CS 534: Machine Learning
Homework 3

(Due Oct 20th at 11:59 PM on Canvas)

Submission Instructions: The homework should be submitted electronically on Canvas. The code can be done in any language that `Euler` supports.

1. **(8+7=15 pts) Multi-class exponential loss**

   For a K-class classification problem, consider the coding $\mathbf{y} = \begin{bmatrix} y_1, & \cdots, & y_K \end{bmatrix}^T$ with

   $$y_k = \begin{cases} 1, & \text{if} G = g_k, \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases}$$

   Let $\mathbf{f} = \begin{bmatrix} f_1, & \cdots, & f_K \end{bmatrix}^T$ with $\sum_{k=1}^K f_k = 0$ and define the generalization of the exponential loss function to the multi-class case as:

   $$L(\mathbf{y}, \mathbf{f}) = \exp\left(-\frac{1}{K}(y_1 f_1 + \cdots + y_K f_K)\right) = \exp\left(-\frac{1}{K}\mathbf{Y}^T\mathbf{f}\right).$$

   Note that when $K = 2$, this reduces to the 2-class exponential loss that you are familiar with $L(y, f) = \exp(-yf(\mathbf{x}))$.

   (a) Using Lagrange multipliers, derive the population minimizer $f^* = \text{argmin}\, E_{Y|\mathbf{x}}[L(\mathbf{y}, \mathbf{f})]$ subject to the zero-sum constraint. Hint: Write the constrained problem and turn it into an unconstrained problem using Lagrange multipliers. You can and should relate these to the class probabilities (by removing the expectation). Then take the derivatives with respect to f_k and λ, the Lagrange multiplier. Finally solve the set of equations to obtain the population minimizer. What are the class probabilities as a function of the population minimizer?

   (b) Show that a multi-class boosting using this loss function leads to a reweighting algorithm similar to AdaBoost. Note that the error remains the same, and the thing that changes is $\alpha^{(m)}$. What is this new $\alpha^{(m)}$ for the multi-class problem? Hint: You'll want to find $\alpha^{(m)}$ the same way it was done for AdaBoost. What can you say about the new algorithm in terms of how it views misclassified points?

2. **(2+8+2+3=15 pts) Decision Tree to Predict Census Income**

   We will consider the Adult Dataset (Blake and Merz, 1998) available on UCL ML repository (`http://archive.ics.uci.edu/ml/datasets/Adult`) which was extracted from the 1994 Census database. The prediction task is to determine whether a person makes over 50K a year. Out of 48,842 samples, about 24% are positive (income $> 50K$). Some of the features have missing values (marked by '?') and there are 14 features (4 categorical and 10 continuous). Use your favorite decision tree software module to predict a person's income.

   (a) Pre-processing: Split the data into a train / test split (justify your selection). Since most decision tree implementations will not readily handle missing values, we will use a very simple method to fill in missing values. For continuous values, you can substitute with the `mean` or `median` value of the particular feature. For categorical values, substitute

with the `mode`. Calculation of these statistics should only be done on the training data. You should also do any additional pre–processing that maybe necessary beyond this – justify whatever you do.

(b) There is debate amongst the community about the use of pruning to control overfitting. Instead, you will tune the decision tree by limiting the depth of the decision tree and the minimum number of samples that each leaf must have (this maps to the `scikit-learn` parameters `max_depth` and `min_samples_leaf`, respectively). Generate a plot that captures the estimated predictive performance using only the training data by varying the two parameters with max depth ranging from 1 to 30 and minimum number of samples ranging from 1 to 50. Plot the estimated accuracy on a single plot (either do a 3D plot or use size of the point to encode the classification accuracy) as a function of the two parameters. What is the best choice of the two parameters?

(c) Evaluate the decision tree performance on the test set using the optimal max depth and minimum number of samples from (b).

(d) Create a visualization of the top 3 levels of your decision tree using the optimal max depth and minimum number of samples.

3. **(20+10+5=35 pts) Gradient Boosting to Predict Blog Feedback**

Consider the BlogFeedback dataset from Homework #1. As a reminder, each sample contains information from the blog post and the goal is to predict the comments the post received in the next 24 hours relative to a basetime. The dataset is split into three subsets: training data based on blog posts from 2010 and 2011, validation data with blog posts in February 2012, and test data with blog posts from March 2012. There are 280 features for each blog post, which are described in detail on the UCL ML repository, BlogFeedback dataset. For this problem, you will implement a gradient boosting algorithm with shrinkage to explore the effects of overfitting. The big change is that your model prediction is now $y^{(t)} = y^{(t-1)} + \nu f_t(\mathbf{x}_i)$. The algorithm is summarized in Algorithm 1.

Algorithm 1 Gradient Boosting with shrinkage

1: Initialize $f_0(\mathbf{x}) = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \phi(\mathbf{x}_i, \gamma))$
2: **for** $m = 1 : M$ **do**
3: Compute gradient residual for all i, $r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x_i})}\right]_{f(\mathbf{x}_i)=f_{m-1}(\mathbf{x}_i)}$
4: Fit regression model, h_m to residual $\mathbf{r}_m$
5: Update model $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu h_m$
6: **end for**
7: Return $f(\mathbf{x}) = f_M(\mathbf{x})$

(a) Implement the gradient boosting algorithm using favorite decision tree implementation. Your boosting model should support the following parameters:

- nIter: number of boosting iterations
- ν: shrinkage parameter
- loss: the loss function in the form of mean squared error and mean absolute error.

(b) For number of boosting iterations between 5, 10, 15, 25, plot the validation error as a function of the parameter $\nu \in [0, 1]$ for the two types of loss. Note that generally speaking, $\nu = 0.1$ is a common one to test. What can you draw from the plots in terms

of optimal parameters and the use of the shrinkage parameter? How do they relate to one another?

(c) For the optimal parameters ν and boosting iterations for each of the loss function, how well does your model do on the test data?

4. **(20 + 7 + 4 + 4 = 35 pts) Almost Random Forest to Detect Thyroid Disease**

We will be using the hyperthyroid dataset `allhyper.data`, which is a subset of the Thyroid Disease Data Set found at `http://archive.ics.uci.edu/ml/datasets/Thyroid+Disease`. The problem is to determine whether a patient referred to the clinic is hypothyroid. The original problem is not a binary classification and contains negative (not hypothyroid), hyperthyroid, and some subnormal functioning. For the purpose of the homework, you will want use hyperthyroid as the positive class and the others as the negative class. Partition the data into 70%–30% train–test split that you will use for this problem. You will implement an adaptation of the random forest for detecting thyroid disease. Instead of subsetting the features for each node of each tree in your forest, you will choose a random subspace (i.e., limit the attributes to consider as the square root of the total number of features) that the tree will be created on. This allows you to use existing decision trees without having to build your own.

(a) Build the adaptation of the random forest using your favorite language. You will want to make sure your forest supports the following parameters:

- nest: the number of trees in the forest
- criterion: the split criterion – either gini or entropy
- maxDepth: the maximum depth of each tree
- minSamplesLeaf: the minimum number of samples per leaf node

In addition, your forest should be able to provide the out-of-bag (OOB) sample accuracy, the importance of the features, in addition to prediction.

(b) For nest = [10, 25], find the best parameters for the split criterion, maxdepth, and minsamples leaf – justify your selection with a few plots.

(c) Using your optimal parameters, how well does your version of the random forest perform on the test data? How does this compare to the OOB sample accuracy?

(d) What are the most important features from your model?