1 Introduction [20 points]

Group Members: Richard Qing Bao, Matthew (Min Hyuk) Kim, June (Joo Eun) Kim
Team Name: 5% tint

Results

- 33rd on 2008 private leaderboard
- 62nd on 2012 private leaderboard
- AUC score of 0.79238 for 2008
- AUC score of 0.76818 for 2012

Contribution

Richard fit and tested the Logistic Regression model to the given data set. He also found the optimal hyperparameters for this model by using grid search. In addition, he tried using Principal Component Analysis (PCA) to reduce the dimensionality of the data set. He wrote a script to compare the performance of different types of weak regressors.

Matthew fit and tested the Random Forest Regressor model to the given data set. He also found the optimal hyperparameters for this model using grid search. In addition, he implemented the model that takes the weighted average of the Logistic Regression, Random Forest Regressor, and Gradient Boosting Regressor. He found the optimal weight distribution that gave the best accuracy.

June first implemented and tested the Simple Gradient Boosting model. She also fit and tested the full Gradient Boosting Regressor to the given data set. She implemented a function that normalizes the output probabilities so that the values are between 0 and 1. In addition, she optimized Gradient Boosting Regressor by finding the best hyperparameters with grid search and creating tests of her own.

2 Overview [20 points]

We started off by examining Logistic Regression. Then we used Random Forest Regressor, Simple Gradient Boosting (with 100 weak regressors), and Gradient Boosting Regressor. To test each of these models, we allocated thirty percent of our training data set to create a validation data set. In addition, we performed hyperparameter optimization on each of these models using Grid Search with cross validation. We also normalized our output probabilities so that the values would be between 0 and 1. We took an unique approach to construct our final model, which was the weighted average of Logistic Regression, Random Forest Regressor, and Gradient Boosting Regressor (all optimized with hyperparameters). Different weight distributions were tested to find the optimal model. We also attempted to use Principal Component Analysis (PCA) to reduce the dimensionality of the data set.

Day:	**Models / Approach:**
Day 1:	Logistic Regression
	Random Forest Regressor
	Simple Gradient Boosting
	Gradient Boosting Regressor
Day 2:	Grid Search on Logistic Regression
	Grid Search on Random Forest Regressor
	Grid Search on Gradient Boosting Regressor
	PCA
	Normalization of output
Day 3:	Weighted average of all 3 optimized models (Logistic Regression, Random Forest Regression, Gradient Boosting Regressor)
	Tested different weights on the model

3 Approach [20 points]

Data

It is useful to first inspect the data. There are 60000 data points in the training set, with a feature space of 381 fields. This feature space is highly complex, with many dimension and lots of noise. Early on, in an attempt to modify the training data to reduce stochastic noise, we considered a linear dimension reduction algorithm called Principal Component Analysis, which involves the application of an orthogonal transform on the input data. This produces an uncorrelated set of linear combinations that captures most of the variance in the data, essentially summarizing the data with only its most significant features. We used PCA to reduce the dimensionality of our feature space to a range of 10-40 principal components, but unfortunately this did not seem to improve the accuracy of our models. Perhaps the patterns in the data are very subtle and this reduction lost too much information. Another possibility is that correlations between features may be non-linear, and therefore linear dimension reduction was inadequate.

We also attempted to implement some unsupervised clustering techniques to help visualize the training data and discover more subtle patterns. For this, we used tSNE and k-means clustering. This clearly illustrated the large amount of noise in the data set, and from a visual perspective, there were no distinctly separable clusters. Another thing of interest in the dataset is the dominating presence of one class, which comprises around 75 percent of the dataset. With certain machine learning approaches, such as neural networks, this may cause unintended skewing in the predictions. This knowledge also provides a useful metric to evaluate the usefulness of our models, as random guessing based on this knowledge yields an accuracy of 75 percent.

Models

The target function maps the feature space to a probability, so clearly the learning task requires a regressive training method. The first model we implemented was simple logistic regression. The advantage of this model is its robustness. Logistic regression is intuitive, fast, and effective even if the signal is subtle or there is a large amount of noise present. It does not assume normal distribution of independent variables or equal or homogenous variance. The parameters interest are the regularization factor and the loss function.

The next type of regressor we tested was the regressive decision tree, implemented from the sklearn library. The decision tree is a natural choice for this learning task, as values in the feature space are highly categorical, often comprising of a few distinct values instead of continuous distributions. Like logistic regression, decision trees easily deals with nonlinear correlations in the data as well. However, the drawback of the decision tree its tendency to overfit on the training data. Furthermore, Information gain in a decision tree with categorical variables gives a biased response for attributes with a greater number of categories. Unfortunately, this effect may have been especially persistent due to the complexity of the training data and the high level of noise pervasive throughout it.

To reduce the variance of the the decision tree and mitigate the risk of overfitting, we utilized bootstrap aggregation to generate a parallel ensemble of regressive decision trees, called a Random Forest. Each decision tree was independently trained on a distinct subset of the training data, and the overall model combined these separate results into a unified prediction. Therefore, the overall model is should be significantly more accurate than any individual decision tree. While in practice it is impossible to guarantee the

mutual independence of these decision trees, as they are trained from the same data, the introduction of randomness helps enforce this condition quite effectively. The parameters of interest are random_state and n_estimators.

In addition to the random forest regressor, we also tested a sequentially gradient-boosted ensemble of weak regressive decision trees. These trees are relatively shallow, so individually they are ineffective, but each additional tree corrects the error of the previous tree, contributing to the complexity and expressiveness of the overall model. Although the gradient boosted ensemble is potentially more accurate than the random forest regressor, it is more prone to overfitting, There were three parameters of interests: number of trees, depth of trees, and learning rate.

We considered several additional types of weak regressors for bootstrap aggregation, such as the Dummy Regressor and the K-Neighbors Regressor, and wrote a script to compare the accuracy of their predictions over a broad range of parameters. However, it was clear that the decision tree is the most effective weak regressor for bootstrap aggregation on our training data.

Finally, after testing these different models, it was evident that each had unique advantages and disadvantages. We realized that it is possible to combine these different models by computing a weighted average of their individual predictions. Our hope was that this would combine the strengths of each model and reduce the overall variance.

4 Model Selection [20 points]

Different optimization objectives were used for the different models that constitute our final model. For logistic regression, we used the L1 penalty because the results trained from the model with the L1 penalty were significantly better (around 0.1 to 0.2 in accuracy) than those from the model with L2. Mean squared error was used to optimize our random forest regressor model, since it uses variance reduction as feature selection criterion. Least squares regression was used for the gradient boosting regressor model, as it is the standard loss function for regression. We scored our models using the AUC metric so that the scores would roughly estimate the performance of our models on Kaggle. A combination of the three models (5% logistic regression, 20% random forest regressor, and 75% gradient boosting regressor) scored the best.

We split the given training data into a smaller training set and a validation set (with 7:3 ratio, randomized) to manually test our models. We also used 5-fold cross validation in our Grid Search. For each model, we manually tested a wide range of each hyperparameter using our validation set to reduce the range of effective hyperparameters, and then ran a Grid Search on a combination of the reduced set of parameters. Then we tested various weight combinations for averaging the three models using the validation set. From the Grid Search on each model, we were able to get the optimal combination of hyperparameters that improve the accuracy without overfitting. For example, for the Gradient Boosting Regressor, maximum depth of 6 and 300 estimators yielded the best results when tested separately, but a combination of maximum depth of 4 and 200 estimators yielded a better result than the model with just maximum depth of 6 (default n_estimators) or the model with just 300 estimators (with default max_depth). This shows how the combination of different parameters could significantly improve the model, with one parameter preventing the overfitting behavior of another. The minimum leaf size parameter was also considered for preventing overfitting, but using maximum depth as a means to prevent overfitting yielded better results.

5 Conclusion [20 points]

The top 10 features that have the most influence on the prediction target are ['PEIO1COW' 'PRMJIND1' 'PEHSPNON' 'GEREG' 'PRCOW1' 'PRMARSTA' 'HETENURE' 'PRTFAGE' 'PEDISOUT' 'PEMARITL'], as our model has the largest weights ([-0.2679067239282478, -0.2707093268351434, -0.27101811337563536, -0.3104934944929694, 0.3535916748020343, 0.3583242299760499, 0.3588392226912022, 0.372430616609862, -0.3973753863829447, -0.4139749861607789]) for these features. 'PRCOW1', 'PRMARSTA', 'HETENURE', and 'PRTFAGE' positively influence the prediction target, and the remaining features negatively influence the prediction target.

We used AUC as our kaggle competition metric because the given data had binary classifications (vote and did not vote). AUC is a good metric to use for binary classification because it deals with true positive rates and false positive rates. In addition, AUC can effectively deal with skewed sample distributions by not overfitting to a specific class. The given data set was skewed heavily (75 percent one class). We dont think there are better metrics for this project. AUC is the most appropriate because the data had binary classifications, output values were probabilities, and and the data set was heavily skewed.

Grid Search can be parallelized when the estimator being parallelized is picklable. We can specify the number of processes to parallelize using the n_jobs parameter in sklearn's GridSearchCV. This parallelism mostly happens in fitting multiple copies of estimators in Grid Search, but it can also be used when predicting, such as when our estimator is Random Forest. Random forest regressor can be parallelized by training a certain number of trees in parallel. We incorporated this into our models by setting the number of jobs parameter to a value greater than equal to 1. This allows use to use concurrent threads to train our trees.

In lectures and homework, we learned about the strengths and weaknesses of each individual model. In this project we learned that combining different models could give us a model better than each of them alone, as it could prevent our results from relying on a single model that potentially overfits the training set. We learned that this allows us to focus on the accuracy of each model without too much regularization, as the averaging method resolves overfitting to some degree.

Overall, we were able to see how we can put together different concepts we learned in class to create a working model in practice. It was interesting to see that the model accuracy that we estimated using our validation methods were in fact representative of the actual accuracy on test methods once we submitted on Kaggle. We also learned the limitation of predicting the future using machine learning, as we did poorly on 2012 dataset, but we were impressed that the most robust models by other people were able to perform decently on the 2012 dataset as well despite the limited information all of us were given.

An obstacle we encountered during the process was the time inefficient nature of grid search. We ran grid search on each of our possible models to optimize each models hyperparameters. For Gradient Boosting Regressor, we tested an extensive number of parameters which took a couple of hours. We could have further expedited the process by eliminating certain parameters that seemed negligible based on our training data set. In addition, we had difficulty reducing the dimensionality of our data set as the signal was lost during PCA. This may have been due to the fact that the features were non-linear (PCA is a linear dimension reduction). To account for the non-linearity, we could have utilized nonlinear dimension reduction techniques such as T-distributed Stochastic Neighbor Embedding (t-SNE).

Although we did fairly well on the 2008 dataset (both public and private), we did very poorly on the 2012 dataset. Since half the 2008 dataset was hidden as well, it seems that the poor performance of our model is not simply due to the data being hidden but also because we did not address the values of each input parameter shifting in 2012. We may have performed better if we trained on a normalized dataset for 2008, then predicted on a normalized dataset of 2012), such that each dataset is normalized around its own average and standard deviation.