Credit card Default

EE 660 Course Project

Project Type: Design a system based on real-world data

Number of student authors: 1

Author names: Qiong Wang

email contacts: wangqion@usc.edu

Date: Dec 4th 2019

1. Abstract

This research focused on the case of customer default payments in Taiwan. This research employed a binary variable, default payment Yes = 1, and No = 0, as the response variable. This study used 23 variables as explanatory variables: Amount of the given credit, Gender, Education, Marital status, Age, Amount of bill statement, Amount of previous payment. Author used different machine learning algorithms to find out the best model. The algorithms include KNN, Logistic Regression and Random forest. In the and the report will show the best model and its test error.

2. Introduction

2.1. Problem Type, Statement and Goals

This research focused on the case of customer default payments in Taiwan which could help banks and financial institutions to reduce risks. The main goal is to predict whether the bank should default payment for a customer on next month. Dataset contains 23 different features and 1 label for this classification problem.

Example sources of difficulty (nontriviality) include:

(1) A physical model that's inherently complicated and hard to abstract.
(2) High dimensionality of feature space.
(3) Significant amounts of preprocessing required.

2.2. Literature Review (Optional)

KNN, Logistic Regression, Random Forest.

2.3. Our Prior and Related Work (Mandatory)

I used KNN to train my dataset, which wasn't mention in EE660 course. Because, KNN is a good method for classification problem, and I have study in EE559, so, I will try this machine learning algorithms.

2.4. Overview of Our Approach

I totally used 3 different model: KNN, Logistic Regression, and random forest. In KNN, I used Euclidean metric, and change the value of K. For different model, I used 5-ford cv for model selection. In Logistic Regression, I used l1 penalty and l2 penalty, and I used different value of λ for different model. In random forest, I used different value of ntree for different model.

3. Implementation

Report your implementation details and results in the following subsections. You should mention which libraries and functions you used but avoid including code in your report. Your description of what your system does should be readable and understandable to a reader that isn't familiar with the functions and libraries you used but is familiar with the algorithms and techniques that were covered in EE 660. (For example, stating "we standardized all real-valued features, and recast all categorical features using one-hot encoding" and also stating the functions used in your code for this, is fine; stating only the functions used in your code is not fine.)

3.1. Data Set

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:

Amount of the given credit	Gender	Education	Marital status	Age	History of past paymen	Amount of bill statement	Amount of previous payment
>0	1 = male 2 = female	1 = graduate school 2 = university 3 = high school 4 = others	1 = married; 2 = single; 3 = others		from April to September, 2005	from April to September, 2005	from April to September, 2005

Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

Amount of bill statement (NT dollar). X12 = amount of bill statement in September 2005; X13 = amount of bill statement in August 2005; . . .; X17 = amount of bill statement in April 2005.

Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

3.2. Preprocessing, Feature Extraction, Dimensionality Adjustment

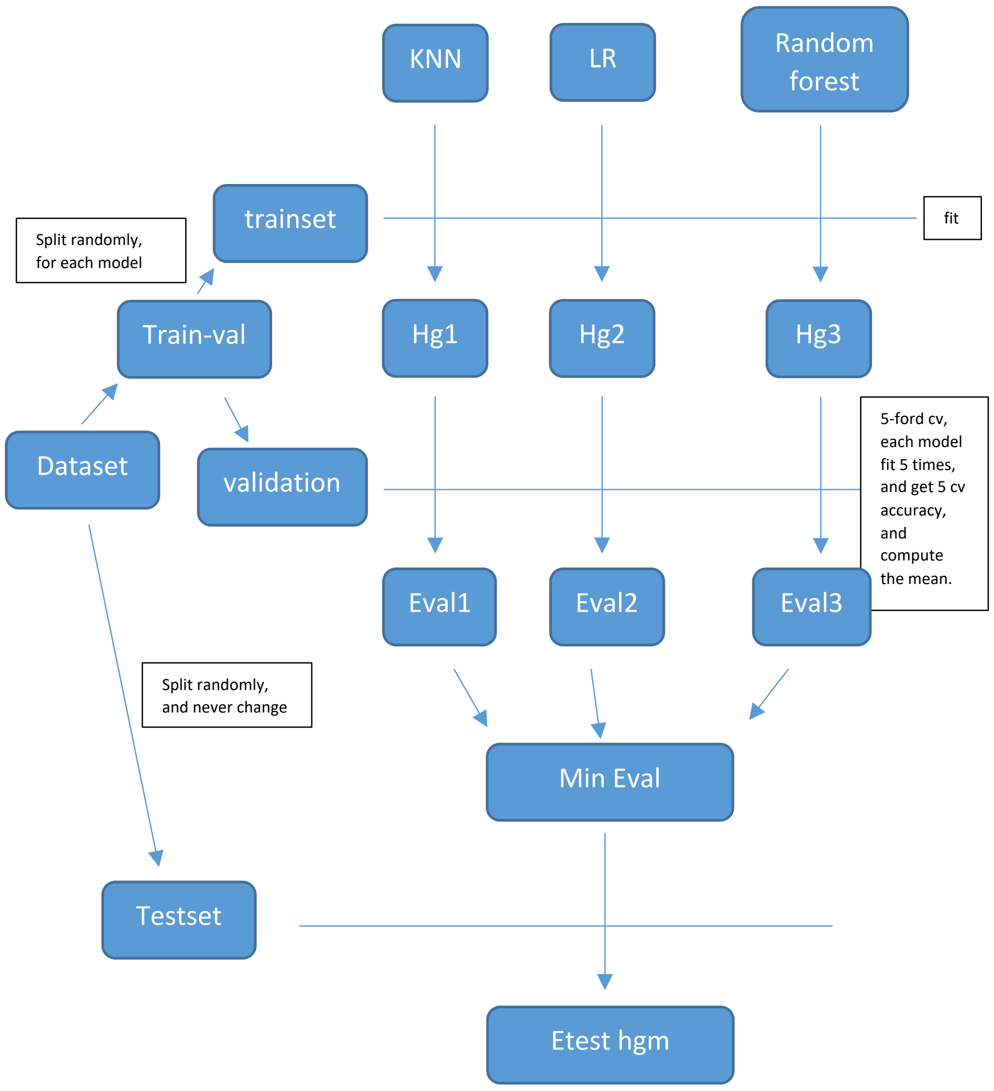
The dataset doesn't have any missing data, I don't need to solve this problem.

For data preprocess, first, I randomly split the data to training data and test data, training: test = 2:1. Then I split training set to training set and validation set. Then I standardized *Amount of bill statement* and *Amount of previous payment* (totally 12 features) in training data and depend on the mean and variance for the training set I used them to standardize the validation set and test set. Moreover, I also extraction some features from dataset, I replace *Amount of bill statement* and *Amount of previous payment* (totally 12 features) to their mean and std, and then I standardized them.

3.3. Dataset Methodology

I split the dataset to training set, validation set and test set. Test set: 10000, validation set: 3333, training set: 16667. I use 5-ford validation for model selection.

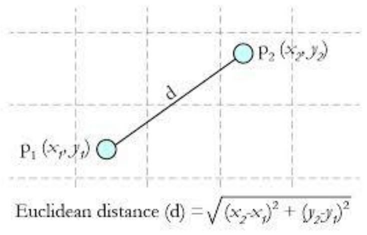
For model selection, I used 5-ford cross validation. Each time, I fit the training set in a new model, then I will use the cross validation to compute cv-accuracy. In order to reduce the random error, I repeat 5 times when I fit the training set for the same model, and I also get 5 cv-accuracy, then, I compute the mean of these cv-accuracy.


KNN
LR
Random forest
trainset
fit
Split randomly, for each model
Train-val
Hg1
Hg2
Hg3
Dataset
validation
5-ford cv, each model fit 5 times, and get 5 cv accuracy, and compute the mean.
Eval1
Eval2
Eval3
Split randomly, and never change
Min Eval
Testset
Etest hgm

3.4. Training Process

1. KNN

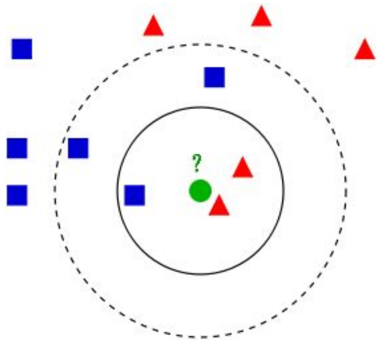
I change the value of k from 1 to 200, and the step is 10. When I compute the distance between each two data points, I used Euclidean metric:



And for n dimensional data space, the formula of Euclidean metric:

$$d(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

Different value of k would get different result, as we can see from the picture below, when I used KNN for classification problem, I vote the k nearest data points to predict an unknow data point. When k = 3, the label of green data point should be red, because 2 red data points, and one blue data point. However, when k = 5, the label of green data point should be blue, because, there are 3 blue data points and 2 res data points. So, for different value of k we have different models, and when k is too large, the model might not complex enough to fit the high dimensional data, and if k is too small, the model will overfitting. So, in order to avoid overfitting, I change the value of K for a large range from 1 to 200, I find that, the average of cross-validation error from the validation sets will decrease and then increase, the minimum value of cross-validation error point to the best model.



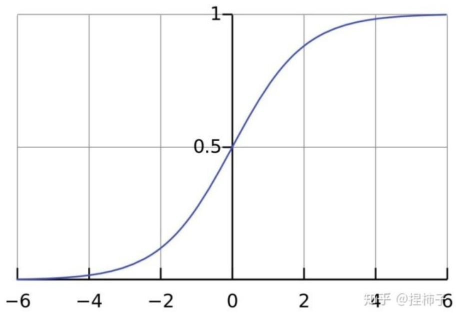
KNN is one of the simplest methods for classification problem, so, I this question, I firstly use KNN model to train my training set. When I use KNN to train the training data, I change the value of K from 1 to 200, the set is 10, so the complexity of KNN is equal to 20. For These 20 different hypotheses, I fit the training set and get the best model. For these 20 different models, I used 5-ford cv to calculate the average cross-validation error from the validation sets, the minimum cv error point to the best model in KNN. Because I have a large range of k value, so, the model experiences the overfitting, best fitting, and lack of fitting, and the average cross-validation error from the validation sets decrease first and then increase.

2. Logistic Regression

When transfer the result of linearly regression to a sigma function, the new result is logistic regression.

Sigma function:

$$g(z) = \frac{1}{1+e^{-z}}$$



When z goes to positive infinite, g(z) close to 1, and when z goes to negative infinite, g(z) close to 0. Therefore, this model is good to solve classification problems. Moreover, sigma function also has a good property:

$$\begin{aligned} g'(z) &= \left(\frac{1}{1+e^{-z}}\right)' = \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

And if we assume that: $z = \theta^T x$, then we get the formulation for logistic regression:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

The objective function:

$$p(y|x;\theta) = h_\theta(x)^y (1-h_\theta(x))^{1-y}$$

The MSE formulation:

$$\begin{aligned} L(\theta) &= p(Y|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1-h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

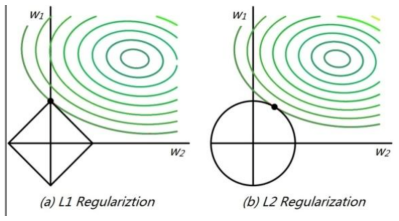
In order to easy computation, I logarithm the MSE formulation, which is also called the loss function:

$$\begin{aligned} J(\theta) &= -\log L(\theta) \\ &= -\sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(x^{(i)}) + \left(1-y^{(i)}\right) \log\left(1-h_\theta(x^{(i)})\right)\right) \end{aligned}$$

In order to get the sparse result, or avoid overfitting, I decided to use the l1-penalty and l2-penality. L1-regularization could produce the sparse result, and l2-regularization could avoid overfitting.

$$\begin{aligned} L_1 &= |w_1| + |w_2| + \ldots + |w_n|, \quad \frac{\partial L_1}{\partial w_i} = sign(w_i) \quad = 1 \text{ or -1} \\ L_2 &= \frac{1}{2} w_1^2 + w_2^2 + \ldots + w_n^2, \quad \frac{\partial L_2}{\partial w_i} = w_i \end{aligned}$$

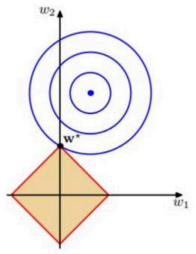
The graph of these two functions:



It is obvious that, l1 regularization could get sparse result, and l2 regularization could avoid overfitting.

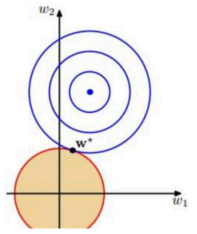
So, the loss function of l1 regularization:

$$L = L_{loss} + \lambda \sum_j |w_j|$$



the loss function of l2 regularization:

$$L = L_{loss} + \lambda \sum_j w_j^2$$



After I used KNN model, I realized that, these 23 features could be highly mutually relevant, redundant and irrelevant for prediction. So, I plan to use l1 regularization and l2 regularization to get a better result.

When I implement the l1 and l2 regularization, value of λ change from 0.01 to 100 by exponent increase. The complexity of KNN is equal to 40. For these different models, I used 5-ford cv to calculate the average cross-validation error from the validation sets, the minimum cv error point to the best model in logistic regression.

3. Random forest
   Train a random forest with the "bag" set and the following parameters: n_estimators = 1 to B, step size 1, B=30. bootstrap = True, max_features = 3. For each value of number of trees (estimators), repeat the experiment 10 times (selecting different bag samples every time), and used 5-ford cv to calculate the average cross-validation error from the validation sets, the minimum cv error point to the best model in random forest.

3.5. Model Selection and Comparison of Results

1. KNN

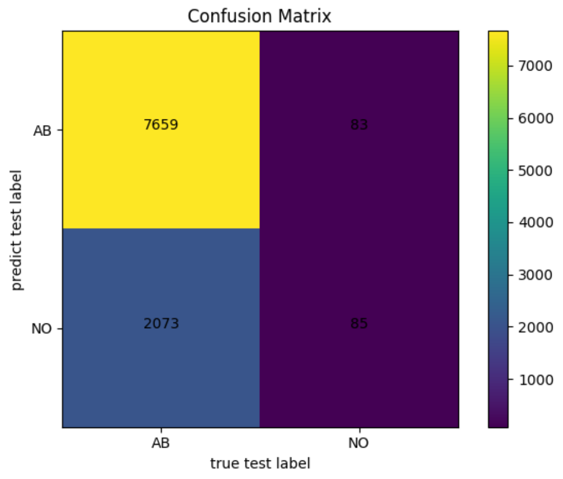
Firstly, I didn't preprocess the dataset, I directly use the raw data for training, the result:

the best model is when k = 91

cv error rate = 0.2202623093176752

test error = 0.21777777777777774

train error = 0.22232122967253287



For each model:

when k = 1 cv error rate = 0.31916296980794523

when k = 11 cv error rate = 0.23413090005580806

when k = 21 cv error rate = 0.2288551615649448

when k = 31 cv error rate = 0.22388017290273599

when k = 41 cv error rate = 0.22463420705364057

when k = 51 cv error rate = 0.22176958198501706

when k = 61 cv error rate = 0.22116683204497378

when k = 71 cv error rate = 0.2213175479455014

when k = 81 cv error rate = 0.22101588882031287

when k = 91 cv error rate = 0.2202623093176752

when k = 101 cv error rate = 0.22056408210493061

when k = 111 cv error rate = 0.2213175479455014

when k = 121 cv error rate = 0.2216193207327566

when k = 131 cv error rate = 0.2219205252096781

when k = 141 cv error rate = 0.22282482061284314

when k = 151 cv error rate = 0.22252361613592164

when k = 161 cv error rate = 0.22101611614444638

when k = 171 cv error rate = 0.2207145706813246

when k = 181 cv error rate = 0.22101600248237951

when k = 191 cv error rate = 0.22071445701925785

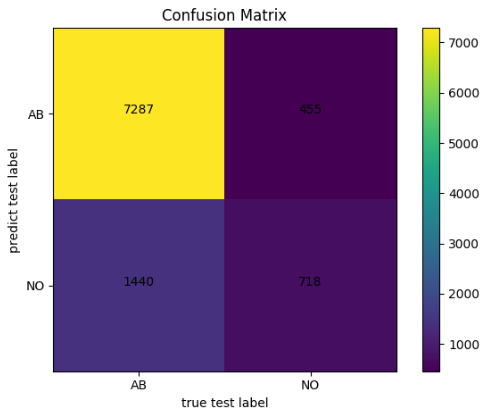
Then I directly standardized the Amount of bill statement and Amount of previous payment, the result is:

the best model is when k = 21

cv error rate = 0.18468289988304176

test error = 0.19141414141414137

train error = 0.17739659909408179



For each model:

when k = 1 cv error rate = 0.26518792317808226

when k = 11 cv error rate = 0.19161844553484253

when k = 21 cv error rate = 0.18468289988304176

when k = 31 cv error rate = 0.18619039987451713

when k = 41 cv error rate = 0.18558764993447385

when k = 51 cv error rate = 0.18739658172700424

when k = 61 cv error rate = 0.187246206812677

when k = 71 cv error rate = 0.18694466134955534

when k = 81 cv error rate = 0.1876984681763263

when k = 91 cv error rate = 0.1889046500288133

when k = 101 cv error rate = 0.19011049089510013

when k = 111 cv error rate = 0.18950751363092344

when k = 121 cv error rate = 0.18980905909404522

when k = 131 cv error rate = 0.1901100362468331

when k = 141 cv error rate = 0.1910144453120649

when k = 151 cv error rate = 0.1905622976104825

when k = 161 cv error rate = 0.18965766122111705

when k = 171 cv error rate = 0.18980849078371131

when k = 181 cv error rate = 0.19026109313356088

when k = 191 cv error rate = 0.1899593203463057

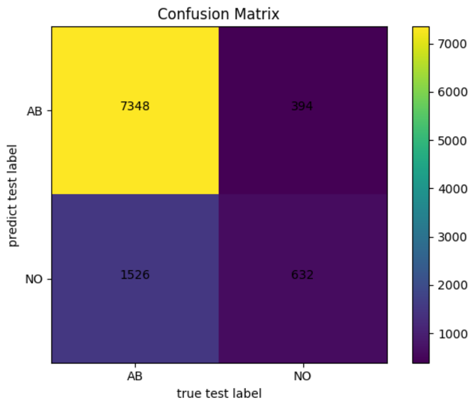
In the end, instead of using the features Amount of bill statement and Amount of previous payment, I calculate the mean and std of Amount of bill statement and Amount of previous payment as 4 new features, and the result:

the best model is when k = 21

cv error rate = 0.18709560457421615

test error = 0.18393939393939397

train error = 0.18229746788445833



For each model:

when k = 1 cv error rate = 0.26111882118797314

when k = 11 cv error rate = 0.19086407039773778

when k = 21 cv error rate = 0.18709560457421615

when k = 31 cv error rate = 0.1907138091454772

when k = 41 cv error rate = 0.19116572952292632

when k = 51 cv error rate = 0.1913165590855206

when k = 61 cv error rate = 0.19176882044916987

when k = 71 cv error rate = 0.1916182182107089

when k = 81 cv error rate = 0.19222130913695268

when k = 91 cv error rate = 0.1919197636738309

when k = 101 cv error rate = 0.19282428640112936

when k = 111 cv error rate = 0.19387929770482193

when k = 121 cv error rate = 0.1946333318557265

when k = 131 cv error rate = 0.19493476365678153

when k = 141 cv error rate = 0.1962917750718629

when k = 151 cv error rate = 0.19779938872540492

when k = 161 cv error rate = 0.19795033195006595

when k = 171 cv error rate = 0.19900579790202566

when k = 181 cv error rate = 0.2011162751576776

when k = 191 cv error rate = 0.20277403640141356

2. Logistic Regression – l1

Firstly, I didn't preprocess the dataset, I directly use the raw data for training, the result:

the best model is when lambda = 0.01584893192461114

cv error rate = 0.18374914327217196

test error = 0.18939393939393945

train error = 0.19053983812281872

For each model:

```
when lambda = 0.01 cv error rate = 0.18377928645227737
when lambda = 0.012589254117941675 cv error rate = 0.18377928645227737
when lambda = 0.01584893192461114 cv error rate = 0.18374914327217196
when lambda = 0.01995262314968881 cv error rate = 0.18374914327217196
when lambda = 0.025118864315095822 cv error rate = 0.18389985917269924
when lambda = 0.03162277660168326 cv error rate = 0.1838395728124883
when lambda = 0.039810717055349776 cv error rate = 0.18386971599259394
when lambda = 0.050118723362727303 cv error rate = 0.18377928645227737
when lambda = 0.06309573444801943 cv error rate = 0.18377928645227737
when lambda = 0.07943282347242829 cv error rate = 0.18380945236479618
when lambda = 0.1000000000000002 cv error rate = 0.18380942963238278
when lambda = 0.125892541179417 cv error rate = 0.18386973872500723
when lambda = 0.15848931924611173 cv error rate = 0.18386976145742062
when lambda = 0.1995262314968885 cv error rate = 0.18399044783990937
when lambda = 0.25118864315095873 cv error rate = 0.1840809455774659
when lambda = 0.3162277660168389 cv error rate = 0.1840809455774659
when lambda = 0.39810717055349853 cv error rate = 0.1840507796649471
when lambda = 0.501187233627274 cv error rate = 0.18396028192739045
when lambda = 0.6309573444801956 cv error rate = 0.18408083191539915
when lambda = 0.7943282347242846 cv error rate = 0.1840506887352935
when lambda = 1.000000000000004 cv error rate = 0.18411097509550467
when lambda = 1.2589254117941726 cv error rate = 0.18438226371645405
when lambda = 1.5848931924611207 cv error rate = 0.18474398187772
when lambda = 1.995262314968889 cv error rate = 0.1845027227748094
when lambda = 2.5118864315095926 cv error rate = 0.18489462958100744
when lambda = 3.1622776601683955 cv error rate = 0.18525650686916717
when lambda = 3.9810717055349936 cv error rate = 0.1853772159840691
when lambda = 5.01187233627275 cv error rate = 0.18504543641118865
when lambda = 6.309573444801969 cv error rate = 0.18582947734771849
when lambda = 7.943282347242862 cv error rate = 0.18655298186749047
when lambda = 10.000000000000062 cv error rate = 0.18637194092755072
when lambda = 12.589254117941753 cv error rate = 0.18619096818485092
when lambda = 15.84893192461124 cv error rate = 0.18573859315913488
when lambda = 19.952623149688932 cv error rate = 0.1855875362724071
when lambda = 25.118864315095976 cv error rate = 0.18622054305462277
when lambda = 31.62277660168402 cv error rate = 0.1876978998659924
when lambda = 39.81071705535002 cv error rate = 0.18860276357949135
when lambda = 50.11872336272761 cv error rate = 0.18926561802043873
when lambda = 63.09573444801982 cv error rate = 0.18848175894321562
when lambda = 79.43282347242878 cv error rate = 0.1897781202794725
```

Then I directly standardized the Amount of bill statement and Amount of previous payment the result is:

```
the best model is when lambda = 0.01

cv error rate = 0.18377928645227737
```

```
test error = 0.1894949494949495

train error = 0.1906140937105517
```

For each model:

```
when lambda = 0.01 cv error rate = 0.18377928645227737

when lambda = 0.012589254117941675 cv error rate = 0.18377928645227737

when lambda = 0.01584893192461114 cv error rate = 0.18377928645227737

when lambda = 0.01995262314968881 cv error rate = 0.18377928645227737

when lambda = 0.025118864315095822 cv error rate = 0.18377928645227737

when lambda = 0.031622776601683826 cv error rate = 0.18377928645227737

when lambda = 0.039810717055349776 cv error rate = 0.18377928645227737

when lambda = 0.050118723362727303 cv error rate = 0.18377928645227737

when lambda = 0.06309573444801943 cv error rate = 0.18377928645227737

when lambda = 0.07943282347242829 cv error rate = 0.18377928645227737

when lambda = 0.1000000000000002 cv error rate = 0.18380945236479618

when lambda = 0.125892541179417 cv error rate = 0.18393011601487164

when lambda = 0.15848931924611173 cv error rate = 0.18389995010235283

when lambda = 0.1995262314968885 cv error rate = 0.18377928645227737

when lambda = 0.25118864315095873 cv error rate = 0.18393011601487164

when lambda = 0.3162277660168389 cv error rate = 0.18393011601487164

when lambda = 0.39810717055349853 cv error rate = 0.18408083191539915

when lambda = 0.501187233627274 cv error rate = 0.18417132965295568

when lambda = 0.6309573444801956 cv error rate = 0.18438237737852092

when lambda = 0.7943282347242846 cv error rate = 0.1842316614779934

when lambda = 1.000000000000004 cv error rate = 0.18429194783820435

when lambda = 1.2589254117941726 cv error rate = 0.1844126342206932

when lambda = 1.5848931924611207 cv error rate = 0.1842316614779934

when lambda = 1.995262314968889 cv error rate = 0.18393000235280488

when lambda = 2.5118864315095926 cv error rate = 0.18402031823105447

when lambda = 3.1622776601683955 cv error rate = 0.18408060459126552
when lambda = 3.9810717055349936 cv error rate = 0.18483452508010334

when lambda = 5.01187233627275 cv error rate = 0.18498524098063085

when lambda = 6.309573444801969 cv error rate = 0.18528667278168598

when lambda = 7.943282347242862 cv error rate = 0.1855277045604632

when lambda = 10.000000000000062 cv error rate = 0.18604018408708323

when lambda = 12.589254117941753 cv error rate = 0.18649262731003946

when lambda = 15.84893192461124 cv error rate = 0.1863417977474452

when lambda = 19.952623149688932 cv error rate = 0.18661308636839458

when lambda = 25.118864315095976 cv error rate = 0.18679383178696107

when lambda = 31.62277660168402 cv error rate = 0.18664288856229994

when lambda = 39.81071705535002 cv error rate = 0.18604025228432342

when lambda = 50.11872336272761 cv error rate = 0.1872456385023431

when lambda = 63.09573444801982 cv error rate = 0.18709503626388246

when lambda = 79.43282347242878 cv error rate = 0.18769824085219278
```

In the end, instead of using the features Amount of bill statement and Amount of previous payment, I calculate the mean and std of Amount of bill statement and Amount of previous payment as 4 new features, and the result:

```
the best model is when lambda = 0.012589254117941675

cv error rate = 0.1846534386753369

test error = 0.18020202020202015

train error = 0.18994579342095497
```

For each model:

```
when lambda = 0.01 cv error rate = 0.18474389094806676

when lambda = 0.012589254117941675 cv error rate = 0.1846534386753369

when lambda = 0.01584893192461114 cv error rate = 0.18474386821565347

when lambda = 0.01995262314968881 cv error rate = 0.1848041545758644

when lambda = 0.025118864315095822 cv error rate = 0.18471372503554784

when lambda = 0.03162277660168382 6 cv error rate = 0.18465346140775019
when lambda = 0.039810717055349776 cv error rate = 0.1848343432207965

when lambda = 0.050118723362727303 cv error rate = 0.18474386821565347

when lambda = 0.06309573444801943 cv error rate = 0.18498503638891073

when lambda = 0.07943282347242829 cv error rate = 0.18483429775596993

when lambda = 0.10000000000000002 cv error rate = 0.18498501365649744

when lambda = 0.125892541179417 cv error rate = 0.18504534548153517

when lambda = 0.15848931924611173 cv error rate = 0.18486446366848885

when lambda = 0.1995262314968885 cv error rate = 0.18489458411618087

when lambda = 0.25118864315095873 cv error rate = 0.18489472051066114

when lambda = 0.3162277660168389 cv error rate = 0.1848344114180367

when lambda = 0.39810717055349853 cv error rate = 0.18495500687087207

when lambda = 0.501187233627274 cv error rate = 0.18522650008354158

when lambda = 0.6309573444801956 cv error rate = 0.1849249546204199

when lambda = 0.7943282347242846 cv error rate = 0.18513604781081183

when lambda = 1.000000000000004 cv error rate = 0.18522656828078188

when lambda = 1.2589254117941726 cv error rate = 0.185437616006347

when lambda = 1.5848931924611207 cv error rate = 0.18561838415732657

when lambda = 1.995262314968889 cv error rate = 0.18528671824651255

when lambda = 2.5118864315095926 cv error rate = 0.1849249546204199

when lambda = 3.1622776601683955 cv error rate = 0.1854074046290014

when lambda = 3.9810717055349936 cv error rate = 0.18528655911961922

when lambda = 5.01187233627275 cv error rate = 0.18549760684518435

when lambda = 6.309573444801969 cv error rate = 0.18570856364109622

when lambda = 7.943282347242862 cv error rate = 0.18595009553296715

when lambda = 10.000000000000062 cv error rate = 0.1861611659909459

when lambda = 12.589254117941753 cv error rate = 0.18591988415562155

when lambda = 15.84893192461124 cv error rate = 0.18573875228602843

when lambda = 19.952623149688932 cv error rate = 0.18549722039415728

when lambda = 25.118864315095976 cv error rate = 0.18591924764804768

when lambda = 31.62277660168402 cv error rate = 0.1858887407493286

when lambda = 39.81071705535002 cv error rate = 0.1866125180580609
```

when lambda = 50.11872336272761 cv error rate = 0.18793877251787627

when lambda = 63.09573444801982 cv error rate = 0.18878341806840415

when lambda = 79.43282347242878 cv error rate = 0.19191930902556376

3. Logistic Regression – l2

Firstly, I didn't preprocess the dataset, I directly use the raw data for training, the result:

the best model is when lambda = 0.125892541179417

cv error rate = 0.21905658211345524

test error = 0.21797979797979794

train error = 0.22395485260265835

For each model:

when lambda = 0.01 cv error rate = 0.22056374111873034

when lambda = 0.012589254117941675 cv error rate = 0.22056374111873034

when lambda = 0.01584893192461114 cv error rate = 0.22056374111873034

when lambda = 0.01995262314968881 cv error rate = 0.22056374111873034

when lambda = 0.025118864315095822 cv error rate = 0.22056374111873034

when lambda = 0.031622776601683826 cv error rate = 0.22056374111873034

when lambda = 0.039810717055349776 cv error rate = 0.22056374111873034

when lambda = 0.050118723362727303 cv error rate = 0.22056374111873034

when lambda = 0.06309573444801943 cv error rate = 0.22056374111873034

when lambda = 0.07943282347242829 cv error rate = 0.22056374111873034

when lambda = 0.1000000000000002 cv error rate = 0.22056374111873034

when lambda = 0.125892541179417 cv error rate = 0.21905658211345524

when lambda = 0.15848931924611173 cv error rate = 0.22056374111873034

when lambda = 0.1995262314968885 cv error rate = 0.22071445701925785

when lambda = 0.25118864315095873 cv error rate = 0.22056374111873034

when lambda = 0.3162277660168389 cv error rate = 0.22056374111873034

```
when lambda = 0.39810717055349853 cv error rate = 0.22056374111873034

when lambda = 0.501187233627274 cv error rate = 0.22056374111873034

when lambda = 0.6309573444801956 cv error rate = 0.22056374111873034

when lambda = 0.7943282347242846 cv error rate = 0.22056374111873034

when lambda = 1.000000000000004 cv error rate = 0.22056374111873034

when lambda = 1.2589254117941726 cv error rate = 0.22056374111873034

when lambda = 1.5848931924611207 cv error rate = 0.22056374111873034

when lambda = 1.995262314968889 cv error rate = 0.22056374111873034

when lambda = 2.5118864315095926 cv error rate = 0.22056374111873034

when lambda = 3.1622776601683955 cv error rate = 0.22056374111873034

when lambda = 3.9810717055349936 cv error rate = 0.22056374111873034

when lambda = 5.01187233627275 cv error rate = 0.22056374111873034

when lambda = 6.309573444801969 cv error rate = 0.22071445701925785

when lambda = 7.943282347242862 cv error rate = 0.22056374111873034

when lambda = 10.000000000000062 cv error rate = 0.22056374111873034

when lambda = 12.589254117941753 cv error rate = 0.22056374111873034

when lambda = 15.84893192461124 cv error rate = 0.22056374111873034

when lambda = 19.952623149688932 cv error rate = 0.22056374111873034

when lambda = 25.118864315095976 cv error rate = 0.22056374111873034

when lambda = 31.62277660168402 cv error rate = 0.22056374111873034

when lambda = 39.81071705535002 cv error rate = 0.22056374111873034

when lambda = 50.11872336272761 cv error rate = 0.22056374111873034

when lambda = 63.09573444801982 cv error rate = 0.22056374111873034

when lambda = 79.43282347242878 cv error rate = 0.22056374111873034
```

Then I directly standardized the Amount of bill statement and Amount of previous payment, the result is:

```
the best model is when lambda = 0.01

cv error rate = 0.18377928645227737

test error = 0.18939393939393945
```

```
train error = 0.1906140937105517


For each model:
when lambda = 0.01 cv error rate = 0.18377928645227737
when lambda = 0.012589254117941675 cv error rate = 0.18377928645227737
when lambda = 0.01584893192461114 cv error rate = 0.18377928645227737
when lambda = 0.01995262314968881 cv error rate = 0.18377928645227737
when lambda = 0.025118864315095822 cv error rate = 0.18377928645227737
when lambda = 0.031622776601683826 cv error rate = 0.18377928645227737
when lambda = 0.039810717055349776 cv error rate = 0.18377928645227737
when lambda = 0.050118723362727303 cv error rate = 0.18377928645227737
when lambda = 0.06309573444801943 cv error rate = 0.18377928645227737
when lambda = 0.07943282347242829 cv error rate = 0.18377928645227737
when lambda = 0.1000000000000002 cv error rate = 0.18377928645227737
when lambda = 0.125892541179417 cv error rate = 0.18377928645227737
when lambda = 0.15848931924611173 cv error rate = 0.18377928645227737
when lambda = 0.1995262314968885 cv error rate = 0.18377928645227737
when lambda = 0.25118864315095873 cv error rate = 0.18377928645227737
when lambda = 0.3162277660168389 cv error rate = 0.18377928645227737
when lambda = 0.39810717055349853 cv error rate = 0.18393011601487164
when lambda = 0.501187233627274 cv error rate = 0.18393011601487164
when lambda = 0.6309573444801956 cv error rate = 0.18393011601487164
when lambda = 0.7943282347242846 cv error rate = 0.18393011601487164
when lambda = 1.000000000000004 cv error rate = 0.18393011601487164
when lambda = 1.2589254117941726 cv error rate = 0.1839302296769384
when lambda = 1.5848931924611207 cv error rate = 0.18408105923953266
when lambda = 1.995262314968889 cv error rate = 0.1839302296769384
when lambda = 2.5118864315095926 cv error rate = 0.18453332060318195
when lambda = 3.1622776601683955 cv error rate = 0.18468403650370946
when lambda = 3.9810717055349936 cv error rate = 0.18468403650370946
when lambda = 5.01187233627275 cv error rate = 0.1845332069411152
```

when lambda = 6.309573444801969 cv error rate = 0.1848346387421701

when lambda = 7.943282347242862 cv error rate = 0.18513595688115847

when lambda = 10.000000000000062 cv error rate = 0.18483452508010334

when lambda = 12.589254117941753 cv error rate = 0.1854373886822135

when lambda = 15.84893192461124 cv error rate = 0.18528678644375274

when lambda = 19.952623149688932 cv error rate = 0.18513607054322512

when lambda = 25.118864315095976 cv error rate = 0.18498524098063096

when lambda = 31.62277660168402 cv error rate = 0.18513618420529188

when lambda = 39.81071705535002 cv error rate = 0.18528678644375274

when lambda = 50.11872336272761 cv error rate = 0.18528644545755246

when lambda = 63.09573444801982 cv error rate = 0.18588942272172915

when lambda = 79.43282347242878 cv error rate = 0.1858893090596624

In the end, instead of using the features Amount of bill statement and Amount of previous payment, I calculate the mean and std of Amount of bill statement and Amount of previous payment as 4 new features, and the result:

the best model is when lambda = 0.01995262314968881

cv error rate = 0.1845328659549148

test error = 0.18020202020202015

train error = 0.19016856018415385

For each model:

when lambda = 0.01 cv error rate = 0.18483429775596993

when lambda = 0.012589254117941675 cv error rate = 0.18483429775596993

when lambda = 0.015848931924611134 cv error rate = 0.18468358185544242

when lambda = 0.01995262314968881 cv error rate = 0.1845328659549148

when lambda = 0.025118864315095822 cv error rate = 0.1845328659549148

when lambda = 0.031622776601683826 cv error rate = 0.18483429775596993

when lambda = 0.039810717055349776 cv error rate = 0.18483429775596993

when lambda = 0.050118723362727303 cv error rate = 0.1849851273185642

```
when lambda = 0.06309573444801943 cv error rate = 0.1849851273185642

when lambda = 0.07943282347242829 cv error rate = 0.1851358432190917

when lambda = 0.1000000000000002 cv error rate = 0.1849851273185642

when lambda = 0.125892541179417 cv error rate = 0.18528678644375274

when lambda = 0.15848931924611173 cv error rate = 0.18528667278168598

when lambda = 0.1995262314968885 cv error rate = 0.18498524098063085

when lambda = 0.25118864315095873 cv error rate = 0.18513618420529188

when lambda = 0.3162277660168389 cv error rate = 0.18543772966841376

when lambda = 0.39810717055349853 cv error rate = 0.18558844556894127

when lambda = 0.501187233627274 cv error rate = 0.18543772966841376

when lambda = 0.6309573444801956 cv error rate = 0.1858898773699963

when lambda = 0.7943282347242846 cv error rate = 0.18558844556894127

when lambda = 1.000000000000004 cv error rate = 0.18573916146946878

when lambda = 1.2589254117941726 cv error rate = 0.18573927513153554

when lambda = 1.5848931924611207 cv error rate = 0.18543772966841365

when lambda = 1.995262314968889 cv error rate = 0.18558855923100792

when lambda = 2.5118864315095926 cv error rate = 0.18573916146946878

when lambda = 3.1622776601683955 cv error rate = 0.18543750234428025

when lambda = 3.9810717055349936 cv error rate = 0.18588965004586266

when lambda = 5.01187233627275 cv error rate = 0.18604036594639017

when lambda = 6.309573444801969 cv error rate = 0.18573893414533527

when lambda = 7.943282347242862 cv error rate = 0.18558821824480776

when lambda = 10.000000000000062 cv error rate = 0.1858898773699963

when lambda = 12.589254117941753 cv error rate = 0.1854376160063469

when lambda = 15.84893192461124 cv error rate = 0.1858898773699963

when lambda = 19.952623149688932 cv error rate = 0.18483429775596982

when lambda = 25.118864315095976 cv error rate = 0.1854373886822135

when lambda = 31.62277660168402 cv error rate = 0.185588104582741

when lambda = 39.81071705535002 cv error rate = 0.18528667278168598

when lambda = 50.11872336272761 cv error rate = 0.18573893414533527

when lambda = 63.09573444801982 cv error rate = 0.1867940591110946
```

when lambda = 79.43282347242878 cv error rate = 0.18694488867368886

4. Random Forest

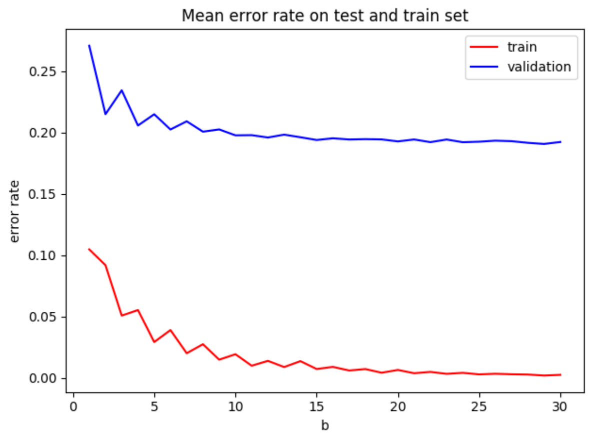
Firstly, I didn't preprocess the dataset, I directly use the raw data for training, the result:

the best model is when b = 29

cv error rate = 0.19077475474567543

train error = 0.0019157941635107977

test error = 0.1890909090909091



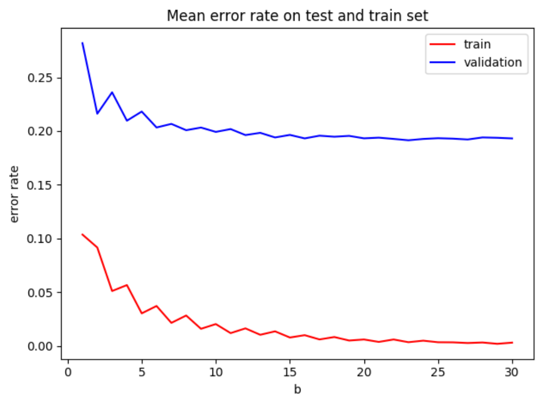
Then I directly standardized the Amount of bill statement and Amount of previous payment, the result is:

the best model is when b = 23

cv error rate = 0.19140782972513107

train error = 0.003341501447983952

test error = 0.18939393939393945



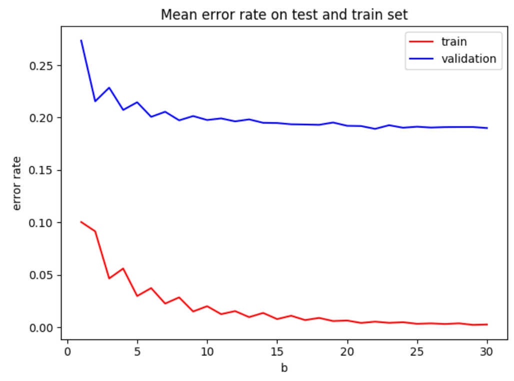
In the end, instead of using the features Amount of bill statement and Amount of previous payment, I calculate the mean and std of Amount of bill statement and Amount of previous payment as 4 new features, and the result:

the best model is when b = 22

cv error rate = 0.1892057863084948

train error = 0.005212742258854974

test error = 0.1794949494949495



4. Final Results and Interpretation

When I use KNN:

1. No preprocessing:

   the best model is when k = 91

   cv error rate = 0.2202623093176752

   test error = 0.21777777777777774

   train error = 0.22232122967253287

2. Standardized:

   the best model is when k = 21

   cv error rate = 0.18468289988304176

   test error = 0.19141414141414137

   train error = 0.17739659909408179

3. Features extraction:

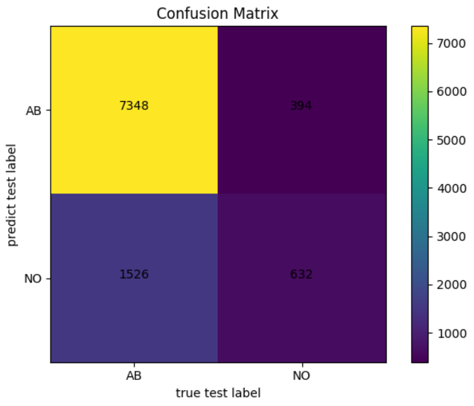
   the best model is when k = 21

   cv error rate = 0.18709560457421615

test error = 0.18393939393939397

train error = 0.18229746788445833

So, the best model is when I use features extraction, and k=21.



When I use logistic regression – l1:

1. No preprocessing

   the best model is when lambda = 0.01584893192461114

   cv error rate = 0.18374914327217196

   test error = 0.18939393939393945

   train error = 0.19053983812281872

2. Standardized

   the best model is when lambda = 0.01

   cv error rate = 0.18377928645227737

   test error = 0.1894949494949495

   train error = 0.1906140937105517

3. Features extraction:

   the best model is when lambda = 0.012589254117941675

   cv error rate = 0.1846534386753369

   test error = 0.18020202020202015

   train error = 0.18994579342095497

So, the best model is when I use features extraction and lambda = 0.012589254117941675

When I use logistic regression – l2:

1. No preprocessing

    the best model is when lambda = 0.125892541179417

    cv error rate = 0.21905658211345524

    test error = 0.21797979797979794

    train error = 0.22395485260265835

2. Standardized

    the best model is when lambda = 0.01

    cv error rate = 0.18377928645227737

    test error = 0.18939393939393945

    train error = 0.1906140937105517

3. Features extraction:

    the best model is when lambda = 0.01995262314968881

    cv error rate = 0.1845328659549148

    test error = 0.18020202020202015

    train error = 0.19016856018415385

So, the best model is when I use features extraction, and lambda = 0.01995262314968881.

When I use random forest:

1. No preprocessing

    the best model is when b = 29

    cv error rate = 0.19077475474567543

    train error = 0.0019157941635107977

    test error = 0.1890909090909091

2. Standardized

    the best model is when b = 23

    cv error rate = 0.19140782972513107

    train error = 0.003341501447983952

    test error = 0.18939393939393945

3. Features extraction:

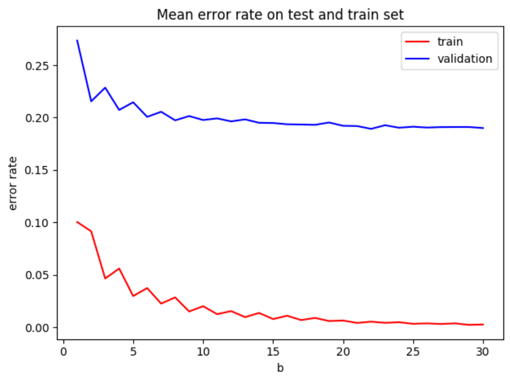
the best model is when b = 22

cv error rate = 0.1892057863084948

train error = 0.005212742258854974

test error = 0.17949494949494949495

So, the best model is when I use features extraction, and b = 22.



All in all, the best model is when I use features extraction, and random forest, and b = 22. In this case:

train error = 0.005212742258854974

test error = 0.17949494949494949495

5. Contributions of each team member

I did all the job.

6. Summary and conclusions

	KNN	l1-penality	l2-penality	random forest
NA	0.217778	0.189394	0.217980	0.189091
Standardization	0.191414	0.189495	0.189394	0.189394
extract new feature	0.183939	0.180202	0.180202	0.179495

The test error is shown above, as we can see, the minimum test error is 0.179 when I used random forest. However, this error rate is still very high, in order to improve my accuracy and minimized the error rate, I will use the neural network to solve this problem.

7. References

[1] https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

8. Appendix

Appendix I.

- You have to create a “main” file that outputs test result if we run the “main” file.
- “main” file should be named “main.py” or “main.m” depending on the choice of language.
- “main” file should be located in the base directory of your extracted zip folder.
- If dataset is too big to be uploaded (size > 50 MB), you can provide URL link with “wget” linux command. (Please see the example below)
- The “main” file should read the data files in “data” folder and output the results of your final system in your report.
- You should not let “main” file to start training. “main” file should load trained parameters and reproduce the test results.

Example of extracted zip folder:

```
├── main.py (or main.m, main.sh – Mandatory file)
├── modules.py (or modules.m, modules.cpp)
├── utils.py (or utils.m, utils.cpp)
├── data
│    ├── data1_training_data.csv
│    ├── data1_validation_data.csv
│    ├── data1_test_data.csv
│    ├── data2_training_data.csv
│    ├── data2_validation_data.csv
│    └── data2_test_data.csv
```

if dataset is too big to be downloaded:

```
├── main.py (or main.m, main.sh – Mandatory file)
├── modules.py (or modules.m, modules.cpp)
├── utils.py (or utils.m, utils.cpp)
├── data
│    └── download.sh
```

in “download.sh” you specify the download link as in the following example:

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.data
wget https://archive.ics.uci.edu/ml/machine-learning-
databases/haberman/haberman.names
```