EMORY UNIVERSITY

Department of Computer Science & Informatics

Masoud Nateghi

CS534 HW4

Professor Joyce Ho

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

By employing the exponential loss function, we can determine the parameters for the additive basis function at the m-th iteration through the use of the following formula:

$$(\beta_m, G_m) = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta G(x_i))] = \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta y_i G(x_i))$$

Then we first find the solution for optimal G. It can be written for any $\beta > 0$ as:

$$G_m = \underset{G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

We use the G_m which we calculated above and plug into the first equation. G_m predicts labels as either 1 or -1. Thus, $y_i G(x_i)$ is either +1 when $y_i = G(x_i)$ or -1 when $y_i \neq G(x_i)$.

$$\beta_m = \underset{\beta}{\operatorname{argmin}}\, e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

From above formula, we take partial derivative with respect to β and then setting it to be zero.

$$-e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)} = 0$$

$$e^{2\beta} = \frac{\sum_{y_i = G(x_i)} w_i^{(m)}}{\sum_{y_i \neq G(x_i)} w_i^{(m)}} \rightarrow \beta_m = \frac{1}{2} \log\left(\frac{\sum_{y_i = G(x_i)} w_i^{(m)}}{\sum_{y_i \neq G(x_i)} w_i^{(m)}}\right)$$

Now we define err_m to be:

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$$

Thus, we can write:

$$\beta_m = \frac{1}{2} \log\left(\frac{1 - err_m}{err_m}\right)$$

2. (a) 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.

Following the procedure in HW3, I initially divided the data into training and test datasets, randomly selecting **20%** of the entire dataset for the **test** set. To fine-tune the hyperparameters, I employed grid search, utilizing **5-fold cross-validation**. For further comparison of the models in HW3 and HW4, I used a **random seed = 42**.

2. (b) 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.

In the initial phase, I addressed the problem of **categorical features** in the dataset by employing suitable encoding methods such as **ordinal encoding and one-hot encoding**. This process resulted in a total of **41 features**. Furthermore, I used the 'earliest_cr_line' column, indicating that individuals with a strong (earlier) credit history are more likely to meet their loan obligations. To facilitate this, I conducted straightforward preprocessing and determined that the latest year in this column was 2011. Consequently, I computed the number of months from the 'earliest_cr_line' entries to December 2011.

Following this, the dataset was **divided into training and testing sets**. Utilizing standard scaling, the features were **normalized and scaled to have a mean of zero and a variance of one**. Subsequently, I computed the **correlation matrix between the features** to identify and **remove one of the features from any highly correlated pair**. Columns 0, 2, 6, 10, 12, 19, 21, 26, and 28 were consequently removed resulting in a total of **32 features**.

Subsequent to this, I computed the **correlation between the features and labels** to **select features displaying strong correlations**, whether positive or negative. Such features were retained for the subsequent analysis, as a strong correlation with the label can indicate their informative and practical nature. Ultimately, 25 features with the highest positive or negative correlations were selected, resulting in a total of **25 features** after the preprocessing stages.

Lastly, I employed the **PCA** (Principal Component Analysis) technique to **decorrelate** the data, ultimately aiming to achieve an identity matrix for the covariance matrix. This involved using the 'fit_transform' method from the PCA on the training data, followed by the application of the 'transform' method from the PCA on the test data.

2. (d) 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?

We explored various configurations for the **hidden layer sizes**, including **(10, 30, 10), (100, 50), (40, 20), (20,), and (40,)**. Similarly, we experimented with different **activation functions** such as **ReLU, tanh, and logistic**. Additionally, we varied the **regularization parameter** across a range of values, namely **[0.001, 0.01, 0.1, 1, 10]**. Our configuration also involved setting **'max_iter' to 100, 'solver' to 'adam', and 'learning_rate' to 'adaptive'**.

Following the implementation outlined in section 2c, we derived the **optimal hyperparameters** as follows:

- Hidden layer sizes: **(100, 50)**
- Activation function: **logistic**
- Regularization parameter: **0.01**

2. (e) Evaluate the best neural model using the hyperparameters from 2d on your test data. What are the AUC, F1, and F2 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!

We employed **'random_state=42**' in the 'train_test_split' function in both codes for HW3 and HW4. This approach ensured an identical data partitioning for both models, resulting in the **same test dataset for comparison**. The table on the left displays the optimal results for the decision tree models, where we set 'max_depth' to 3 and 'min_samples_leaf' to 2. Meanwhile, the table on the right showcases the performance of the neural network, utilizing the hyperparameters as stated in 2d.

DT	AUC	F1-score	F2-score
Train	0.7220	0.6150	0.4996
Test	0.7382	0.6454	0.5321

NN	AUC	F1-score	F2-score
Train	0.7238	0.6176	0.5147
Test	0.7336	0.6356	0.5338

We observe that the **performance** of the two models is **quite similar**. Thus, for this dataset, neither model exhibits clear superiority over the other.

2. (f) 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.

As demonstrated in 2e, the **performance** of the two models is **closely the same** for this dataset. However, it is important to note that **Neural Networks** entail significantly **higher computational complexity** compared to Decision Trees. Utilizing the 'time' module, we measured the duration required for training and prediction with both the Decision Tree and Neural Network models. Our findings revealed that the Decision Tree model only took 15 ms, whereas the Neural Network model took 1.54 s, making it approximately **100 times slower** than the Decision Tree.

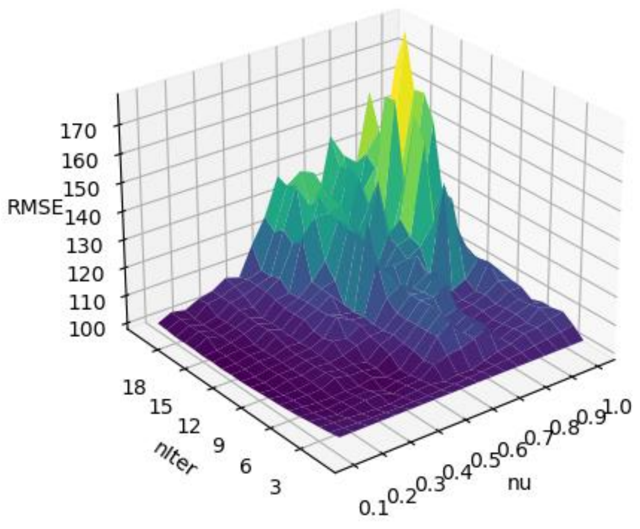
Furthermore, **Neural Networks** involve a **multitude of parameters**, including the number of hidden layers, the number of hidden units, the choice of activation function, learning rate, optimizer, and regularization parameter. These parameters far exceed the hyperparameters typically associated with a Decision Tree model, contributing to the intricacy and potential challenges of tuning Neural Networks.

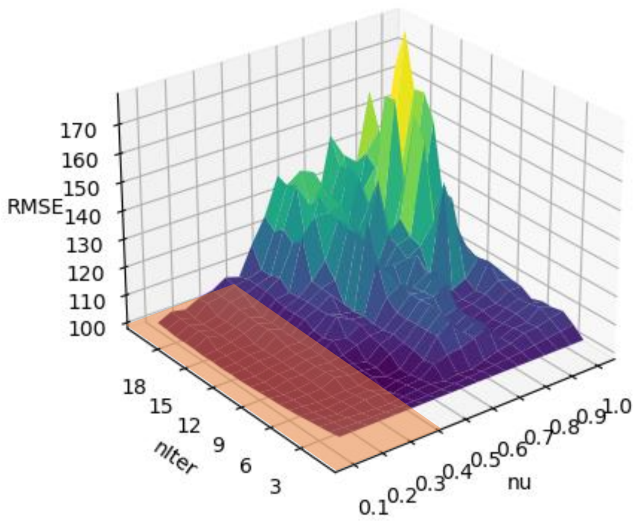
3. (d) Using all the training data (q = 1), plot the validation RMSE as a function of the parameter ν ∈ [0, 1] and the number of boosting iterations (you can choose however you plan to present the information, whether using different lines, 3D plot, etc.) Note that ν = 0.1 is a common parameter value 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?

We employed nIter values ranging from 1 to 21 and ν values from 0.1 to 1 to monitor the changes in RMSE concerning these parameters. A 3D plot was generated to illustrate the RMSE variations with respect to nIter and ν, considering a fixed max depth of 3 and q of 1.

The left figure clearly indicates that an increase in both ν and nIter is likely to result in worse model performance. However, the dark blue region identified in the figure represents a safe area, suggesting a reasonable model performance. Thus, it would be advisable to focus the search within the range of ν = (0, 0.4] and nIter = [1, 20] as highlighted in the right figure for the hyperparameters.

Utilizing grid search, we determined the optimal hyperparameters to be: nIter = 5, ν = 0.4, q = 1, and max depth = 3.





The table presented below illustrates the performance of the model, utilizing the optimal hyperparameters, on both the train and test datasets, measured by the RMSE and R2 metrics.

GTB	RMSE	R2
train	94.05	0.1890
test	93.52	-0.0591

3. (f) Tune for the best validation RMSE using 3e including different subsampling rates (q ∈ [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?

As discussed earlier in the 3D analysis, we can **limit our search area**, as shown in the plot on the right, to the range of ν = (0, 0.4] and nIter = [1, 20]. Presently, we will expand our search to include q values within the range of [0.6, 0.7, ..., 1].

By employing grid search, we identified the optimal hyperparameters as follows: nIter = 6, ν = 0.4, q = 0.8, and max depth = 3.

3. (g) 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)?

The table below displays the performance of the SGTB model with optimal hyperparameters on both the train and test datasets, utilizing RMSE and R2 metrics. We compared these results with the best model from HW1, which was Lasso linear regression with λ=822. Upon comparison, it is evident that **ridge regression outperforms the SGTB model** on this dataset.

	RMSE	R2
SGTB train	93.11	0.2052
SGTB test	103.59	-0.2996
eval_ridge2 (λ = 822) train	96.74	0.2110
eval_ridge2 (λ = 822) test	**83.64**	0.1520

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

The table below summarizes the performance of GTB and SGTB on test data.

	RMSE	R2
GTB train	94.05	0.1890
GTB test	**93.52**	-0.0591
SGTB train	93.11	0.2052
SGTB test	103.59	-0.2996

The **Gradient Tree Boosting (GTB) demonstrates superior performance on the test data** for this dataset, primarily because it is **trained using the entire dataset** and not just a portion of it as the case in SGTB. However, the dataset exhibits a notably poor structure due to the disparity between the records in the train and test datasets, which belong to entirely different periods of the year. Specifically, the train data comprises records from 1/11/2016 to 5/7/2016, while the test dataset contains records from 5/7/2016 to 5/27/2016, exhibiting no overlap and representing distinct periods of the year.

Stochastic Gradient Tree Boosting introduces an **additional hyperparameter, q**, compared to the Gradient Tree Boosting method. Consequently, **Gradient Tree Boosting**, with its **fewer hyperparameters**, is **more favorable** in terms of the effort required to fine-tune these parameters. Additionally, the utilization of a **subsample** of the dataset in **Stochastic Gradient Tree Boosting** inherently **reduces runtime**, making it a **more suitable** choice for us in terms of computation time.