Credit Card Default Model

Performance Validation Guide

PREDICT 498 Capstone with Dr. Bhatti, Winter 2018

M.S. Predictive Analytics

Northwestern University

Author: Ahmed Mustakim

Date: March 6, 2018

Contents

1. The Production Model

In the model development guide, we focused on creating predictive analytics models in R that predicted credit card payment defaults using a data set covering 30,000 sample Taiwanese customers at a financial institution. The data set for these customers included 24 attributes covering their credit card payment history (such as credit limit, their monthly bill, how much of that bill they paid, how late they were in making payments and if they actually defaulted on their payments) and socio-demographic attributes (such as age, gender, education level and marital status) or 6 consecutive months from April 2005.

Prior to constructing those models, we imputed out-of-range values in range-bound categorical variables and ignored the late payment indicator fields because they were extremely sparse. We also engineered new features that allowed us to normalize and benchmark payment, credit utilization and other metrics across individual customer records. We experimented with four different advanced predictive modeling techniques on the data which were random forest, gradient boosting, logistic regression with backwards variable selection and support vector machine. The logistic regression model was initialized based on a pool of the top 10 relevant variables identified by the random forest and gradient boosting methods and then backwards variable selection was employed to prune the list of predictor variables further.

Of these four models, random forest yielded the highest model but the simplest model, which was the logistic regression model, had a better balance of true positive rate vs. false positive rate with marginal degradation in model accuracy. As such as the logistic regression was the model of choice for the production environment. The following tables provide the model parameters including predictor variables, coefficients, standard error and p-value:

Predictor Variable	Coefficient	Standard Error	p-value
Intercept	-1.42E+00	1.04E-01	< 2e-16
TOTAL_PAY_RT	7.45E-01	1.73E-01	1.67E-05
PAY_RT2	-2.25E-01	9.61E-02	0.019232
PAY_RT3	-3.33E-01	9.50E-02	0.000465
PAY_AMT1	-2.59E-05	3.80E-06	9.06E-12
PAY_AMT2	-1.48E-05	3.36E-06	9.88E-06
PAY_AMT3	-4.20E-06	2.37E-06	0.075695
LIMIT_BAL	-1.97E-06	2.16E-07	< 2e-16
AGE	1.03E-02	2.11E-03	9.64E-07
UTIL_DIFF1	-3.14E-01	2.02E-01	0.119898
UTIL_RT1	-4.14E-01	2.42E-01	0.08733
UTIL_RT2	7.82E-01	2.33E-01	0.000787
UTIL_RT4	4.67E-01	1.52E-01	0.002143
TOTAL_UTIL_VEL	-1.80E+00	8.57E-01	0.035427

Table 1.1: *Model parameters*

.2. Model Development Performance

The following table shows the model performance on the train and test data sets – i.e. the performance metrics discussed thus far:

Performance Metric	Training	Test
True Positive Rate	0%	79%
False positive rate	23%	22%
Model accuracy	77%	78%

Table 2.1: *Performance metrics for logistic regression*

These performance metrics were quite helpful in selecting the most optimal model out of several candidates. However once the model is deployed in a production environment, we will need to monitor the following additional metrics (illustrated conceptually in figure 2.1 below) to understand how the model performs over its life-time and also to ascertain when to make additional improvements or implement a new model altogether. These new metrics are in fact based on the true positive rate (TP) and false positive rate (FP) metrics discussed earlier:

- **ROC** – "A receiver operating characteristics (ROC) graph is a technique for visualizing, organizing and selecting classifiers based on their performance… ROC graphs are two-dimensional graphs in which TP (true positive) rate is plotted on the Y axis and FP (false positive) rate is plotted on the X axis. An ROC graph depicts relative tradeoffs between benefits (true positives) and costs (false positives)." *(Fawcett 2005).*
- **AUC** – "A common method is to calculate the area under the ROC curve, abbreviated AUC (Bradley, 1997; Hanley and McNeil, 1982). Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0. However, because random guessing produces the diagonal line between (0, 0) and (1, 1), which has an area of 0.5, no realistic classifier should have an AUC less than 0.5." *(Fawcett 2005).*

When we extend the model performance measures to include the ROC curves and the AUC statistics for the train and test data sets, we observe the following ROC charts each with an AUC of 0.66 (rounded to 2 decimal points):

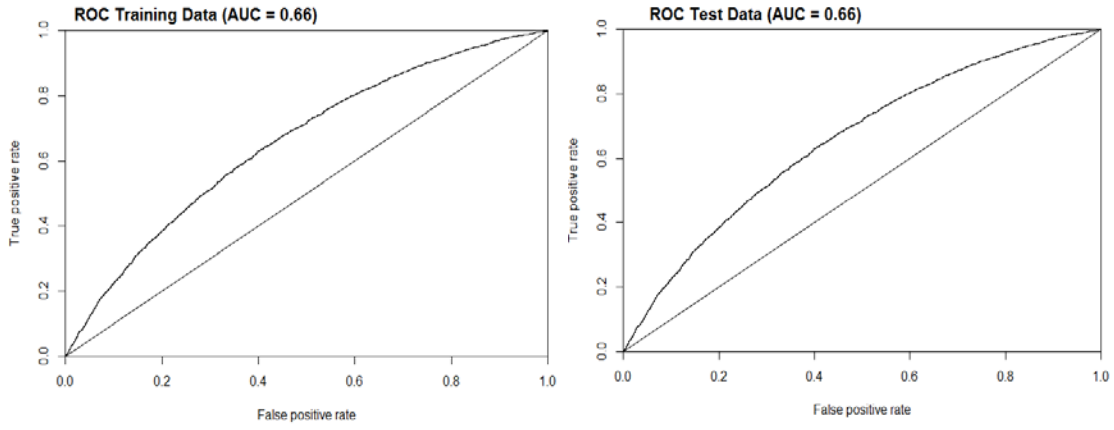


Figure 2.1: *ROC and AUC for logistic regression (training and test data)*

In addition to the ROC-AUC plots, we also look at the lift chart and K-S statistic in a production environment. A lift chart measures "the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model" *(Hamilton 2012)* and it does so by first calculating the probability for each observation; ranking these probabilities in decreasing order; building deciles with each group having almost 10% of the observations and then calculating the correct and incorrect response rates at each deciles.(*Srivastava 2016).* This systematic approach allows us to look at the cumulative "lift" or gain as a result of implementing this model and then ascertain whether we should continue to use this model in the future. *(Hamilton 2012)*

K-S or Kolmogorov-Smirnov "is a measure of the degree of separation between the positive and negative distributions. The K-S is 100, if the scores partition the population into two separate groups in which one group contains all the positives and the other all the negatives. On the other hand, If the model cannot differentiate between positives and negatives, then it is as if the model selects cases randomly from the population. (In that case) The K-S would be 0. In most classification models the K-S will fall between 0 and 100, and that the higher the value the better the model is at separating the positive from negative cases." (*Srivastava 2016)*

The following lift charts along with the Kolmogorov-Smirnov (KS) statistics in table 2.2 and table 2.3 below provide an additional measure of model performance covering both the train and test data sets. They show that the highest K-S is achieved in the 5^{th} deciles in both data sets:

- Training K-S: 22.7%
- Test K-S: 23.7 %

Decile	Obs	Target (Y=1)	NonTarget (Y=0)	Target Density	NonTarget Density	Target CDF	NonTarget CDF	KS Stat
1	1,518	627	891	18.3%	7.6%	18.3%	7.6%	10.7%
2	1,518	503	1,015	14.7%	8.6%	33.0%	16.2%	16.8%
3	1,518	439	1,079	12.8%	9.2%	45.8%	25.4%	20.4%
4	1,518	394	1,124	11.5%	9.6%	57.3%	34.9%	22.4%
5	1,518	350	1,168	10.2%	9.9%	67.6%	44.9%	**22.7%**
6	1,518	306	1,212	8.9%	10.3%	76.5%	55.2%	21.3%
7	1,518	269	1,249	7.9%	10.6%	84.4%	65.8%	18.6%
8	1,518	226	1,292	6.6%	11.0%	91.0%	76.8%	14.2%
9	1,518	182	1,336	5.3%	11.4%	96.3%	88.2%	8.1%
10	1,518	127	1,391	3.7%	11.8%	100.0%	100.0%	0.0%
Totals	**15,180**	**3423**	**11,757**	**100.0%**	**100.0%**			

Table 2.2: *Lift Chart and K-S Statistic – Training Data*

Decile	Obs	Target (Y=1)	NonTarget (Y=0)	Target Density	NonTarget Density	Target CDF	NonTarget CDF	KS Stat
1	733	269	464	17.3%	8.0%	17.3%	8.0%	9.2%
2	732	232	500	14.9%	8.7%	32.2%	16.7%	15.5%
3	732	202	530	13.0%	9.2%	45.2%	25.9%	19.2%
4	732	201	531	12.9%	9.2%	58.1%	35.1%	22.9%
5	732	165	567	10.6%	9.8%	68.7%	45.0%	**23.7%**
6	733	127	606	8.2%	10.5%	76.8%	55.5%	21.4%
7	732	116	616	7.5%	10.7%	84.3%	66.1%	18.1%
8	732	105	627	6.7%	10.9%	91.0%	77.0%	14.0%
9	732	89	643	5.7%	11.2%	96.7%	88.2%	8.6%
10	733	51	682	3.3%	11.8%	100.0%	100.0%	0.0%
Totals	**7,323**	**1557**	**5,766**	**100.0%**	**100.0%**			

Table 2.3: *Lift Chart and K-S Statistic – Test Data*

However, when we look at figures 2.2 and 2.3 below, which shows the lift values against the cut-off thresholds of the predicted probabilities from the logistic regression (e.g. if the predicted probability is above 0.5 mark an observation as “Default” and below that as “Not

Default"), we notice that for the training data the optimal cut off is slightly above 0.4 but for the test data it is above 0.5. It is therefore quite likely that the cut-off threshold will need to adjusted in the production model as part of routine maintenance.

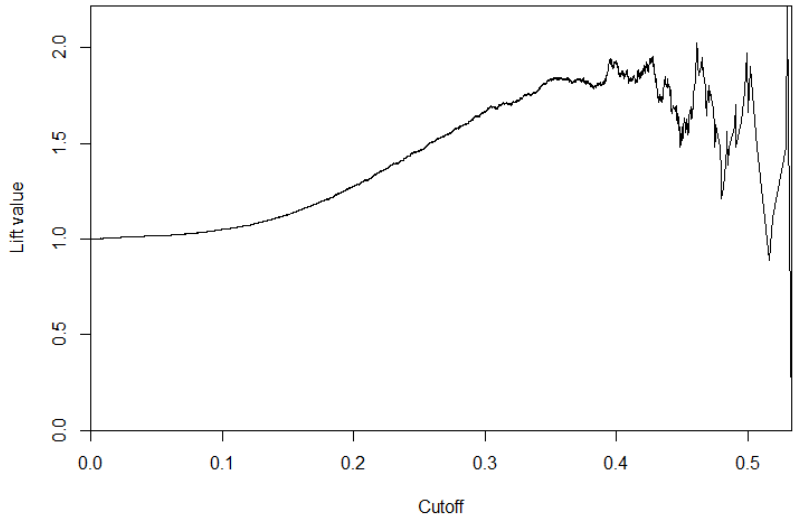


Figure 2.2: *Lift values vs. Cut-off threshold of predicted probabilities (training data)*

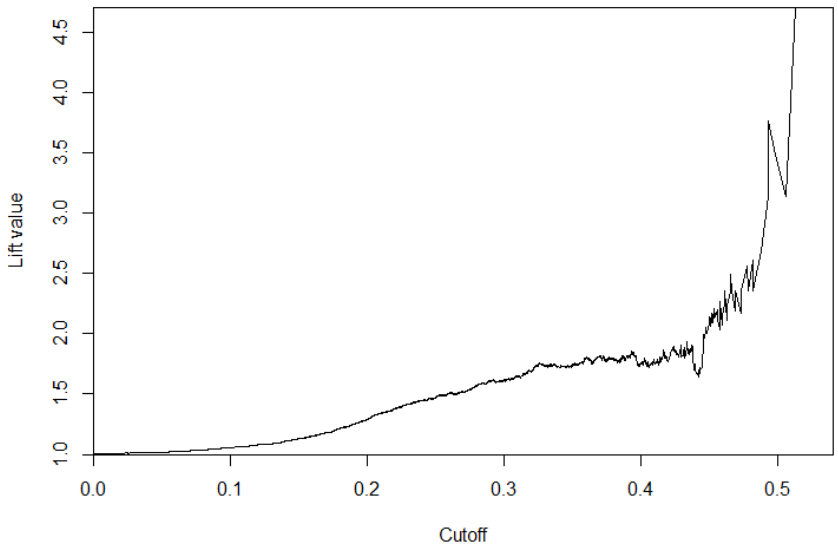


Figure 2.3: *Lift values vs. Cut-off threshold of predicted probabilities (test data)*

In fact the K-S statistic will guide the routine maintenance of the model. This can be explained by visualizing the K-S statistics (tables 2.2 and 2.3 above) as illustrated in figure 2.4 below. These charts show that when the largest gap occurs between the baseline performance and the performance achieved by the logistic regression model in a particular decile, that is the most optimal decile (in this case the 5th decile) that separates the 2 predicted classes (default vs. non-default). Therefore the idea is to maintain the largest gap between the model performance and the baseline performance in each decile, in the production environment. If the K-S statistic starts getting closer to the baseline performance in production, then it starts to indicate that the model will need to be adjusted. The closer the gap is, the more severe the model remediation action will be. This kind of monitoring and maintenance will need to occur regularly and consistently and a systematic performance monitoring plan will outline the appropriate actions that will need to be taken.

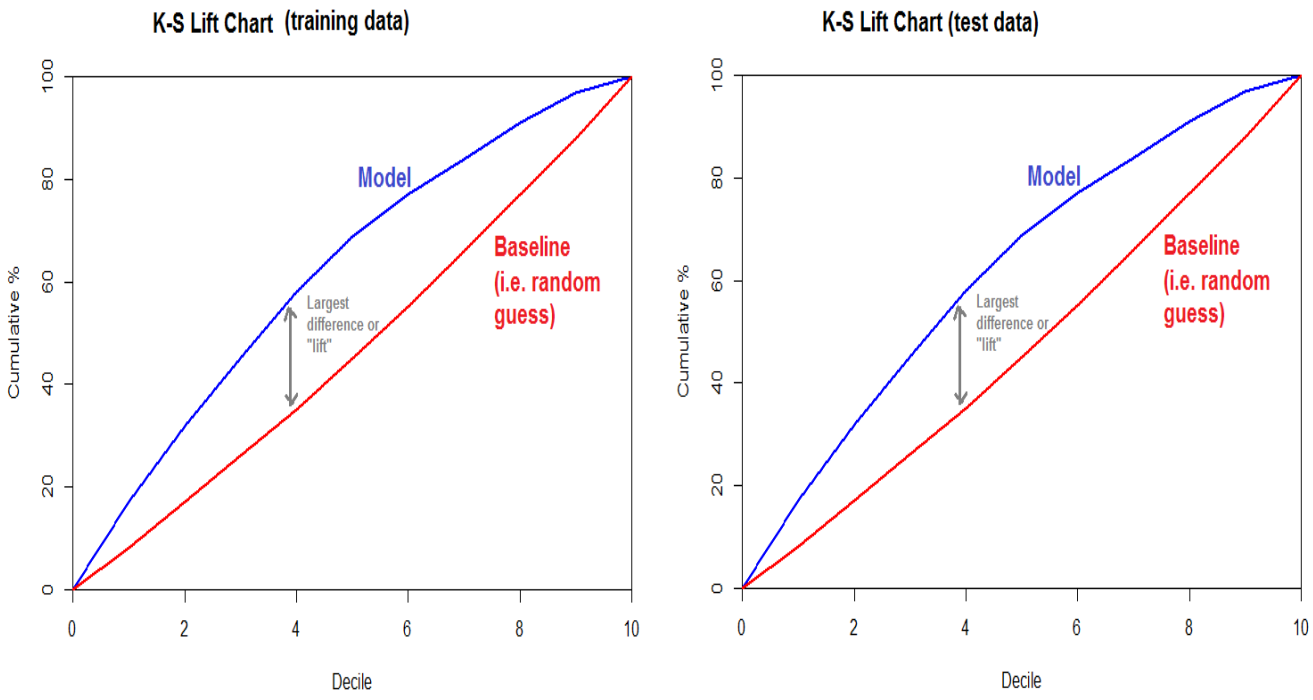


Figure 2.4: *KS-Lift chart (training and test data)*

3. Performance Monitoring Plan

As mentioned earlier, in a production environment the model will need to be routinely maintained or adjusted by consistently monitoring the K-S statistic performance metric. The following RAG (Red-Amber-Green) statuses as explained in table 3.1 below, form the basis of this performance monitoring plan:

Red	Model needs re-development
Amber	Model needs to be re-validated in 3 months
Green	Model is performing as expected

Table 3.1: RAG (Red, Amber and Green) Performance Statuses

Using this RAG status rubric, the following table outlines the metric thresholds for the KS statistic in each decile, which are set using somewhat subjective ("eye-ball") assessments. This table is based on the test K-S statistics in table 2.3 above and shows the absolute change (not relative changes) to the KS statistic. So for instance, after the model is deployed in production and run against the validation data set and the KS stat for the 5th decile is 12% then the model will need to be redeveloped.

Decile	KS Stat (at this number or higher)		
1	3%	5%	9.2%
2	6%	8%	15.5%
3	5%	10%	19.2%
4	8%	15%	22.9%
5	8%	15%	23.7%
6	8%	15%	21.4%
7	5%	10%	18.1%
8	6%	8%	14.0%
9	3%	5%	8.6%
10	0.0%	0.0%	0.0%

Table 3.2: KS-RAG Performance Monitoring Plan

4. Performance Monitoring Results

The following is the lift-chart along with the K-S statistic for the validation data set (which is a stand-in for real life production data in this capstone simulation):

Decile	Obs	Target (Y=1)	NonTarget (Y=0)	Target Density	NonTarget Density	Target CDF	NonTarget CDF	KS Stat
1	750	301	449	18.2%	7.7%	18.2%	7.7%	10.5%
2	750	253	497	15.3%	8.5%	33.5%	16.2%	17.3%
3	749	214	535	12.9%	9.2%	46.4%	25.4%	21.0%
4	750	190	560	11.5%	9.6%	57.9%	34.9%	22.9%
5	749	148	601	8.9%	10.3%	66.8%	45.2%	**21.6%**
6	750	155	595	9.4%	10.2%	76.1%	55.4%	20.7%
7	750	130	620	7.9%	10.6%	84.0%	66.0%	18.0%
8	749	114	635	6.9%	10.9%	90.9%	76.9%	14.0%
9	750	87	663	5.3%	11.4%	96.1%	88.3%	7.9%
10	750	64	686	3.9%	11.7%	100.0%	100.0%	0.0%
Totals	**7,497**	**1656**	**5,841**	**100.0%**	**100.0%**			

Table 4.1: KS-RAG Performance Monitoring Plan

Comparing this result against the KS-RAG Performance Monitoring Plain in table 3.2 above, leads us to the conclusion that the model will need to be re-validated in 3 months, which will be our recommendation to model governance. Having said that, the KS stat values are still pretty close to the thresholds in table 3.2 above, which indicates that the model is performing really well against the validation data set.

5. Bibliography

UCI Machine Learning Repository (2016, January 26). Default of credit card client's data set. Retrieved February 26, 2018, from https://archive.ics.uci.edu/ml/datasets/default%20of%20credit%20card%20clients

Mustakim, A (2018, February 28). Credit Card Default - Model Development Guide. Northwestern University. Retrieved March 4, 2018 from https://canvas.northwestern.edu/

Fawcett, T. (2005). An introduction to ROC analysis. Elseiver - Institute for the Study of Learning and Expertise. Retrieved March 5, 2018, from https://ccrma.stanford.edu/workshops/mir2009/references/ROCintro.pdf

Dernoncourt, F. (2015, January 9). What does AUC stand for and what is it? Retrieved March 5, 2018, from https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it

Hamilton, H. (2012, June 8). Cumulative Gains and Lift Charts. Retrieved March 6, 2018, from http://www2.cs.uregina.ca/~dbd/cs831/notes/lift_chart/lift_chart.html

Srivastava, T. (2016, February 19). 7 Important Model Evaluation Error Metrics Everyone should know. Retrieved March 6, 2018, from https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/

Vogler, R. (2015, June 23). Illustrated Guide to ROC and AUC. Retrieved March 5, 2018, from https://www.r-bloggers.com/illustrated-guide-to-roc-and-auc/

6. Appendix

6.1 Finalized data preparation steps in R

```
rm(list=ls())

# Define path and file;
# You will need to change the path value to match the location on your own computer;
my.path <- 'C:\\Users\\amustaki\\Documents\\mspa\\498\\data\\';
my.file <- paste(my.path,'credit_card_default.RData',sep='');


# Read the RData object using readRDS();
credit_card_default <- readRDS(my.file)
#Backup original data set
###############
credit_card_default_bkp <- credit_card_default

#New variables
###############

#Mute EDUCAION for unknown values
credit_card_default$EDUCATION[credit_card_default$EDUCATION==0] <- NA
credit_card_default$EDUCATION[credit_card_default$EDUCATION==5] <- NA
credit_card_default$EDUCATION[credit_card_default$EDUCATION==6] <- NA

#Mute MARRIAGE for unknown values
credit_card_default$MARRIAGE[credit_card_default$MARRIAGE==0] <- NA

#Mute PAY_X for unknown values
credit_card_default$PAY_0[credit_card_default$PAY_0==-2] <- NA
credit_card_default$PAY_0[credit_card_default$PAY_0==0] <- NA

credit_card_default$PAY_2[credit_card_default$PAY_2==-2] <- NA
credit_card_default$PAY_2[credit_card_default$PAY_2==0] <- NA

credit_card_default$PAY_3[credit_card_default$PAY_3==-2] <- NA
credit_card_default$PAY_3[credit_card_default$PAY_3==0] <- NA

credit_card_default$PAY_4[credit_card_default$PAY_4==-2] <- NA
credit_card_default$PAY_4[credit_card_default$PAY_4==0] <- NA

credit_card_default$PAY_5[credit_card_default$PAY_5==-2] <- NA
credit_card_default$PAY_5[credit_card_default$PAY_5==0] <- NA

credit_card_default$PAY_6[credit_card_default$PAY_6==-2] <- NA
credit_card_default$PAY_6[credit_card_default$PAY_6==0] <- NA

#Engineered Features - Utilization Rate
util_rate_func <- function (x,y){
  ifelse( y != 0 ,
        ifelse(x>y,1,x/y),
        ifelse(x>0,1,0)
```

```
 )
}

credit_card_default$UTIL_RT1 <- util_rate_func(credit_card_default$BILL_AMT1, credit_card_default$LIMIT_BAL)
credit_card_default$UTIL_RT2 <- util_rate_func(credit_card_default$BILL_AMT2, credit_card_default$LIMIT_BAL)
credit_card_default$UTIL_RT3 <- util_rate_func(credit_card_default$BILL_AMT3, credit_card_default$LIMIT_BAL)
credit_card_default$UTIL_RT4 <- util_rate_func(credit_card_default$BILL_AMT4, credit_card_default$LIMIT_BAL)
credit_card_default$UTIL_RT5 <- util_rate_func(credit_card_default$BILL_AMT5, credit_card_default$LIMIT_BAL)
credit_card_default$UTIL_RT6 <- util_rate_func(credit_card_default$BILL_AMT6, credit_card_default$LIMIT_BAL)

#PAyment Rate
pay_rate_func <- function (x,y) {
 ifelse( x!=0,
       ifelse(y>x,1,y/x),
       1
 )
}

credit_card_default$PAY_RT1 <- pay_rate_func(credit_card_default$BILL_AMT1,credit_card_default$PAY_AMT1)
credit_card_default$PAY_RT2 <- pay_rate_func(credit_card_default$BILL_AMT2,credit_card_default$PAY_AMT2)
credit_card_default$PAY_RT3 <- pay_rate_func(credit_card_default$BILL_AMT3,credit_card_default$PAY_AMT3)
credit_card_default$PAY_RT4 <- pay_rate_func(credit_card_default$BILL_AMT4,credit_card_default$PAY_AMT4)
credit_card_default$PAY_RT5 <- pay_rate_func(credit_card_default$BILL_AMT5,credit_card_default$PAY_AMT5)
credit_card_default$PAY_RT6 <- pay_rate_func(credit_card_default$BILL_AMT6,credit_card_default$PAY_AMT6)


#Utilization Velocity
credit_card_default$UTIL_VEL1 = credit_card_default$UTIL_RT1 - credit_card_default$UTIL_RT2
credit_card_default$UTIL_VEL2 = credit_card_default$UTIL_RT2 - credit_card_default$UTIL_RT3
credit_card_default$UTIL_VEL3 = credit_card_default$UTIL_RT3 - credit_card_default$UTIL_RT4
credit_card_default$UTIL_VEL4 = credit_card_default$UTIL_RT4 - credit_card_default$UTIL_RT5
credit_card_default$UTIL_VEL5 = credit_card_default$UTIL_RT5 - credit_card_default$UTIL_RT6


#Total Utilization Velocity
credit_card_default$TOTAL_UTIL_VEL = ((credit_card_default$UTIL_VEL1 +
                            credit_card_default$UTIL_VEL2 +
                            credit_card_default$UTIL_VEL3 +
                            credit_card_default$UTIL_VEL4 +
                            credit_card_default$UTIL_VEL5)/5)



#Difference from min utilization
credit_card_default <- transform(credit_card_default, min_UTIL_RT = pmin(UTIL_RT1,
                                          UTIL_RT2,
                                          UTIL_RT3,
                                          UTIL_RT4,
                                          UTIL_RT5,
                                          UTIL_RT6
))

credit_card_default$UTIL_DIFF1 = credit_card_default$UTIL_RT1 - credit_card_default$min_UTIL_RT
```

```
credit_card_default$UTIL_DIFF2 = credit_card_default$UTIL_RT2 - credit_card_default$min_UTIL_RT
credit_card_default$UTIL_DIFF3 = credit_card_default$UTIL_RT3 - credit_card_default$min_UTIL_RT
credit_card_default$UTIL_DIFF4 = credit_card_default$UTIL_RT4 - credit_card_default$min_UTIL_RT
credit_card_default$UTIL_DIFF5 = credit_card_default$UTIL_RT5 - credit_card_default$min_UTIL_RT
credit_card_default$UTIL_DIFF6 = credit_card_default$UTIL_RT6 - credit_card_default$min_UTIL_RT



#Total PAy Rate
credit_card_default$TOTAL_PAY_RT = ((credit_card_default$PAY_RT1 +
                          credit_card_default$PAY_RT2 +
                          credit_card_default$PAY_RT3 +
                          credit_card_default$PAY_RT4 +
                          credit_card_default$PAY_RT5 +
                          credit_card_default$PAY_RT6)/6)

#Age bins
credit_card_default$AGE_60_ABOVE <- as.numeric(credit_card_default$AGE>=60)
credit_card_default$AGE_45_59 <- as.numeric( credit_card_default$AGE>= 45 & credit_card_default$AGE<60)
credit_card_default$AGE_40_44 <- as.numeric( credit_card_default$AGE>= 40 & credit_card_default$AGE<45)
credit_card_default$AGE_36_39 <- as.numeric( credit_card_default$AGE>= 36 & credit_card_default$AGE<40)
credit_card_default$AGE_33_35 <- as.numeric( credit_card_default$AGE>= 33 & credit_card_default$AGE<36)
credit_card_default$AGE_30_32 <- as.numeric( credit_card_default$AGE>= 30 & credit_card_default$AGE<33)
credit_card_default$AGE_29_BELOW <- as.numeric(credit_card_default$AGE < 30)

credit_card_default$AGE_BIN <- with(credit_card_default, ifelse(
 credit_card_default$AGE_60_ABOVE > 0, '60_and_above', ifelse(
  credit_card_default$AGE_45_59 > 0, '45_to_59', ifelse(
   credit_card_default$AGE_40_44 > 0, '40_to_44', ifelse(
    credit_card_default$AGE_36_39 > 0, '36_to_39', ifelse(
     credit_card_default$AGE_33_35 > 0, '33_to_35', ifelse(
      credit_card_default$AGE_30_32 > 0, '30_to_32', '29_and_below'
     )
    )
   )
  )
 ) ) )



#Convert discrete variables to factor allow EDA plotting
credit_card_default$SEX <- as.factor(credit_card_default$SEX)
credit_card_default$EDUCATION <- as.factor(credit_card_default$EDUCATION)
credit_card_default$MARRIAGE <- as.factor(credit_card_default$MARRIAGE)
credit_card_default$PAY_0 <- as.factor(credit_card_default$PAY_0)
credit_card_default$PAY_2 <- as.factor(credit_card_default$PAY_2)
credit_card_default$PAY_3 <- as.factor(credit_card_default$PAY_3)
credit_card_default$PAY_4 <- as.factor(credit_card_default$PAY_4)
credit_card_default$PAY_5 <- as.factor(credit_card_default$PAY_5)
credit_card_default$PAY_6 <- as.factor(credit_card_default$PAY_6)

#Class boundaries
######################
```

```
credit_card_default$UTIL_RT2_0.49_AND_ABOVE <- as.numeric(credit_card_default$UTIL_RT2 >= 0.49)
credit_card_default$UTIL_RT2_BELOW_0.49 <- as.numeric(credit_card_default$UTIL_RT2 < 0.49)

credit_card_default$UTIL_RT3_0.45_AND_ABOVE <- as.numeric(credit_card_default$UTIL_RT3 >= 0.45)
credit_card_default$UTIL_RT3_BELOW_0.45 <- as.numeric(credit_card_default$UTIL_RT3 < 0.45)

credit_card_default$UTIL_RT4_0.48_AND_ABOVE <- as.numeric(credit_card_default$UTIL_RT4 >= 0.48)
credit_card_default$UTIL_RT4_BELOW_0.48 <- as.numeric(credit_card_default$UTIL_RT4 < 0.48)

credit_card_default$UTIL_RT5_0.68_AND_ABOVE <- as.numeric(credit_card_default$UTIL_RT5 >= 0.68)
credit_card_default$UTIL_RT5_BELOW_0.68 <- as.numeric(credit_card_default$UTIL_RT5 < 0.68)

credit_card_default$UTIL_RT6_0.62_AND_ABOVE <- as.numeric(credit_card_default$UTIL_RT6 >= 0.62)
credit_card_default$UTIL_RT6_BELOW_0.62 <- as.numeric(credit_card_default$UTIL_RT6 < 0.62)


credit_card_default$TOTAL_PAY_RT_0.18_AND_ABOVE <- as.numeric(credit_card_default$TOTAL_PAY_RT >=
0.18)
credit_card_default$TOTAL_PAY_RT_BELOW_0.18 <- as.numeric(credit_card_default$TOTAL_PAY_RT < 0.18)

credit_card_default$LIMIT_BAL_125k_AND_ABOVE <- as.numeric(credit_card_default$LIMIT_BAL >= 125000)
credit_card_default$LIMIT_BAL_BELOW_125k <- as.numeric(credit_card_default$LIMIT_BAL < 125000)

credit_card_default$UTIL_RT2_BIN <- with(credit_card_default, ifelse(
  credit_card_default$UTIL_RT2_0.49_AND_ABOVE > 0
  , '0.49_AND_ABOVE'
  , 'BELOW_0.49'
)
)

credit_card_default$UTIL_RT3_BIN <- with(credit_card_default, ifelse(
  credit_card_default$UTIL_RT3_0.45_AND_ABOVE > 0
  , '0.45_AND_ABOVE'
  , 'BELOW_0.45'
)
)

credit_card_default$UTIL_RT4_BIN <- with(credit_card_default, ifelse(
  credit_card_default$UTIL_RT4_0.48_AND_ABOVE > 0
  , '0.48_AND_ABOVE'
  , 'BELOW_0.48'
)
)

credit_card_default$UTIL_RT5_BIN <- with(credit_card_default, ifelse(
  credit_card_default$UTIL_RT5_0.68_AND_ABOVE > 0
  , '0.68_AND_ABOVE'
  , 'BELOW_0.68'
)
)

credit_card_default$UTIL_RT6_BIN <- with(credit_card_default, ifelse(
  credit_card_default$UTIL_RT6_0.62_AND_ABOVE > 0
```

```
  , '0.62_AND_ABOVE'
  , 'BELOW_0.62'
)
)


credit_card_default$TOTAL_PAY_RT_BIN <- with(credit_card_default, ifelse(
  credit_card_default$TOTAL_PAY_RT_0.18_AND_ABOVE > 0
  , '0.18_AND_ABOVE'
  , 'BELOW_0.18'
)
)

credit_card_default$LIMIT_BAL_125k_AND_ABOVE <- as.numeric(credit_card_default$LIMIT_BAL >= 125000)
credit_card_default$LIMIT_BAL_BELOW_125k <- as.numeric(credit_card_default$LIMIT_BAL < 125000)

credit_card_default$LIMIT_BAL_BIN <- with(credit_card_default, ifelse(
  credit_card_default$LIMIT_BAL_125k_AND_ABOVE > 0
  , '125k_AND_ABOVE'
  , 'BELOW_125k'
)
)

cc_train <- credit_card_default[credit_card_default$train == 1,]

#remove temporary and ineffective variables
drop <- c("ID",
        "PAY_0",
        "PAY_2",
        "PAY_3",
        "PAY_4",
        "PAY_5",
        "PAY_6",
        "u",
        "train",
        "test",
        "validate",
        "data.group",
        "min_UTIL_RT",
        "AGE_BIN",
        "UTIL_RT2_BIN",
        "UTIL_RT3_BIN",
        "UTIL_RT4_BIN",
        "UTIL_RT5_BIN",
        "UTIL_RT6_BIN",
        "TOTAL_PAY_RT_BIN",
        "LIMIT_BAL_BIN"
)

cc_train = cc_train[,!(names(cc_train) %in% drop)]

cc_test <- credit_card_default[credit_card_default$test == 1,]
```

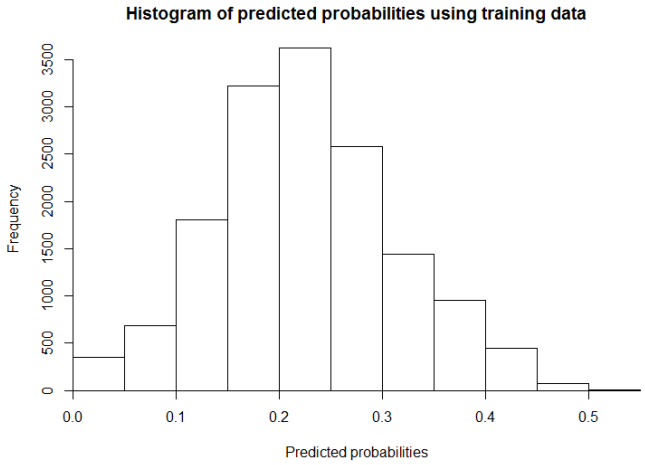
```
cc_vald <- credit_card_default[credit_card_default$validate == 1,]
```

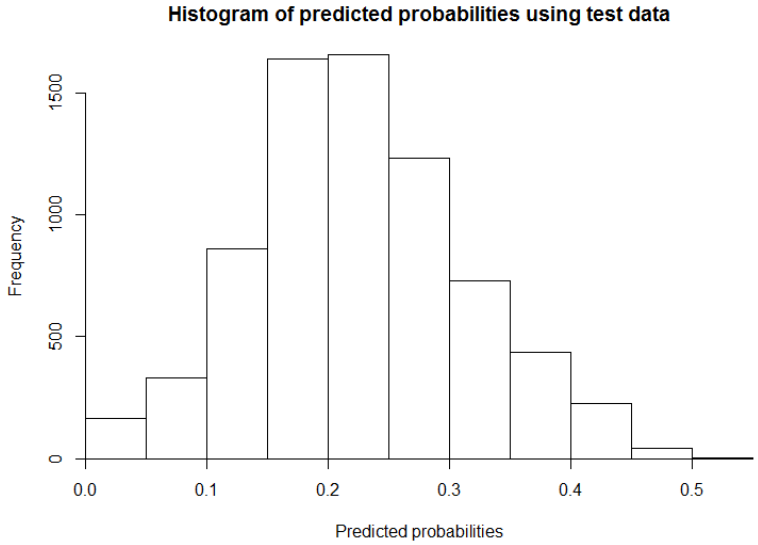
6.2 Production Model in R

```
model_logR <- glm(formula = DEFAULT ~ TOTAL_PAY_RT +
            PAY_RT1 +
            PAY_RT2 +
            PAY_RT3 +
            PAY_AMT1 +
            PAY_AMT2+
            PAY_AMT3 +
            LIMIT_BAL +
            AGE +
            UTIL_DIFF1 +
            UTIL_RT1 +
            UTIL_RT2 +
            UTIL_RT4 +
            UTIL_VEL1 +
            TOTAL_UTIL_VEL,
          data = cc_train ,
          family = binomial(link='logit'))

backwards = step(model_logR)
```

6.3 Predicted Probabilities using R

```
#Predicted probabilities - train
cc_logR_pred <- predict(backwards, cc_train, type = 'response')

#Predicted probabilities - test
cc_logR_pred_test <- predict(backwards, cc_test, type='response')

#validate
cc_logR_pred_vld <- predict(backwards, cc_vald, type='response')

hist(cc_logR_pred,
    main="Histogram of predicted probabilities using training data",
    xlab='Predicted probabilities')

hist(cc_logR_pred_test,
    main="Histogram of predicted probabilities using test data",
    xlab='Predicted probabilities')

hist(cc_logR_pred_vld,
    main="Histogram of predicted probabilities using validation data",
    xlab='Predicted probabilities')
```


Histogram of predicted probabilities using training data
Frequency
0
500
1000
1500
2000
2500
3000
3500
0.0
0.1
0.2
0.3
0.4
0.5
Predicted probabilities


Histogram of predicted probabilities using test data
Frequency
0
500
1000
1500
0.0
0.1
0.2
0.3
0.4
0.5
Predicted probabilities

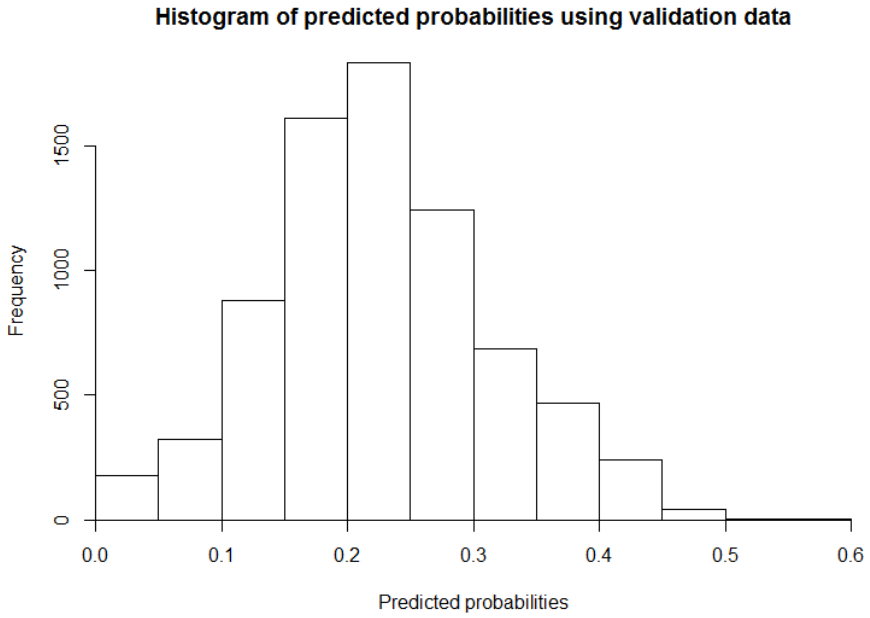


Figure 6.3.1: *Histograms of predicted probabilities using logistic regression*

6.4 ROC-AUC plots and calculations in R

```
library(ROCR)

#TRAIN
a<-list(as.numeric(cc_logR_pred), as.numeric(cc_train$DEFAULT))
pred <- prediction(a[1], a[2])
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc.perf)
abline(a=0, b= 1)

perf <- performance(prediction.obj, measure = "auc")
perf # look at y.values

#TEST
b<-list(as.numeric(cc_logR_pred_test), as.numeric(cc_test$DEFAULT))
pred <- prediction(b[1], b[2])
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc.perf)
abline(a=0, b= 1)

perf <- performance(prediction.obj, measure = "auc")
perf # look at y.values

#VALIDATE
```

```
c<-list(as.numeric(cc_logR_pred_vld), as.numeric(cc_vald$DEFAULT))
pred <- prediction(c[1], c[2])
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc.perf)
abline(a=0, b= 1)

perf <- performance(prediction.obj, measure = "auc")
perf # look at y.values
```

6.5 K-S calculations and lift charts in R

```
library(ROCR)
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc.perf)
abline(a=0, b= 1)

performance(pred, measure = "auc")
# look at y.values
# train 0.6558284
# test 0.6558284
#validate 0.6549111

roc.lift = performance(pred, measure = "lift")
plot(roc.lift)
#outputs in figures 2.2 and 2.3
```