ECS 171 - Final Project Report

1. Team Information:

Kaggle Team Name: Geese Four

Name and Kerbos ID:

Jiayi Wang(wjiayi); Cheng Gai(cgai);
Haidou Bai(tbc3211); Xiang Wang(shawwwn)

2. Command to generate the predicted results:

```
# convert dataset to csv
python convert_csv.py ecs171train.npy 0 -h
python convert_csv.py ecs171test.npy 0

# train & predict
python train_predict.py
# will generate "out.csv"
```

3. Final submitted model:

3.1 Model Description:

Our model consists of 3 parts: **feature selection**, **classification**, and **regression**. In feature selection, Pearson Correlation Test[1] was used to determine the level of correlation between a feature and the row's output. Bivariate features(result of arithmetic operation from a feature pair) was used in addition to univariate features. Top 100 features/feature pairs in the correlation test were appended to our dataset. For classification, we binarized all Y(all positive number being set to 1), and train our dataset on a Gradient Boosting Classifier. For regression, we first selected a subset of data in which Y!=0, then applied transformation to Y in selected data so that Y's distribution become more like normal distribution. These data were subsequently trained on a Random Forest Regressor. Finally, regression result was merged with classification result to get the final output.

3.2 Kaggle MAE:

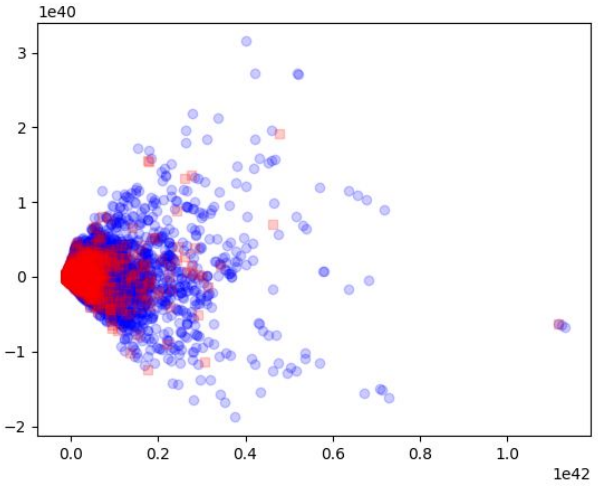
Combining all the things we have done (below), we were able to achieve 0.50281 on Kaggle.

[1] https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.pearsonr.html

4. Previous model attempts and Details:

4.0 Preprocessing

Principal Components Analysis(PCA)[2] was used to perform dimensionality reduction on the training data. The result was plotted on the right hand side(red represents a loan is default, blue represents not default). The scatter plot clearly indicated that the raw dataset was not linearly separable, so we need to find some non-linear model(trees) to train our data.



There were many NAs in the given dataset. In order to get rid of them, we use **sklearn.preprocessing.Imputer**[3] to batch set NA to column mean.

We also did not have prior knowledge about each feature and its distribution. To test and plot each feature in histogram can be impractical. To save time, we used a scaler in an attempt to scale all features to standard normal distribution. We first tried **sklearn.preprocessing.StandardScaler** but later found out **sklearn.preprocessing.RobustScaler**[4] performed better. This could be due to the fact that robust scaler only scales data within certain quantile[5], therefore insensitive to outliers.

```
from sklearn.preprocessing import Imputer, RobustScaler
sd = RobustScaler()
imp = Imputer(strategy='median')
x1= imp.fit_transform(x_train)
x2= sd.fit_transform(x1)
x_train = x_train
```

For testing and validation purposes, we used **sklearn.model_selection.train_test_split** to split our training data into 2:8 ratio -- 20% for validation, and 80% for trail-run training. **sklearn.metrics.mean_absolute_error** was also used as a metric for regression validation. We adopted the all-zero prediction as our baseline, any result that scores better than the baseline would be regarded as an improvement. All validation code was removed from final submission.

4.1 Feature Selection:

---

[2] https://en.wikipedia.org/wiki/Principal_component_analysis
[3] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Imputer.html
[4] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
[5] http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

We first tried to run our classifier on raw data directly, but accuracy was even worse than the baseline result(all-zero prediction). We then tried using the 100 most correlated univariate features from the pearson test[6], the result was not ideal.

top-10 univariate features: (feature name, pearson score)

```
[('f322', 0.12368640390548852), ('f323', 0.1111639738746235), ('f314',
0.10915947155091422), ('f324', 0.10292815790000039), ('f376', 0.10193015512176404),
('f377', 0.10176647597125642), ('f315', 0.09983570122563766), ('f25',
0.09955351550890662), ('f26', 0.09583189943780852), ('f31', 0.09450508807733166)]
```

Finally, we tested all bivariate features by adding, subtracting, multiplying, and dividing pairs of feature produced by combination and permutation(i.e., f1-f2, f1+f2, f1*f2, f1/f2). To our surprise, after adding the top 100 most correlated bivariate features to the original dataset, the accuracy has greatly improved.

top-10 bivariatefeatures: (feature name, pearson score)

```
[('f528-f274', 0.24825328292710389), ('f528-f527', 0.1701660607337399),
('f527-f274', 0.1569532758193278), ('f322-f68', 0.1427473586239801), ('f322-f13',
0.14266040573200417), ('f253-f766', 0.14227795511957936), ('f253-f404',
0.14224032896221173), ('f263-f766', 0.14202666454770593), ('f263-f404',
0.14198817696743674), ('f114-f766', 0.14178362653184115)]
```

4.2 Classification:

We tested following non-linear classifiers(on default parameters unless specified):

Classifier	**Accuracy**
KNeighborsClassifier	0.9077
DecisionTreeClassifier	0.9825
RandomForestClassifier(n_estimators=100)	0.9622
DecisionTreeClassifier(max_depth=9)	0.9825
GradientBoostingClassifier	0.9834
GradientBoostingClassifier(max_depth=9)	0.9895
GradientBoostingClassifier(n_estimators=400, max_depth=9)	0.9905
AdaBoostClassifier(n_estimators=50)	0.9803
AdaBoostClassifier(n_estimators=100)	0.9841

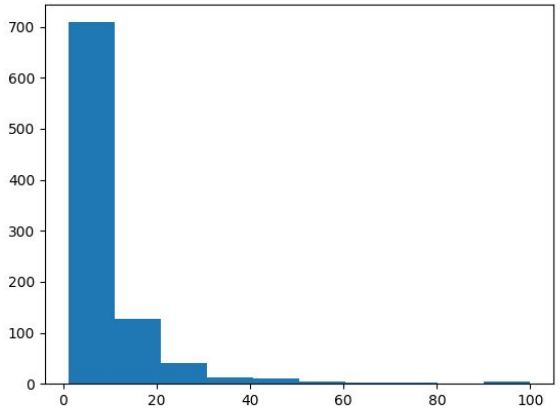
By adding 100 bivariate features, we were able to get 98.34% accuracy in classification(based on in-data validation) with **sklearn.ensemble.GradientBoostingClassifier**[7] on default parameters, and a Kaggle score of 0.76834.

[6] http://blog.datadive.net/selecting-good-features-part-i-univariate-selection/
[7] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

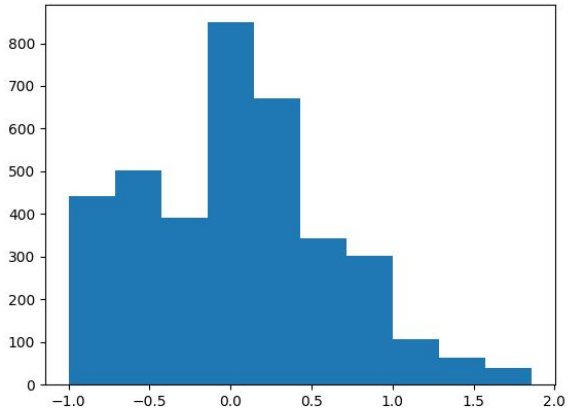
4.3 Regression:

Since there are way more non-default(y=0) than default(y!=0) in the dataset, regression on whole dataset would never give correct result because mean will dragged down by all those zeroes. Therefore, we only train our regressor on the default set. After testing various regression model and found that **sklearn.ensemble.RandomForestRegressor**[8] yields the best result on given data. Apply regression directly on default dataset yield a Kaggle score of 0.56094.

In order to further improve our score, we did a histogram plot of its Y.



From the image above, we can clearly see the Y's must follow some sort of exponential distribution. In order for our model to work better, we need to transform the Y's into a distribution that is close to standard normal distribution.



We apply log transformation[9] to the Y's (i.e., $Y = log(Y)$) and scale the data using a scaler. The second plot is what the Y's look like after the transformation.

To be able to get the raw prediction value back, we just need to apply inverse transformation on our prediction(i.e., $y = e^y$).

This got our Kaggle score down to 0.50507.

[8] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[9] https://stats.stackexchange.com/questions/137059/find-distribution-and-transform-to-normal-distribution

Finally, we used **sklearn.model_selection.GridSearchCV**[10] to find the best parameter for our ML model. We found that when *n_estimators=1000, [max_depth=9]*, our classifier and regressor works best.

The regressor's prediction is not as accurate as classifier's. Therefore, we need to merge classifier's result with regressor's. The underlying logic is: if both classifier & regressor predicts loss, then take regressor's value; if regressor/classifier doesn't agree with each other, then take the classifier's result(Y=1).

Doing all the above has improved our Kaggle score to 0.50281.

5. Future Improvement:

We observed that values from our regression never exceeded 30. This is because majority of Y's are concentrated in the first quantile (less than 25) thus dragging the means down. Since regression will always have the tendency to move towards the mean, it's unlikely to predict a large value with the training data we have. What if we first classify all Y's with a large value and then train on these Y's? In our in-sample test set(10000 sample), there are 978 Y>0, but only 47 Y>50. Unfortunately, there are too few Y's for any model to give an acceptable result. Alternatively, we can use some easy-to-overfit models(i.e., decision tree) to train on all Y>0, then cross-compare the result with the previous default/not-default classification result. This way, almost 2-3 large Y's will be predicted correctly(out of 47). Its influence on the MAE score is negligible so we didn't proceed further.

[10] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html