SEIS 736 Machine Learning

Project Report

December 9, 2015

Give Me Some Credit

Team Members: Princewill Eneh, Suraj Jois, Xiaozhao Song, Yalei Yan, Ruogu Wang

Table of Content

Introduction & Objective

Banks play a crucial role in market economies. They decide who can get finance and on what terms and can make or break investment decisions. For markets and society to function, individuals and companies need access to credit.

Credit scoring algorithms, which make a guess at the probability of default, are the method banks use to determine whether or not a loan should be granted. This project requires us to improve on the state of the art in credit scoring, by predicting the probability that somebody will experience financial distress in the next two years.

The goal of this project is to build a model that performs better than 77% accuracy which is the specified threshold to beat.

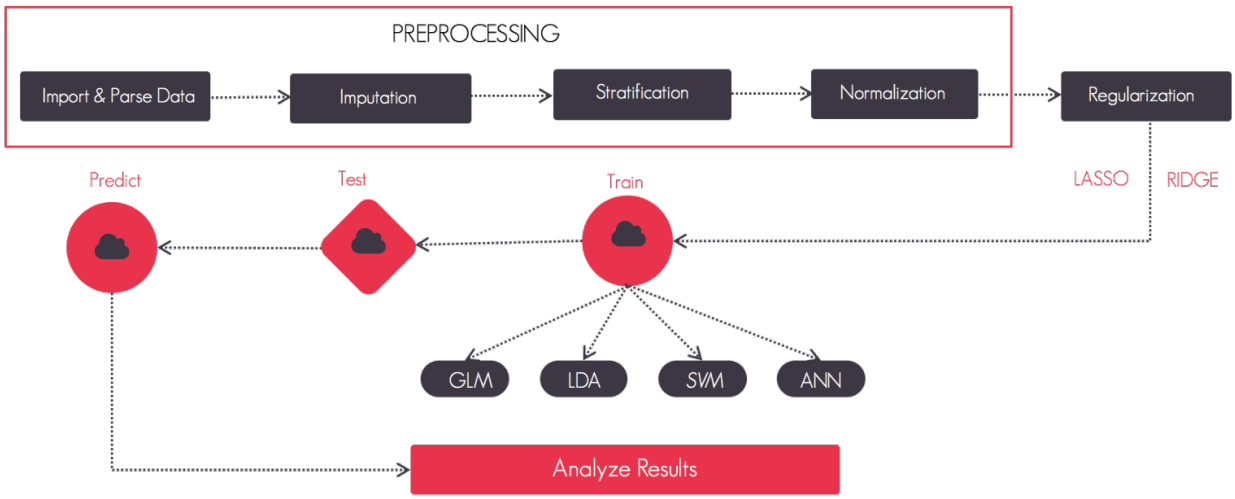
Datasets

- Our dataset consists of two csv files. One for training and the other for testing. The csv files contain about 150,000 records of customers for training and 100,000 for testing.
- Below is the data dictionary of the dataset.

Variable Name	Description	Type
SeriousDlqin2yrs	Person experienced 90 days past due delinquency or worse	Y/N
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage
age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income	percentage
MonthlyIncome	Monthly income	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer

Methodology

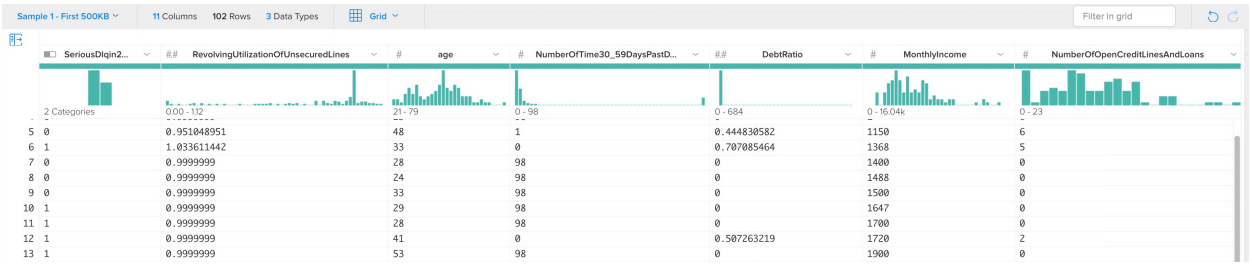
We plan to use several Machine learning techniques and methodologies. Below is the plan of action for achieving the projects set goal.



Tools

The primary tool used for this project was **R**. We chose R because it is a free flexible statistical analysis toolkit. All of the standard data analysis tools are built right into the R language: from accessing data in various formats, to data manipulation (transforms, merges, aggregations, etc.), to traditional and modern statistical models (regression, ANOVA, GLM, tree models, etc.). All are included in an object-oriented framework that makes it easy to programmatically extract out and combine just the information you need from the results, rather than having to cut-and-paste from a static report.

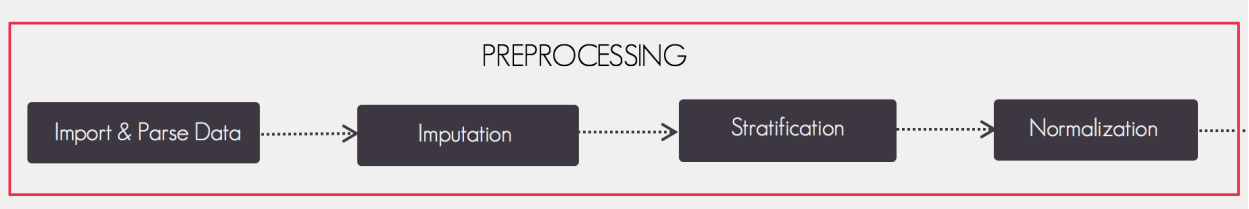
We also used a data wrangling tool called **Trifecta Wrangler (https://www.trifacta.com)**. The tool was helpful in us cleaning up the data as Data wrangling is the critical first step of the analysis process – and one that has traditionally consumed a significant amount of time and effort. Successful analysis relies upon accurate, well-structured data that has been formatted for the specific needs of the task at hand. In fact, wrangling is as much a part of the data analysis process as the final results. So we employed this tool to visualize and understand our data before any data analysis is performed.



Finally, we wanted to see how some of the Algorithms performed when used on an in memory database like **SAP HANA**. We ran our dataset on AWS against Decision Tree and BPNN algorithms.

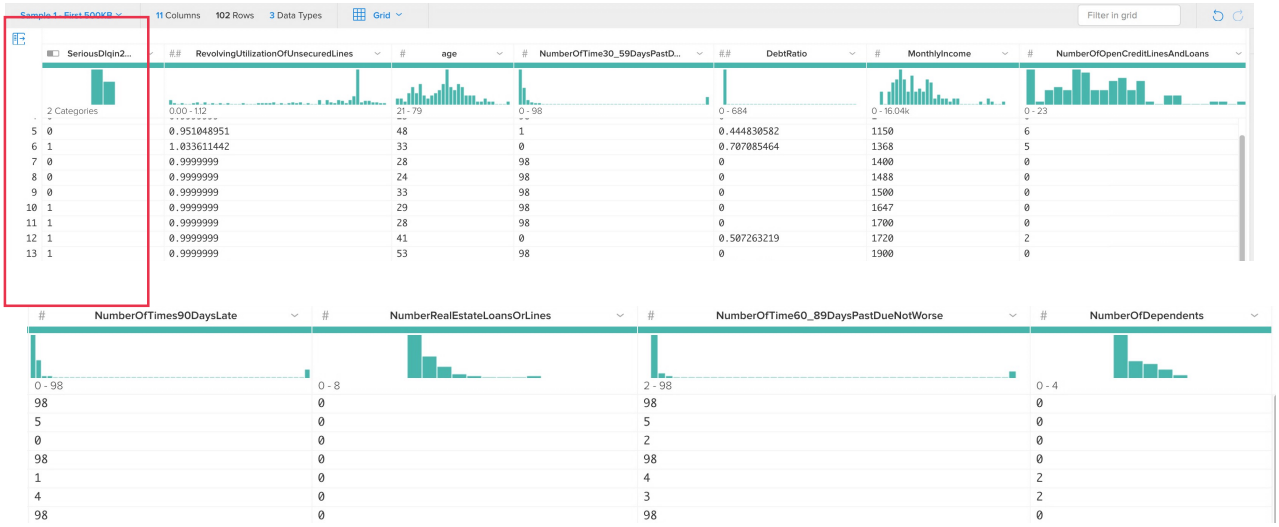
Preprocessing

The preprocessing phase of this project was critical as we wanted to have the best data for the predictive analysis we wished to perform. Some of the techniques employed include Imputation, Stratification and Normalization.

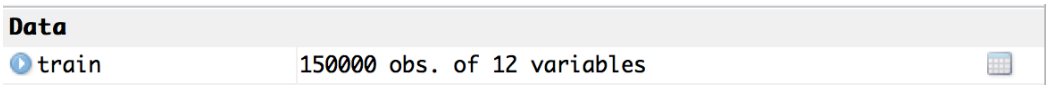


Import Data

We plan to understand our dataset as much as possible therefore before working on it, we need to know the distribution of each attribute. The first task is to import the raw data into Trifecta Wrangler. Once the data is imported we can see the distribution of all attribute. Below is the preview of the distribution.



We then import the data into R and get a summary of the data. The summary indicates we have 150000 observations and 12 variables.

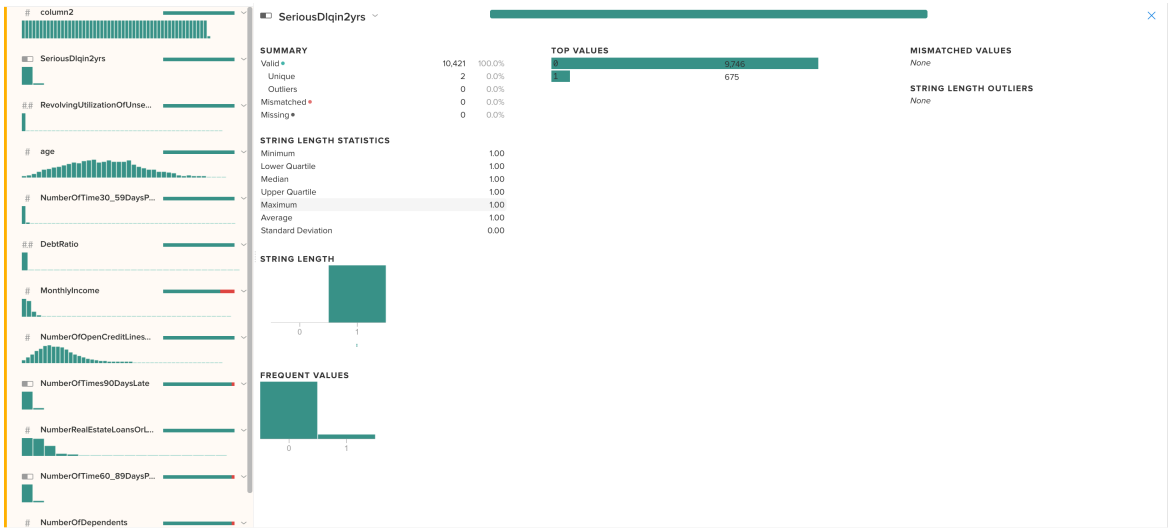


Data Exploration

Next we checked for colinearity, skewness and missing data since we are know that the presence of those will result in problematic model .

Stratified Sampling

Looking at our dataset we noticed that the training data had about 93% of 0s and 7% of the ones. Using this dataset, the way it is to train our data resulted in a good accuracy but precision was poor and the model could not generalize to our test data.



To combat this issue, we decided to stratify our training data and make the 1's and 0's the same size that is, 50% for 1's and 50% for 0's. This will ensure that our model is robust and can generalize to any test data.

Data Cleansing Process

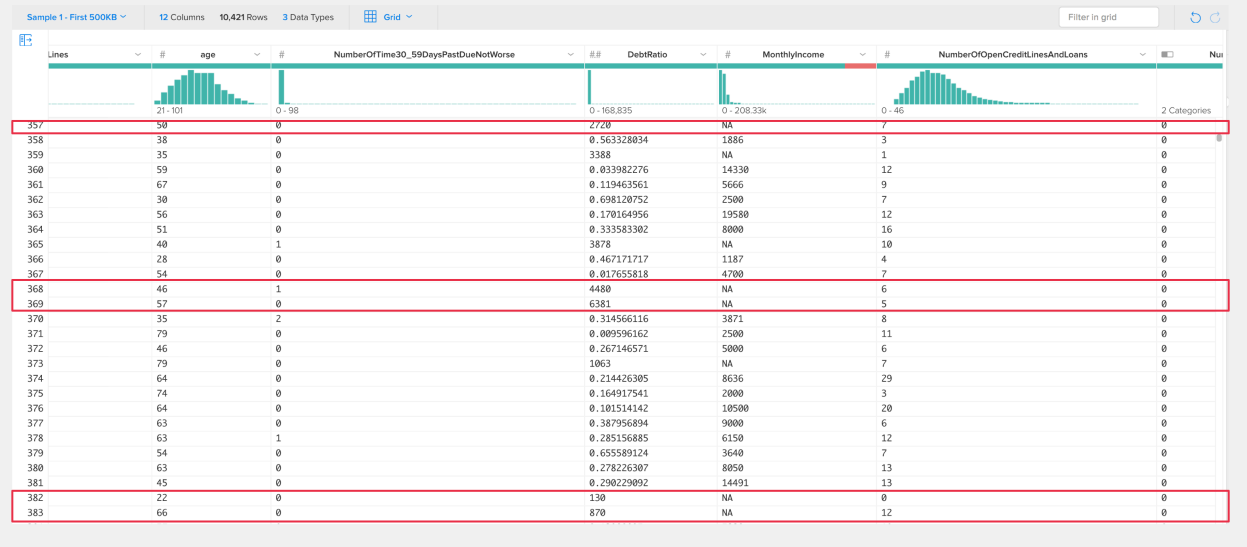
As is always the case with most datasets, the problem of missing values is one that always needs to be resolved for us to generate models from GLM, to SVM etc.

Imputation

Algorithms like logistic regression, Neural network, Support vector machine etc. would not work if there are missing values in the table. So to solve this, we had two choices;

1. Impute the missing values
2. Remove rows with missing values in them

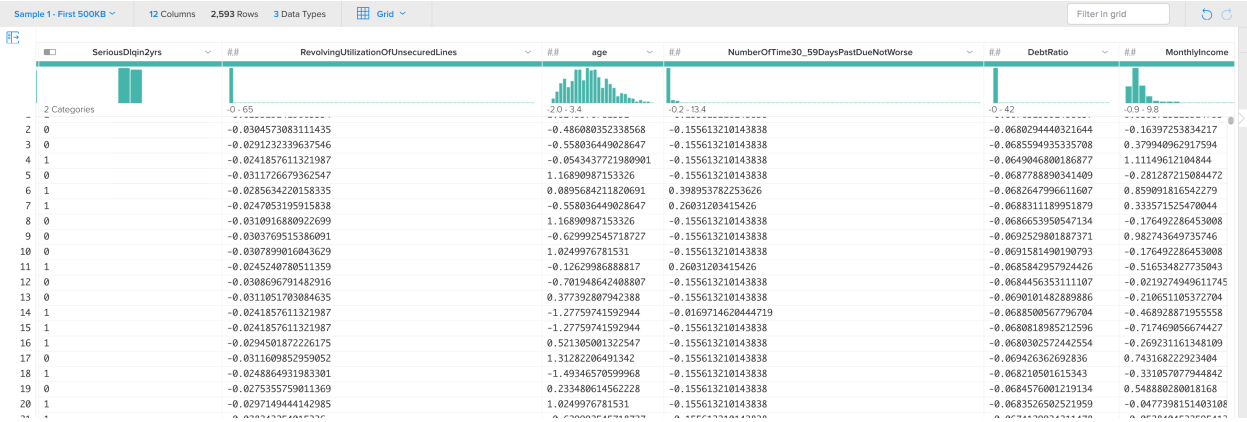
Each choice had its pros and cons. For choice one we will be retaining all the rows in our dataset which gives us more rows to train with. But doing this takes a lot of time and processing power. Choice two was the quickest but meant we will be removing about 30,000 records from our 150,000 record dataset. We decided to go with this choice as we concluded that 120,000 records for training is still a decent amount of records.



	Lines	age	NumberOfTime30_59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	Nu
		21 - 101	0 - 98	0 - 168,835	0 - 208.33k	0 - 46	2 Categories
357		50	0	2720	NA	7	0
358		38	0	0.563328034	1886	3	0
359		35	0	3388	NA	1	0
360		59	0	0.033982276	14330	12	0
361		67	0	0.119463561	5666	9	0
362		30	0	0.698120752	2500	7	0
363		56	0	0.170164956	19580	12	0
364		51	0	0.333583302	8000	16	0
365		40	1	3878	NA	10	0
366		28	0	0.467171717	1187	4	0
367		54	0	0.017655818	4700	7	0
368		46	1	4480	NA	6	0
369		57	0	6381	NA	5	0
370		35	2	0.314566116	3871	8	0
371		79	0	0.009596162	2500	11	0
372		46	0	0.267146571	5000	6	0
373		79	0	1063	NA	7	0
374		64	0	0.214426305	8636	29	0
375		74	0	0.164917541	2000	3	0
376		64	0	0.101514142	10500	20	0
377		63	0	0.387956894	9000	6	0
378		63	1	0.285156885	6150	12	0
379		54	0	0.655589124	3640	7	0
380		63	0	0.278226307	8050	13	0
381		45	0	0.290229092	14491	13	0
382		22	0	130	NA	0	0
383		66	0	870	NA	12	0

Normalization

Looking at the imported data, we notice a vast range of scale and unit differences in the attributes. Since algorithms like SVM require normalized data, we normalized the training data. Below is a preview of the result.



	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30_59DaysPastDueNotWorse	DebtRatio	MonthlyIncome
	2 Categories	0 - 65	-2.0 - 3.4	-0.2 - 13.4	0 - 42	-0.9 - 9.8
2	0	-0.0304573083111435	-0.486080352338568	-0.155613210143838	-0.0680294440321644	-0.16397253834217
3	0	-0.0291232339637546	-0.558036449028647	-0.155613210143838	-0.0685594935335708	0.379940962917594
4	1	-0.0241857611321987	-0.0543437721980901	-0.155613210143838	-0.0649046800186877	1.11149612104844
5	0	-0.0311726679362547	1.16890987153326	-0.155613210143838	-0.0687788890341409	-0.281287215084472
6	1	-0.0285634220158335	0.0895684211820691	0.398953782253626	-0.0682647996611607	0.859091816542279
7	1	-0.0247053195915838	-0.558036449028647	0.26031203415426	-0.0688311189951879	0.333571525470044
8	0	-0.0310916880922699	1.16890987153326	-0.155613210143838	-0.0686653950547134	-0.176492286453008
9	0	-0.0303769515386091	-0.629992545718727	-0.155613210143838	-0.0692529801887371	0.982743649735746
10	0	-0.0307899016043629	1.0249976781531	-0.155613210143838	-0.0691581490190793	-0.176492286453008
11	1	-0.0245240780511359	-0.12629986888817	0.26031203415426	-0.0685842957924426	-0.516534827735043
12	0	-0.0308696791482916	-0.701948642408807	-0.155613210143838	-0.0684456353111107	-0.021927494961174
13	0	-0.0311051703084635	0.377392807942388	-0.155613210143838	-0.0690101482889886	-0.210651105372704
14	1	-0.0241857611321987	-1.27759741592944	-0.0169714620444719	-0.0688500567796704	-0.468928871955558
15	1	-0.0241857611321987	-1.27759741592944	-0.155613210143838	-0.0680818985212596	-0.717469056674427
16	1	-0.0294501872226175	0.521305001322547	-0.155613210143838	-0.0680302572442554	-0.269231161348109
17	0	-0.0311609852959052	1.31282206491342	-0.155613210143838	-0.069426362692836	0.743168222923404
18	1	-0.0248864931983301	-1.49346570599968	-0.155613210143838	-0.068210501615343	-0.331057077944842
19	0	-0.0275355759011369	0.233480614562228	-0.155613210143838	-0.0684576001219134	0.548880280018168
20	1	-0.0297149444142985	1.0249976781531	-0.155613210143838	-0.0683526502521959	-0.047739815140310

Regularization

Regularization helps us find the best predictors and prevent overfitting. We wanted to make sure that our model does not overfit. We perform regularization using lasso and ridge. First we start with Ridge.

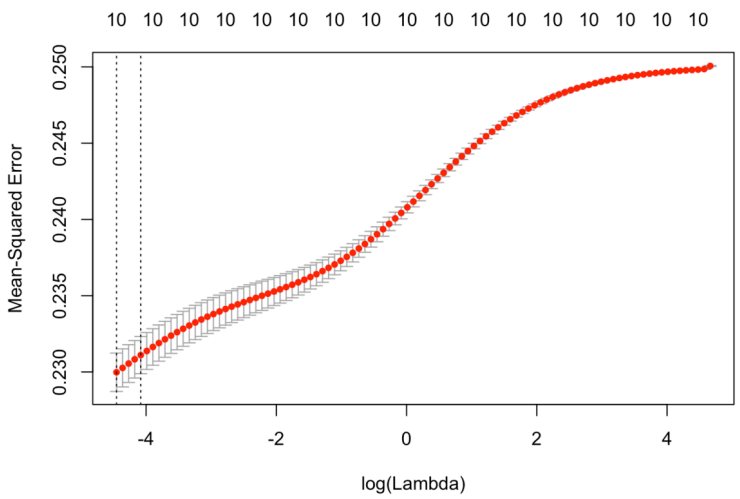
RIDGE

```
x.tr <- model.matrix(SeriousDlqin2yrs ~ RevolvingUtilizationOfUnsecuredLines  + age + NumberOfTime30.59DaysPastDu
eNotWorse
                     + DebtRatio  + MonthlyIncome +NumberOfOpenCreditLinesAndLoans + NumberOfTimes90DaysLate +
                       NumberRealEstateLoansOrLines +  NumberOfTime60.89DaysPastDueNotWorse
                     + NumberOfDependents, data = training_data)[, -1]
y.tr <- training_data$SeriousDlqin2yrs


x.val <- model.matrix(SeriousDlqin2yrs ~ RevolvingUtilizationOfUnsecuredLines  + age + NumberOfTime30.59DaysPastD
ueNotWorse
                      + DebtRatio  + MonthlyIncome +NumberOfOpenCreditLinesAndLoans + NumberOfTimes90DaysLate +
                        NumberRealEstateLoansOrLines +  NumberOfTime60.89DaysPastDueNotWorse
                      + NumberOfDependents, data = testing_data)[, -1]
y.val <- testing_data$SeriousDlqin2yrs
```

Cross Validation to obtain best lambda

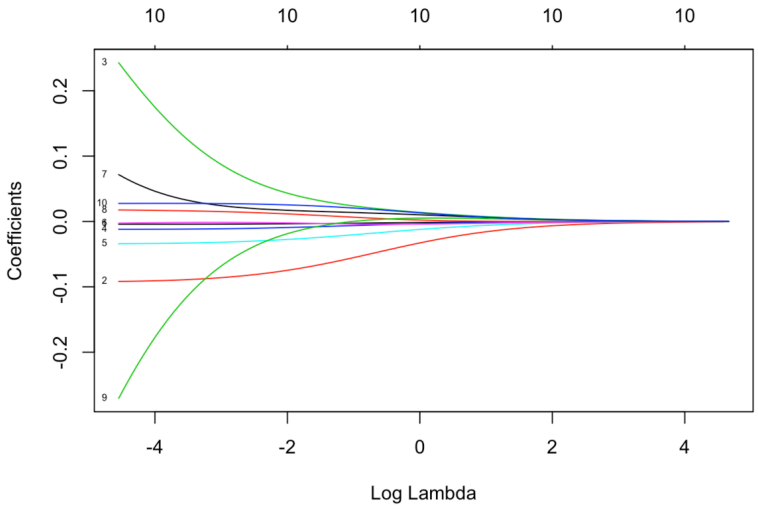
```
set.seed(10)
rr.cv <- cv.glmnet(x.tr, y.tr, alpha = 0)
```



First we turn our training data table into a Matrix *(line 1 above)* so we can perform regularization. Next we set seed *(line 19 above)* so we can repeat the process and get the same result. On line 20 we perform a cross validation to get the best Lambda for our ridge regularization. The Graph above denotes that our best lambda is between **-4.5** and **-4**.

Next we try to find the best predictors. Line 1 on the next page shows how this is done in R. The graph on the next page shows that almost all the predictors converged at **0** at the same time. We expected this might happen as an initial logistic regression model we generated should very significant p-value for all the **10** independent variables.

```
1  rr.fit <- glmnet(x.tr, y.tr, alpha = 0)
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26  rr.pred <- predict(rr.fit, s = rr.bestlam, newx = x.val)
27  rr.pred  <- ifelse(rr.pred  > 0.5,1,0)
28
29  table(rr.pred ,testing_data[,1 ])
30
31
32  ##
33  ## rr.pred    0    1
34  ##        0 2427 1464
35  ##        1 1748 2718
36
37  MSE <-mean(rr.pred != testing_SeriousDlqin2yrs)
38  MSE
39
40  ## [1] 0.3843485
41
42  print(paste('Accuracy',(1-MSE)*100,"%"))
43
44  ## [1] "Accuracy 61.5651549599138 %"
45
46
```



We now try to predict our dependent variable using the model we generated after applying the best lambda and eliminating not so useful predicators according to ridge regularization (line 26-28).

Now, it's time to see how ridge regularization performed. We compare the predicted output to the actual output from our testing data. The confusion matrix on line 33 shows a high **FP** and **FN** values. The accuracy is **61%.**

We anticipated the above result as we mentioned earlier. Almost all the independent variables are useful in predicting our independent variable. Therefore, the use of ridge regularization resulted in an underfitting model.

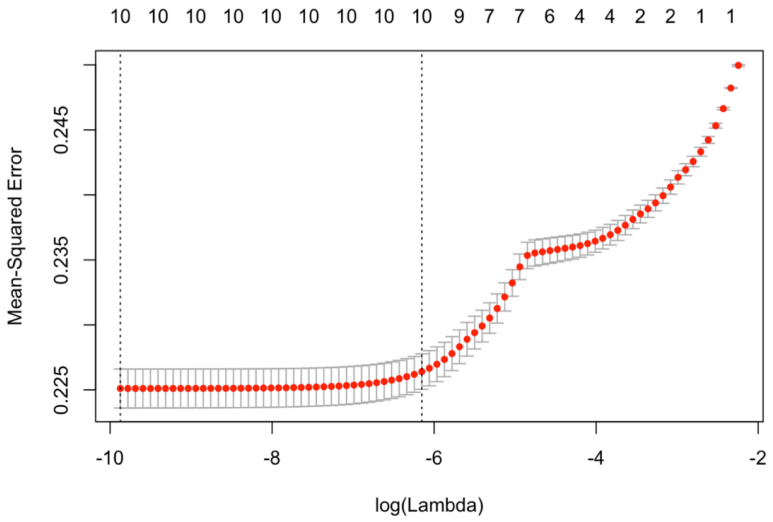
LASSO

```
x.tr <- model.matrix(SeriousDlqin2yrs ~ RevolvingUtilizationOfUnsecuredLines  + age + NumberOfTime30.59DaysPastDu
eNotWorse
                     + DebtRatio  + MonthlyIncome +NumberOfOpenCreditLinesAndLoans + NumberOfTimes90DaysLate +
                       NumberRealEstateLoansOrLines +  NumberOfTime60.89DaysPastDueNotWorse
                     + NumberOfDependents, data = training_data)[, -1]
y.tr <- training_data$SeriousDlqin2yrs


x.val <- model.matrix(SeriousDlqin2yrs ~ RevolvingUtilizationOfUnsecuredLines  + age + NumberOfTime30.59DaysPastD
ueNotWorse
                      + DebtRatio  + MonthlyIncome +NumberOfOpenCreditLinesAndLoans + NumberOfTimes90DaysLate +
                        NumberRealEstateLoansOrLines +  NumberOfTime60.89DaysPastDueNotWorse
                      + NumberOfDependents, data = testing_data)[, -1]
y.val <- testing_data$SeriousDlqin2yrs
```

CV to obtain best lambda

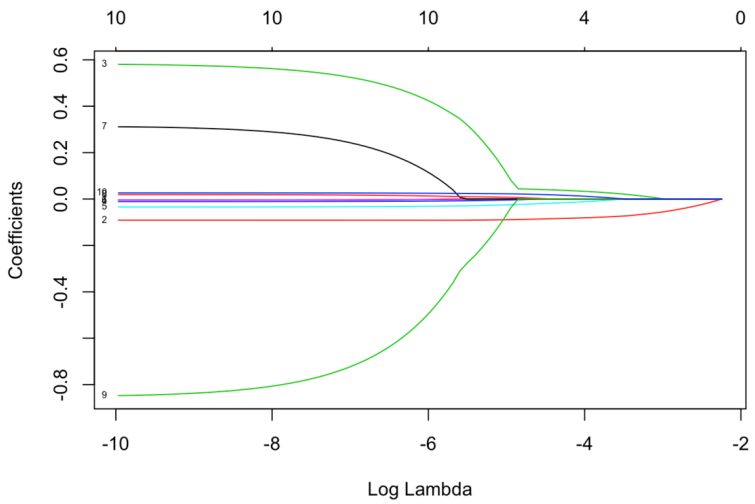
```
set.seed(10)
las.cv <- cv.glmnet(x.tr, y.tr, alpha = 1)
```



Like ridge we also have to turn our training data table into a Matrix *(line 1 above)* so we can perform lasso regularization. Next we set seed *(line 19 above)* so we can repeat the process and get the same result. On line 20 we perform a cross validation to get the best Lambda for our ridge regularization. The Graph above denotes that our best lambda is between **-9.8** and **-6.2**.

Next we try to find the best predictors. Line 1 on the next page shows how this is done in R. The graph on the next page shows that almost all the predictors converged at **0** at the same time except for predictor **2** and **3**. The coefficients converge to **0** at log Lambda of **-6** which is earlier that ridge which was around **2**. We expected this might happen as lasso converges coefficients to zero faster that ridge.

```
las.fit <- glmnet(x.tr, y.tr, alpha = 1)
```




```
las.pred <- predict(las.fit, s = las.bestlam, newx = x.val)

las.pred <- ifelse(las.pred > 0.5,1,0)
table(las.pred,testing_data[,1 ])
```

```
##
## las.pred    0    1
##        0 2636 1575
##        1 1539 2607
```

```
MSE <-mean(las.pred != testing_SeriousDlqin2yrs)
MSE
```

```
## [1] 0.3726218
```

```
print(paste('Accuracy',(1-MSE)*100,"%"))
```

```
## [1] "Accuracy 62.7378245781979 %"
```

We now try to predict our dependent variable using the model we generated after applying the best lambda and eliminating the not so useful predicators according to lasso regularization (line 27).

Now, it's time to see how lasso regularization performed. We compare the predicted output to the actual output from our testing data. The confusion matrix on line 33 shows a high FP and FN rates. The accuracy is 62%.

We anticipated the above result as we mentioned earlier. Almost all the independent variables are useful in predicting our independent variable. Therefore, the use of lasso regularization resulted in an underfitting model.

Although lasso regularization was fractionally better that ridge in this case, it's still not require for our particular problem. As we will demonstrate later, the result we got using both ridge and lasso regularization were not significantly better than when we did not use them.

Algorithms

Logistic Regression

We choose Logistic Regression as one of the classification algorithm. As mentioned in the tools section, we use R as our tool to get the logistic regression result. The commands don't have any big difference from the MATLAB commands.

The below graphs are for model results.

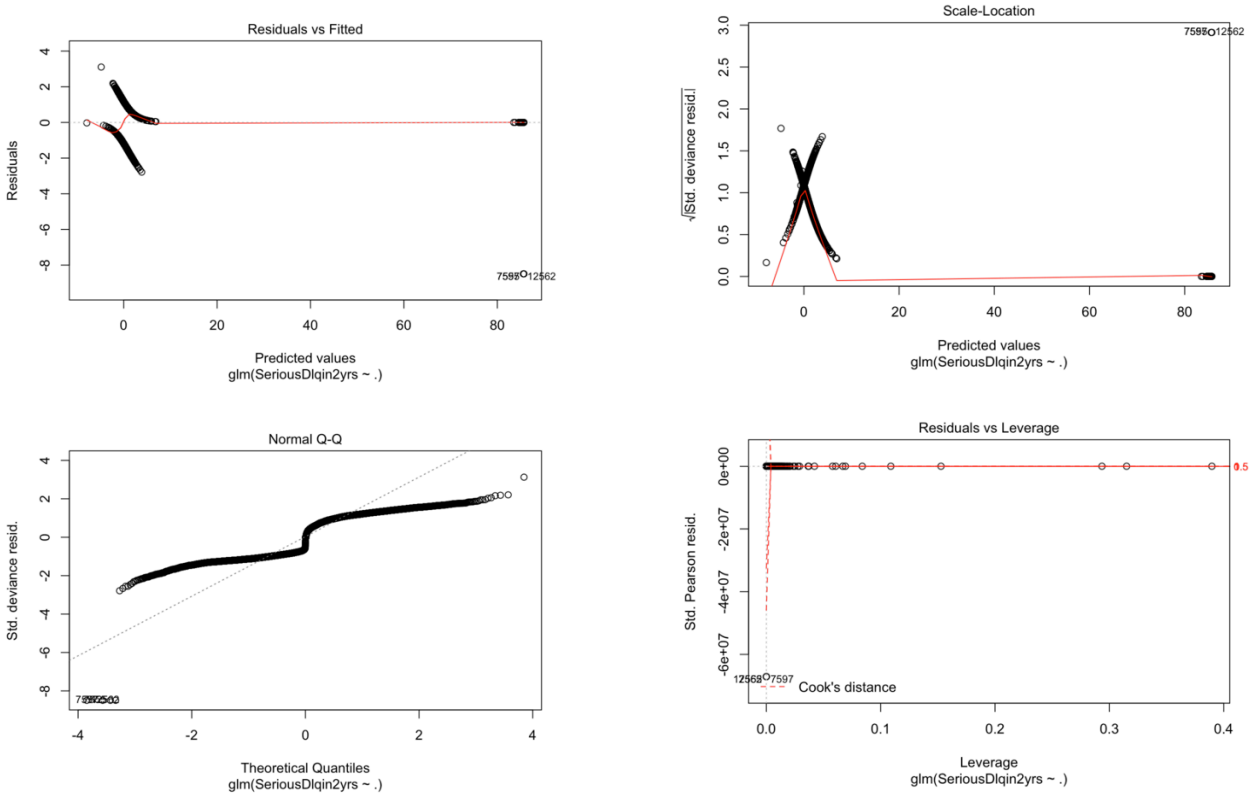
```
1
2  glmdl <- glm(SeriousDlqin2yrs ~.,family=binomial(link='logit'),data=training_data)
3
4  ## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
5
6  summary(glmdl)
7
8
9  ##
10 ## Call:
11 ## glm(formula = SeriousDlqin2yrs ~ ., family = binomial(link = "logit"),
12 ##     data = training_data)
13 ##
14 ## Deviance Residuals:
15 ##     Min       1Q   Median       3Q      Max
16 ## -8.4904  -1.0161  -0.2558   1.0741   3.1066
17 ##
18 ## Coefficients:
19 ##                                       Estimate Std. Error z value Pr(>|z|)
20 ## (Intercept)                            0.49780    0.03893  12.789  < 2e-16
21 ## RevolvingUtilizationOfUnsecuredLines -0.01224    0.02210  -0.554 0.579748
22 ## age                                   -0.37683    0.02539 -14.841  < 2e-16
23 ## NumberOfTime30.59DaysPastDueNotWorse  3.62786    0.20618  17.596  < 2e-16
24 ## DebtRatio                             -0.06161    0.02726  -2.260 0.023799
25 ## MonthlyIncome                         -0.20013    0.03543  -5.648 1.62e-08
26 ## NumberOfOpenCreditLinesAndLoans        0.01395    0.02793   0.500 0.617396
27 ## NumberOfTimes90DaysLate                3.18999    0.26769  11.917  < 2e-16
28 ## NumberRealEstateLoansOrLines           0.11153    0.02835   3.934 8.35e-05
29 ## NumberOfTime60.89DaysPastDueNotWorse -0.53621    0.32775  -1.636 0.101827
30 ## NumberOfDependents                     0.09467    0.02452   3.861 0.000113
31 ##
32 ## (Intercept)                           ***
33 ## RevolvingUtilizationOfUnsecuredLines
34 ## age                                   ***
35 ## NumberOfTime30.59DaysPastDueNotWorse ***
36 ## DebtRatio                             *
37 ## MonthlyIncome                         ***
38 ## NumberOfOpenCreditLinesAndLoans
39 ## NumberOfTimes90DaysLate                ***
40 ## NumberRealEstateLoansOrLines           ***
41 ## NumberOfTime60.89DaysPastDueNotWorse
42 ## NumberOfDependents                     ***
43 ## ---
44 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
45 ##
46 ## (Dispersion parameter for binomial family taken to be 1)
   ##
   ##     Null deviance: 11585.3  on 8356  degrees of freedom
   ## Residual deviance:  9917.1  on 8346  degrees of freedom
   ## AIC: 9939.1
   ##
   ## Number of Fisher Scoring iterations: 8
```

In line 2 we use the **logit** family of binomial classification and specify our training data.We also indicate that our dependent variable is **SerioulyDlqin2yrs** and the rest are independent variable

We run this line to generate the model and the resulting model is summarized.

Line 17 to 21 shows us the coefficients of the independent variable as well as their respective p-values. R has a nice way of showing which p-values are significant. (***) means a very significant p-value or a p-value less than 0.001. As you can tell the All but two of the predictors have a significant p-value.

A plot of the the residual vs fitted value, Scale-location, NormalQ_Q and Residual vs leverage of the resulting model can be seen below.



Next we try to predict using the model that we generated in the previous page. Line 1 show we are using the response type of Response to generate the probabilities of each instance belonging to the 1 or 0. The variable **Glmdlpredit** now has a vector of all the predicted probabilities.

```
1  glmdlpredict <- predict(glmdl,testing_data,type='response')
2  glmdlpredict <- ifelse(glmdlpredict > 0.5,1,0)
3
4  table(glmdlpredict,testing_data[,1 ])
5
6  ##
7  ## glmdlpredict    0    1
8  ##            0 3182 1525
9  ##            1  993 2657
10
11 MSE <-mean(glmdlpredict != testing_SeriousDlqin2yrs)
12 MSE
13
14 ## [1] 0.3013043
15
16 print(paste('Accuracy',(1-MSE)*100,"%"))
17
18 ## [1] "Accuracy 69.8695704200072 %"
19
20 sqrt(mean((glmdlpredict - testing_data$SeriousDlqin2yrs)^2))
21
22 ## [1] 0.5489119
23
24 confusionMatrix(glmdlpredict, testing_SeriousDlqin2yrs)
25
26 ## Confusion Matrix and Statistics
27 ##
28 ##           Reference
29 ## Prediction    0    1
30 ##          0 3182 1525
31 ##          1  993 2657
32 ##
33 ##                Accuracy : 0.6987
34 ##                  95% CI : (0.6887, 0.7085)
35 ##     No Information Rate : 0.5004
36 ##     P-Value [Acc > NIR] : < 2.2e-16
37 ##
38 ##                   Kappa : 0.3975
39 ##  Mcnemar's Test P-Value : < 2.2e-16
40 ##
41 ##             Sensitivity : 0.7622
42 ##             Specificity : 0.6353
43 ##          Pos Pred Value : 0.6760
44 ##          Neg Pred Value : 0.7279
45 ##              Prevalence : 0.4996
46 ##          Detection Rate : 0.3808
   ##    Detection Prevalence : 0.5632
   ##       Balanced Accuracy : 0.6987
   ##
   ##        'Positive' Class : 0
   ##
```

The next step on line two is to assign 1's and 0's to those probabilities with probabilities >0.5 assigned a 1 class and those less that 0.5 assigned as 0 class.

Once this is done we then compare this result to our testing data line 4 and get the MSE which is 0.301 in this case.

We then plot the confusion matrix (line 24) to see how good our model performed against our test data. We will provide and in-depth review of this result in the result and Analysis section of this report.

Linear Discriminant Analysis

The purpose of linear discriminant analysis (LDA) is to find the linear combinations of the original variables (10 in our dataset) that gives the best possible separation between the groups (default/ not default) in our data set.

If we want to separate the records by to give credit or not, the records come from 2 different variants, so the number of groups (G) is 2, and the number of variables is 10. The maximum number of useful discriminant functions that can separate the records is the minimum of G-1, and so in this case it is the minimum of 1 to 10.

```
ldamdl<-lda(SeriousDlqin2yrs ~., data = training_data)
ldamdl
```

```
## Call:
## lda(SeriousDlqin2yrs ~ ., data = training_data)
##
## Prior probabilities of groups:
##         0         1
## 0.5004188 0.4995812
##
## Group means:
##   RevolvingUtilizationOfUnsecuredLines        age
## 0                         0.014683021  0.2230021
## 1                        -0.009784796 -0.2055636
##   NumberOfTime30.59DaysPastDueNotWorse   DebtRatio MonthlyIncome
## 0                          -0.1142063  0.01536177    0.07281932
## 1                           0.0859057 -0.01843612   -0.08549153
##   NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
## 0                      0.04646429             -0.10069598
## 1                     -0.04778837              0.07623902
##   NumberRealEstateLoansOrLines NumberOfTime60.89DaysPastDueNotWorse
## 0                  0.016906134                          -0.08432083
## 1                  0.004875361                           0.05711844
##   NumberOfDependents
## 0        -0.08777666
## 1         0.08663385
##
## Coefficients of linear discriminants:
##                                         LD1
## RevolvingUtilizationOfUnsecuredLines -0.02546800
## age                                  -0.60302787
## NumberOfTime30.59DaysPastDueNotWorse  3.88982613
## DebtRatio                            -0.07301906
## MonthlyIncome                        -0.22669692
## NumberOfOpenCreditLinesAndLoans      -0.03388304
## NumberOfTimes90DaysLate               2.10898015
## NumberRealEstateLoansOrLines          0.12768022
## NumberOfTime60.89DaysPastDueNotWorse -5.70389196
## NumberOfDependents                    0.17499667
```

We can carry out a linear discriminant analysis using the "lda()" function from the R "MASS" package. To use this function, we first need to install the "MASS" R package.

To calculate the separation achieved by discriminant function, we first need to calculate the value of each discriminant function, by substituting the variables' values into the linear combination for the discriminant function (eg. -0.025*V2 - 0.603*V3 +3.889*V4 -0.073*V5 - 0.226*V6 -0.033*V7 - 1.661*V8 +2.108*V9 + 0.127*V10 -5.703*V11 +0.174*V12 for the first discriminant function).

```
ldamdlpredict<-predict(ldamdl, newdata = testing_data[,c(2,3,4,5,6,7,8,9,10,11)])$class
```

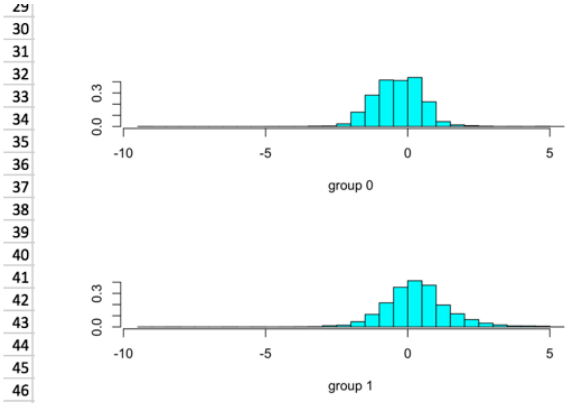
```
table(ldamdlpredict,testing_data[,1 ])
```

```
##
## ldamdlpredict    0    1
##             0 2640 1577
##             1 1535 2605
```

We use the predict function in R to do the prediction against test data. We then build the confusionMatrix between ldamdlpredict and Testing_SeriousDlq dataset. To get the above confusion matrix results.

There are 1535+1577=3122 samples that are misclassified, out of (1535+1577 +2640+2605=) 8537 samples, therefore the accuracy is 0.627. The misclassification rate is quite high, and therefore the accuracy of the allocation rule is low.

To get a more accurate idea of how well the discriminant functions separates the groups, we would need to see a stacked histogram of the values for the two using some unseen "test set", that is, using a set of data that was not used to calculate the linear discriminant function.



A nice way of displaying the results of a linear discriminant analysis (LDA) is to make a stacked histogram of the values of the discriminant function for the samples from different groups

We can do this using the "ldahist()" function in R. For example, to make a stacked histogram of the first discriminant function's values for samples of the 2 different groups, we type:

> ldahist(data = testing_data.lda.values$x[,2,3,4,5,6,7,8,9,10,11], g= $class)

We can see from the histogram that groups/classes 1 and 2 are not well separated by the discriminant functions, since the values for the zero class are between -4 and +3, while the values for class 1 are between -3 and 4, and so there is overlap in values. This clearly makes it challenging for the model to give good prediction accuracy.

Support Vector Machine

As a powerful tool of pattern recognition, support vector machine (SVM) was used to build a prediction model using the training data, and pass the test data to evaluate the quality of the prediction model with the ultimate goal of using the model to forecast the likelihood of experiencing serious financial distress for potential loaners.

```
svmmdl <- ksvm(SeriousDlqin2yrs ~ ., data=training_data, type = "C-svc", kernel = "rbfdot",kpar = list(sigma =
0.1), C = 10, prob.model = TRUE)
```

```
## line search fails -1.01618 -0.08577839 1.823325e-05 7.463505e-06 -1.64527e-08 -9.743073e-09 -3.727038e-13
```

```
svmmdl
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 10
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.1
##
## Number of Support Vectors : 5150
##
## Objective Function Value : -48507.17
## Training error : 0.249252
## Probability model included.
```

In this project, we adopted kernel function of SVM , which is available in the kernlab package in R, to avoid the curse of dimensionality. More specially, we used the **rbfdot** kernel function, which is also referred to as Radial Basis kernal “Gaussian”. **C-svc** was passed as a parameter to do C classification.

Other parameters used in model building include hyper-parameter **sigma=0.1** (inverse kernel width for rbfdot, **C=10** (cost of constraints violation), and **prob.model=TRUE**(for calculating class probabilities).

```
svmmdlpredict <- predict(svmmdl,testing_data[,c(2,3,4,5,6,7,8,9,10,11)])
table(svmmdlpredict,testing_data[,1 ])
```

```
##
## svmmdlpredict    0    1
##             0 3572 1570
##             1  603 2612
```

```
MSE <-mean(svmmdlpredict != testing_SeriousDlqin2yrs)
MSE
```

```
## [1] 0.2600215
```

```
print(paste('Accuracy',(1-MSE)*100,"%"))
```

```
## [1] "Accuracy 73.9978461170276 %"
```

```
confusionMatrix(svmmdlpredict, testing_SeriousDlqin2yrs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3572 1570
##          1  603 2612
##
##                Accuracy : 0.74
##                  95% CI : (0.7304, 0.7494)
##     No Information Rate : 0.5004
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4801
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8556
##             Specificity : 0.6246
##          Pos Pred Value : 0.6947
##          Neg Pred Value : 0.8124
##              Prevalence : 0.4996
##          Detection Rate : 0.4274
##    Detection Prevalence : 0.6153
##       Balanced Accuracy : 0.7401
##
##        'Positive' Class : 0
##
```

As a result, 5150 support vectors (out of 25071 training data points) were identified. Support vectors are the data points that lie closest to the decision surface. The range of the number of support vectors can be wide, depending on how much slack we allow and the complexity of our model. The percentage of support vectors is 20.5% for our training data, suggesting that the possibility for our model to overfit the train data is low.

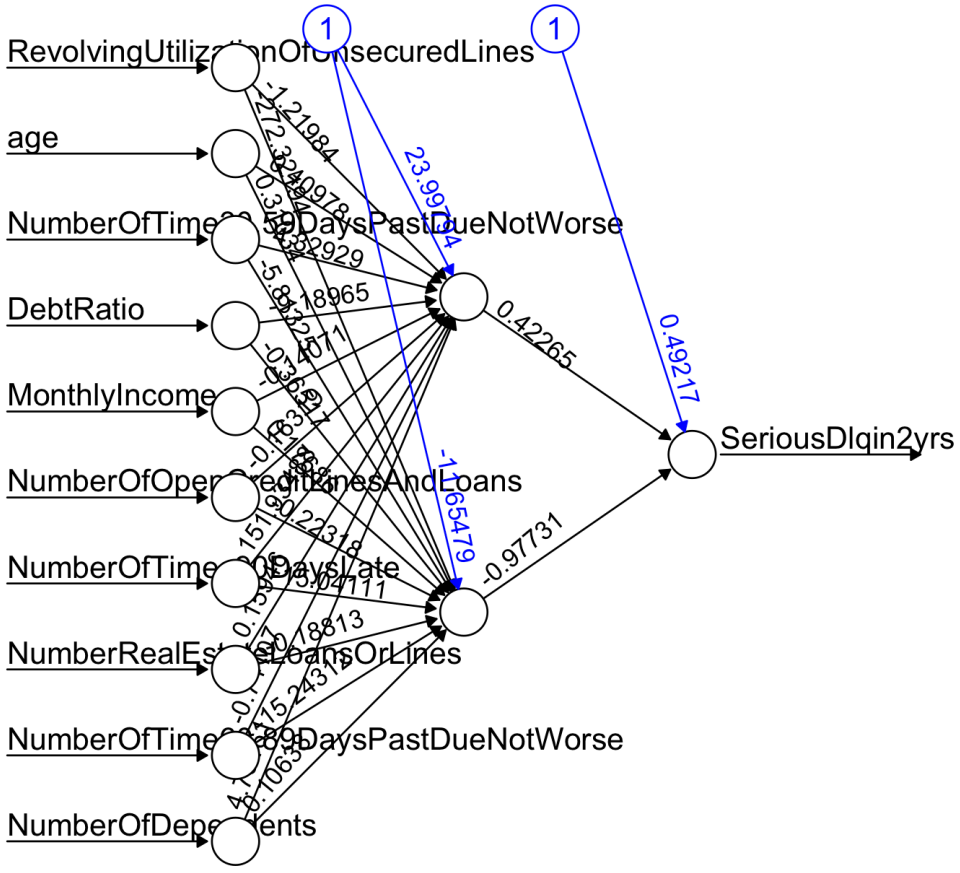
Using the test data (a total of 8357 records), the accuracy of our SVM model is 0.74, with a total of 2173 records being misclassified. In SVM, the accuracy depends on the trade-off between the complexity of a model and a large-margin that misclassifies some of the training data in the interest of better generalization. We will explore the results in details in the Analysis section.

Neural Network

For neural network we proceed the same way we did for all the other algorithms. First we train using our training data. Some of the parameters in the training function include hidden and threshold. **Hidden** refers to a vector of integers specifying the number of hidden neurons (vertices) in each layer. **Threshold** refers to a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.

```
nnmdl <- neuralnet(SeriousDlqin2yrs ~ RevolvingUtilizationOfUnsecuredLines  + age + NumberOfTime30.59DaysPastDueN
otWorse
                       + DebtRatio  + MonthlyIncome +NumberOfOpenCreditLinesAndLoans + NumberOfTimes90DaysLate +
                         NumberRealEstateLoansOrLines +  NumberOfTime60.89DaysPastDueNotWorse
                       + NumberOfDependents, data = training_data, hidden=2, threshold=0.01)
```

This model will be used as a parameter in our ANN predict function to predict the class of each of the dependent variable.



The image above represents the plot of the **nnmdl** model that was generated using our training dataset.

```
nnmdlpredict <- compute(nnmdl, testing_data[,c(2,3,4,5,6,7,8,9,10,11)])

results <- data.frame(actual = testing_SeriousDlqin2yrs, prediction = nnmdlpredict$net.result)
nnmdlpredict <- round(results$prediction)
```

```
table(nnmdlpredict,testing_data[,1 ])
```

```
##
## nnmdlpredict    0    1
##            0 3234  973
##            1  941 3209
```

```
MSE <-mean(nnmdlpredict != testing_SeriousDlqin2yrs)
MSE
```

```
## [1] 0.2290295561
```

```
print(paste('Accuracy',(1-MSE)*100,"%"))
```

```
## [1] "Accuracy 77.0970443939213 %"
```

```
confusionMatrix(nnmdlpredict, testing_SeriousDlqin2yrs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3234  973
##          1  941 3209
##
##                Accuracy : 0.7709704
##                  95% CI : (0.7618078, 0.7799438)
##     No Information Rate : 0.5004188
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.5419435
##  Mcnemar's Test P-Value : 0.4785833
##
##             Sensitivity : 0.7746108
##             Specificity : 0.7673362
##          Pos Pred Value : 0.7687188
##          Neg Pred Value : 0.7732530
##              Prevalence : 0.4995812
##          Detection Rate : 0.3869810
##    Detection Prevalence : 0.5034103
##       Balanced Accuracy : 0.7709735
##
##        'Positive' Class : 0
##
```

On line 1 of the code above, we predict the dependent variable in our test set using the model generated in the previous page. Once the prediction is complete we generate a confusion matrix in line 27. Neural network give us an Accuracy of 77.1%(line 31) the highest of the algorithms so far. We will discuss this further in our result analysis section.

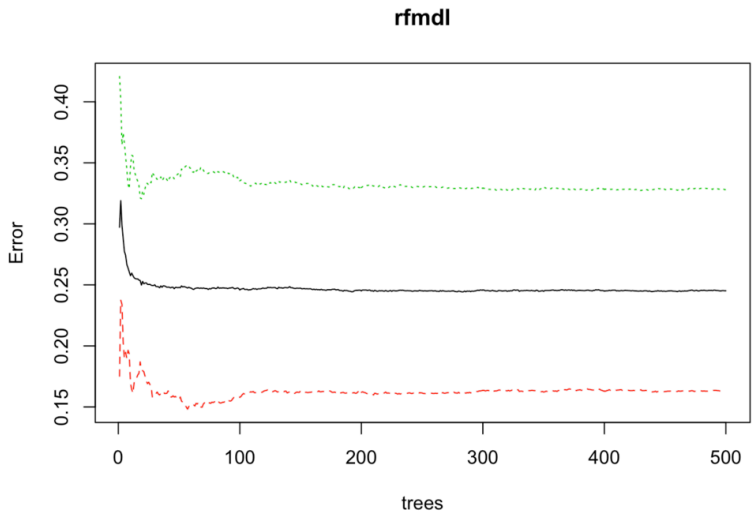
Random Forest

So far we've been working with algorithms that we've talk about in class. But out of curiosity we decided to test out some other highly talked about algorithm in the data science community.

Random forest is one of such algorithm. It's a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set (Wikipedia).

```
set.seed(100)
rfmdl <- randomForest(training_data[,-c(1,2,7,12)], factor(training_data$SeriousDlqin2yrs),
                      sampsize=1000, do.trace=TRUE, importance=TRUE, ntree=500, forest=TRUE)
```

```
plot(rfmdl)
```



First we set a seed so we can replicate our result. Then we build our model. Some of the parameters in the training function include sampsize, do.trace, importance,ntree, and forest.

sampsize	Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.
importance	Should importance of predictors be assessed?
localImp	Should casewise importance measure be computed? (Setting this to TRUE will override importance.)
do.trace	If set to TRUE, give a more verbose output as randomForest is run. If set to some integer, then running output is printed for every do.trace trees.

```
rfmdlpredict <- data.frame(SeriousDlqin2yrs=predict(rfmdl,testing_data[,-c(1,2,7,12)],type="prob")[,2])
rfmdlpredict <- ifelse(rfmdlpredict > 0.5,1,0)

table(rfmdlpredict,testing_data[,1 ])
```

```
##
## rfmdlpredict    0    1
##            0 3420 1401
##            1  726 2810
```

```
MSE <-mean(rfmdlpredict != testing_SeriousDlqin2yrs)
MSE
```

```
## [1] 0.2545172
```

```
print(paste('Accuracy',(1-MSE)*100,"%"))
```

```
## [1] "Accuracy 74.5482828766304 %"
```

On line 1 on the code above, we predict the dependent variable in our test set using the model generated in the previous page. Once the prediction is complete we gernerate a confusion matrix . Neural network give us an Accuracy of 74.5%(line 19 below) a little less accurate than neural network. We will discuss this further in our result analysis section.

```
confusionMatrix(rfmdlpredict, testing_SeriousDlqin2yrs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3420 1401
##          1  726 2810
##
##                Accuracy : 0.7455
##                  95% CI : (0.736, 0.7548)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4916
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8249
##             Specificity : 0.6673
##          Pos Pred Value : 0.7094
##          Neg Pred Value : 0.7947
##              Prevalence : 0.4961
##          Detection Rate : 0.4092
##    Detection Prevalence : 0.5769
##       Balanced Accuracy : 0.7461
##
##        'Positive' Class : 0
##
```

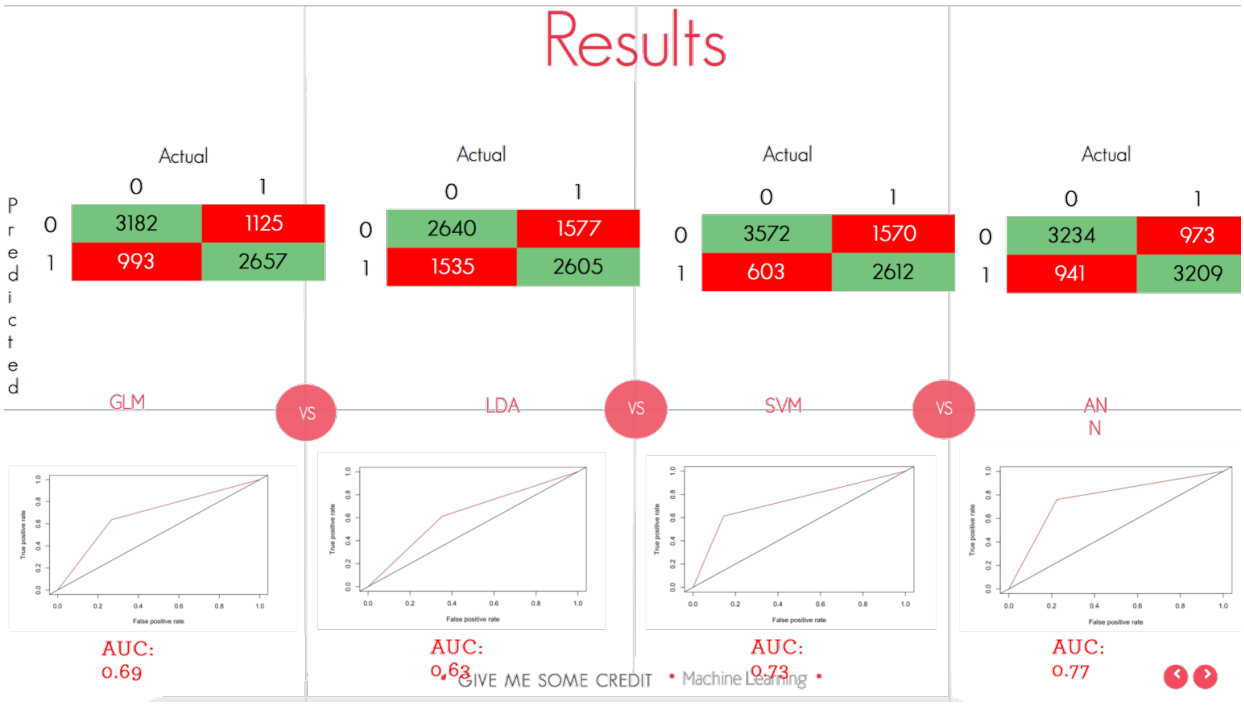
```
rfmdlpredict<- as.numeric(rfmdlpredict)

rfroc<-prediction(rfmdlpredict, testing_SeriousDlqin2yrs, label.ordering = NULL)

rfroc.perf <- performance(rfroc, measure = "tpr", x.measure = "fpr")
plot(rfroc.perf, col = "dark red")
abline(a=0, b= 1)
```

Results Evaluation and Algorithm comparison

Now that we've generated all the confusion matrix and the corresponding ROC curves, we will now evaluate the result of these algorithm. A snapshot of the result all 4 algorithms can be seen below



A grand view of all the results indicated that **LDA** was the worst performing algorithm. We suspect the reason could be that Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. And since our data set is not normally distributed, LDA found it difficult to correctly classify some instances. It should be mentioned that LDA assumes normal distributed data, features that are statistically independent, and identical covariance matrices for every class.

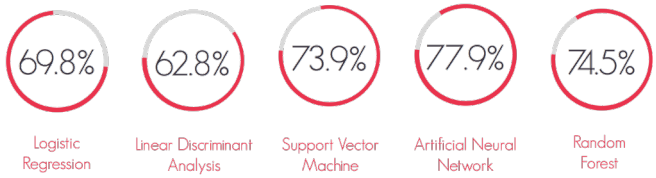
Logistic Regression with an AUC of 0.69 is the second to the last performing in terms of accuracy. Unlike LDA Logistic regression has several advantages over discriminant analysis some which are that it is more robust: the independent variables don't have to be normally distributed, or have equal variance in each group and It does not assume a linear relationship between the independent variable and the dependent variable. Unfortunately, the advantages of logistic regression come at a cost: it requires much more data to achieve stable, meaningful results. With standard regression, and LDA, typically 20 data points per predictor is considered the lower bound. For logistic regression, at least 50 data points per predictor is necessary to achieve stable results. This is not the case with our dataset which had about 10 predictors.

Third on the list is **SVM**, which had an AUC of 0.73. SVM was good in this case because SVM has a regularization parameter, which makes the user think about avoiding over-fitting. it also uses the kernel trick, so you can build in expert knowledge about the problem via engineering the kernel.

Finally, we examine the result of ANN. **Artificial Neural Network** performed the best with an AUC of 0.77. As in most cases it's usually difficult to explain why ANN gives the best results but our inclination is that you'll find that finite ANNs with even one hidden layer can approximate a huge class of functions. This property turns out to be fairly robust relative to the exact choice of network: by this we mean, you don't have to have the choice of network exactly right in order to approximate a particular function - there may be a network which is ideal for that function, but you can still do quite well with a similar network (of sufficient complexity).

With a new appreciation for the computational power of neural networks, we can return to the problem of learning real-world patterns like the one we have. We use ANN in situation were we suspect there is a rule governing how this a value changes as a function of the input. However, the scenario is very complicated, and you are not sure how to model it with an equation or an algorithm, this is where we ask ANN to model the rule for us. This in most cases returns very good results. This we suspect is the case here.

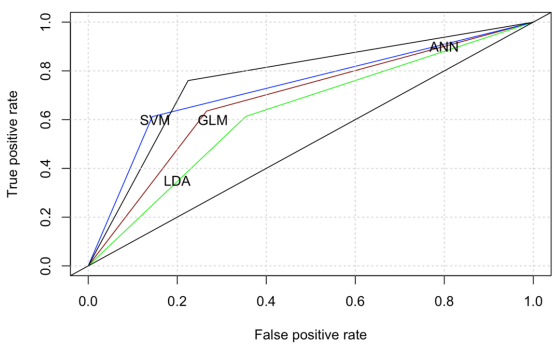
Below is a snapshot of the Accuracy of all the algorithms.



The ROC shows also the performance of all 4 algorithms with ANN occupying the most area and LDA doing not as well.

plot all ROCS

```
plot( glmroc.perf, col = "dark red")
plot(ldaroc.perf, add = TRUE, col = "green")
plot(svmroc.perf, add = TRUE, col = "blue")
plot(nnroc.perf, add = TRUE, col = "black")
grid()
abline(a=0, b= 1)
text(.8,.9,"ANN")
text(.28,.6,"GLM")
text(.15,.6,"SVM")
text(.2,.35,"LDA")
```

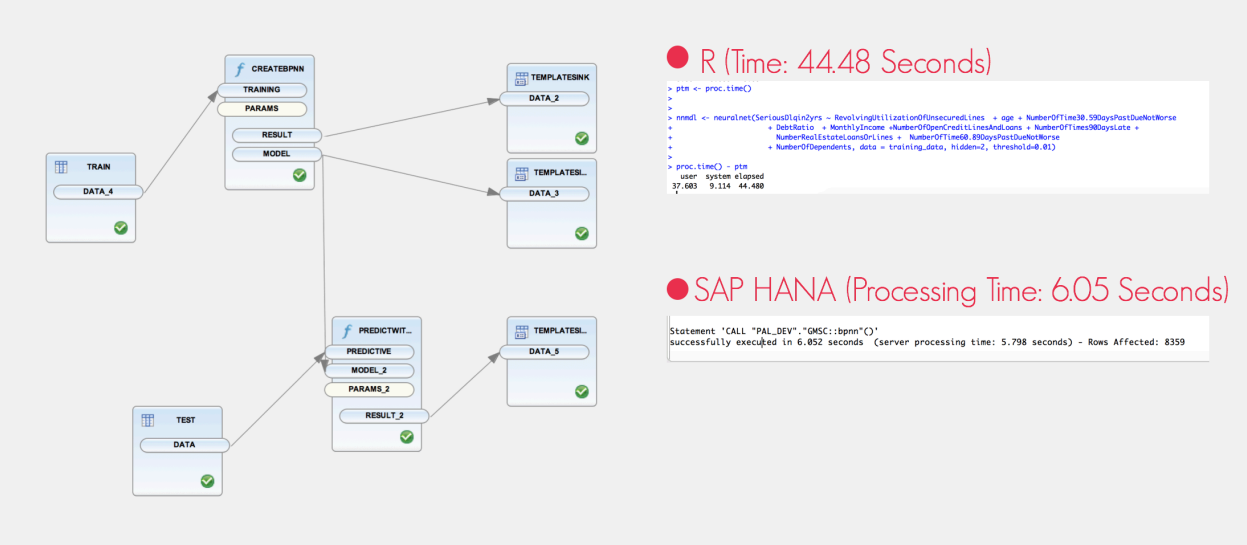


Performance Comparison with SAP HANA

We wanted to see how the algorithm performed when ran in an in memory database like SAP HANA. To this we wanted to compare the algorithm that took the longest time to run in R and compare it with the run on SAP HANA.

ANN averaged about 40 seconds when ran on our dataset in R environment. The longest time of all the other algorithms. This dropped down to about 6 seconds when ran on SAP HANA.

We can easily conclude that given the computational complexity of ANN, we would recommend running ANN on an in memory database especially if you ANN structure has several layers and neurons.



Lessons Learned and Challenges

There were several challenges that we encountered and lessons we learnt throughout this project but these three where particularly impactful.

Preprocessing and wrangling the data: Preprocessing is an essential part of predictive analysis. This part of the project took the most time and was very critical when it came time to use the algorithms. From normalization to stratification and outlier elimination it played a key role in making sure we are not feeding garbage to our algorithms.

Handling the Skew of the data: Skew in data can impact the result you get. This we discovered earlier in our wrangling process. Skew data can produce results with height accuracy and low precision. A lesson we've learnt and tend to look out for in future work with datasets.

Choosing the right parameters for the algorithms: Choosing the right parameters can impact your results especially for parametric algorithms like the ones we examined in this project. We did not explore a lot of the parameters present for most of the algorithms. We suspect that a more thorough examination and use of these would improve our model significantly.

Conclusion

In conclusion, we were able to improve fractionally by .9% the predictive accuracy of the credit scoring sample model, which had an accuracy of 77% by using Artificial Neural Network. ANN performed the best out of all the other algorithm for our particular use case.

References

G. C. Cawley and N. L. C. Talbot, Over-fitting in model selection and subsequent selection bias in performance evaluation, Journal of Machine Learning Research, 2010. Research, vol. 11, pp. 2079-2107, July 2010. (pdf)

https://victorfang.wordpress.com/2011/05/10/advantages-and-disadvantages-of-logistic-regression/

https://www.quora.com/Why-do-Artificial-Neural-Networks-work-the-way-they-do