CS274a Homework #4

Introduction to Machine Learning: Fall 2013

Due: Tuesday November 19th, 2013

Write neatly (or type) and show all your work!

Please remember to turn in at most two documents, one with any handwritten solutions, and one PDF file with any electronic solutions.

Problem 0: Project teams

Please identify your team name and group members for the project.

Problem 1: VC Dimension

(a) Consider a decision tree for binary classification ($y \in \{-1, 1\}$) with *one* real-valued feature x_1, and depth (at most) d. What is the VC dimension of this learner? Explain.

(b) Now consider a decision tree with *two* real-valued features. For $d = 1$, how many parameters does this decision tree have, and what are their domains? (For split nodes, parameters include the feature index to split on and its threshold; for predict nodes, they include its prediction value.) Show by example that for $d = 1$ (a decision "stump"), the learner can shatter three points, but argue that it cannot shatter four. How does this compare to your answer in the previous part?

(c) Now consider a decision tree with *three* features, still with $d = 1$. In this case, how many parameters does the model have? Modify your previous example to show that this learner now *can* shatter four points.

Hint for parts b & c: the decision stump classifier depends only on the *order* of the data points in each dimension, not their actual values (you should convince yourself of this). So you can just reason about the possible orderings of the points in each dimension.

Problem 2: Support Vector Machines

In this problem, we'll train a (separable) SVM using a QP solver. Build your separable binary classification data as in the previous homework (no need to do training/validation this time):

```
iris=load('data/iris.txt');     % load the text file
X = iris(:,1:2); Y=iris(:,end); % get first two features
XA = X(Y<2,:); YA=Y(Y<2);       % get class 0 vs 1
```

Now, use either Matlab's `quadprog` or Octave's `qp` function to solve the SVM quadratic program. Unfortunately, `quadprog` is only available in the Optimization toolbox, so if you don't have access to that and can't use the on-campus lab machines, I suggest using Octave. If you don't want to install Octave, you can likely do this problem on an online version of Octave, e.g., `http://www.compileonline.com/execute_matlab_online.php` (You'll need to copy and paste your data in.)

Recall that the primal SVM form is:

$$\min_{w,b} \sum_i w_i^2 \qquad \text{s.t.} \qquad y^{(i)}\left(wx^{(i)} + b\right) \geq 1$$

Manipulate this form until it matches a "standard" form used by `qp` or `quadprog`, e.g., `quadprog(H,f,A,b,Aeq,Beq,lb,ub)` and design the necessary matrices `H,f,A,b`, etc. (These will be a bit different for Octave.) Output the resulting linear parameters θ^*, and use your perceptron classifier from last homework to check that it separates the data and plot its decision boundary using `plotClassify2D`.

Now, we'll solve the dual form instead. Compute the Gram matrix of dot products, $K_{ij} = x^{(i)} \cdot x^{(j)}$. Recall that the dual form is:

$$\max_{\alpha \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \, y^{(i)} y^{(j)} K_{ij} \qquad \text{s.t.} \qquad \sum_i \alpha_i y^{(i)} = 0$$

Identify the support vectors from your solution α^*. Verify that your solution is identical, i.e.,

$$\theta^* = \sum_i \alpha_i^* y^{(i)} x^{(i)}$$

and mark the support vectors' locations in your classification boundary plot.

Problem 3: Bagging with Decision trees

Download the SpamBase data set from UCI's machine learning repository:
`http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data`

These data are pre-processed features of emails, along with a class indicating whether the associated email was spam or not. They are a bit out of date; modern spam prediction data sets would have hundreds of thousands of features, rather than a few dozen.

Pre-process and split the data into training and validation sets:

```
rand('state',0); randn('state',0);
data = load('data/spambase.data');
X = data(:,1:57); Y=data(:,end);            % extract features & class labels
[X,Y] = shuffleData(X,Y);                   % permute out of class-based order
[Xt,Xv,Yt,Yv] = splitData(X,Y,.6);          % divide into training & test
[Xt,S] = rescale(Xt); Xv = rescale(Xv,S);   % pre-process data scaling
```

(a) Use either Matlab's or my own provided decision tree code (`treeClassify`) to learn a decision tree classifier for the data, and report its training and validation accuracy. Report the settings you used to learn the tree (i.e., leaf criteria, split impurity score function, etc.)

(b) Now search over values for the maximum depth cutoff for the tree, and score both training and test. Plot the results and report what value of depth you would select and why.

(c) Now, learn a bagged ensemble of decision trees. (See the pseudocode from lecture slides.) For your individual learners, don't use any complexity control (no depth cutoff, minParent, etc.); explain why we should believe that for these parameter settings we are overfitting. For the bootstrap process, draw the same number of data as in the original data set ($N' = N$). (You may find `bootstrapData.m` helpful, although it is very easy to do yourself.) Plot the training and validation error as a function of the number of learners you include in the ensemble.

(d) Now, choose and fix a number of learners to include in the ensemble (say, 25). Re-run your bagged ensemble learning algorithm, but now sample fewer data points per bootstrap set. (I suggest spacing values of N' logarithmically between say $0.1\,N$ and N.) Again, plot training and validation error as a function of N'. Explain briefly what is happening and why.

Problem 4: Boosting

Consider the following classification problem. You wish to use boosting to learn a classifier consisting of a decision stump – using a single feature, a threshold on the value of x_1 (horizontal axis) or x_2 (vertical axis) at each level. Find the first two classifiers learned by AdaBoost, where your weak learning algorithm at each stage directly minimizes the weighted classification error over all possible decision stumps (there are only 5 viable classifiers to check). At each step, compute the weighted error of your newly selected classifier, the resulting classifier weight alpha, and the new example weights to be used at the next stage. Show how you computed these values.

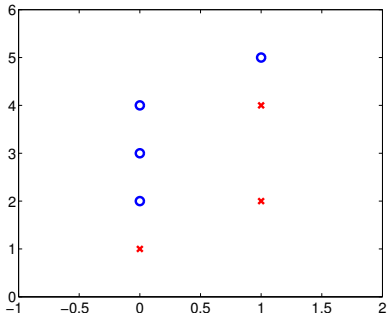

Figure 1: Data for AdaBoost problem.

If you compute the error rate of the overall (ensemble) classifier (not required), you will notice that it does not improve between the first and second weak learners. (It does improve on the 3rd.) Why is this the case?