Anthony DePalatis (ald2796)

Tianyi Bi (tb25947)

EE 379K

Final Project Report

***Algorithms/Methods Attempted***

For the final project, we have tried 4 different machine learning algorithms: Logistic Regression, XGBoost, Random Forest, and Decision Tree. For each method, we used a slightly different approach. Overall, we got a variety of different results from each method, and saw that some produced significantly more accurate models than others. (XGBoost is the final method we used, so it will be in next session)

**Logistic Regression**

Since most of the starting code we saw used logistic regression and the datasets amazon provided are mostly categorical, we tried this at the beginning and the final score of this is around .88. We tweaked parameters of the model, decreased the features of the prediction, and we also tried to do multiple test predictions and add those predictions to the original dataset as new features. But none of this would improve the final score, it wouldn’t get higher than the starting code, so we decided to try some other models with our original method and see how the result goes.

**Decision Tree:**

Decision trees are - from a high-level view - a set of “choices,” each leading to another subset of choices until eventually a decision is made on the overall behavior of a data point with regard to the characteristic under investigation. Overall this method was not very successful for this project. We had the most success with the ‘gini’ impurity (which was slightly more accurate than ‘entropy,’ which was counter-intuitive since we had such accurate results using logistic regression). The best results occurred when we maximized the depth of the decision tree and used a relatively high ‘maxBins’ value (specifically, 55). In the end, however, we were only able to barely exceed 70% accuracy with this model.

**Random Forest:**

Even after tweaking the parameters and removing some features from the dataset, we never got very accurate results using a random forest model. In fact, no combination of the variations we tried with this model got above 51% accuracy. Because we didn’t see any slight progress, we abandoned this model pretty quickly so we could put our efforts into models that were more rewarding right off the bat.

***Final Methodology***

As we mentioned in the beginning of the report, the final method we used to solve the problem was XGBoost.

**XGBoost(eXtreme Gradient Boosting):**

XGBoost is an advanced implementation of the gradient boosting algorithm. From our initial research for the project, we found that XGBoost has several advantages over the other predictive models, and it gave us the second highest accuracy of all the methods we tried (the most accurate being the logistic regression model used in the starter code).

**Why XGBoost**

Due to its high flexibility, XGBoost allows users to define custom optimization objectives and evaluation criteria, so there is very little limitation on what we can do with this model. It handles missing values very well which, although there are no missing from the amazon data, is still a nice feature to have. One of its key feature is regularization, which helps to reduce overfitting for us. The one downside to XGBoost is the time it takes to finish each run. Since we did an AUC check in a for loop that iterates 10 times and it took close to 5 minutes to complete a single run.

**What have we done**

After tweaking the XGBoost model, we settled on the following values for the parameters:

* **Learning\_rate:** We set this parameter to 0.1 for time efficiency. This is the sweet spot value that doesn’t take too long and still gives us the high AUC.
* **Max\_depth:** We set max\_depth to 11. We found that increasing the max\_value to this value gives us the maximum AUC before it starts to level off.
* **N\_estimator:** We set this to 1000.

The other parameters were kept at their default values, because they either had minimal effect on the accuracy, or changing them decreased accuracy.

**Why we used this method and coding**

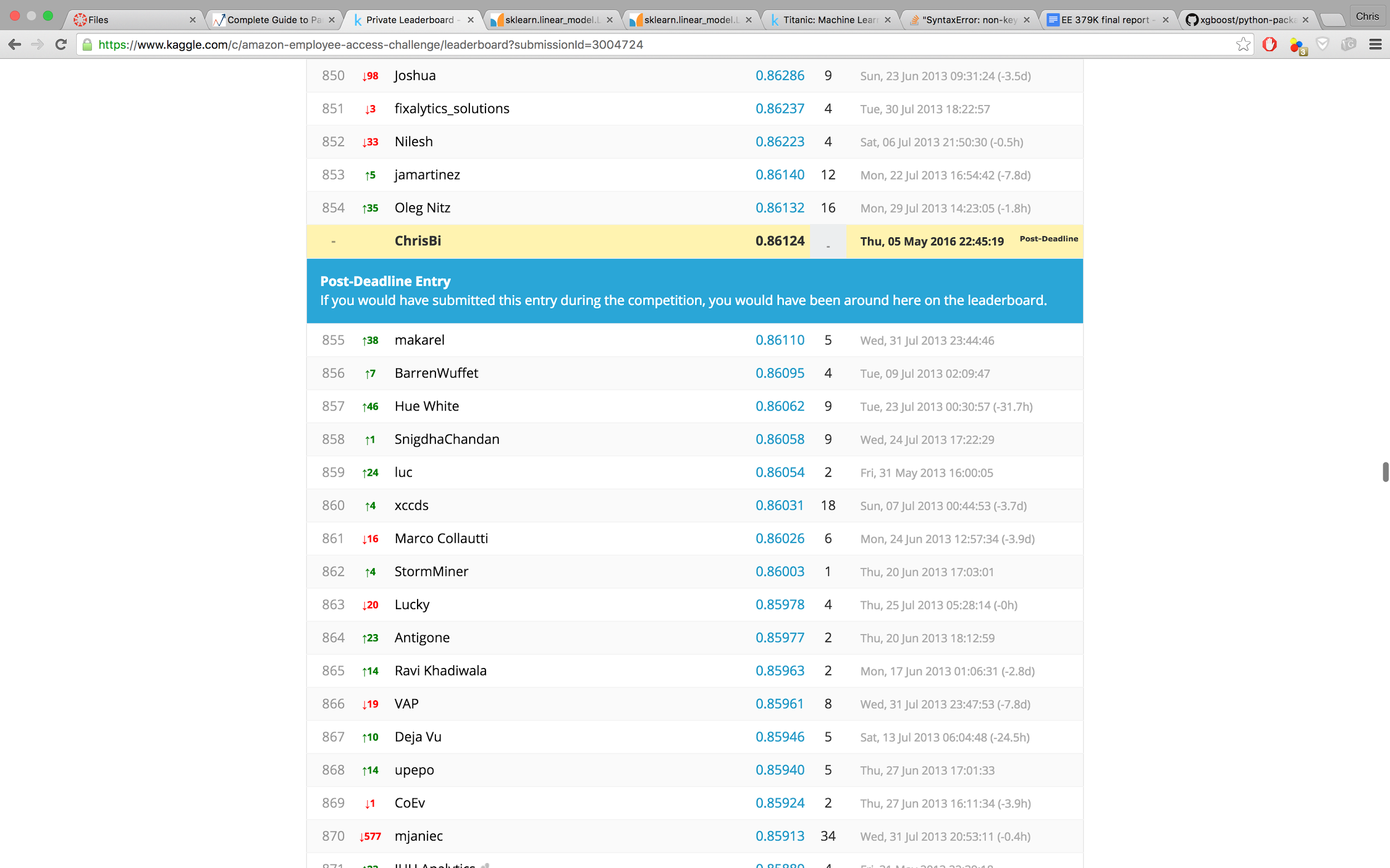
Although the starting code gave us 0.88 final score on the private dataset, we couldn’t find a way to change the code to get a significantly better score. Plus, we wanted to be a little more original, which is why we selected XGBoost as our final method.

In our code, we first parse the data using the pandas library, (we also tried numpy but we ended up sticking with pandas in the end). We used sklearn.preprocessing.OneHotEncoder to transform the data. We also tried standardscaler, but it didn’t worked so well, with its highest score being around 70%.

We did borrow 2 portions of code from couple starting codes, one of them is testing the AUC with different train-test split on train.csv, and the other one is output all the prediction to a csv file.

**Final Score and How to Run Our Code:**

The final score we got on the private data set with this setup is 0.86124.



The actual python file, **xgb.py**, is in the XGBoost folder and the resultant csv file is in the submission folder. You can run it by enter *$ python xgb.py* in the terminal, it will give you the mean AUC of 10 runs, and will output all the prediction to a csv file with a name of your choosing.