CSE 417T: Homework 6

Due: 22nd of Nov 2016 – 10am

Notes:

- There are 3 problems on 4 pages in this homework.
- This homework consists of two parts.
- **Written part:** Problems 1 and 2 (**no** group work)
  - Submit all written answers by committing a single pdf file named **`YOUR_WUSTL_KEY_hw6.pdf`** to the `hw6` folder in your SVN repository.
- **Implementation part:** Problem 3 (you may work in groups of 2)
  - You may work in pairs on the **implementation part**. Each group should turn in the source code only once. Please do not collaborate with anyone other than your partner. Indicate group work by adding both wustlkeys separated by a newline to the `partners.txt` file and commit this file to both repositories (partnerships must me mutually accepted!).
  - Stubs for your implementations are in your SVN repository. Your implementations will be autograded. **Follow the submission instructions exactly.** Do **not** add additional functions and do **not** change the names or signatures of the functions as this will lead the autograder to fail in which case you cannot get credit!
  - Comment your code properly.

Autograder: The autograder for this class is used for grading only. **It is not meant to be used for debugging.** We will run it after the submission deadline. We offer **one** *autograder deadline* prior to the submission deadline, where we grade your work and give you the feedback of the autograder. This deadline is optional, but highly recommended!

- **THU 17th of Nov 10am: deadline for optional autograder feedback**
- **TUE 22nd of Nov 10am: final submission deadline**
- You may use up to two **late days** from your late day contingent for this submission. (No extensions beyond this!)
- Keep in mind that Nov 23rd - 27th is **Thanksgiving break** (**no** office hours and limited Piazza question answering.)

If you have any questions with respect to the autograder, ask on Piazza using the **autograder tag**.

Problems:

1. **(15 points) Entropy for Decision Tree Learning**

   When deciding which split (feature and category or threshold) to use for the nodes when learning a decision tree, we aim to maximize label purity in the child nodes. One way to achieve this goal is to **maximize the divergence from the uniform label distribution**.

   (a) Show that maximizing the divergence of the label distribution in the child nodes to the uniform distribution is equivalent to **minimizing entropy**. HINT: use KL-divergence (Kullback-Leibler Divergence) to quantify the distance between two discrete probability distributions.

   (b) In the lecture we **maximized information gain** to decide on the splits. Show that maximizing information gain is equivalent to minimizing entropy. HINT: you can assume binary splits.

   (c) What is the time complexity to find the best split using entropy assuming *binary features*? What is the time complexity assuming *continuous features*?

2. **(35 points) Efficient Decision Tree Learning**

   You are building a regression tree (CART), and your recursive function is called with data $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$ (where we have *continuous attributes* $x_i \in \mathbb{R}^d$ and *continuous labels* $y_i \in \mathbb{R}$). For a leaf let the prediction be the average predictor $p \leftarrow \bar{y} = \frac{1}{|S|} \sum_{(\vec{x}_i, y_i) \in S} y_i$, where S are all datapoints in that leaf. We use *label variance* as impurity measure.

   (a) Show that p minimizes the average squared loss $\mathcal{L}(D) = \frac{1}{|D|} \sum_{(\vec{x}_j, y_j) \in D} (y_j - p)^2$. Explain why this must be the case.

   (b) If the termination conditions are not met (i.e. you do not create a leaf), you need to split. For this, you need to find the best split for any feature f. Assume we have sorted our data set according to some feature f, such that $f_1 \leq f_2 \leq \cdots \leq f_n$, where f_i is the value of feature f in example $\vec{x}_i$. Let all relevant splits be $c_1 \leq c_2 \leq \cdots \leq c_{n-1}$, for example $c_i = \frac{f_i + f_{i+1}}{2}$.

      i. Define the predictors $\bar{y}_L^i, \bar{y}_R^i$ of the left and right sub-trees, after cutting at c_i (left $\leq c_i$) in terms of $y_1, \ldots, y_n$.

      ii. Write down the expressions for the loss $\mathcal{L}_L^i$ and $\mathcal{L}_R^i$ on the left and right sub-trees, after cutting at c_i. What is the time complexity of computing both terms for any cut point c_i?

      iii. Express $\bar{y}_L^{i+1}, \bar{y}_R^{i+1}$ in terms of $\bar{y}_L^i, \bar{y}_R^i$ and y_{i+1}.

      iv. Let $s^i = \sum_{j=1}^{i} y_j^2$ and $r^i = \sum_{j=i+1}^{n} y_j^2$, express $\mathcal{L}_L^{i+1}, \mathcal{L}_R^{i+1}$ in terms of $y_{i+1}, \bar{y}_L^i, \bar{y}_R^i, s^i, r^i$.

      v. Write a full update rule for iteration $i+1$, assuming you have $\bar{y}_L^i, \bar{y}_R^i, s_i, r_i, \mathcal{L}^i, \mathcal{R}^i$.

      vi. What is the time complexity to find the best split using *this method* assuming your data is already sorted? What is the time complexity for sorting D for all features? Compare the time complexity to *build an entire tree* for the approach developed in this problem to the time complexity of building an entire tree using entropy as in problem 1(c).

3. **(50 points) Implementation: Bagged and Boosted Decision Trees**

In this assignment you will implement a decision tree algorithm and then use it for bagging and boosting. Update your SVN repository. All of the following stub files are in the `hw6` folder:

Stubs:	
`id3tree.m`	Returns a decision tree using the minimum entropy splitting rule (implemented for you in `entropysplit.m`)
`evaltree.m`	Evaluates a decision tree on a test data.
`prunetree.m`	Prunes a tree using a validation set.
`forest.m`	Builds a forest of id3trees.
`evalforest.m`	Learns and evaluates a forest.
`boosttree.m`	Applies adaboost to your id3tree.
`evalboost.m`	Learns and evaluates a boosted classifier.

Already implemented:	
`entropysplit.m`	This function implements minimum entropy splitting using information gain/entropy.
`cart_tictoc.m`	This function computes train and test accuracy and runtime for all your algorithms (uses `analyze.m`).
`cart_visual.m`	This function visualizes trees (uses `vistree.m` and `scat.m`).

Datasets:	(download from Piazza resources)
`iris.mat`	Subset of the Iris flower dataset (`https://en.wikipedia.org/wiki/Iris_flower_data_set`)
`heart.mat`	Another dataset for binary classification.

As MATLAB does not support pointers and is really only good with matrices, we will represent a tree through a **matrix** T. A tree T with q nodes must be of dimensions $6 \times q$, where each column represents a different node in the tree. The very first column is the root node and each column represents the following information:

(1) prediction at this node
(2) index of feature to cut
(3) cutoff value c ($<= c$: left, and $> c$: right)
(4) index of left subtree (0 = leaf)
(5) index of right subtree (0 = leaf)
(6) parent (0 = root)

`entropysplit.m` searches through all features and returns the best split (feature, cut-threshold combination) based on information gain (in this case entropy). You don't need to implement this, it is already done for you in the file `entropysplit.m`. Please take a minute and familiarize yourself with the implementation. It will make subsequent tasks easier.

(a) Implement the function `id3tree.m` which returns a decision tree based on the minimum entropy splitting rule. The function takes training data, test data, a maximum depth, and the weigh of each training example (used in entropy splitting). You can

visualize your tree with the command `visualhw7`. Maximum depth and weight are optional arguments. If they are not provided you should make maximum depth infinity and equally weight each example. (Hint: To speed up your code make sure you initialize the matrix T with the command `T = zeros(6, q)` with some estimate of q.) You should use the function `entropysplit.m`

(b) Implement the function `evaltree.m`, which evaluates a decision tree on a given test data set. You can test the accuracy of your implementation with `hw7tictoc`.

(c) Decision trees (without depth limit) are **high variance** classifiers. Consequently, overfitting is a serious problem. One cure around it is to prune your tree to stop making "silly" splits that overfit to the training set. You prune a tree bottom up, by removing leaves that do not reduce the validation error (**reduced error pruning**). Implement the function `prunetree.m`, which returns a decision tree pruned for a validation data set. The accuracy on the validation set should be the same or higher after pruning.

(d) A more effective way to prevent overfitting is to use **bagging**. Implement the function `forest.m`, which builds a *forest* (**not** a random forest) of ID3 decision trees. Each tree should be built using training data drawn by randomly sampling n examples from the training data with replacement, you can use MATLAB's `randsample` function to do this. (Do **not** randomly sample features.)

(e) Implement the function `evalforest.m`, which evaluates a forest on a test data set. Remember that random forests make predictions by majority vote.

(f) Another option to improve your decision trees is to build trees of small depth (e.g. only depth=3 or depth=4). These do not have high variance, but instead suffer from **high bias**. You can reduce the bias of a classifier with **boosting**. Implement the function `boosttree.m`, which applies ADABOOST on your `id3tree` functions.

(g) Implement the function `evalboost.m`, which evaluates an ensemble of boosted decision trees.

Once, you have implemented all functions you can run `cart_tictoc.m`, which computes training and test error as well as runtime for all four algorithms. Try different datasets (you may also use the ones from the previous homeworks[1].) and play around with the input parameters of your forest and Adaboost methods.

Submit your implementation by committing **all your functions** and the **`partners.txt`** file to the HW6 folder in your SVN repository.

[1] If you want to use `digits` or `faces`, you will need to adapt your boosting algorithm for multi-class classification. (Not required for the homework.)