CS 534: Homework #4

Submission Instructions: The homework is due on Nov 3rd at 11:59 PM ET on Gradescope. A part of your homework will be automatically graded by a Python autograder. The autograder will support Python 3.10. Additional packages and their versions can be found in the `requirements.txt`. Please be aware that the use of other packages and/or versions outside of those in the file may cause your homework to fail some test cases due to incompatible method calls or the inability to import the module. We have split the homework into 2 parts on Gradescope, the autograded portion and the written answer portion. If *either of the two parts is late, then your homework is late.* Format will be the same as the previous 3 assignments.

1. **(10 pts) AdaBoost Update**

   (Written) Derive the expression for the update parameter in AdaBoost (Exercise 10.1 in HTF).

   $$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

2. **(2+3+5+15+5+5=35 points) Predicting Loan Defaults with Neural Networks** Consider the Loan Defaults dataset from Homework #3, `loan_default.csv`. You will be using neural networks to predict whether or not a customer will default on a loan. Note that neural networks can be quite expensive so you might want to use a beefier machine to do this. Your function for 2c should be in a 'q2.py' file.

   (a) (Written) How did you partition the loan data to assess the model performance and choose the hyperparameter? You will find it useful to use the same method from Homework #3 as you will be asked to compare against the decision tree.

   (b) (Written) Preprocess the dataset for neural networks. What did you do and why? You can use your experience from Homework #3 and what you learned about neural networks to guide your choice here.

   (c) (Code) Write a function `tune_nn(x, y, hiddenparams, actparams, alphaparams)` that takes in a numpy 2d array of features, x, a numpy array of labels, y, a list of hidden layer sizes, hiddenparams, a list of activation functions, actparams, and a list of l2 regularization parameters, alphaparams, and determines the optimal parameters for a neural network based on grid search and the AUC metric. Each list element in hiddenparams will be a tuple where each ith element represents the number of neurons for that hidden layer. For example, (100, 25) means the first hidden layer has 100 neurons and the second hidden layer has 25. Each list element in actparams is equivalent to the activation function used for the hidden layer and can take on the values 'logistic', 'tanh', and 'relu'. Your function should return back a dictionary with the following keys 'best-hidden', 'best-activation', 'best-alpha'. The dictionary can also have other keys but these 3 must be present.

   (d) (Written) Build a neural network on your dataset using your code from 2c. What is your search space for the neural network hyperparameters? What are the optimal hyperparameters?

   (e) (Written) Evaluate the best neural model using the hyperparameters from 2d on your test data. What are the AUC, F_1, and F_2 scores for the neural network and the decision tree from Homework #3? Make sure you report the results in a Table and that the results are for the same test set(s) for both models!

(f) (Written) How does the neural network compare against the decision tree? Make sure to comment on the performance, the computational complexity (i.e. runtime), and the ease of parameter tuning. To measure the runtime of the neural network or the decision tree, you may find it helpful to use the `time` module. To find the execution time of a block of code, you can subtract the end time from the start time.

```
import time
start_time = time.time()
# block of code to time
duration = time.time() - start_time
```

3. **(5+15+10+5+5+5+5+5=55 pts) Stochastic Gradient Tree Boosting to Predict Appliance Energy Usage**

Consider the Energy dataset (`energydata.zip`) from Homework #1. As a reminder, each sample contains measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station, and the recorded energy use of lighting fixtures to predict the energy consumption of appliances in a low energy house. For this problem, you will implement a variant of stochastic gradient tree boost to predict the energy usage [1]. There are two major changes from the gradient tree boosting algorithm that was discussed in class: (1) each model is dampened by a factor ν (i.e., model prediction is $y^{(t)} = y^{(t-1)} + \nu f_t(\mathbf{x}_i)$), and (2) there is random subsampling of the dataset (hence the name stochastic).

You have been given the template code, 'sgb.py', which takes in the following 4 parameters.

- $nIter$: number of boosting iterations (non-negative integer)
- ν: shrinkage parameter (float $\in [0, 1]$)
- q: subsampling rate (float $\in [0, 1]$)
- md: max depth of the tree that is trained at each iteration (≥ 1)

(a) (Code) Implement the function `compute_residual` where given a set of predictions, yhat, and the true label, y, it will compute the gradient residual for all i in the sample.

(b) (Code) Implement the function `fit` where given a set of features, x (a numpy 2-D array), and the labels, y (a numpy 1-D array), it will learn the model according to Algorithm 1. The `fit` function is set to return *itself*. This is designed so that it will work naturally with `GridSearchCV` from `sckit-learn`). Within your `fit` function, it *must update* the `train_dict` such that the dictionary contains the following key-value format: key is the iteration and the value is the RMSE of the training data. Note that for the 0th iteration, it should return the RMSE for a model that predicts the mean of y and the 1st iteration would return the RMSE for a single tree fit to the initial set of residuals multiplied by the shrinkage rate.

(c) (Code) Implement the function `predict` where given a new set of samples, x (a numpy 2-D array), predicts the label (a numpy 1-D array) according to the final learned model.

[1] In gradient tree boosting, you re-calculate the weight of each terminal tree node. This is not easy to do if you use most standard tree implementations (i.e., `scikit-learn`), and thus has been omitted for implementation ease. The sacrifice is that your implementation is less likely to do as well as other packages.

Algorithm 1 Stochastic Gradient Tree Boosting with Shrinkage for Regression

1: Initialize $f_0(\mathbf{x})$ to target mean
2: **for** $m = 1 : M$ **do**
3: Subsample training set at rate q
4: Compute gradient residual for all i in subsample, $r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x_i})}\right]_{f(\mathbf{x}_i)=f_{m-1}(\mathbf{x}_i)}$
5: Fit tree model h_m to residual $\mathbf{r}_m$
6: Update model $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu h_m$
7: **end for**
8: Set the final learned model: $f(\mathbf{x}) = f_M(\mathbf{x})$

(d) (Written) Using all the training data ($q = 1$), plot the validation RMSE as a function of the parameter $\nu \in [0, 1]$ and the number of boosting iterations (you can choose however you plan to present the information, whether using different lines, 3D plot, etc.) Note that $\nu = 0.1$ is a common parameter value of ν and maybe helpful to have in your grid search. What conclusions can you draw from the plots in terms of how the shrinkage parameter relates to the number of boosting iterations? What would be optimal parameters for $q = 1$?

(e) (Code) Implement the helper function `tune_sgtb` where given list of boosting iterations, a list of shrinkage parameters, a list of subsampling rates, and a single max depth, identifies the best parameters based on the list. Your function should return a dictionary with the keys "best-nIter", "best-nu", and "best-q" and the optimal single value for each hyperparameter. Your dictionary can have other keys but must have these 3 keys (you may find it helpful to read 3f to see if there are other key-value pairs that you might want to return).

(f) (Written) Tune for the best validation RMSE using 3e including different subsampling rates ($q \in [0.6, 1]$), ν, and M by reporting the results in a table. You can use 3d to reduce the parameter search for ν and M (convince us that there might be certain values we can consider omitting and how you decided it). What are the optimal parameters for all three?

(g) (Written) For the optimal parameters you found in 3f, train your "final model". What is the test error? How does this compare to your results from homework #1 (please report your performance on Homework #1 for grading convenience)?

(h) (Written) Comment on the stochastic gradient tree boosting (i.e., $q \neq 1$) versus gradient tree boosting (i.e., $q = 1$) in terms of computation time (i.e., runtime), performance results, and hyperparameter tuning.