Final Project Report

EE 379K - Architectures of Big Data Science

Rohan Nagar (ran679) and Wenyang Fu (wf2796)

a) How did your team go about tackling this problem?

To begin, we researched the Amazon Employee Access Challenge. This included looking at the winning solutions, reading related blog posts, and reading the Kaggle forums on the challenge. This research was crucial for us because it gave us an idea of where to start and what problems could arise while attempting this challenge.

Next, we decided to start exploring the data ourselves. A lot of this data exploration is done in our amazon_challenge Juypter Notebook. We looked at the distribution of the feature values among all of the training data, and were able to visualize how lopsided the classification problem was. 95% of the employees were labeled with ACTION = 1.0, which means that they were allowed access. Clearly, this means that if we were to just predict all 1's all the time, we would get around a 95% accuracy score. However, this is not a good classifier. That's why it is better to look at the AUC ROC score. This the scoring metric used in the Kaggle competition, but it was good to visualize it ourselves.

We also used pairplots and a heatmap to visualize the correlation between the features. We noticed that the two features ROLE_TITLE and ROLE_CODE are highly correlated, which means that we could probably remove one of the features and not lose much information.

After all of our visualization we finally began to train some models, look at ensembling, and do a little bit of feature selection. These models are discussed in the following questions.

b) Which methods/algorithms did you try?

We started with starter code that performed a Logistic Regression on the data. This gave us a very good baseline to work with, as the AUC with this model alone was around 0.88.

The starter code we used is here:
https://www.kaggle.com/c/amazon-employee-access-challenge/forums/t/4797/starter-code-in-python-with-scikit-learn-auc-885

We then wanted to try some of own models. We began with Random Forests, since they are a relatively simple Decision Tree ensemble model, and they do not require categorical feature encoding. In order to find the best parameters for the Random Forest model, we used scikit-learn's GridSearchCV. This searches over a set of parameters we specify and finds the ones that give the best score. Random Forests performed pretty well as well, scoring up to a .869.

We also tried an AdaBoostClassifier. This model did not work quite as well as we hoped, because our score continued to be under .80 throughout all of our parameter tuning with GridSearchCV.

The model that we spent the most time on was XGBoost (https://github.com/dmlc/xgboost). This model uses gradient boosting, and is very efficient and flexible. We were able to train an XGBClassifier and tune the parameters using scikit-learn's GridSearchCV, since XGBClassifier implements the scikit-learn interface. This did really well, and with good parameter tuning the XGB model alone could get up to a score of 0.866.

The last base model we tried were SVMs. However, we ran into a lot of problems with SVMs and since we started playing with them much closer to the deadline, we didn't have time to finish working with them. They are left out of our final methodology. The training time on SVMs, especially when setting the probability parameter to True (in order to get class probabilities instead of just a 0 or 1 prediction), was very high. This made it hard to test and tune parameters.

Finally, we tried ensembling all of our best models together. We did a rank-average based ensemble approach using the generated submission files from our previous models. This was a great way to do ensembling because our models did not need to be re-trained, and we could quickly and easily run our rank-average code. This method is seen in "rankedavg.py", and is used in our final methodology.

c) What is your final methodology? Walk through it in detail, starting from data pre-processing. Explain all the machine learning algorithm(s) you used as well as the parameters you chose. Also discuss any external tools or libraries that you used.

Our final methodology includes ensembling 2 of our own models in addition to the logistic regression starter code with a rank-average based approach.

The first model is a Logistic Regression. This is the starter logistic regression code that we saw earlier. For this model, the required pre-processing is a One-Hot encoding on the categorical features. One the data is encoded, we train the model and use cross validation to get an accurate idea of how well the model will perform. This model is very basic (but does very well), and the only parameter that is changed from the default value is the C value, which is set to 3 in this model. Finally, we fit our model on the entire training set and then call predict_proba() on the test set to get our model's answers and confidence in those answers. It is important to use predict_proba() rather than predict() so that we can know how confident the model is, and so we can use those probabilities to average in our ensemble.

One of own models is the XGBClassifier. The pre-processing that we did on the data was dropping the "ROLE_CODE" feature. We determined that this feature was highly correlated with "ROLE_TITLE" and was not necessary for our model. In fact, our score went up after dropping this feature. We train the model (again with cross validation to get an accurate sense) with the best parameters that we found during parameter tuning. These parameters are:

- max_depth = 8
- learning_rate = 0.3
- n_estimators = 155
- min_child_weight = 0.6
- subsample = 1.0
- colsample_bytree = 0.45

Again, we found these parameters using GridSearchCV and searching over many possible parameter combinations. We also had help from http://www.slideshare.net/odsc/owen-zhangopen-sourcetoolsanddscompetitions1 when deciding which parameter values to start with. Finally, we again train the model on the entire training set and use predict_proba() on the test set. These are saved to a submission file.

Our last model is a Random Forest. Since tree-based models do well with categorical features, we did not have to do any encoding with this model, either. However, we did again drop the

“ROLE_CODE” feature in order to bump our score up a little bit. Following a similar pattern as above, we trained with cross validation with the following parameters:

- n_estimators = 2000
- criterion = entropy
- max_features = auto
- bootstrap = True

These were the best parameters that we found using GridSearchCV. “n_estimators” is the number of decision trees to use in the forest, and typically this number is a couple thousand, so that makes sense. Entropy turned out to be the best split criteria in this case. “max_features” is the number of features to consider when deciding a split. In our case, ‘auto’ was the best (which is actually equivalent to the ‘sqrt’ parameter, since auto looks at sqrt(num_features). Finally, we use bootstrapping in the model to bootstrap samples while building the model.

With the submission files for these 3 models, we were then able to ensemble them together with a rank-average approach. With help from http://mlwave.com/kaggle-ensembling-guide/, we took the submission files and turned the predicitons in rankings, averaged the ranks, and then normalized those averaged rankings between 0 and 1. This gave us an even distribution in our predictions. Our resulting ensemble model (the resulting submission file) produced a much higher score than each individual model’s submission file.

In terms of external tools and libraries, we used most of the same ones that we have been using all semester. We heavily utilized Juypter notebooks in order to quickly run and test code. This especially came in handy when doing parameter tuning. The libraries we used include numpy, scipy, scikit-learn, matplotlib and seaborn. In addition to these, we also used a library called XGBoost. This stands for “Extreme Gradient Boosting”, and has recently become a popular model. Working with the library was nice and flexible, as it was structured similar to the models in scikit-learn. We also attempted to use a powerful AWS EC2 instance (c4.8xlarge) to speed up the training time of our code, specifically when attempting to train SVMs. Unfortunately, the SVMs still ran quite slowly and were left out of the final model.