Final Project Report

EE379K: Big Data Architecture

By: Tom Ermis [TFE95] + Anish Vaghasia [AV23674]

Summary:

I. Feature Analysis

A. We first wanted to understand how each of the features related to the action value so we could get a better vision of the data.
    1. AnalyzeData.py
        a) Finds every ID for each feature. Then, for each ID, the script counted the total number times this ID was seen and whether it correlated to a 0 or 1 action value. While pretty basic, this gave us a pretty good idea of how often certain ID's popped up and how they correlated to the action values.
    2. AnalyzeMultiFeatures.py
        a) This script takes it a bit further and checks the strengths of combinations of features.

B. We choose to use one hot encoding on our feature set. There was a substantial improvement in accuracy across all models: 0.51925 -> 0.88204, with the case of the logistic regression training model.

II. Model Analysis

A. We first decided to test different models to understand how they performed unaltered for this problem.
    1. We tested Logistic Regression, Logistic Regression CV, 2 different implementations of Xgboost (XGBoostClassifier, XGBClassifier), Random Forest, and Gradient boosting. Our results for accuracy are outlined at the bottom of this report.

B. We choose to use these models to test for multiple reasons
    1. The highest results in the kaggle competition were modified versions of these models.
    2. There is a plethora of examples and starter code that could be analyzed while making our own
    3. The results of these models were combined using some selection algorithm in most of the top code solutions.

C. Logistic Regression/CV
    1. Logistic Regression is the obvious choice for this situation. The features are independent of each other and the result is binary. Miroslaw's example is the main code we looked at for this model.
    2. Logistic Regression CV is along the same mindset, but uses repeated cross validation folds to improve accuracy.

D. RandomForest
    1. Random Forests is a great model to use due to their relatively fast runtime and accuracy.

E. Gradient Boosting
    1. While we weren't as familiar with gradient boosting, we still took a look at the model due to its strong results. It handles classification and regression well and therefore is a good fit for this challenge.

2. Additionally, boosting is a powerful tool to be able to use. It adds new models sequentially. So a new weak, base-learner model is trained by the previous results.

F. XGBoost
    1. XGBoost is relatively new in the ML space but it is getting a great amount of attention. We tested two implementations of XGBoost. The first was from a custom implementation of XGboost, XGBoostClassifier. This implementation routinely scored a higher accuracy than the standard XGBClassifier from sklearn.

G. Extra Trees
    1. Uses a meta estimator that fits a number of randomized decision trees on subsets of data and uses averaging to improve the accuracy and control over-fitting.

III. Example Code Used

A. Starter Code (Code can be found in lecture slides)
    1. Using one hot encoding, logistic regression model training with cross validation. Metrics were compiled to determine mean AUC across multiple cross validation folds (measure which we are being evaluated against).

B. Miroslaw (Code can be found in lecture slides)
    1. Miroslaw implements some of the concepts talked about earlier in the report. It mainly takes advantage of one hot encoding, feature selection/extraction, and Logistic Regression. After a bit of tweaking, and a 3 hour long execution, we were able to get the original score of 0.89852.
    2. Due to the long runtime we used a smaller dataset during parameter testing.
        a) Eventually, we found that the given combination of parameters was indeed the best possible for the model.
        b) Using the same base code, we also attempted to used Random Forest Classifier model instead of a logistic Regression. While it finished for a medium size dataset (6k), we ran out of time to run it on the full dataset. However, in comparison to the Logistic Regression test on the medium size dataset, it achieved a full percentage point better. 60% vs 61%

IV. Final

A. Our final code methodology is the following:
    1. Import data as numpy array/pandas dataframe
    2. Compile list of multiple models with parameters to determine best accuracy
    3. Use one hot encoding to convert the numpy array/pandas dataframe into sparse csr_matrix. Every feature now has multiple states
    4. Train predetermined models on training dataset

5. Predict probabilities of trained models given test data
6. Save results as CSV file

B. We had tried a variety of different avenues to tackle this problem and we learned a lot in the process while we experimented with these different models. To get the highest score we decided to combine the different models already created by expert Kagglers. We hoped that, by doing so, each model could actually cover the weak points of other models. Thereby improving the overall score.
   1. Models Used
      a) Modified Starter Code
      b) Miroslaw Logistic Regression
   2. Combine.py
      a) This script takes multiple Kaggle Submission files and combines for the optimal submission. It does this by taking the "confidence level" or action probability for each 'id' and comparing it across files. Then it selects the highest probability for the Combined-Submission file. This assumes the model used to create each file has provided an accurate prediction for it's "confidence level". (The further it is away from .5, the more confident the prediction)
   3. Combine2.py
      a) This script also takes multiple Kaggle Submission files and combines for the optimal submission. It does this by averaging the scores from each submission file for the final submission
      b) In tests done, this method worked better than the one above

V. Concluding Thoughts

A. Unfortunately, our results from XGBoost and RandomForests were not as accurate as expected. However, we believe highest accuracy can be achieved by selecting our features, running them through one hot encoding, training models based off XGBoost and RandomForest, and then ensembling them with our custom scripts mentioned above.

**Please note: All models below run on default settings (with the exception of seed for consistency)

Model	**Accuracy** (without one hot encoding)	**Accuracy** (with one hot encoding)
Logistic Regression	0.51925	0.88204

Logistic Regression CV	0.51922	0.88184
Xgboost	0.86394	0.85776
Random Forest	0.80456	0.84499
Gradient Boosting	0.75635	-N/A-
Extra Trees	0.76006	0.84440

Random notes:

- Using class_weight='balanced' in logistic regression model actually reduces performance.
- Mean AUC actually improved by decreasing test size in cross validation train/test split, private score remained same
- Increasing number of folds does not increase performance. Mean AUC decreases, private score stays same