Ultimatum Game with Preplay Contracting

Jonathan Hilgart
DSCI6003: Machine Learning I

To summarize, this analysis looked at a variant of the Ultimatum Game. This version of the ultimatum game involved two players , A and B, where each made a decision simultaneously in this game. Each player could choose to 'push', or give, a set amount to the other player ($6-$7) hoping that the other player would also 'push' ($6-$7) to them. Alternatively, a player could choose to 'pull', or take, money from the pot ($3-$4). In addition, in the second stage of this game players could offer a side payment to entice the opposite player to 'push' to them. Below is an overview of the algorithms used in the analysis of how to predict your payoff from this game.

K-nearest neighbors regression - (1)

For each data point, find the nearest neighbors to this data point using a distance metric (such as Euclidean distance). Find a best *k* of nearest neighbors, based on RMSE (can also look at silhouette plot or gap statistic here). For each neighborhood, calculate the average of the desired predicted value (can also incorporate a weighted average using points further away with less weight). This learns clusters of your data to predict new data off of.

GLM/Elastic net

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1).$$

(2)

The glm/elastic net algorithm takes a linear regression and adds a regularization terms to the coefficients (B). The goal here is to shrink the influence of any individual features to produce a more robust model overall (reduce variance) (3). Typically, there is a parameter, alpha, that would regulate how much of a L1 vs L2 error that you assign to the coefficients. In addition, there is typically a learning rate, lambda, that indicates how much influence the regularization term will have for the updated coefficients (B). This learns a linear decision boundary for your model.

Gradient Boosting

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)); \quad (4)$$

The idea of gradient boosting is that the combination of weak learners (these are typically decision trees that can predict the outcome slightly better than 50% of the time), can produce a strong learner. This is the idea behind boosting in general, that you train each new model on the errors of the previous model. For gradient boosting, you take the gradient of the loss function, multiplied by the learning rate, and then train the subsequent model on this error term. Usually, these models are decision stumps (stump due to the fact that tree depth is only one) because stumps are good ‘weak’ leaners.

Extreme Gradient Boosting

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant \quad (5)$$

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Extreme gradient boosting works in a similar fashion to gradient boosting. However, the biggest differences are three-fold. First, the objective function (above) is defined use the first derivative and second derivative of the loss function (of the actual values Yi and the predicted values from the previous time period Ybar. Second, extreme gradient boosting is much faster than gradient boosting because to find the best split points the algorithm first sorts the instances and then can scan through the instances in linear time to find the best split point. In addition, extreme gradient boosting adds a regularization term when determining the number of splits to make to form new leaves. If the gain from splitting is larger than the regularization term, then you will add another term to the tree. Otherwise, you are finished.

Random Forest

Random Forest employs a principal called bootstrap aggregating. (6) This is a technique of sampling multiple times from your data, building a model on each of these samples, and then aggregating the results of these models together. The combination of these models is used to reduce variance in the overall model. In particular, for this data set random forest probably worked well due to the fact that there is a lot of variance with a small data set. In addition, random forest takes longer to overfit that some other models; therefore, we can train this model with a large number of trees in an attempt to minimize test RMSE. (6)

Ensemble: Linear Model

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p + \varepsilon \quad \textbf{(7)}$$

Now, we have five fully trained models on the input data. The next step is to created a stacked generalization of these models (8). The first level is inputting the individual features into each model to create a prediction. Then, train a multiple linear regression model on each model's prediction to learn the bias of each model. For example, in the equation above x1 could be the predictions from KNNregression and B1 could be the bias associated with this model. Finally, using these models we can predict new data points linearly.

Meta-ensemble: Random Forest Model

In addition to a linear model combining the predictions from each model, I wanted to explore adding the original features alongside these predictions into a meta-ensemble learner. For this learner, I used a Random Forest to hopefully reduce the variance of the predictions. The steps here are as follows. First, create predictions from each model above. Then, combine these predictions with the original features (scaled for convenience of using KNN and GLMnet together). Finally, train a new meta-model of a Random Forest on these data points. Unfortunately, this approach did not work well for this data.

Resources

(1) https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
(2) https://en.wikipedia.org/wiki/Elastic_net_regularization
(3) Everitt B.S. (2002) Cambridge Dictionary of Statistics (2nd Edition), CUP. ISBN 0-521-81099-X
(4) Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine." (February 1999)
(5) http://xgboost.readthedocs.io/en/latest/model.html
(6) https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
(7) http://www.saedsayad.com/mlr.htm
(8) http://www.machine-learning.martinsewell.com/ensembles/stacking/Wolpert1992.pdf