ECS 171 Loan Prediction Report

Abstract: In this report we will be looking at how we improved our model. Through changing our steps and processes that in turn gave us our group **Abelian Group** a score of Score:**0.53782** with a current rank of **8** as of **12/15/17**. The main solution to this problem boils down to two parts of the assignment with changes and alterations to our method of classification and method of regression.
Keywords: ridge regression, lasso regression, regression, classification, features selection, testing, overfitting, regularization, gradient boosting regressor, gradient boosting classification

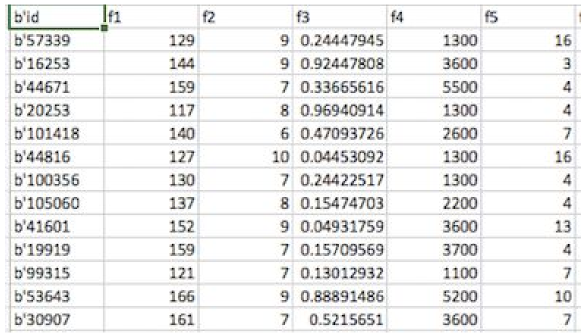
Team: Abelian Group

Student:	**Kerbos:**	**Student ID:**
1)David Zheng	davez	913131529
2)Zekai Zhao	zekai917	912663522
3)Jiayue Yao	yao2007	914377874
4)Alan Zhang	zhangti	912692250

In our initial outline of the project we had a bunch of ideas we wanted to implement; based on what we learned in class and out of class. The issue with this, was that (or with most Machine Learning cases) most people are eager to implement algorithms without being fully aware of their data. Early on, we realized that the features were plentiful and that if we could use almost any algorithms.

However, the pitfall of doing this type of implementation of these types of algorithms is that we have an issue of overfitting or underfitting.

We had discussions on whether to implement what types of algorithm to use, but we first learned how to understand the data, before we even implemented an actual learning algorithm. Therefore, the first step we had to do was to change the data value into a CSV to try to deduce our understanding of the problem better. After reading a couple of the rows and columns we realized that the 700+ features were too much handle, so we used the scikit-learn function **SelectKBest**.[1] This along with a simple scikit regression library had us beat the professors We then used this to train our 40,000 training set and used the other 10,000 for validation. The 10,000 was shuffled randomly because we wanted to make our validation process to be more accurate. Doing this in a random fashion helped make our validation and classification more accurate.



b'id	f1	f2	f3	f4	f5
b'57339	129	9	0.24447945	1300	16
b'16253	144	9	0.92447808	3600	3
b'44671	159	7	0.33665616	5500	4
b'20253	117	8	0.96940914	1300	4
b'101418	140	6	0.47093726	2600	7
b'44816	127	10	0.04453092	1300	16
b'100356	130	7	0.24422517	1300	4
b'105060	137	8	0.15474703	2200	4
b'41601	152	9	0.04931759	3600	13
b'19919	159	7	0.15709569	3700	4
b'99315	121	7	0.13012932	1100	7
b'53643	166	9	0.88891486	5200	10
b'30907	161	7	0.5215651	3600	7

[2]

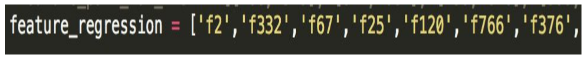
Now we moved onto our initial model for classification and moved to our first method of regression. In our function **def regression**, we initially used a basic regression function that prompted a score that was **1.59**. We realized that we had two issues at hand, either our features were incorrect, or our regression was wrong or both. So running through more tests, we realized that we should have made non-zeros be 1 before classification to be more accurate in our classification of the data. We then ran it through with a small tweak with our regression was using the 10 best selected feature from **SelectKBest**. This led to us having a score that lost to the professor's score by **0.010236**. While this was not, bad we wanted to improve on our model to be more precise. We went back to the drawing board and used a better regression model called **ridge regression**. The reason why we thought that ridge regression was a good regression model was because of the main job that ridge regression does compared to the other popular regression models (**lasso regression**).

In ridge regression, we don't exactly know when a type of regression does better than another model. However, ridge regression main strength is how it produces a 00set of coefficient estimate for different tuning parameters values. Therefore, we knew that it was good to use the results of ridge regression along with our cross validation. Ridge regression was great for regularization where we could keep variance under control. As a group, our initial score (**1.59**) we believe that because of the huge amounts of features, we realized that if our algorithm was suboptimal we would have large discrepancies, hence the bad score.

Ridge regression was great at handling biases as to put it simply it is almost analogous to accepting a scale weight of 2 pounds light but can go up to 2.5 pounds or as little as 1.5 pounds. The "2.5" and "1.5" were the biases we were willing to

[1] We also realized that the majority of the data terms were mostly 0's and nonzeros (where we later realized a better classifying method).
[2] Some of the features listed in our dataset

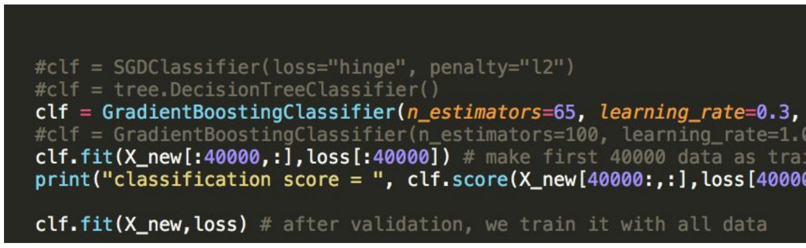
accept. While this was a good start, we knew that we could improve on our score. We played around with the learning rate of 0.1, 0.001, 0.0001 but not much really changed our score. We soon came to an epiphany. We realized the problem was not the regression (at the time, but we will speak more about that later) but the feature selection. We realized that the problem with using the scikit-learn library was that we did not know the guidelines in how **SelectKBest** selected the 10 best features i.e (we don't know if the 10 best features selected are really the 10 "best" features). The dangers of not realizing the implementation of the library, and blindly using a package was our exact initial point in our report. Luckily, we realized that early on, and began implementing a better feature selection. Looking back at our **CSV file**, we realized that were duplicate columns and other "noise" which were hurting the prediction of our model. Therefore, we removed the duplicate columns from our dataset. This in addition to a new method of selecting our features by having better classification of double for loop that naively looped through the training and trained until there was minimal loss after our 500th iteration. While, there were better methods to probably do this feature selection, we were crunched for time and used a naive method to quickly select our "best features". We then used these features as our new feature list.



```
feature_regression = ['f2','f332','f67','f25','f120','f766','f376',
```

(these are just some of the features we found, there was a total of 16 features that were selected as our best feature). We implemented the **gradient boosting classifier** (similar rules like SGD or GD) and followed the same initial rule of our first 40,000 being used and the last 10,000 to validate our classifier. These features and tweaks then improved our score to a sub **.63-.62** overall score. Lastly, we felt that our features were the best it could have been. So we focused our attention on the regression part of the assignment. The overall assignment outline was split to the classification and regression. We felt that we had a good selection of features and would only marginally improve our score at this point if we tried any further.

In thinking about our method of regression we realized that our ridge regression was a good starter step, but could be further improved. Through further research and discussion we looked for a better method and came upon **gradient**



```
#clf = SGDClassifier(loss="hinge", penalty="l2")
#clf = tree.DecisionTreeClassifier()
clf = GradientBoostingClassifier(n_estimators=65, learning_rate=0.3,
#clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.
clf.fit(X_new[:40000,:],loss[:40000]) # make first 40000 data as tra
print("classification score = ", clf.score(X_new[40000:,:],loss[4000

clf.fit(X_new,loss) # after validation, we train it with all data
```

[3]

boosting regressor and **gradient boosting classifier**(from earlier paragraph above)**.** The gradient boosting classifier was powerful because of how it was not too hard to implement in lectures as the professor stated previously in lectures. However, there are some models that are powerful, the pitfall is that they are very complex to implement. Even then our gradient boosting model helped optimized our arbitrary function where we had our classes regression tree mesh well with our features through each iteration. The regressor was perfect, because as the professor has stated before about the power of decision trees in

[3] Gradient Boosting Classifier function that was used throughout as our main classification

lecture, we took that idea to heart and tried to implement as such. The gradient boosting regressor helped give each stage a better regression tree as we went on. The main part about the regressor and classifier help control the fitting behavior where the arguments (hyperparameters) main part of our gradient boosting regression tree were the: regression trees(), depth of each(), our loss function() and our learning_rate(). In practice, we realized that the power with gradient boosting was because it is very robust out of box classifier and regressor that does not require the function to be well defined. It learns the function and classification through it's iteratively improvement. Also, we chose this as opposed to something like linear models because of the weakness that linear models entail. While easier to use, it is very susceptible to overfitting and with the bountiful features we believe that the outliers can heavily impact our model "too well". The power in decision trees is that does not do what linear models do, it makes no assumptions and are good at learning complex non-linear decisions. While, we could have only done a decision tree, we realized that like most models, they can easily overfit on the data hence the need for gradient boosting. Gradient boosting boils down to three steps that make it so powerful

1) Loss function to optimize (can be vague)
2) Weak initial learning model for predictions
3) Adding and tweaking the model make the learning model minimize loss function.

Overall, using **gradient boosting regressor** and **classifier** gave us our current best score **0.53782**. To run the program ***my_prediction.py***, use ***python3 my_prediction.py*** with the source code listed with the dataset. Which would print out the submission file.

Work Cited and Packages Used:

1) https://codingstartups.com/practical-machine-learning-ridge-regression-vs-lasso/
2) https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/
3) https://onlinecourses.science.psu.edu/stat857/node/155
4) https://machinelearningmastery.com/an-introduction-to-feature-selection/
5) http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
6) http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
7) http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
8) http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
9) https://en.wikipedia.org/wiki/Gradient_boosting
10) https://gormanalysis.com/gradient-boosting-explained/
11) https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d