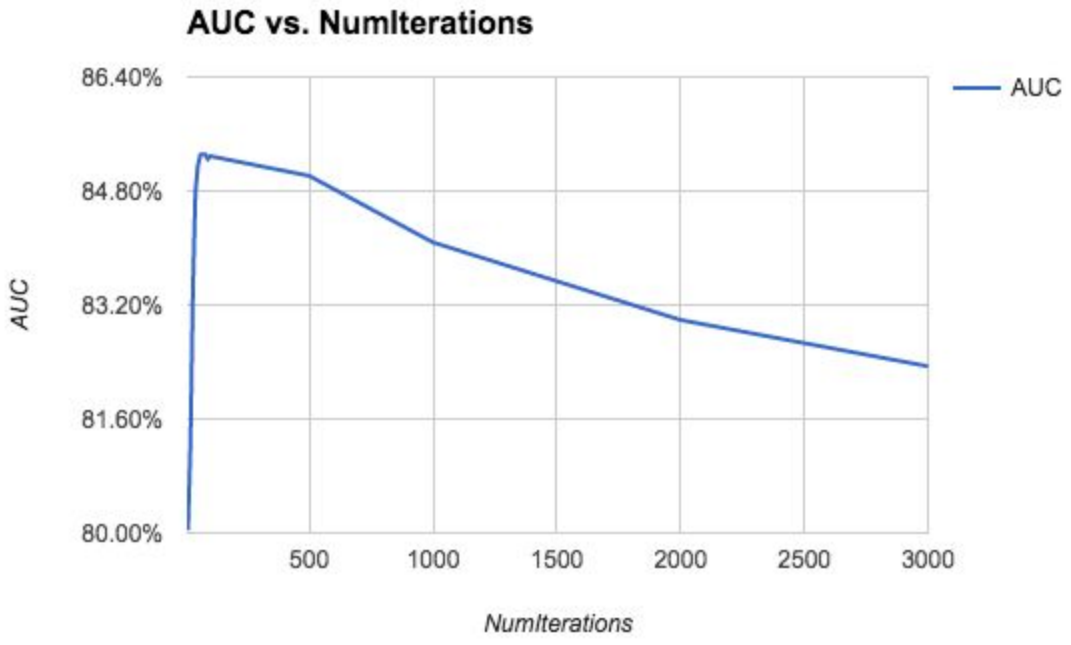
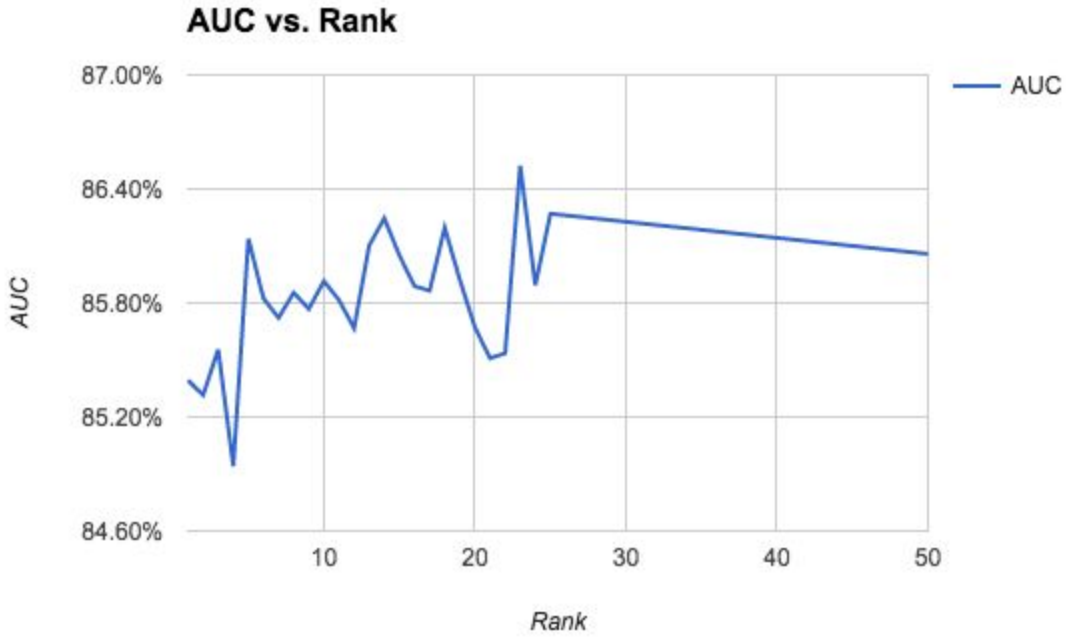
Arvind Sundaram AM62379
Carlos Rios CR32859

EE379K: Amazon Employee Access Challenge Project Report
(CODE ALSO LOCATED AT: https://github.com/criostrevino93/ee379k_project)

Part 1: Factorization Machines

At the start of the project, we used the a library called FastFM for Factorization Machines along with the scikit library. Initially, we used the Asymmetric Least Squares Regression (ALS) model with initial parameters of als.FMRegression(n_iter=1000, init_stdev=0.1, rank=2, l2_reg_w=0.1, l2_reg_V=0.5). These were parameters suggested by the FastFM library documentation. This was used after using the One Hot Encoder on the data.
This initial model had the resulting private score of **0.82785**, and our team thought we could do better. We tested different models within the FastFM library and saw no major difference in the scoring until we tested the the Markov Chain Monte Carlo (MCMC) model. This model with the initial parameters of mcmc.FMClassification(n_iter=1000, rank=2, init_stdev=0.1) gave us a private score of **0.85055**. Then, we ran some tests to optimize the MCMC model's parameters. The following charts illustrate how the AUC score changed when we varied the number of iterations and the rank.





We found that the highest AUC happened with 80 iterations and a rank of 25. With the final parameters of mcmc.FMClassification(n_iter=80, rank=25, init_stdev=0.1), we increased our private score to **0.87936** (submission_FM_mcmc80_r25.csv).

This part of the project can be found on the python script called amazon_fm.py

Part 2: XGBoost

Our second test consisted of using the the XGBClassifier to make the predictions. This part of our project also relies on the use of One Hot Encoding on the dataset. The initial parameters we used were xgb.XGBClassifier(max_depth=5, n_estimators=1000, learning_rate=0.05). These were also parameters suggested by the examples on the documentation of the XGB library. This initial submission generated a private score of **0.83526**.

After testing and researching online to find the best parameters, we found that the parameters that yielded the best parameters were a max depth of 100, 200 estimators, column sample of 0.3 and a learning rate of 0.5. With the parameters xgb.XGBClassifier(max_depth=100, we increased the private score to **0.87127(submissionXGB_optimized.csv)**.

This part of the project can be found on the first part of the python script called amazon.py. This code also contains a second part that is very similar approach to the starter code by using One Hot Encoding and Linear Regression. In this script, we also tested other models such as

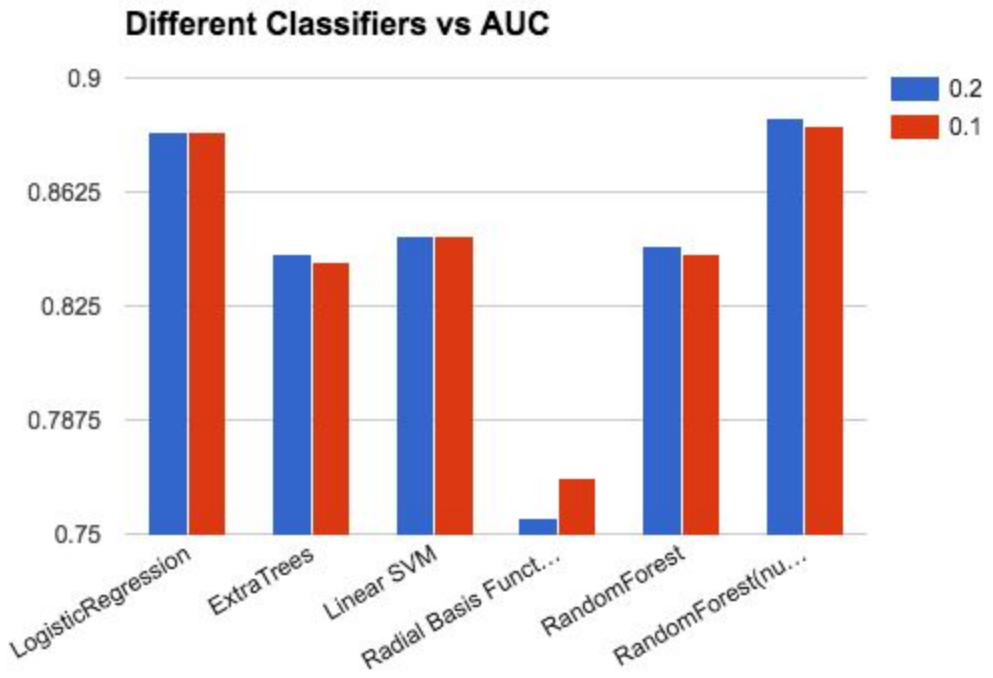
SGDClassifier and the GNB model that don't necessarily have results worth mentioned. It does, however, illustrate that we tried a lot of different models to increase our private score.

Part 3: Support Vector Machines (SVMs), Classifiers, & Test Size

Our third test looked at SVMs from the sklearn library as well as different classifiers from the ensemble library. Using the starter code for this part, the *LogisticRegression (LogisticRegression1.csv &LogisticRegression2.csv)* classifier gave an AUC of **0.88204** for both testsize=0.2 and testsize=0.1. We decided to test some classifiers from our previous labs first, specifically the rbf and linear kernel SVMs. Using the *Radial Basis Function (rbf)* kernel for the SVM gave us an AUC of **0.7553** for testsize=0.2 and **0.7686** for testsize=0.1 and took about 45min to run 10 tests and store the results,. Since the AUC was so low in testing for *RBF,* it wasn't used in the submissions on kaggle. Then using the *Linear* kernel for the SVM (*LinearSVM1.csv&LinearSVM2.csv*) gave us an AUC of **0.84811** for both testsize=0.2 and testsize=0.1 and it took around 30min to run to run 10 tests and store the results. The rbf kernel didn't help at all with around a **.76** AUC for both test sizes and the linear kernel was slightly lower than the starter code with an AUC of 0.8481.

Moving forward we wondered how other classifiers in this library compared to the *LogisticRegression*, for this we tested the *ExtraTrees* and *RandomForest* Classifiers. *ExtraTrees (ExtraTrees1.csv&ExtraTrees2.csv)* gave an AUC of **0.84242** for testsize=0.2 and **0.83926** for testsize=0.1. It took around 30min to run 10 tests and store the results. *RandomForest (RandomForest1.csv&RandomForest2.csv)* gave an AUC of **0.84476** for testsize=0.2 and **0.84223** for testsize=0.1. It took longer to run, about 40min to run 10 tests and store the results. Further looking at *RandomForest*, we changed the num_iterations to 100 and 250. For *RandomForest* with num_iter=100 (*RFNI1.csv&RFNI2.csv*) the AUC was **0.88683** for testsize=0.2 and **0.88399** for test size=0.1, and took 1hr and 20min to run 10 tests and store the results. For *RandomForest* with num_iter=250 *(RFNI250.csv)* the AUC was **0.88905** for testsize=0.2 and it took 3hr and 35min to run 10 tests and store the results. Due to the length of this test, AUC was not computed for testsize=0.3. Ultimately the *RandomForest* classifier with num_iterations set at 250 gave the **highest AUC at 0.88905.** The results and graph for the different classifiers, test sizes, and SVMs can be seen below. The code for the different classifiers and results is found in *amazon1.py* & the *submissions* folder on github.

TestSize	0.2	0.1
LogisticRegression	0.88204	0.88204
ExtraTrees	0.84242	0.83926
Linear SVM	0.84811	0.84811
Radial Basis Function (rbf) SVM	0.7553	0.7686
RandomForest	0.84476	0.84223
RandomForest(num_iter=100)	0.88683	0.88399
RandomForest(num_iter=250)	**0.88905**	n/a



Part 4: Ensembling Models

For this part of this project, we combined the predictions of the different models tested by averaging the predictions together. As mentioned before, the best models from our results were FM MCMC, XGBClassifier, RandomForests, and the starter code using LinearRegression (As a Baseline, its private score was 0.88204). These models were the ones we attempted to combine. The following is a list of the private scores after averaging a the predictions from a combinations of the models.

- Averaging starterLinearRegression + InitXGBClassifier = 0.88525
- Averaging starterLinearRegression + InitFM = 0.85450
- Averaging starterLinearRegression + OptimizedXGBClassifier = 0.88854
- Averaging starterLinearRegression + OptimizedMCMC = 0.88854
- Averaging OptimizedXGBClassifier + OptimizedFM = 0.87127
- Averaging starterLinearRegression + InitialXGBClassifier + InitialFM = 0.88421
- Averaging starterLinearRegression + OptimizedXGBClassifier + OptimizedMCMC = **0.88882** (submission_optimized3models.csv)
- Averaging starterLinearRegression + OptimizedXGBClassifier + OptimizedMCMC + BestRF = 0.49886
- Averaging OptimizedXGBClassifier + BestRF = 0.49885

As seen from the list, the best ensembling results was averaging the starter code LinearRegression model, our Optimized XGBClassifier model, and our OptimizedMCMC model yielding a private score of **0.88882.** This is a small increase from the baseline starter code private score. However, our best score still resulted from the RandomForest model mentioned in Part B which was **0.88905.** In fact, any average that included the best RandomForest predictions did worse than the sample submission. Our guess as to why this happens is that some of the values of the predictions in the RandomForest model are 1, so when averaged together, it worsens the roc score.

The averaging is located in the python script called average.py. Note: If you wish to run this code, you have to modify the code to select the models that you want to average.