Machine Learning Final Project - Team 4

Ricky Chen, Yifei Ren, Halle Steinberg, Robert Wei

Emory Goizueta Business School

Our project examines many different possible approaches that can be taken to predict the probability of a click for on-line advertisements. We tried multiple classification methods to determine a best model, including decision trees, logistic regression, neural networks and XGBoost. Before modeling, we did some data cleaning and manipulation on both our training and test data sets, including splitting our training data into smaller samples since the original data set contains over 30 million observations.

Data Preparation

With such a large data set, we understood that high dimensionality could be an issue while running our models. All the variables in our data are categorical, and categorical variables increase dimensionality by the number of categories minus one, whereas numerical variables only add to the overall dimensionality by one. Because of this, we wanted to explore further just how large the dimensionality of our data set could be, so we looked into the total number of unique values within each of our variables, as this would represent the total number of categories for that variable. We took this unique count and divided it by the total number of observations in the data and found that both ‘device_id’ and ‘device_ip’ had a very high percentage of unique values out of the total data. This showed us that each value is almost completely uniquely categorized, and having a unique category for nearly all observations would not add to the

predictive ability of our models. Because of this, we decided to remove these variables from any further analysis.

There were other high-dimensionality variables in the data that we still wanted to consider including in our models. We noticed that the variables ‘site_id’, ‘side_domain’, ‘app_id’, and ‘device_model’ had a moderately high number of unique categories, but based on our knowledge of what these variables might represent, we wanted to still consider their contribution to predicting a click. In order to keep these variables in our model, we attempted to reduce their dimensionality. By doing this, we could decrease any chance of over-fitting and reduce the overall computation time for our models.

To reduce dimensionality for our variables, we wanted to consider grouping categories together to decrease the number of unique categories we have. We decided that if we can reduce these categories down to a smaller number, then we could group everything else together into a general “others” category. We set the threshold at 20 total categories for each of our variables, and we chose to determine which 20 categories to keep based on those with higher-frequency counts. After looking at the distributions of our variables, we saw that the majority (around 80-85%) of our data was in about 20 categories in each variable. We first grouped each unique category within each variable and obtained a count of each category, then based on these counts, we chose the top 19 categories, and labeled the rest as “others”. These categories in “others” represent the less frequent categories for that variable, indicating a more unique and possibly rare representation of the entire population.

We also needed to consider the fact that the test set could have new categories for some variables. We wanted to make sure that we would still be able to include those variables in our models, so we needed to incorporate an “others” category for all variables in our training data

set. Ultimately, we decided to repeat the process above for all variables, using the top 19 categories (or whatever the maximum is if there are less than 19), and categorizing the rest as "others".

In order to make sure our categories are consistent in both our training and test data sets, we applied the categories that we came up with for each of our variables in the training data set to the test data set as well. There are 12 variables in the test data set that have categories that do not appear in the training data, so for any new instance in the test set that does not appear in the training set, we placed it into the "others" category. By doing so, we are still able to capture that variable's predictive ability in all of the models that we decide to run.

Lastly, we took both the hour part and the day part from the 'hour' variable in order to take into consideration any effect that time of day or day of the week might have on our predictions. We made these into two different categorical variables, with the time of day split into four categories for different time periods throughout the day and the day variable left as is. We chose not to use month or year since all of the data is from the same month and year. We then applied all of the data changes mentioned above to the test data set, categorizing any new categories that aren't present in the training set as "others".

Once we decided which variables to keep and adjusted the categories within those variables, we then used R to shuffle the entire training data set and randomly choose the first three million observations to make a sample to train our models. We used the next one million observations to make a validation sample to test and tune our models, and then the other next one million observations to test our ensemble methods.

Modeling

To begin our modeling, we chose to select a three-million-row training data sample at random from the entire training data set and use this to train our models. We chose another random one-million-row sample as our validation data set. We used these two data sets to build and evaluate all of our models so that we had consistency across all of our models. We made sure all of our variables were made factors for all of our models. Finally, we made predictions on a second one-million-row validation data set and applied an ensemble method to get an optimal model combination.

Logistic Regression

The first model we trained using our data was logistic regression with lasso regularization. We used the 'glmnet' function with an 'alpha' equal to 1 in order to achieve this. Since we have many variables for which we don't truly know the business context, it is hard to make any business decisions about some variables as to whether to remove them or keep them. Because lasso performs automatic feature selection in the model training, we wanted to use it to see if it would remove any multicollinear variables and reduce over-fitting.

We wanted to optimize the lambda parameter for our model, so we ran the 'cv.glmnet' function, which performs k-fold cross-validation and returns an optimal lambda based on our data. From the cv.glmnet, the best lambda we got was 9.425 e -05, so we then ran the logistic regression using that lambda and lasso regularization and calculated log loss on the predictions made with our validation set. The lasso regression removed many factors from our model, most of which were in the 'site_id', 'side_domain', and app_id' categories. We ended up with a log loss of 0.423.

Neural Network

Next, we wanted to train the data using multiple neural networks. We attempted to build three models, using different combinations of layers and nodes within each layer. We understand how sensitive neural nets are to scaling, so we scaled all of our X variables in the training, validation, and test data sets to be between 0 and 1.

After our variables were scaled properly, we started by creating a neural network with four layers of four neurons each. We used a ReLu activation function for the four layers and a sigmoid function for the final layer in order to obtain probability outputs. We then trained the neural network model using our training data sample and then used the validation data set to make probability predictions. From this model with four layers and four neurons in each layer, we got a log loss of 0.421.

In order to try and optimize our neural network approach, we wanted to try to make our model wider by using ten neurons in each of the four layers. Again, we used the ReLu activation function and a sigmoid function for the final layer. After calculating predictions on the validation data, we got a log loss of 0.456. We found that this model appears to overfit the training data.

Between the two neural networks that we ran up to this point, the best one was the one with four layers and four neurons in each layer. We decided to run the same model but using softmax as the activation function in the final layer instead of sigmoid. In this case, we used two neurons in the softmax layer for our output and then used our model to make predictions on the validation data. We obtained a log loss of 0.432.

Out of the three neural network models we ran, our best one was the model with four layers of four neurons each, which resulted in a log loss of 0.421. We will consider this model when comparing against our other methods.

Decision Tree

Since a decision tree is ideal for prediction with an imbalanced response variable categories and it is easily interpretable, we decided to try a classification tree on our data set. We first split our X variables and Y variables for both our training and validation sets and proceeded to set up a tree control with different parameter values. We tried many different values and found that, eventually, there is no difference in performance. We settled on a 'minsize' of 20000 and a 'mincut' of 5000. We tried both 'gini' and 'deviance' as splitting criteria and found that 'gini' resulted in the best log loss on the validation data.

After setting up a tree control with the tuned parameters, we then trained the model with our training set using all the variables we settled on above. We used our validation set to make predictions for the probability of a click or not and used the probability of a click to evaluate the log loss. We ended up with a log loss of 0.410.

XGBoost

After researching possible ways to improve our models, we discovered XGBoost, which is an implementation of gradient boosted decision trees, designed for speed and performance. We implemented parameter tuning for the XGBoost model by using GridSearchCV in Python to search through various values for different parameters. We tried tuning 'learning_rate', 'max_depth', and 'min_child_weight', using the log loss to make the parameter selections.

Because learning rate usually fluctuates between 0.05 and 0.30, and max depth and minimal child weight can range from 1 to 10, we used these ranges to perform our cross-validation grid search for our optimal parameters. Our final tuned parameters were 'learning_rate' = 0.15, 'max_depth' = 5, and 'min_child_weight' = 4.

When training our XGBoost model, we applied boosting of 1500 rounds. To avoid over-fitting, we also used a validation data set and set early stopping to 80 rounds. This means that if validation performance is not able to be improved in 80 consecutive rounds, then the algorithm will use the last best model. Boosting for our model stopped after 1014 rounds. Lastly, we made our predictions using the validation data set, and our XGBoost model resulted in a log loss of 0.402.

Final Predictions

After tuning our models and finalizing our parameters based on our training and validation sets, we wanted to create predictions and evaluate our models using a brand new data set that the models had not yet seen and that hadn't been used to tune any models. We generated another new random sample of one million observations to act as a second validation sample and then used our models to create predictions and calculate log loss based on the brand new data set. Based on these log losses from the new validation data sets, we planned to decide on our best two models and use an ensemble method to average the predictions to act as our final predictions for the final test data set. Finally, our decision tree received a log loss of 0.4095, XGBoost was 0.4008, logistic regression was 0.4228, and neural net was 0.4215.

After compiling log losses on the second validation set for all of our models, we experimented with various ensemble methods to ensure we were choosing the right combination

of models for making our predictions. We found that by using the average of the predictions for the decision tree model and the XGBoost model from the second validation set, we get a log loss of 0.403. This is the best log loss we got from all of the ensemble methods we tested on the second validation set, so we used this combination to make our final predictions on the test data set.

Finally, we read in the test data set and made predictions using both our decision tree model and our XGBoost model. To obtain our final predictions, we averaged the predictions from both models, and saved these along with the ID's to our final file.

Code Files Included in Submission

- Project-Code1-Team4.R
    - Includes code for data cleaning, decision tree modeling, logistic regression modeling, neural networks output, predictions on second validation set, and compiling predictions
- Project-Code2-Team4.py
    - Includes code for neural networks modeling
- Project-Code3-Team4.py
    - Includes code for XGBoost modeling