Data Mining on Loan Default Prediction

Boston College
Haotian Chen, Ziyuan Chen, Tianyu Xiang, Yang Zhou

May 1, 2015

Abstract

This Final Project investigates a variety of data mining techniques both theoretically and practically to predict the loan default rate. We have examined logistic regression, decision tree, extra tree classifier, generalized regression neural network and gradient boosted regression tree. The performance was evaluated by mean absolute error, and the best result we have for this figure is 0.51787 produced by two stage gradient boosted regression. The implementation of these methods was conducted both on Matlab and Python with scikit-learn library.

1. Introduction

The main problem that we try to solve in our final project is to predict the loan default rate. Accurate prediction of whether an individual will default on his or her loan, and how much loss it will incur has a practical importance for banks' risk management. Nowadays, banks have included a large amount of information in its evaluation of loan issuance, and some of these information has a vague causal relationship with the loan default rate. The growing amount of data due to improved data capture and data storage technology has bring us to a new perspective on this problem which is used to be accomplished by financial and economic analysis. Therefore, in this project, we would apply our knowledge from data mining class and test a variety of data mining approaches on this problem.

This data mining task, in nature, is a regression task as the target attribute, loan default rate is a continuous numerical value. However, as we improve our model, there is some variations. At first, we consider it as a one-stage regression model described above. We later found that defining this task as a two-stage regression model could improve the performance. Specifically, a two-stage task will first conduct a classification task to separate the default ones from the non-default ones, and then it will conduct a regression task in order to determine for each default one, how much loss it will incur. In the past, several sophisticated machine learning methods have been employed such as neural networks, random forests, support vector machine and particularly, the most popular one, logistic regression (Hand, 2009). A benchmark paper of two-stage model was written by Loterman where 5 datasets were tested (Loterman, 2012). Some of his major findings state that non-linear techniques, such as support vector machine and artificial neural nets outperform traditional linear techniques. Later literature tends to be in favor of his conclusion (Tobback, 2014). This brief literature review has pointed us to a good direction for this project.

Our project will include intensive amount of theoretical research and implementation as well. The comparative study of data mining techniques will help us build a comprehensive understanding of data mining as a subject and also find a specific application of it in financial industry. We also seek a hands-on experience and get familiarized with standard tools for data mining such as scikit-learn in python. The deliverables of this project will consist of two parts. Haotian Chen and Yang Zhou will be mainly responsible for the methodology research and literature review; therefore, they deliver the research paper as well as the presentation. On the other hand, Ziyuan Chen and Tianyu Xiang will be mainly responsible for the implementation of data mining techniques; thus, they produce necessary statistics and diagrams for the project.

Based on our preliminary research, we are expecting some serious challenges. The dataset we found for this project is quite large, and it is much larger than the ones we have worked with in our class in terms of dimensionalities and number of instances. The project requires strong computational power for some training models as well as long period of training time. Dimensionality reduction is necessary during prepossessing considering our limited computational power. Also, the large dataset has a significant amount of missing values that has to be treated for some training models. Moreover, familiarizing scikit-learn toolkit will take some effort as we have little experience with this platform.

2. Dataset Description and Performance Evaluation Criteria

The dataset we worked on is provided by Imperial College London (Imperial College London, 2015). The dataset was provided for the purpose of a world-wide data mining competition. It is separated into two parts, a training set and a testing set. For the training set, it has 770 attributes and a target attribute, “loss”. The “loss” attribute has been normalized to a score from 0 to 100 where 0 means no default, and the larger the score, the larger the loss incurred. This dataset contains 105,476 pieces of loan history, but in order to protect the privacy of borrowers, the name of these attributes are all erased and replaced with non-descriptive names such as “f1” and “f2”. We will train our model on the training set and test the model on the testing set. The dataset in average has 9% defaults, where the average loss incurred is 8.6. Figure 1 has shown its distribution. The percentage of instances that have missing value is 50%.

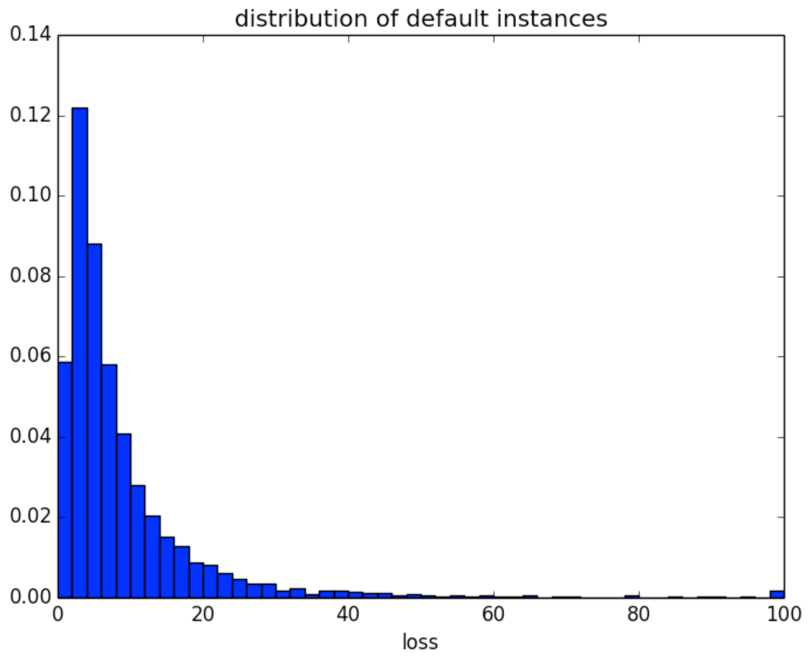


Figure 1. Default Instances Distribution Histogram

The choice of measurement metrics is fairly self-explanatory as the bank want to minimize the prediction error. The project will be evaluated by the mean absolute error (MAE), which is defined below:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

- n is the number of rows
- $\hat{y}_i$ is the predicted loss
- y_i is the actual loss

The absolute difference of actual loss and predicted loss is averaged over the size of test dataset. The lower the MAE, the better the prediction result is. The stubborn predictor of all default has a MEA of 0.8 which is a benchmark for our project. We don’t have a theoretical performance upper limit for comparison, but we will try to minimize MAE as much as possible.

3. Methodology

3.1 Generalized Regression Neural Network

The first model we used is the Generalized Regression Neural Network (GRNN), which is a kind of neural network that specializes in solving function approximation problems (Ahangar, Yahyazadehfar , & Pournaghshband , 2010). The GRNN model is generally constructed with four layers: Input Layer, Pattern Layer, Summation Layer, and Output Layer.

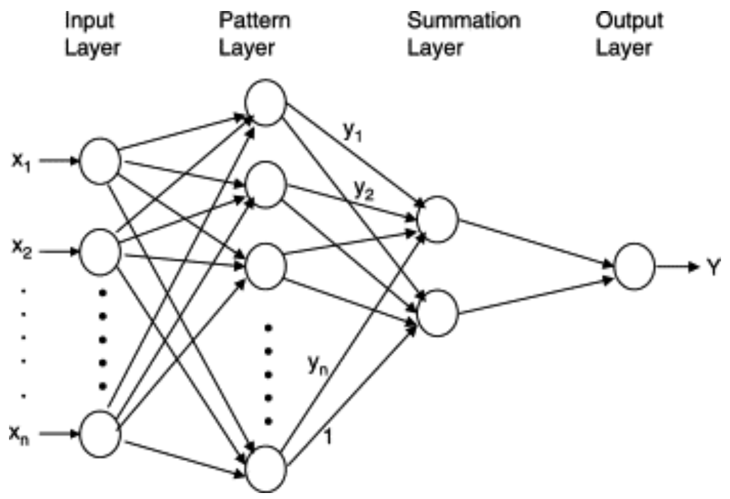


Figure 2. General Regression Network Architecture (Cigizoglu & Alp, 2006)

The first layer is designed to receive information. Each input neuron corresponds to each selected predictive feature. The pattern neurons are used to process the information and build up the relationships between input data and each training sample. Thus, the number of pattern neurons is equal to the number of training data. The pattern neurons are then combined through summation neurons to produce the final output.

A simple algorithm of the network is:

```
1. Input sample vector $x$ of input neurons
2. For each training vector $t_i$ :
        Calculate the Euclidean Distance:
            $di^2 = (x - t_i)^T(x - t_i)$
        Calculate the weights (Pattern Neurons) using the activation function:
            $w_i = e^{-\frac{di^2}{2\sigma^2}}$
        $\sigma$ is the smoothing parameter that needs to be tuned.
3. Calculate the two summation neurons:
    N = $\sum Y_i w_i$  ($Y_i$ is the output of the training sample $x_i$ )
```

D = $\sum w_i$

4. Output for x is

$$Y(x) = \frac{N}{D}$$

This one stage regression model is implemented on Matlab. Since the training data contains 770 attributes, which requires a huge computational power, we first apply the correlation-based attribute subset selection method for feature extraction. Also, we chose the mean value substitution method to deal with the missing values in the dataset. Compared to case deletion method, mean substitution is a more appropriate treatment in this case. Given the dataset contains non-descriptive features and large number of NaN values, mean substitution can guarantee that no relevant feature is eliminated, and the large number of instances also ensures the replaced values are reasonable.

After the preprocessing step, four features are selected and we use the linear combinations of these four as the predictor variables. With the training data and predictive features, we create the network using the build-in function "newgrnn". Then, the testing sample is applied to the network to predict the loss rate for each loan. MAE of the model is computed in the end in order to evaluate the model's performance.

3.2 Regression Decision Tree

We have been quite familiar with the classification decision tree from our class, but this task requires a regression decision tree which can produce continuous value as the result. The regression decision tree works in a similar fashion. The core algorithm for building decision tree is developed by Quinlan called ID3 (Quinlan, 1986). It's a top-down, greedy search through the space of possible branches. The ID3 algorithm can construct a regression decision tree by measuring standard deviation reduction for each step. The advantage of using regression decision tree is the fact that the algorithm will conduct the feature selection whereas for many other methods, we will have to figure out an appropriate feature selection before application of those methods. However, the disadvantage about this method is that we might into the problem of overfitting, and we have to find a good stopping criteria.

3.3 Other Techniques Explored

We have also chosen a few techniques from the scikit-learn package and test them based on our evaluation metrics. Some of these techniques include logistic regression, gradient boosted regression trees (GBRT), and supported vector machine. For feature selection, we tried extra tree classification, F regression and logistic classifier. Gradient boosted regression was able to produce the lowest MAE result. This technique was introduced by Friedman in 2001, and it was learned by different loss functions (Friedman, 2001). Detailed implementation was summarized by Natekin and Knoll who states that it is effective in capturing non-linear function dependencies despite high memory consumption required (Natekin & Knoll, 2013).

4. Result

The efficiency of the GRNN model is extremely low. It takes about 500 times for a total of about 200 thousand iterations with 16GB RAM and 8-core CPU. The MAE obtained by using this one-stage regression model is 0.81617. This result is much worse than our expectations and it does not even beat the benchmark, which is 0.81375.

The one-stage regression decision tree improves the result, and figure 3 shows its result over various maximum tree depths.

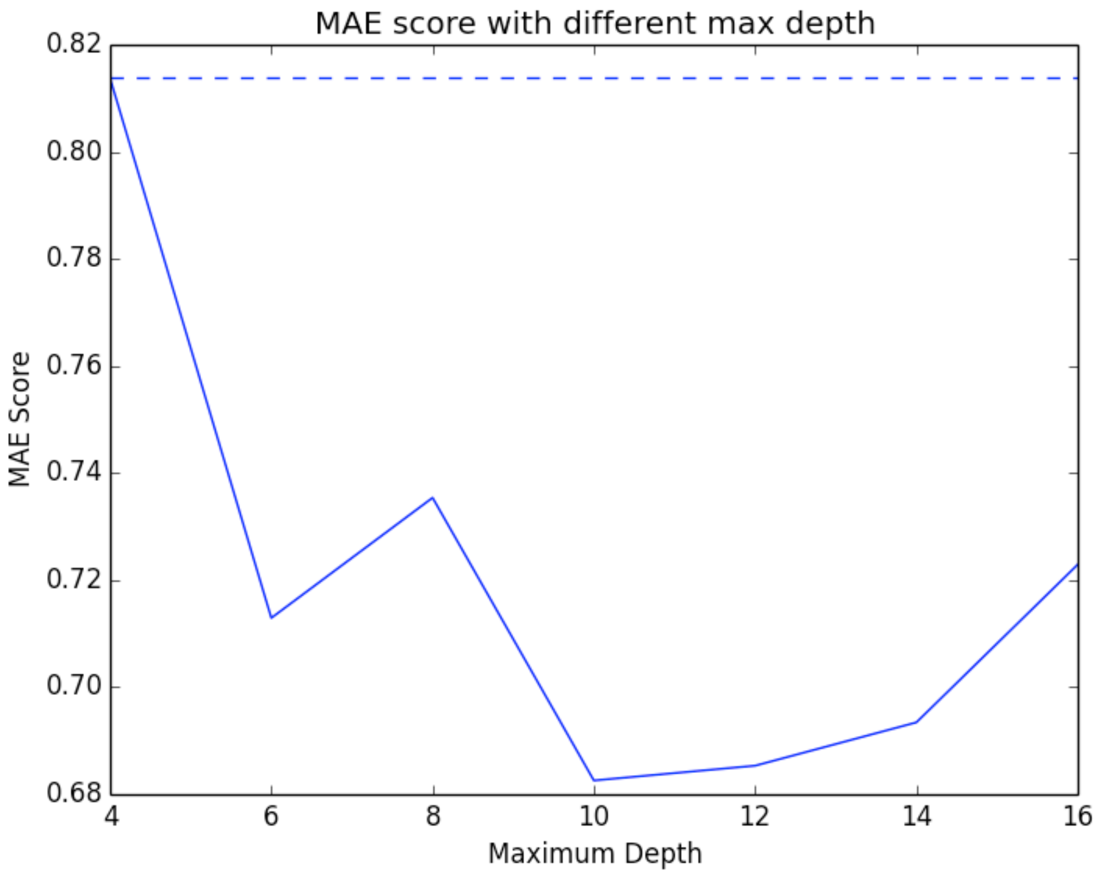


Figure 3. One-Stage Regression Decision Tree MAE over Depth

The lowest MAE we can reach using this method is 0.68258.

Among all the techniques we have explored, the best result was found using gradient boosted regression tree with a two-stage approach. Specifically, we first use gradient boosted classifier to predict a binary target, default or not, by training on the whole dataset. Next, we conduct the gradient boosted regression tree only on those that are predicted to default by training only on the default instances. This method gives us a MAE of 0.51. We also test a variety combination of classifier and regression method on these two stages. For example, decision tree on both stages has a MAE of 0.69; decision tree on first stage and gradient boosted regression tree on second stage have a MAE 0.57.

Both logistic regression and supported vector machine consume two much computational power and take too long to train and we have to discard the result.

5. Conclusion

In our experiment, the one stage GRNN model does not perform well. An alternative feature extraction may be used in the future to select more relevant attributes. Moreover, since the network we built with the training data is relatively large, input all the testing data to the network is extremely time consuming. A combination of classification and regression is more advantageous in improve the efficiency and gain a more accurate result. Based on the above reasoning, we found the two-stage gradient boosted regression tree was able to minimize the MAE. Among 675 teams in this competition, our score has a rank of 71, and we believe it is still an achievement for beginners in this field.

6. References

Ahangar, R. G., Yahyazadehfar , M., & Pournaghshband , H. (2010, February). The Comparison of Methods Artificial Neural Network with Linear Regression Using Specific Variables for Prediction Stock Price in Tehran Stock Exchange. *International Journal of Computer Science and Information Security, 7*(2), 41.

Cigizoglu, H. K., & Alp, M. (2006, February). Generalized Regression Neural Network in Modelling River Sediment Yield. *Advances in Engineering Software, 37*(2), 63-68.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics, 29*(5), 1189-1232.

Hand, D. J. (2009). Mining the past to determine the future: Problems and possibilities. *International Journal of Forecasting, 25*(3), 441-451.

Imperial College London. (2015). Retrieved from https://www.kaggle.com/c/loan-default-prediction/data

Loterman, G. B. (2012). Benchmarking regression algorithms for loss given default modeling. *International Journal of Forecasting, 28*, 161-170.

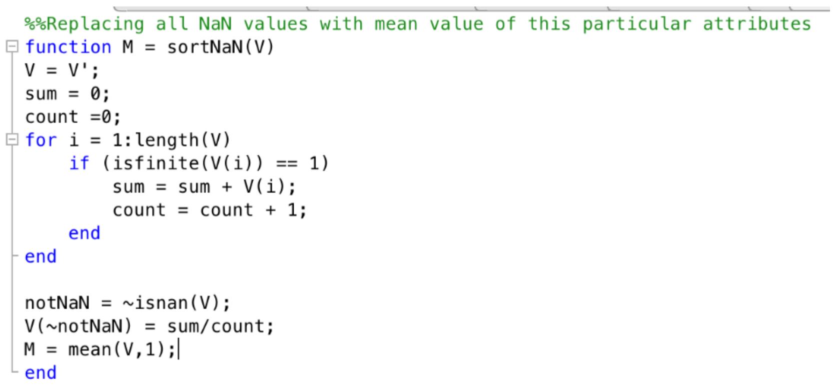
Quinlan, R. J. (1986). Induction of Decision Trees. *Machine Learning, 1*(1), 81-106.

Tobback, E. M. (2014). Forecasting Loss Given Default models: impact of account characteristics and the macroeconomic state. *Journal of the Operational Research Society, 65*, 376–392.

7. Appendices

GRNN Implementation Code

```
import pandas as pd
from scipy.stats.stats import pearsonr
import numpy
tp = pd.read_csv('train_v2.csv', iterator=True, chunksize=1000)
df = pd.concat(tp, ignore_index=True)

A = []
B = []

##### Example #########
for i in range(105470):
        A.append(df.values[i][2])
        B.append(df.values[i][3])

print pearsonr(A,B) #(corr,pvalue)
#compute the correlation of f2 and f3

#later on we find corr of f271 and f274 >.99
#and corr of f527,f528 >.99
```



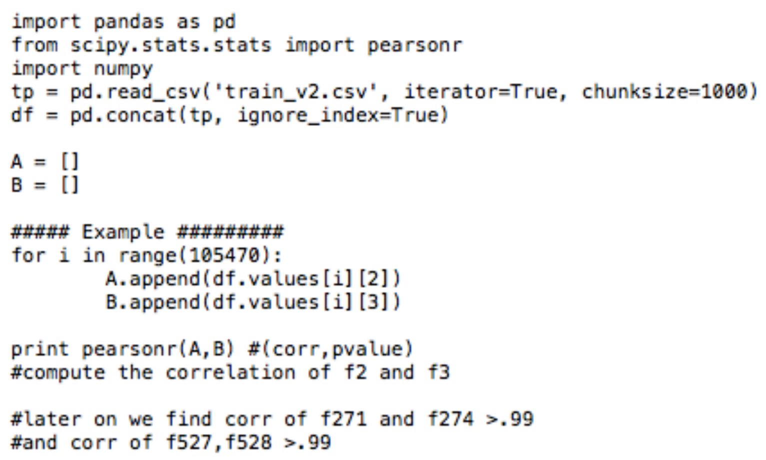
```
%%Replacing all NaN values with mean value of this particular attributes
function M = sortNaN(V)
V = V';
sum = 0;
count =0;
for i = 1:length(V)
    if (isfinite(V(i)) == 1)
        sum = sum + V(i);
        count = count + 1;
    end
end

notNaN = ~isnan(V);
V(~notNaN) = sum/count;
M = mean(V,1);
end
```

```
% clear all the NaN and replace it with mean value
n271 = sortNaN(copy271);
n274 = sortNaN(copy274);
n527 = sortNaN(copy527);
n528 = sortNaN(copy528);
nloss = sortNaN(losscopy);

%GRNN - 2 General Regression Neural Network (GRNN)
input = [n271-n274; n528-n527 ];
T = nloss;
net = newgrnn(input, T);
```

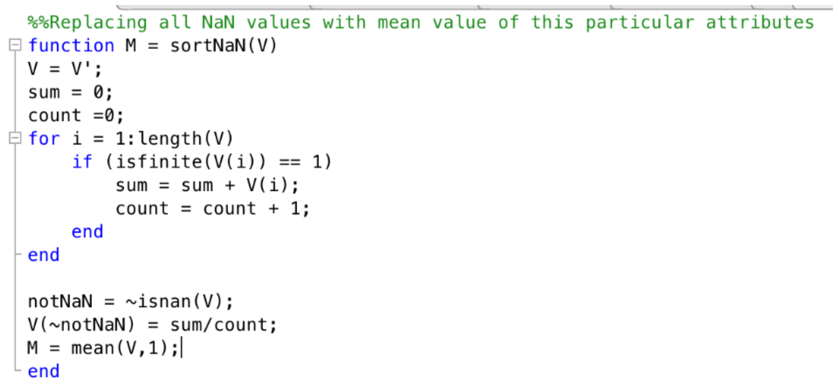
```
% clear all the NaN and replace it with mean value
nt271 = sortNaN(test271);
nt274 = sortNaN(test274);
nt527 = sortNaN(test527);
nt528 = sortNaN(test528);

%SIMULATION
temp = [nt271 - nt274, nt528 - nt527];

for i = 1:210945
    Predict = temp(:,i);
    newresult(i) = sim(net,Predict);
end
```

Appendices

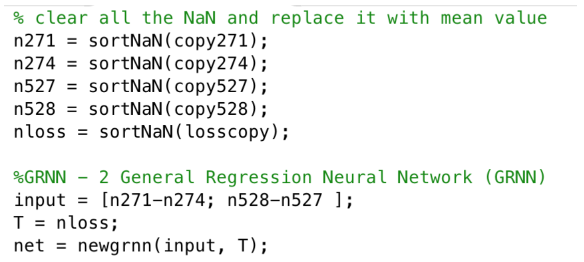
GRNN Matlab Implementation Code



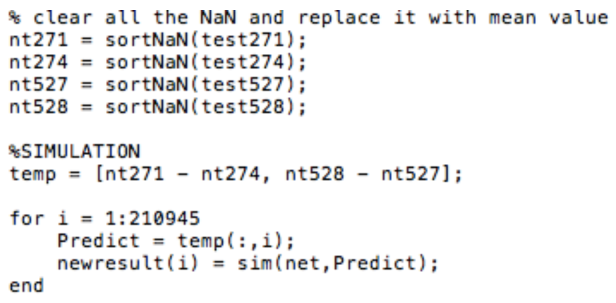
```
import pandas as pd
from scipy.stats.stats import pearsonr
import numpy
tp = pd.read_csv('train_v2.csv', iterator=True, chunksize=1000)
df = pd.concat(tp, ignore_index=True)

A = []
B = []

##### Example #########
for i in range(105470):
        A.append(df.values[i][2])
        B.append(df.values[i][3])

print pearsonr(A,B) #(corr,pvalue)
#compute the correlation of f2 and f3

#later on we find corr of f271 and f274 >.99
#and corr of f527,f528 >.99
```



```
%%Replacing all NaN values with mean value of this particular attributes
function M = sortNaN(V)
V = V';
sum = 0;
count =0;
for i = 1:length(V)
    if (isfinite(V(i)) == 1)
        sum = sum + V(i);
        count = count + 1;
    end
end

notNaN = ~isnan(V);
V(~notNaN) = sum/count;
M = mean(V,1);
end
```



```
% clear all the NaN and replace it with mean value
n271 = sortNaN(copy271);
n274 = sortNaN(copy274);
n527 = sortNaN(copy527);
n528 = sortNaN(copy528);
nloss = sortNaN(losscopy);

%GRNN - 2 General Regression Neural Network (GRNN)
input = [n271-n274; n528-n527 ];
T = nloss;
net = newgrnn(input, T);
```



```
% clear all the NaN and replace it with mean value
nt271 = sortNaN(test271);
nt274 = sortNaN(test274);
nt527 = sortNaN(test527);
nt528 = sortNaN(test528);

%SIMULATION
temp = [nt271 - nt274, nt528 - nt527];

for i = 1:210945
    Predict = temp(:,i);
    newresult(i) = sim(net,Predict);
end
```

Other Python Implementation Code

```
import numpy as np
from sklearn import tree
from sklearn.ensemble import GradientBoostingRegressor, GradientBoostingClassifier

def load_data():
    train = np.genfromtxt(open('train_v2.csv', 'rb'), delimiter=',', skip_header=1)
    test = np.genfromtxt(open('test_v2.csv', 'rb'), delimiter=',', skip_header=1)
    # clean data
    train = clean_data(train)
    test = clean_data(test)
    # separate instances and target
    xs = train[:, range(1, 770)]
    ys = train[:, -1]
    ts = test[:, range(1, 770)]
    # clear out two features causing overflow
    xs = np.delete(xs, 388, 1)
    ts = np.delete(ts, 388, 1)
    xs = np.delete(xs, 616, 1)
    ts = np.delete(ts, 616, 1)
    # make a binary target
    ysb = np.zeros(len(ys))
    ysb[ys > 0] = 1
    return xs, ys, ysb, ts

def clean_data(data):
    means = np.nanmean(data, axis=0)
    nan_index = np.where(np.isnan(data))
    data[nan_index] = means[nan_index[1]]
    return data

from sklearn.ensemble import ExtraTreesClassifier

# select 154 features using ExtraTreesClassifier
def feature_selection(xs, ys, ts):
    forest = ExtraTreesClassifier(n_estimators=250, random_state=0)
    forest.fit(xs, ys)
    importances = forest.feature_importances_
    std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
    indices = np.argsort(importances)[::-1]
    xs_selected = xs[:, indices[:150]]
    ts_selected = ts[:, indices[:150]]
    xs_selected = add_golden_features(xs, xs_selected)
    ts_selected = add_golden_features(ts, ts_selected)
    return xs_selected, ts_selected

# see Tianyu's code for computation of indices of these features
def add_golden_features(datas, tops):
    fs = np.array([datas[:, 520] - datas[:, 519]]).T
    fs = np.hstack((fs, np.array([datas[:, 520] + datas[:, 519]]).T))
    fs = np.hstack((fs, np.array([datas[:, 520] - datas[:, 271]]).T))
    fs = np.hstack((fs, np.array([datas[:, 520] + datas[:, 268]]).T))
    fs = np.hstack((fs, tops[:, 0:tops.shape[1]]))
    return fs

def single_stage_decision_tree(xs, ys, ts):
    xs_selected, ts_selected = feature_selection(xs, ys, ts)
    clf = tree.DecisionTreeRegressor(max_depth=10)
    clf = clf.fit(xs_selected, ys)
    Y = clf.predict(ts_selected)
    output_prediction(Y)

def two_stage_gradient_boosting(xs, ys, ysb, ts):
    # classification stage
    xsb, tsb = feature_selection(xs, ysb, ts)
    clf = GradientBoostingClassifier(n_estimators=200, learning_rate=0.3, min_samples_split=30,
min_samples_leaf=5)
    clf.fit(xsb, ysb)
    Y_bin = clf.predict(tsb)
    # regression stage
    ind_tsr = np.where(Y_bin > 0)[0]
    ts = ts[ind_tsr]
    ind_defaults = np.where(ys > 0)[0]
    xs = xs[ind_defaults]
    ysr = ys[ind_defaults]
    xsr, tsr = feature_selection(xs, ysr, ts)
    reg = GradientBoostingRegressor(n_estimators=200, learning_rate=0.1, min_samples_split=30,
min_samples_leaf=5, loss='lad')
    reg.fit(xsr, ysr)
    Y_defaults = reg.predict(tsr)
    Y = np.zeros(210944)
    Y[ind_tsr] = Y_defaults
    output_prediction(Y)

def output_prediction(Y):
    f = open('output.csv', 'w')
    f.write('id,loss\n')
    for i in range(len(Y)):
        if Y[i] > 100:
            Y[i] = 100
        elif Y[i] < 0:
            Y[i] = 0
        f.write(str(i+105472) + ',' + str(np.float(Y[i])) + '\n')
    f.close()

def main():
    two_stage_gradient_boosting(load_data())
```