving algorithm in which $b(T) = \frac{\log_2 N}{T}$ where N is the number of experts and the true concept ϕ is known to be one of the experts.

In the batch setting you are given a training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ which is drawn independently from some distribution $\mathbf{P}(X, Y)$ and you are asked to return a hypothesis $\tilde{H}$ which is then evaluated on data drawn from the same distribution. Ultimately, we want to find an $\tilde{H}$ which minimizes the probability that $\tilde{H}$ will be incorrect (the generalization error):

$$\text{error}(\tilde{H}) = \mathbf{P}\left(\tilde{H}(X) \neq Y\right) \ .$$

Consider the following procedure for constructing $\tilde{H}$ for the batch setting using the online learning algorithm A. You begin by obtaining a training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ which is drawn iid from some unknown distribution $\mathbf{P}(X, Y)$. You then run the online learning algorithm A sequentially on the training set producing the sequence of hypotheses $(H_0, \ldots, H_{m-1})$ where $H_i = A(H_{i-1}, (x_i, y_i))$ for $i \geq 1$. To build $\tilde{H}$ from the sequence of $(H_0, \ldots, H_{m-1})$, we randomly choose with equal probability, a hypothesis from $(H_0, \ldots, H_{m-1})$ for every new test point. That is for each new test point x, $\tilde{H}(x)$ randomly chooses a hypothesis H_i from $(H_0, \ldots, H_{m-1})$ and then predicts $H_i(x)$.

3.1 Understanding the Problem [Points: 6 pts]

Let's begin with a few warm-up questions to build a better understanding of the problem setting.

1. Suppose we had alternatively defined $\tilde{H}(x) = H_0(x)$; how could this lead to a poor test performance?
2. Suppose we had alternatively defined $\tilde{H}(x) = H_{m-1}(x)$; how could this lead to a poor test performance?

3.2 Bounding the Performance [Points: 14 pts]

We now derive a bound for the expected error rate for $\tilde{H}$ from the bounds for the online learning algorithm.

1. Using the definition of expectation argue that:
$$\text{error}(\tilde{H}) = \mathbf{P}\left(\tilde{H}(X) \neq Y\right) = \mathbf{E}\left[\mathbb{I}\left[\tilde{H}(X) \neq Y\right]\right]$$
2. Using properties of expectations argue that:
$$\mathbf{E}\left[\mathbb{I}\left[\tilde{H}(X) \neq Y\right]\right] = \frac{1}{m}\sum_{i=1}^{m} \mathbf{E}\left[\mathbb{I}\left[H_{i-1}(X) \neq Y\right]\right]$$
3. Using the iid and independence assumptions show that:
$$\frac{1}{m}\sum_{i=1}^{m} \mathbf{E}\left[\mathbb{I}\left[H_{i-1}(X) \neq Y\right]\right] = \frac{1}{m}\sum_{i=1}^{m} \mathbf{E}\left[\mathbb{I}\left[H_{i-1}(X_i) \neq Y_i\right]\right]$$
4. Using the assumptions about A and the previous steps to argue that:
$$\text{error}(\tilde{H}) \leq b(m)$$
5. Suppose we use the halving algorithm for A and assume that ϕ is one of the experts; how does the bound we obtained here compare to the PAC bounds for consistent learning in the finite hypothesis spaces? Write and label both bounds and then briefly describe their similarities and differences in terms of what they bound and the assumptions they make.

4 Decision trees, Bagging and Boosting [Points: 80 pts]

In this part of the assignment you will empirically compare the performances of the following three algorithms both with and without label noise:

- Decision trees
- AdaBoost with decision stumps as base classifier
- Bagging with decision trees as base classifier

The data you will be working with is a subset of UCI spambase dataset[1]. The classification problem is to predict whether an email is considered `spam` or `ham`.

4.1 Comparing the methods

1. Download the data from the course website www.cs.cmu.edu/~ggordon/10601/hws/hw5/hw5data.zip.
2. Load the Matlab file `spam.mat`. The file should contain a struct:

   data: This struct contains all of your data. The fields in the struct are:

   X: Feature matrix, rows are examples and columns are features

   Y: The labels of each example, 1 for `spam` and -1 for `ham`.
3. In comparing the three algorithms use 10-fold cross validation and report the average errors. Make sure to use the same splits across different algorithms.

4.1.1 Decision tree

Train and test a decision tree. Matlab provides a function `classregtree` to build decision trees. Use defaults settings. An example call is the following:

```
tree=classregtree(trainX,trainY,'method','classification');
trainYpred=str2double(eval(tree,trainX));
```

Report the average training and testing errors across the 10 folds. View the tree you generated (you do not need to report).

4.1.2 Bagging decision trees

Bagging creates ensemble of strong learners and combines the predictions of these learners into a final hypothesis using majority voting. Implement the bagging algorithm; use strong decision trees as the base learner (keep the default setting). In each step, create a bootstrap replicate of the training data with the same size. For this, you can use `randsample` function of Matlab with `replace` option set to `true` (this setting provides sampling with replacement). Bag $T = 1000$ decision trees. Store the training and test errors for the classifier as you grow the ensemble, $t = 1 \ldots 1000$. Plot the average training and test errors for the entire voted ensemble, using 10-fold cross validation, as a function of the number of trees in the ensemble: the x-axis is the ensemble size $t = 1 \ldots 1000$, and the y-axis is the average error calculated over 10 folds at the given step. The plot should include two curves: the average training error and the average test error. Describe and discuss the plot you generated.

[1] http://archive.ics.uci.edu/ml/datasets/Spambase

4.1.3 Boosting decision trees

AdaBoost trains a sequence of weak classifiers and combines the hypothesis of these classifiers into a final decision. Let m be the number of training examples, which are of the form $(\mathbf{x}_i, y_i)$, y_i is the true label and $\mathbf{x}_i$ is the feature vector for example i , where $i = 1, \ldots, m$. At each iteration t, a classifier, $h_t(\mathbf{x}) \rightarrow \{-1, 1\}$, is trained on the training data to minimize the weighted classification error, $\sum_{i=1}^{n} D_t(i) \cdot \mathbb{I}\left[h_t(\mathbf{x}_i) \neq y_i\right]$. Here, $\mathbb{I}\left[\cdot\right]$ is the indicator function (0 if the predicted and true labels match, and 1 otherwise) and $D_t(i)$ indicates the importance of example i. The final prediction of AdaBoost is a linear combination of these classifiers, $H_T(\mathbf{x}) = \mathsf{sign}(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$, where T is the number of boosting iterations (number of decision trees in the final ensemble).

1. As your base learner you will use decision stump. Implement the decision stump which minimizes the weighted error.

2. Implement AdaBoost algorithm. Run boosting for $T = 1000$ steps. Store the training and test errors for the single classifier learned at each step, $t = 1 \ldots 1000$, as you grow the ensemble. Plot the average training and test errors for the entire voted ensemble, using 10-fold cross validation, as a function of the ensemble size: the x-axis is the ensemble size $t = 1 \ldots 1000$, and the y-axis is the average error calculated over 10 folds at the given step. The plot should include two curves: the average training error and the average test error.

4.1.4 Comparing the performances

Overlay the test errors of the tree methods onto a single graph, the x-axis is the ensemble size $t = 1 \ldots 1000$. The y-axis is the average test errors. There should be two curves one for bagging one for boosting. Plot the decision tree test error as a line on the same graph, constant across the ensemble size. Discuss in detail the plot you generated.

4.2 Comparing the performances of the three methods with noisy training data

Repeat the comparison of the tree algorithms this time under uniform label noise. Begin with injecting noisy training labels, by flipping the true labels of the example to the other class uniformly at random with probability 0.2. Use the same settings for the classifiers. Repeat the experiments 10 times, each time with a random set of labels are noisy (10 x 10 CV folds, total of 100 runs). Generate the same plots as you did in the previous section. How do performances of the methods change as compared to the non-noisy condition? Discuss each method and the reasons of why it would perform as such.