**IDS 572 DATA MINING FOR BUSINESS**

**ASSIGNMENT 3 - FUNDRAISING 2**

**Kaushik Kompella**

**Jaideep Adusumelli**

**Sushaanth Srirangapathi**

**1. Modeling Partitioning - Partition the dataset into 60% training and 40% validation (set the seed to 12345). In the last assignment, you developed decision tree, logistic regression, naïve Bayes, random forest and boosted tree models. Now, develop support vector machine models for classification. Examine different parameter values, as you see suitable. Report on what you experimented with and what worked best.**

**How do you select the subset of variables to include in the model? What methods do you use to select variables that you feel should be included in the model(s)? Does variable selection make a difference? Provide a comparative evaluation of performance of your best models from all techniques (including those from part 1, i.e. assignment 2)** (Be sure NOT to include “TARGET−D” in your analysis.)

**Data Selection:** Missing Values

In order to focus our efforts on the set of variables that would provide the most meaningful information we decided to start by looking for variables with excessive amounts of missing values.

We saw an advantage in choosing to eliminate variables with >= 85% of all values missing. This helped us to eliminate a number of variables. We chose this route because we knew that including these variables with greater numbers of missing values would not be useful in predicting donor response, because there is very little information contained within those variables, well below even the low response rate of 5.1%.

Certain notable variables were:

|  |  |
| --- | --- |
| **Attributes Eliminated** | **Number of Missing values (out of 9999 values)** |
| MAILCODE | 9857 |
| PVASTATE | 9847 |
| RECP3 | 9728 |
| RECPGVG | 9983 |
| CHILD03 | 9884 |
| CHILD12 | 9840 |
| CHILD18 | 9746 |
| SOLP3 | 9976 |
| MAJOR | 9969 |
| HOMEE | 9902 |
| PHOTO | 9504 |
| KIDSTUFF | 9822 |
| CARDS | 9869 |
| PLATES | 9933 |
| RDATE\_3 | 9947 |
| RDATE\_5 | 9997 |
| RDATE\_6 | 9896 |
| RAMNT\_3 | 9947 |
| RAMNT\_4 | 9954 |
| RAMNT\_5 | 9997 |
| RAMNT\_6 | 9896 |
| RECSWEEP | 9826 |

These Variables did not even have enough number of values for the actual data which has only 5.1% response rate. So these variables even though contain very important information cannot contribute well towards creating a good model. Hence we eliminated all these variables.

**We did not eliminate those variables that were coded as X and null values. We transformed many of those attributes and will discuss those in the ‘transform’ section.**

**Data Selection:** Relevance

Next, we chose to focus on only including variables which would have relevance on predicting donation probability. We did this by scanning through the variable descriptions in the data dictionary to only retain those that were relevant.

An example of a variable that we chose to eliminate was DW2. This variable corresponded to the locale-specific percentage of single-unit detached structures. Eliminating these variable and other variables similar to this allowed us to lessen the total number of variables that we analyzed.

In addition to those variables that we found to be irrelevant, we noticed that there was a great deal of overlap between certain variables. For instance, the ‘Wealth1’ and ‘Wealth2’ variables had very similar descriptions in the data dictionary. However, we noticed that Wealth2 contained a more robust description of the variable –

“Wealth rating uses median family income and population statistics from each area to index relative

wealth within each state. The segments are denoted 0-9, with 9 being the highest income group and zero being the lowest. Each rating has a different meaning within each state”.

Whereas “Wealth1” simply contained the description: wealth rating. Since “Wealth1” has a similar description to “Wealth2” we know that there is a lot of information overlap, which allows us to justify dropping the less informative of the two: “Wealth1”.

We also found some variables which had no relationship with “TARGET\_B” but have a relationship with “TARGET\_D”. Some variables of this kind are:

**Data Reduction:** PCA

We experimented with creating many different sets of PCAs but found that in many cases we were including many variable sets that had numerous missing values, or had summary variables that were already existing. Therefore, we only created 3 PCAs that were valuable PCA\_CENSUS\_X, PCA\_EMP\_X and PCA\_INTERESTS\_X.

PCA\_CENSUS\_X included census variables for neighborhoods

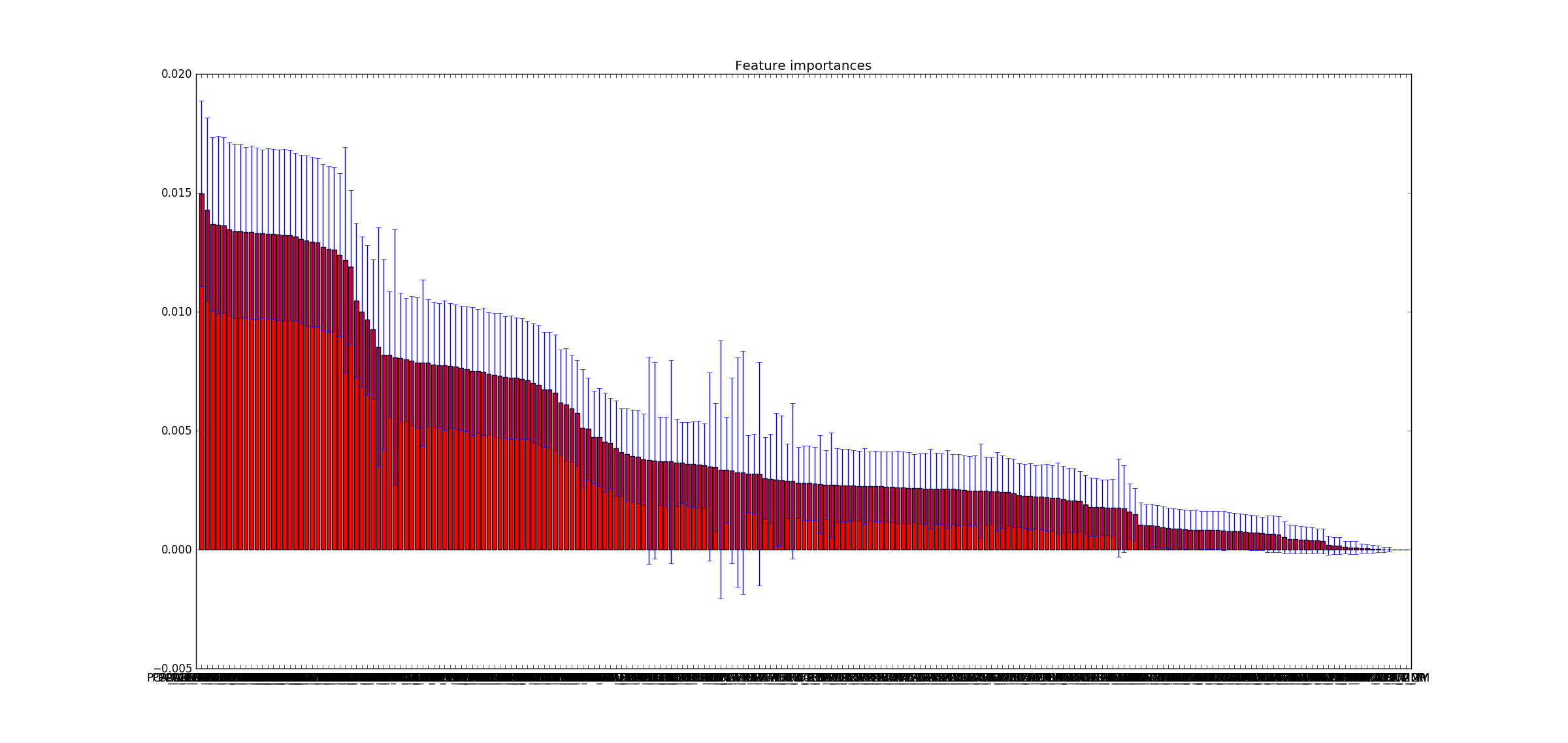
PCA\_EMP\_X included Employment information of the people i.e. Whether they are army or military veterans, state government employees or federal government employees, etc.

PCA\_INTERESTS\_X included the 18 ‘interest/hobby’ based variables.

**Data Reduction**: Random Forests

We ran random forests and plotted the variable importance graphs for all the variables for all the remaining attributes after the above processes were carried out. And then we selected all those variables which had the “Decrease in Gini Index” more than or equal to 0.0064 which gave us 68 variables. These included PCAs and mostly summary variables.

Most of the variables were not at all included in the split and we were able to eliminate them to improve the overall performance of our model.



**Data Selection:** Missing Value Replacement

We utilized two rapid miner operators in our process to ensure that those variables that had missing values below our cutoff were transformed. The two operators we used were the Replace Missing Values and the Map operators.

Replace Missing Values allowed us to impute the values when necessary while the Map operator allowed us to recode those variables that had encoded characters such as “MDMAUD”. This allowed us to code the extraneous characters as null values giving us a more accurate picture of which variables actually contained many more missing values than originally depicted in our first glance.

**Data Selection:** The Variables Included

CLUSTER DOMAINTYPE WEALTH1   
INCOME PEPSTRFL MAXRAMNT  
DOMAINSES HIT VIETVETS  
WWIIVETS IC1 CARDPROM  
MAXADATE CARDPM12 RAMNTALL   
NGIFTALL CARDGIFT MINRAMNT  
MINRDATE MAXRDATE LASTGIFT  
FISTDATE TIMELAG AVGGIFT  
CONTROLN TOTALDAYS RECENCYFREQ   
LOWINCOME MEDINCOME HIGHINCOME  
PCA\_NEIGH\_PER\_1-5 PCA\_NEIGH\_MMM\_1-4 PCA\_EMP\_1-4

**Decision Trees**

We chose the J48 version of the decision tree as we were able to obtain the most accurate results.

The parameter that allowed us to build the best model was **.4** as a **threshold**. We selected Laplace smoothing parameter for the J48 and chose an overall **confidence** of **.35**.

We created two main PCA’s which we removed from the model initially. The following confusion matrix is our J48 without these three PCAs:

Accuracy of Training Data set for the best tree: 94.97

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 3796 | 159 | 95.98 |
| Predicted Bad | 143 | 1901 | 93.00 |
| Class recall | 96.37 | 92.28 |  |

Accuracy of Test Data set for the best tree: 55.67

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1683 | 895 | 65.28 |
| Predicted Bad | 878 | 544 | 38.26 |
| Class recall | 65.72 | 37.80 |  |

Our optimal J48 model showed a **55.67**% accuracy on the overall test data. The class precision for true 1’s and predicted 1’s was **38.26**% and overall class recall was **37.80**%. These results were close to training data accuracy so it implies that we reached optimal model.

**Logistic Regression**

Once again, we ran the model with and without the PCAs that we generated for the J48 model. In this case we also found that the inclusion of PCA’s was hurting overall model performance. Below is the confusion matrix associated with logistic regression.

Accuracy of Training Data set for the best tree: 54.49

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1476 | 267 | 84.68 |
| Predicted Bad | 2463 | 1793 | 42.13 |
| Class recall | 37.47 | 87.04 |  |

Accuracy of Test Data set for the best tree: 48.08

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 814 | 330 | 71.15 |
| Predicted Bad | 1747 | 1109 | 38.83 |
| Class recall | 31.78 | 77.07 |  |

We then compared the training results from our Logistic Regression to the validation data and found our best LR model. The associated test data confusion matrix is seen above.

We could maintain an overall model accuracy of **48.08**%. The class precision from the test data was very similar to that of the training data. We obtained a **38.83**% precision with an overall class recall of **77.07**%.

We used default parameters for the Logistic regression as these gave us the highest overall accuracy.

**Naïve Bayes**

After running J48 and Logistic regression we attempted to find better model using Naïve Bayes. We applied same subset of variables both with and without PCAs to find best Naïve Bayes model. Our best model again did not contain any of the PCA’s but as with the other models, contained all of our created variables described in the initial data preprocessing steps.

Accuracy of Training Data set for the best tree: 63.68

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2888 | 1128 | 71.99 |
| Predicted Bad | 1051 | 932 | 47.00 |
| Class recall | 73.32 | 45.24 |  |

Accuracy of Test Data set for the best tree: 59.35

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1745 | 810 | 68.30 |
| Predicted Bad | 816 | 629 | 43.53 |
| Class recall | 68.14 | 43.71 |  |

The overall model accuracy is **59.35**% with a class precision of **43.53**% and a class recall of 4**3.71**%

We compared the accuracy of our Naïve Bayes model with the associated testing data. The resulting confusion matrix is as above.

In this case we see a rise in overall model accuracy on the validation data set. Also, we noticed similar class precision and class recall between our testing and training data suggesting we had found our best, and most stable Naïve Bayes model.

**Support Vector Machines**

Initially we tried to deploy SVM model on our data but were getting perfect over fit. We were using a kernel type radial based function on default settings. In addition, we changed the **threshold** to **.3, .5, and .65** but were still getting massive over fit on our training data.

Below is the confusion matrix associated with the SVM training data:

Accuracy of Training Data set for the best tree: 59.21

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 3064 | 1572 | 66.09 |
| Predicted Bad | 875 | 488 | 35.80 |
| Class recall | 77.79 | 23.69 |  |

Accuracy of Test Data set for the best tree: 59.55

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2048 | 1105 | 64.95 |
| Predicted Bad | 513 | 334 | 39.43 |
| Class recall | 79.97 | 23.21 |  |

After analyzing many different kernel types such as- polynomial, dot, and Gaussian combination. We also experimented with the C penalty parameters and determined that the support vector machine wasn’t the best suited algorithm to our data and did not generalize well.

We could generate an **overall accuracy** of **59.21**% on this model, but with a lower **class precision** of **35.8**%. The **class recall** was also worse than all other models with a **23.69**% recall on true 1’s/predicted 1’s.

Our test data had an **overall accuracy** of **59.55**% but still with very low **class precision** of **39.43**% and **class recall** of **23.21**%.

**KNN**

We used the following parameters to generate our best KNN model:

K = 150; Measure: Mixed, Mixed-Measure: Mixed Eucledian

This included using a k-value of **150**, with threshold of **.33** along with mixed measure of Euclidean as a distance metric.

Below is the confusion matrix of our training data for KNN:

Accuracy of Training Data set for the best tree: 54.53

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2004 | 793 | 71.65 |
| Predicted Bad | 1935 | 1267 | 39.57 |
| Class recall | 50.88 | 61.50 |  |

Accuracy of Test Data set for the best tree: 50.48

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1151 | 571 | 66.84 |
| Predicted Bad | 1410 | 868 | 38.10 |
| Class recall | 44.94 | 60.32 |  |

Using the parameters above we obtained an overall accuracy of **54.3**% on the training data. The class precision was lower than that of other models at **39.57**%. The class recall for predicted and true 1’s was **61.5**% overall.

Our testing confusion matrix gave an accuracy of **50.48**%, with a class precision of **38.1**%, but an overall class recall of **60.32**% on predicted 1’s and true 1’s.

**Random Forest**

Initially we used the same set of attributes that we deployed in our SVM model and were only able to generate a ~30% Test accuracy but with a ~80% Training data accuracy. We allowed too many trees to build and as a result this model was over fit.

By limiting depth of the trees, we could generate a much higher accuracy on test data. Below is our Random Forest confusion matrix for our training data:

Accuracy of Training Data set for the best tree: 60.06

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2101 | 558 | 79.01 |
| Predicted Bad | 1838 | 1502 | 44.97 |
| Class recall | 53.34 | 72.91 |  |

Accuracy of Test Data set for the best tree: 53.80

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1212 | 499 | 70.84 |
| Predicted Bad | 1349 | 940 | 41.07 |
| Class recall | 47.33 | 65.32 |  |

We obtained an accuracy of **60.06**% overall on training data, but with a class precision of **44.97**% and a class recall **72.91**% for predicted and true 1’s

On testing data set we obtained a reasonably close accuracy of **53.8**%. The class precision was **41.07**% with a class recall close to that of the training data at **65.32**%.

**Gradient Boosted Trees:**

Accuracy of Training Data set for the best tree: 72.03

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2706 | 445 | 85.88 |
| Predicted Bad | 1233 | 1615 | 56.71 |
| Class recall | 68.70 | 78.40 |  |

Accuracy of Test Data set for the best tree: 55.62

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1428 | 642 | 68.99 |
| Predicted Bad | 1133 | 797 | 41.30 |
| Class recall | 55.76 | 55.39 |  |

The parameters used were:

Number of trees: 40, Maximal Depth: 5, Minimum Rows: 20, Minimum Split Improvement: 0.0025, Number of Bins: 25, Learning Rate: 0.1, Sample Rate: 1, Distribution: Auto

From the above matrix we found that gradient boosted trees were the best model we could get with optimal accuracy and precision and high recall which would help us to maximize the profits.

**2. Our overall goal is to identify which individuals to target for maximum donations (profit). We will try two approaches for this:**

**(i) using the response model, together with average donation and mailing costs information, to identify the most profitable individuals to target (Q 2.1 below)**

**(ii) develop a second model on TARGET\_D, and combine this with the response model to identify the most profitable individuals to target (Q 2.2 below)**

**2.1 (a) What is the ‘best’ model for each method in Question 1 for maximizing revenue? Calculate the net profit for both the training and validation set based on the actual response rate (5.1%). We can calculate the net profit from given information - the expected donation, given that they are donors, is $13.00, and the total cost of each mailing is $0.68. Note: to calculate estimated net profit (on data with the ‘natural’ response rate of 5.1%), we will need to “undo” the effects of the weighted sampling, and calculate the net profit that reflects the actual response distribution of 5.1% donors and 94.9% non-donors.)**

After choosing the best models from the above methods, we could determine that our best model was the **Gradient Boosted Trees.** This is not just based on the accuracy itself but the net profit we got.

We needed to undo the effects of the weighted sampling which we did in the following way:

Firstly, adjust the weighted profit and the weighted cost.

**Weighted profit**: 13-$0.68 = **$12.32**

**Weighted Cost** = ($0.68)

After this, we used the following formula to calculate maximum profit:

**(12.32\*.051)/(3499/9999) = 1.795**

Next, we used the following formula to calculate the adjusted cost:

**(($0.68)\*0.9491)/(6500/9999) = .992**

The lift we achieved for the logistic regression was **306.679** as max profit.

**(b) Summarize the performance of the ‘best’ model from each method, in terms of net profit from predicting donors in the validation dataset; at what cutoff is the best performance obtained?**

**Draw profit curves: Draw each model’s net cumulative profit curve for the validation set onto a single graph. Are there any models that dominate?**

**Best Model: From your answers above, what do you think will be the “best” model to implement? (What criteria do you use to determine ‘best’?)**

Our best model is Gradient Boosted Trees. We used max lift criteria but also considered the highest class recall (**55.39**%) and class precision (**41.30**%). Our next best model (“Logistic Regression”) has a class precision of **38.83**%, and a class recall of **77.07**%, but a higher lift curve. It does not make sense from a business standpoint to evaluate our best model on the criteria of lift alone because then the J48 wouldn’t generalize well to predict donors accurately.

The performance of all models are as follows:

|  |  |
| --- | --- |
| **Model** | **Profit (Performance)** |
| J-48 | 105.5 $ |
| Logistic | 257.63 $ |
| Naïve Bayes | 319.58 $ |
| Random Forests | 349.10 $ |
| KNN | 159.34 $ |
| SVM | 90.63 $ |

The cutoff we used in Gradient Boosted Trees was a confidence of: **0.496901**.

The best model that was J-48 according to the lift chart. However, Gradient Boosted Trees had the best chance to generalize well on unseen data.

**2.2. (a) We will next develop a model for the donated amount (TARGET\_D). Note that TARGET\_D has values only for those individuals who donors (that is, TARGET\_D values are defined only for cases where TARGET\_B is 1). What data will you use to develop a model for TARGET\_D? (Non-donors, obviously, do not have any donation amount -- should you consider these as $0.0 donation, or impute missing values here? Should non-donors be included for developing the model to predict donation amount? Also, should cases with rare very large donation amounts be excluded? [Hah! – leading questions]**

**Develop a model for TARGET\_D. What modeling method do you use (report on any one). Which variables do you use? What variable selection methods do you use? Report on performance.**

For the cases with rare very large donation amounts, we will not exclude these cases as we will lose out on predicting our most valuable donors. And these donations might be large because of a supportive reason from the values of other variables. Hence we did not want to lose Important Information.

We will use linear Regression model for predicted TARGET\_D with a split validation of 60-40 on the data set.

**Variable selection methods**

After applying the data cleaning and missing value treatment as applied on dataset for TARGET\_B, we looked at the scatter plots of the independent variables against TARGET\_D and eliminated those variables that did not show any significant trend and/or relationship with TARGET\_D. Given below are a few variables that we have selected for regression modelling for Target-D and their scatter plots with Target D.

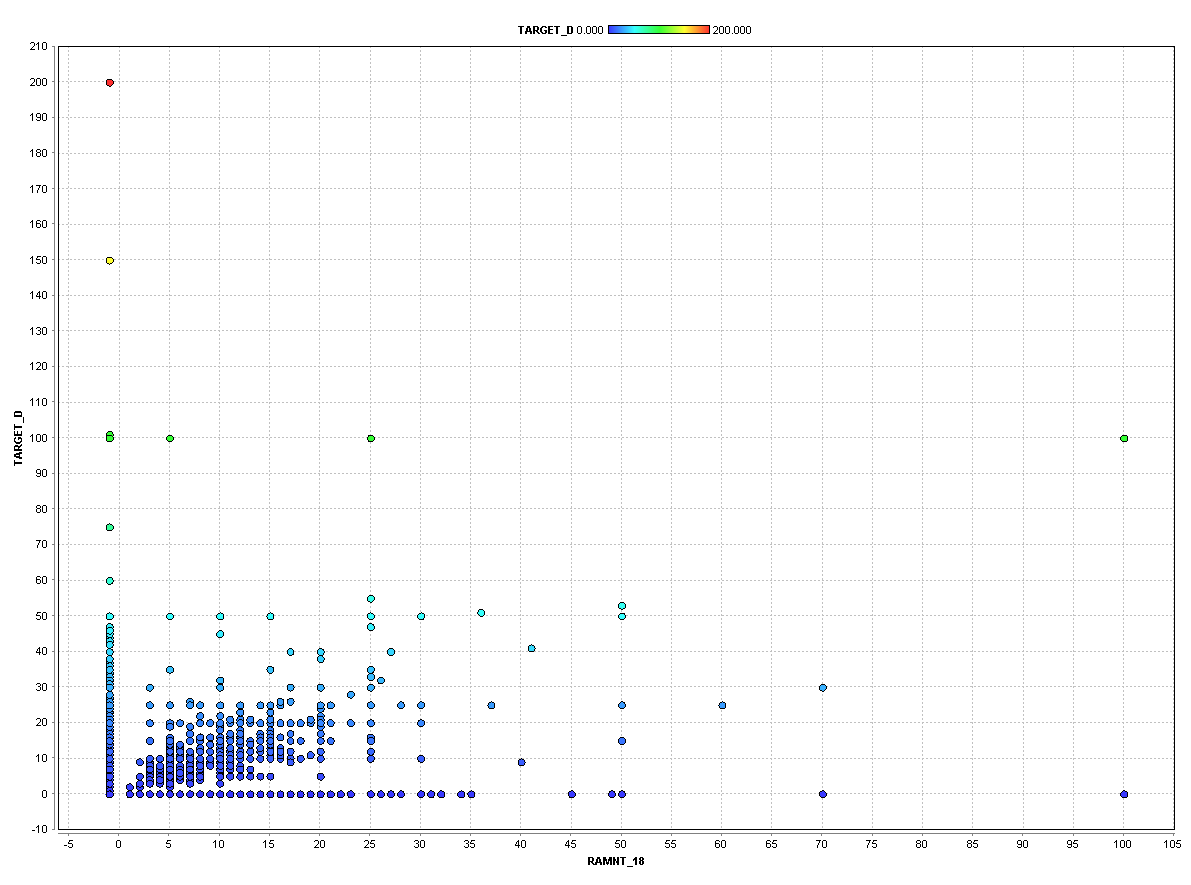


Fig.1 RAMNT\_18 vs TARGET\_D

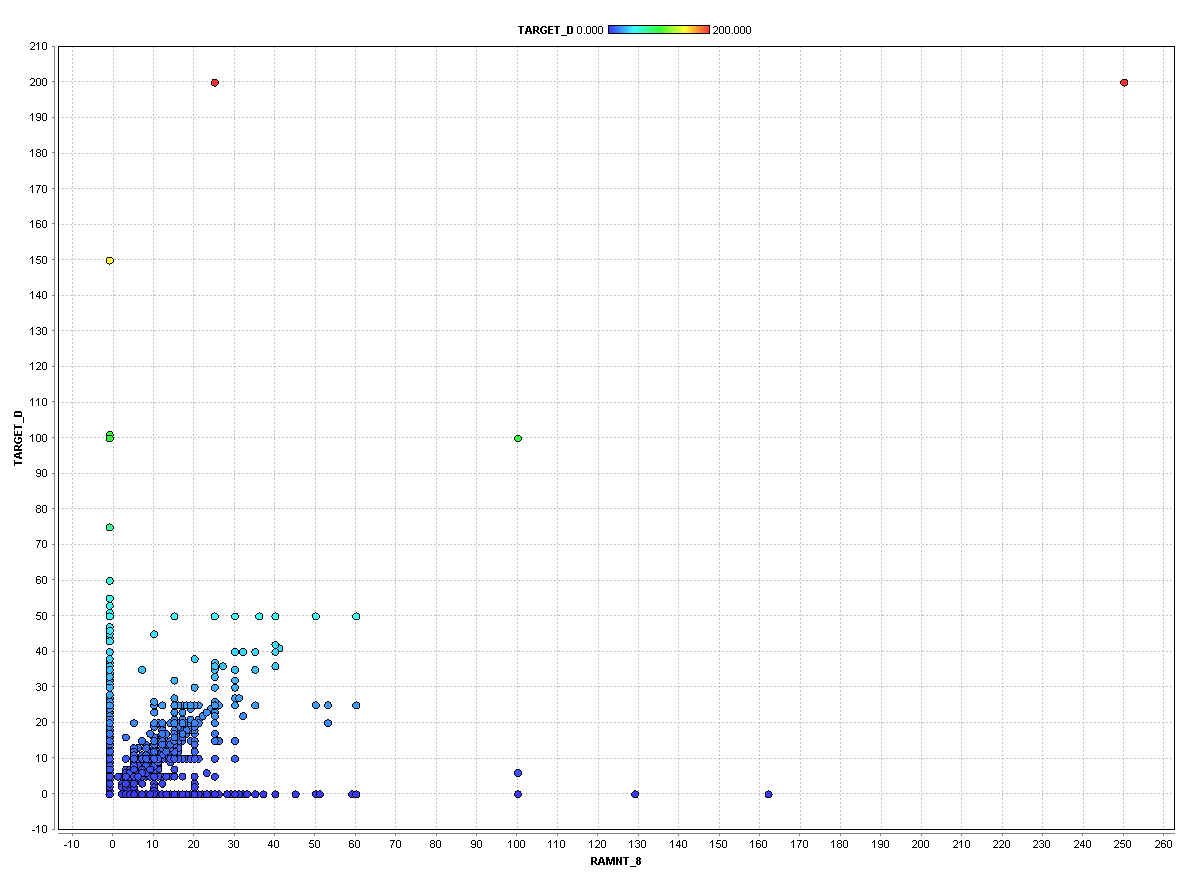


Fig.2 RAMNT\_8 vs TARGET\_D

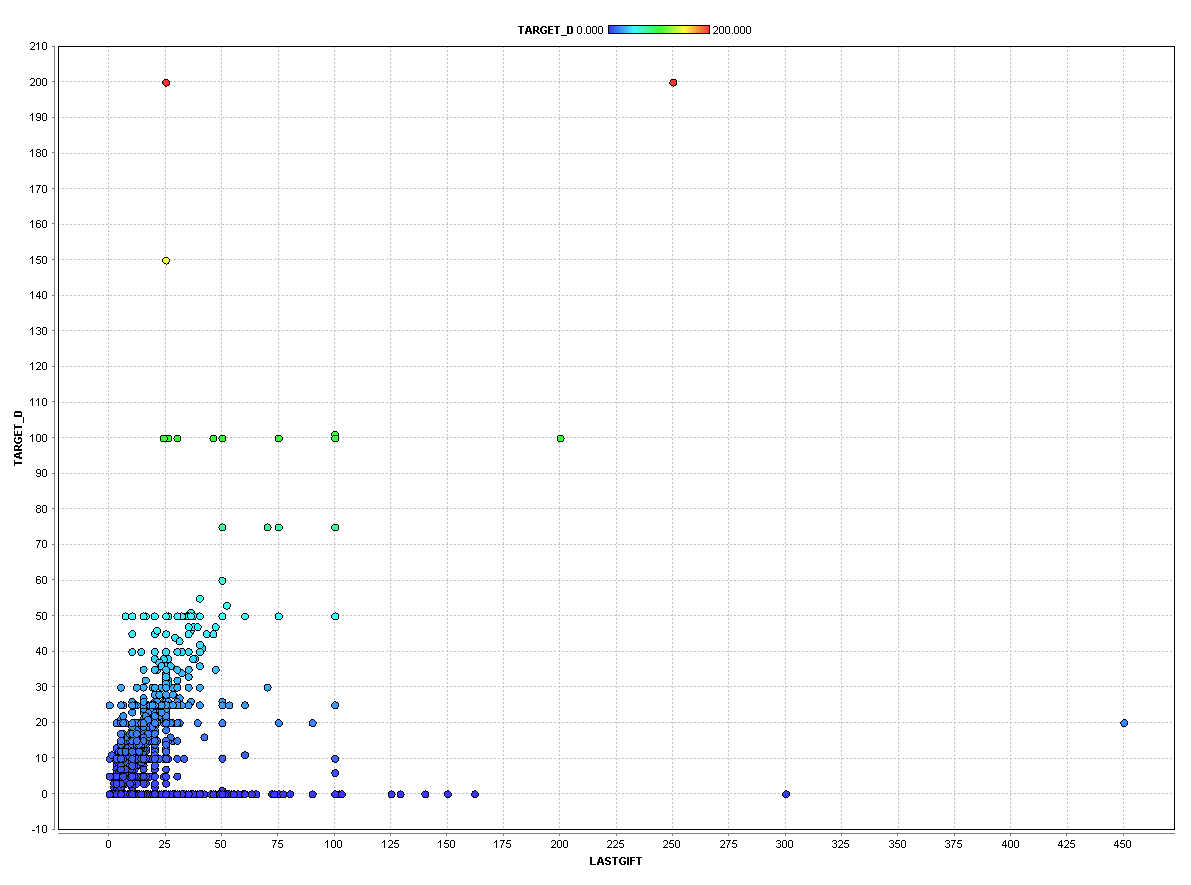


Fig.3 LASTGIFT vs TARGET\_D

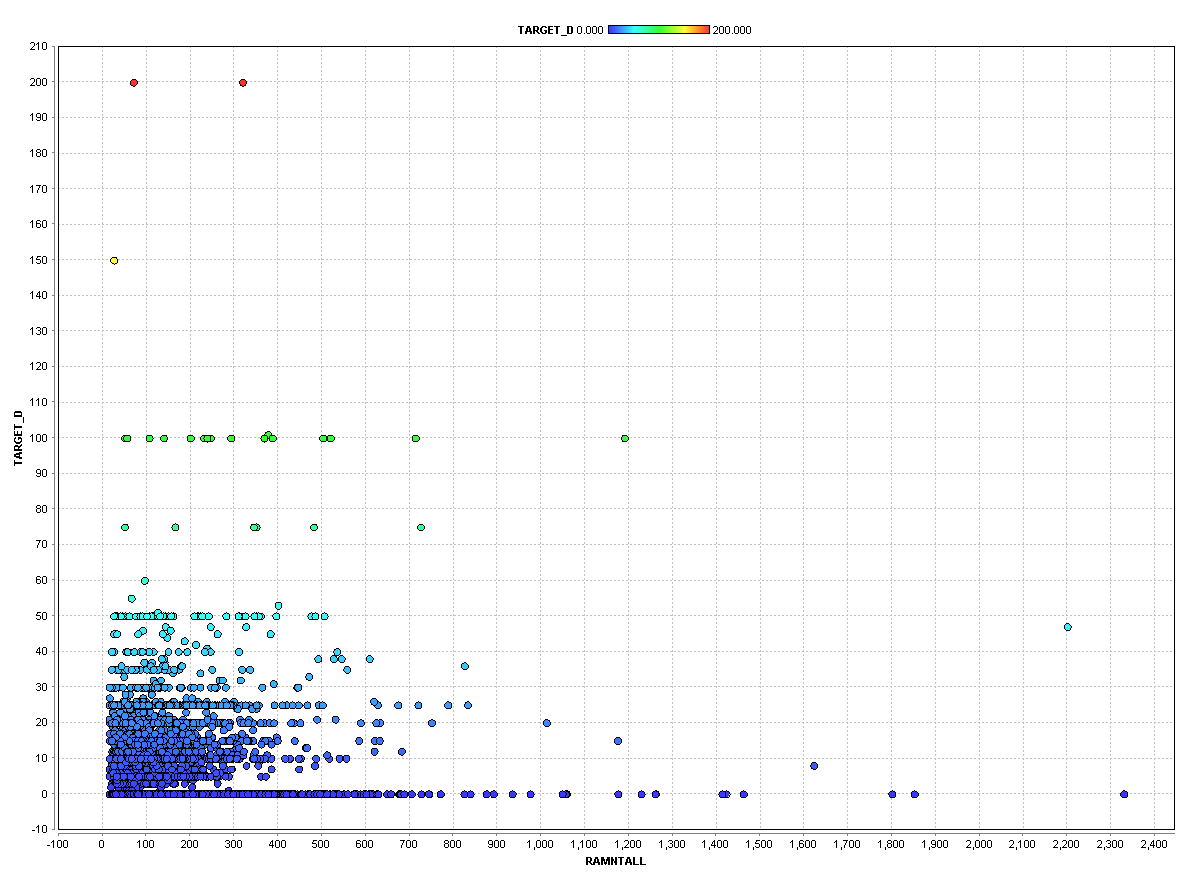


Fig.4 RAMNTALL vs TARGET\_D

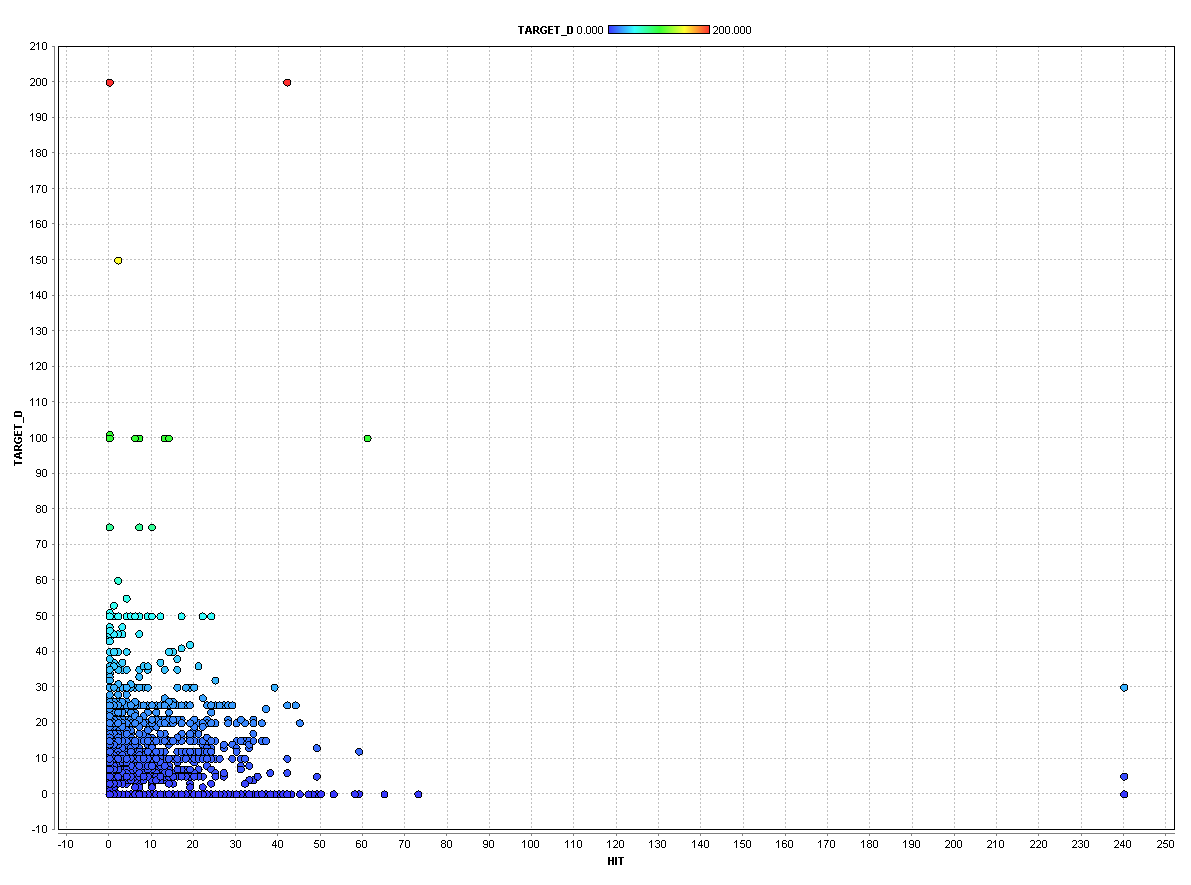


Fig.5 HIT vs TARGET\_D

As can be seen from the plots, the above variables had a significant relationship with TARGET\_D, hence we selected these variables for the linear regression modelling for the prediction of TARGET\_D Variable.

Given below are the scatter plots of variable with TARGET\_D for the ones which we eliminated.

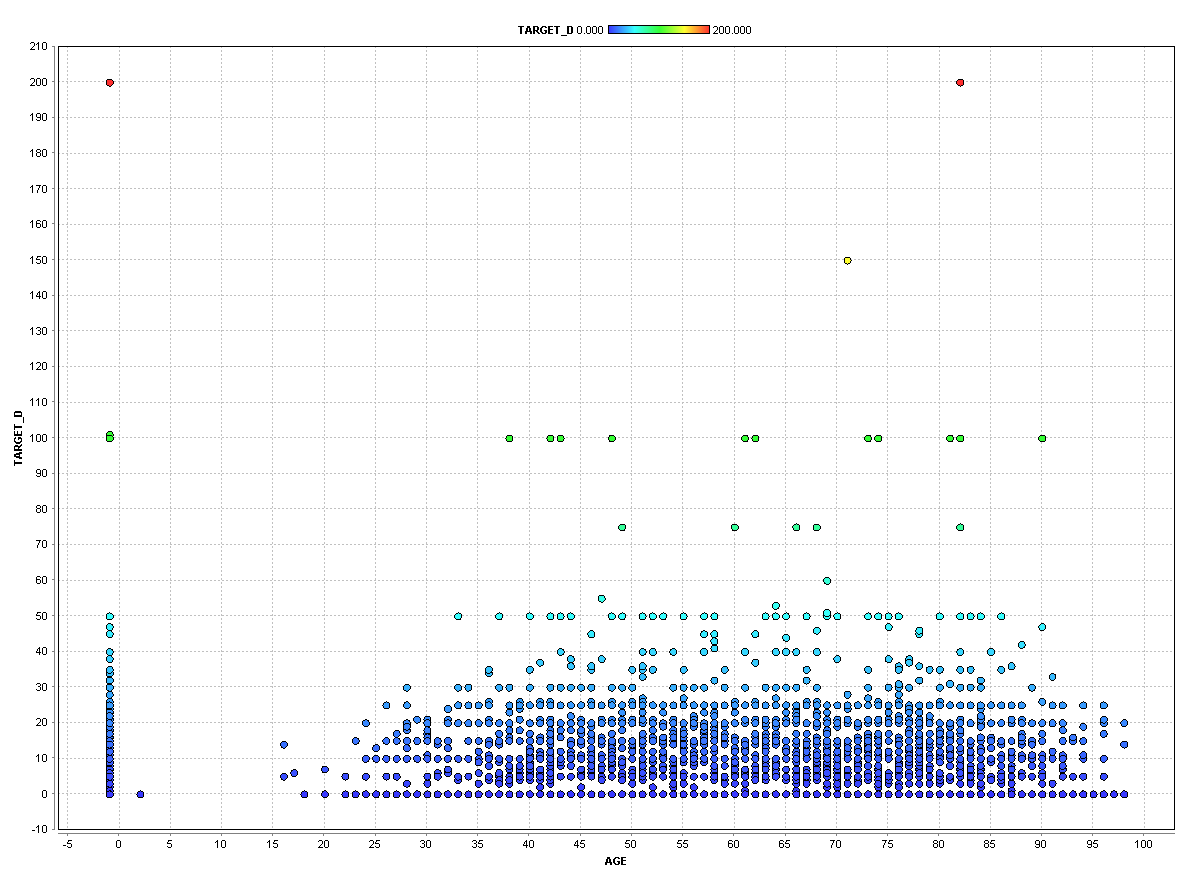


Fig.1 AGE vs TARGET\_D

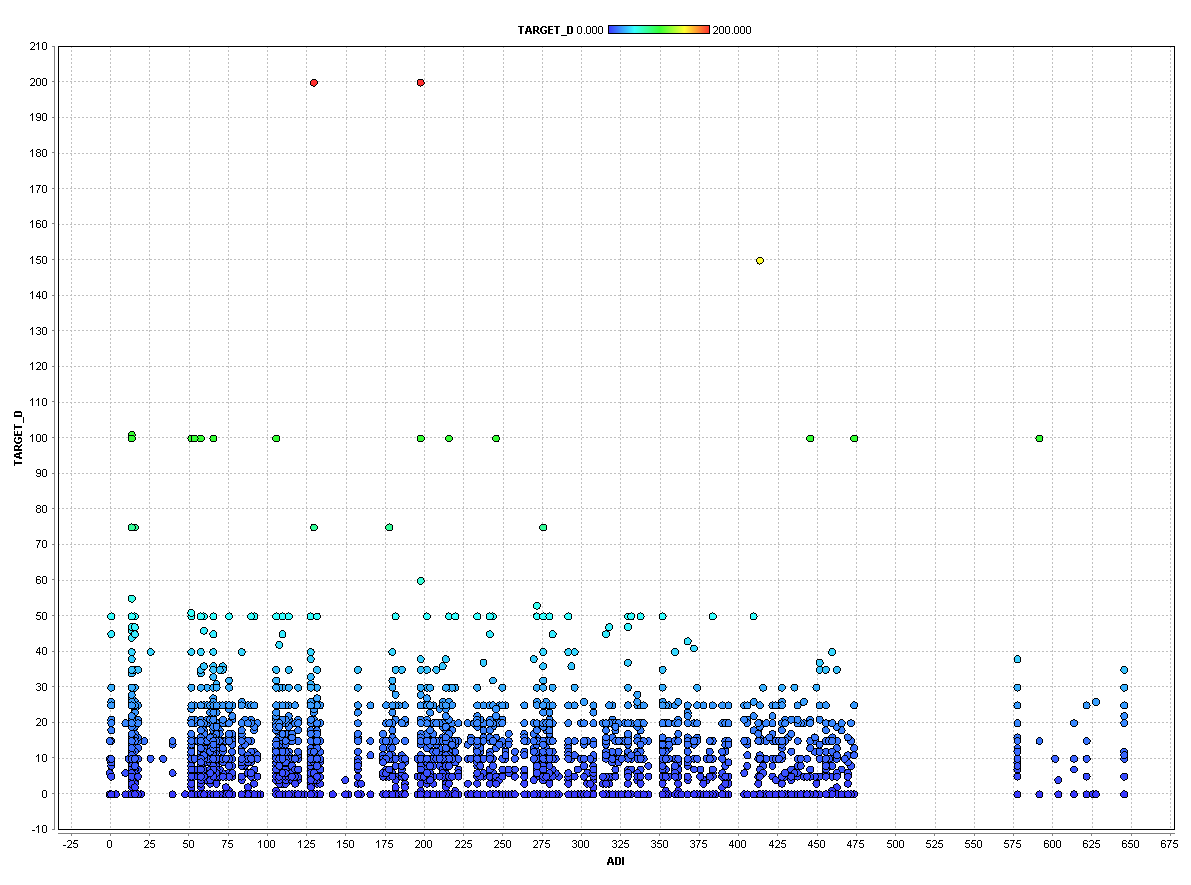


Fig.2 ADI vs TARGET\_D

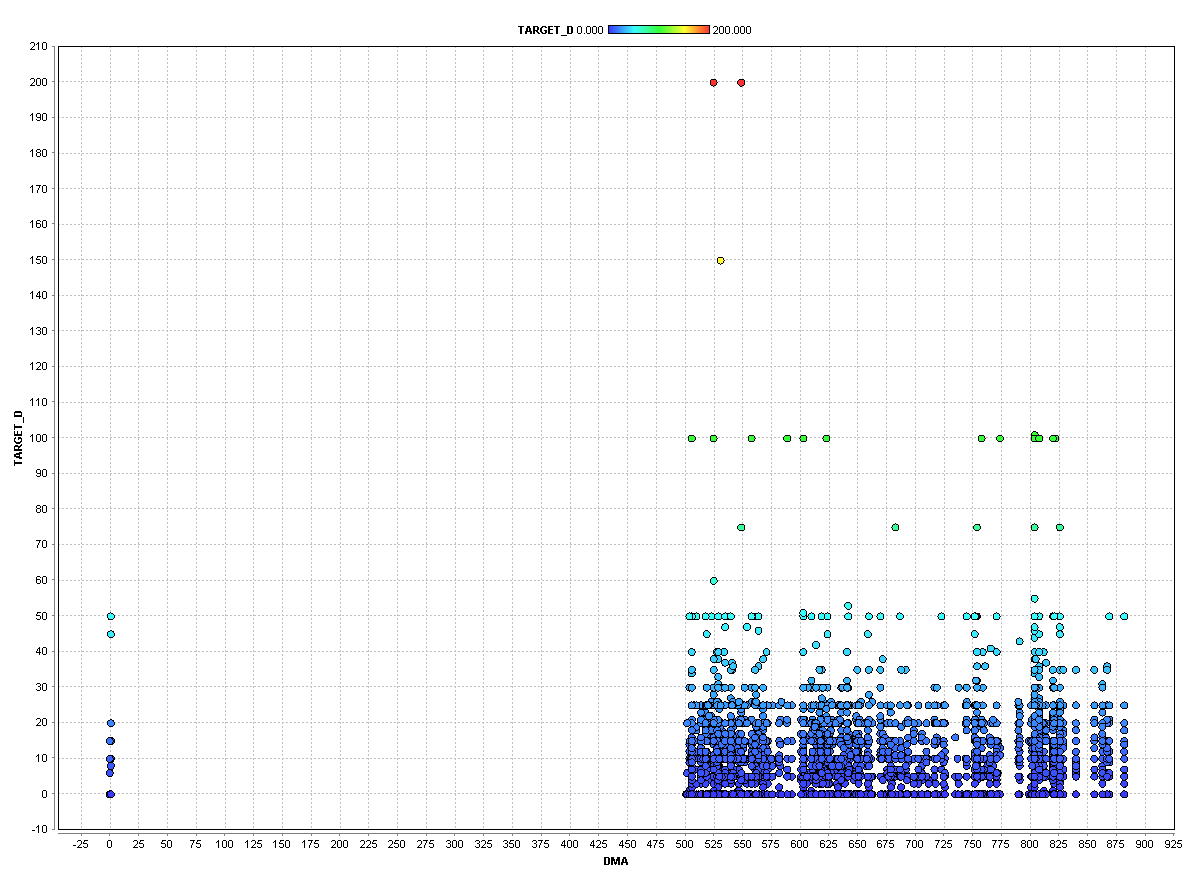


Fig.3 DMA vs TARGET\_D

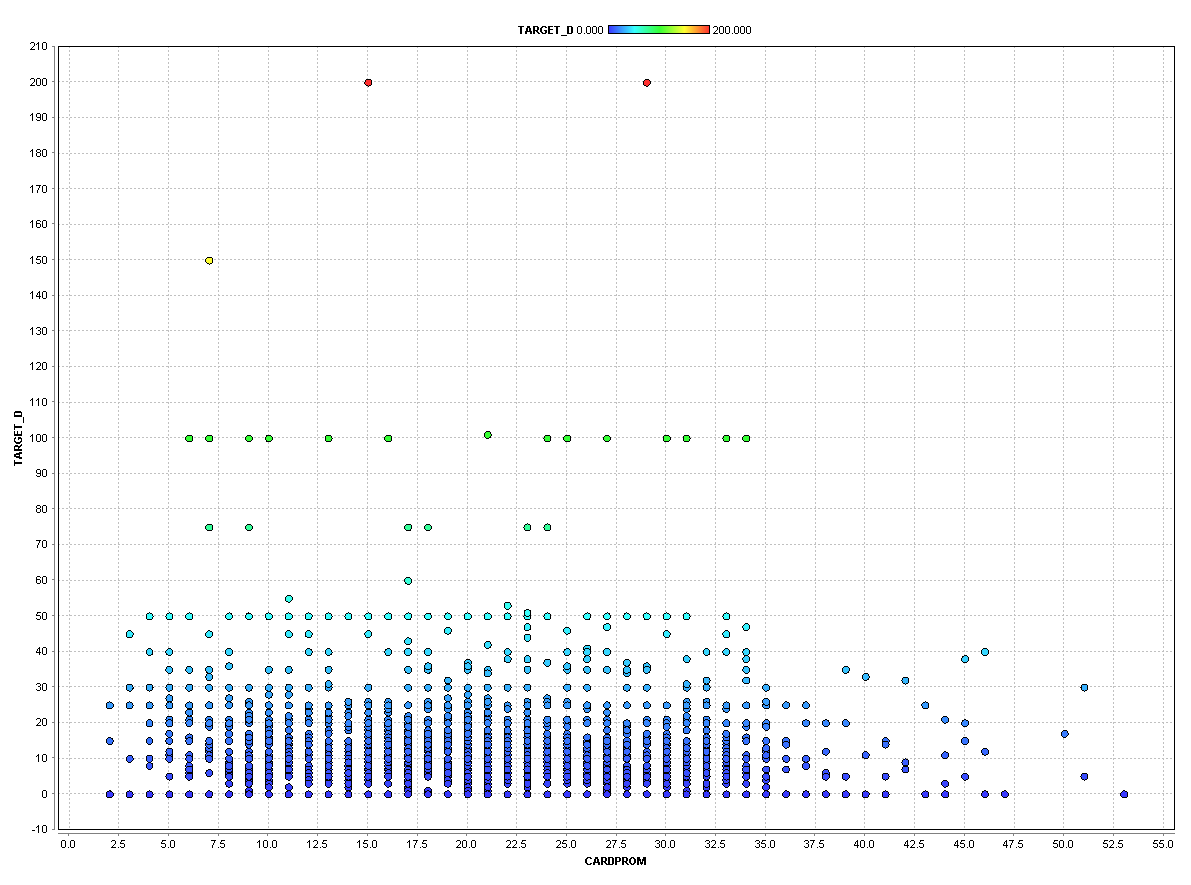
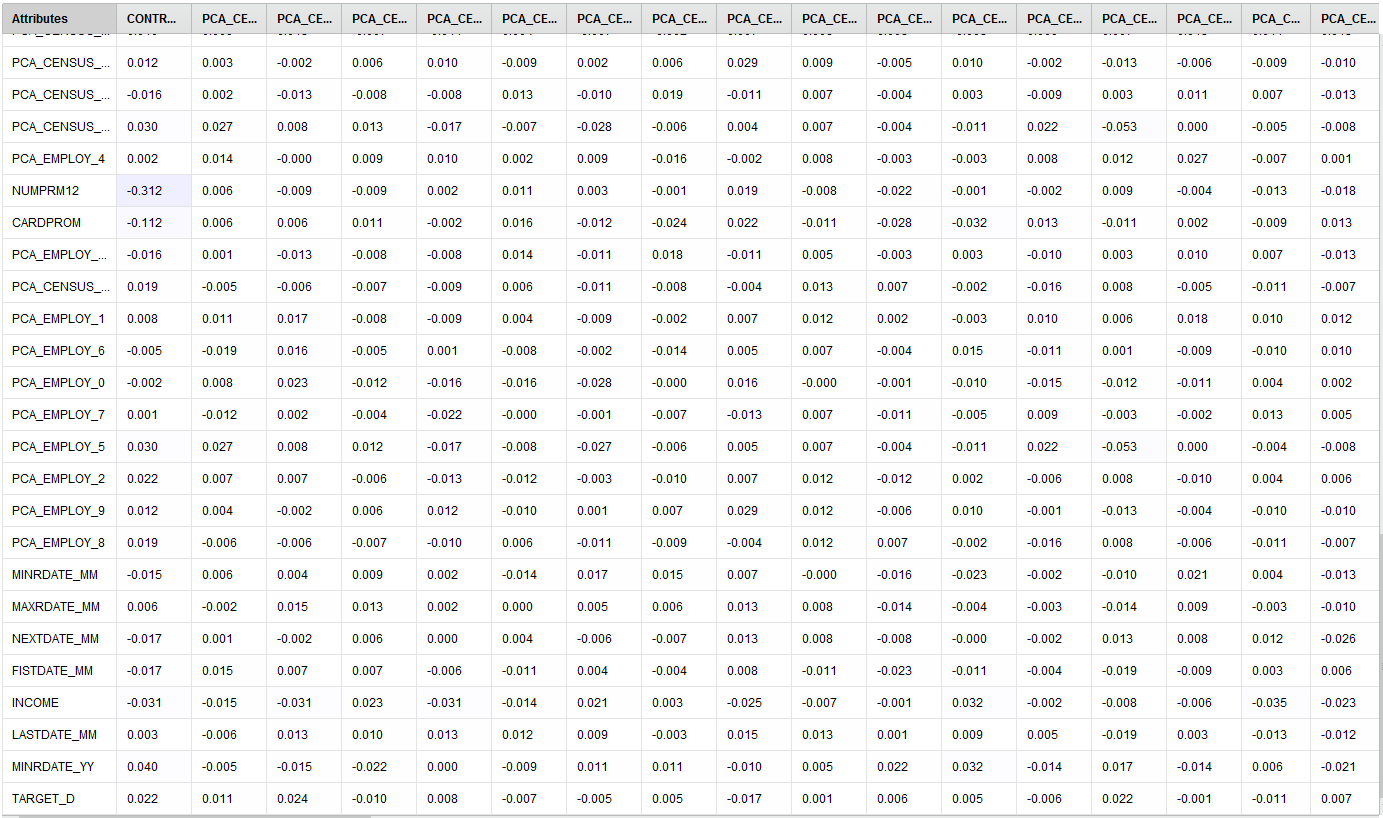


Fig.4 CARDPROM vs TARGET\_D

These variables didn’t show any particular trend with the TARGET\_D variable hence were eliminated.

After Scatter plots, we narrowed down the list of variables from 481 to 261. And then we used correlation matrix to finally select variables for modelling.



The highest correlation (R) values of independent variables with TARGET\_D was of 0.2 from LASTGIFT followed by these variables - NGIFTALL, AVGGIFT, RAMNTALL, MINRAMNT. We choose all the variables having >= 0.05 as R value with TARGET\_D. We finalized a list of 15 independent variables in our dataset to run the model. Here is the list of variables that we used in our best linear regression model through the T-Test feature selection.

The variable we got were:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HIT | WEALTH\_1 | NUMPROM | RAMNT\_8 | RAMNT\_12 |
| RAMNT\_14 | RAMNT\_16 | RAMNT\_18 | RAMNT\_22 | RAMNTALL |
| NGIFTALL | MINRAMNT | MAXRAMNT | LATGIFT | AVGGIFT |

Performance-

Our best model has given the following values for Root mean squared error and squared correlation

Root-Mean Squared error: 9.949 +/- 0.000

**(b) How can you use the results from the response model together with results from the donation amount model to identify targets? (Hint: The response model estimates the probability of response p(y=1|x). The donation amount model estimates the conditional donation amount, E [amt | x, y=1]. The product of these gives …..? )**

**How do you identify individuals to target, when combining information from both models? [Sorted values of predicted donation amount? What threshold would you use? Or maybe you prefer to target all individuals with predicted amounts greater than $0.68 (why?), and/or…..]**

Given the confidence probabilities of donors (TARGET\_B=1) from the classification model and the predicted donation amounts (TARGET\_D) from the regression model we multiply the two values to combine the two models. The multiplication of the **confidence (1)** from the classification model and the probable donation from the donor predicted by the linear regression model gives us the most probable donation by the responder.

Hence with the classification model we only predicted how many individuals are going to respond to our promotion, on the other hand the regression model gives us the probable donation amount by the responders obtained from the classification model. Hence combining the two models plays a very important role.

The modelling technique we used is Gradient Boosted Trees, which has an accuracy of approximately **50%,** hence we have selected all those individuals who are donating **2$** or more (as the accuracy is 50%, we assume that out of all the predicted responders out every 2 one would surely respond, hence we want that one individual to cover our promotional charges which was downscaled to **-0.9928 $** for two individuals i.e. **1.9856 $.**

Hence, in order to cover our expenses, we would need individuals who would donate a minimum amount of **~2 $.**

**5. Testing – chose one model, either the one from 2.1 or 2.2 above, based on performance on the test data.**

**The file FutureFundraising.xls contains the attributes for future mailing candidates. Using your “best”**

**model from Step 2 which of these candidates do you predict as donors and non-donors? List them in descending order of probability of being a donor/prediction of donation amount. What cutoff do you use? Submit this file (xls format), with your best model’s predictions (prob of being a donor).**

For testing the FutureFunraising.xls file, the best model which we have used is **Gradient Boosted Trees.**

After applying the model, the below are the predicted information inferred

**Number of Donors : 10857  
Number of Non-Donors : 9143**

**Cumulative Profit : ~24604 $**

**The cut off used to predict donor/non-donors is the confidence of predicting 1s = 0.496901**

**Note:**The list of predictions using the best model is uploaded to the blackboard. (Future\_Data\_Prediction.xls).

**Appendix-1**

Python code used for cleaning the data-

import numpy as np

import pandas as pd

drop\_column = ["ODATEDW", "OSOURCE", "TCODE", "ZIP", "MAILCODE", "PVASTATE", "DOB", "NOEXCH", "CLUSTER",

"AGEFLAG", "NUMCHLD", "HIT", "DATASRCE", "MALEMILI", "MALEVET", "VIETVETS", "WWIIVETS", "LOCALGOV",

"STATEGOV", "FEDGOV", "GEOCODE", "HHP1", "HHP2", "DW1", "DW2", "DW3", "DW4", "DW5", "DW6", "DW7", "DW8",

"DW9",

"HV1", "HV2", "HV3", "HV4", "HU1", "HU2", "HU3", "HU4", "HU5", "HHD1", "HHD2", "HHD3", "HHD4", "HHD5",

"HHD6",

"HHD7", "HHD1", "HHD2", "HHD3", "HHD4", "HHD5", "HHD6", "HHD7", "HHD8", "HHD9", "HHD10", "HHD11",

"HHD12", "HUR1",

"HUR2", "RHP1", "RHP2", "RHP3", "RHP4", "HUPA1", "HUPA2", "HUPA3", "HUPA4", "HUPA5", "HUPA6", "HUPA7",

"RP1",

"RP2", "RP3", "RP4", "MSA", "ADI", "DMA", "MC1", "MC2", "MC3", "TPE1", "TPE2", "TPE3", "TPE4", "TPE5",

"TPE6", "TPE7",

"TPE8", "TPE9", "PEC1", "PEC2", "TPE10", "TPE11", "TPE12", "TPE13", "ANC1", "ANC2", "ANC3", "ANC4",

"ANC5", "ANC6",

"ANC7", "ANC8", "ANC9", "ANC10", "ANC11", "ANC12", "ANC13", "ANC14", "ANC15", "POBC1", "POBC2", "LSC1",

"LSC2",

"LSC3", "LSC4", "VOC1", "VOC2", "VOC3", "ADATE\_2", "ADATE\_3", "ADATE\_4", "ADATE\_5", "ADATE\_6", "ADATE\_7",

"ADATE\_8",

"ADATE\_9", "ADATE\_10", "ADATE\_11", "ADATE\_12", "ADATE\_13", "ADATE\_14", "ADATE\_15", "ADATE\_16",

"ADATE\_17",

"ADATE\_18", "ADATE\_19", "ADATE\_20", "ADATE\_21", "ADATE\_22", "ADATE\_23", "ADATE\_24", "MAXADATE",

"RDATE\_3",

"RDATE\_4", "RDATE\_5", "RDATE\_6", "RDATE\_7", "RDATE\_8", "RDATE\_9", "RDATE\_10", "RDATE\_11", "RDATE\_12",

"RDATE\_13",

"RDATE\_14", "RDATE\_14", "RDATE\_15", "RDATE\_16", "RDATE\_17", "RDATE\_18", "RDATE\_19", "RDATE\_20",

"RDATE\_21",

"RDATE\_22", "RDATE\_23", "RDATE\_24", "MINRDATE", "MAXRDATE", "LASTDATE", "FISTDATE", "NEXTDATE",

"CONTROLN",

"TARGET\_D", "HPHONE\_D", "RFA\_2R", "RFA\_2F", "RFA\_2A", "MDMAUD\_R", "MDMAUD\_F", "MDMAUD\_A", "CLUSTER2",

"GEOCODE2", "MDMAUD"]

df = pd.read\_csv("C:\\Users\\tyrion\\Documents\\IDS\_572\_notes\\assign2\\pvaBal35Trg.csv", sep=',', na\_values=[' '],

low\_memory=False)

df.drop(drop\_column, axis=1, inplace=True)

list\_string = []

non\_list\_string = []

# filling numeric columns with -1

for c in df.columns:

li = df[c].values.tolist()

a = np.asarray(li)

# print type(a)

flag = 0

for x in np.nditer(a):

si = x.tolist()

if x != "nan":

if type(si) != str:

flag = 1

break;

# filling NAN for all numeric entries

if flag == 1:

df[c].fillna(-1, inplace=True)

# print "was in"

# print c

else:

if df[c].isnull().values.any():

# print type(df[c])

list\_string.append(c)

else:

non\_list\_string.append(c)

# print c

# replacing columns having "X" to 1 and "NaN" to 0

for val in list\_string:

str\_l = df[val].values.tolist()

a = np.asarray(str\_l)

# print type(a)

flag = 0

for x in np.nditer(a):

if x == "X":

flag = 1

break

if x == "M":

flag = 2

break

if x == "Y":

flag = 3

break

if flag == 1:

# print val

df[val] = df[val].replace({'X': 1}, regex=False)

df[val].fillna(0, inplace=True)

if flag == 2:

df[val].fillna(-1, inplace=True)

df[val] = df[val].replace({'M': 1}, regex=False)

df[val] = df[val].replace({'F': 0}, regex=False)

if flag == 3:

df[val] = df[val].replace({'Y': 1}, regex=False)

df[val].fillna(-1, inplace=True)

if val == "HOMEOWNR":

df[val].fillna(0, inplace=True)

df[val] = df[val].replace({'H': 1}, regex=False)

new\_attri = []

for val in list\_string:

if val.find("RFA",0) == 0:

df[val].fillna("Z5Z", inplace=True)

r = val + "\_R"

f = val + "\_F"

c = val + "\_C"

df[r] = df[val].str.extract('([FNALISZ])',expand = True)

df[f] = df[val].str.extract('(\d)', expand=True)

df[c] = df[val].str.extract('[a-zA-Z][\d]([a-zA-Z])', expand=True)

"""for che in range(1,9999,1):

if df[val].iloc[che] == "Z5Z": print che

print df[f].iloc[128]

"""

df[r] = df[r].replace({'F': 0,'N': 1,'A': 2,'L': 3,'I': 4,'S': 5,'Z': 6}, regex=False)

df[c] = df[c].replace({'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6,'Z': 7}, regex=False)

new\_attri.append(r)

new\_attri.append(f)

new\_attri.append(c)

del df[val]

val = "RFA\_2"

df[val].fillna("Z5Z", inplace=True)

r = val + "\_R"

f = val + "\_F"

c = val + "\_C"

df[r] = df[val].str.extract('([FNALISZ])',expand = True)

df[f] = df[val].str.extract('(\d)', expand=True)

df[c] = df[val].str.extract('[a-zA-Z][\d]([a-zA-Z])', expand=True)

"""for che in range(1,9999,1):

if df[val].iloc[che] == "Z5Z": print che

print df[f].iloc[128]

"""

df[r] = df[r].replace({'F': 0,'N': 1,'A': 2,'L': 3,'I': 4,'S': 5,'Z': 6}, regex=False)

df[c] = df[c].replace({'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6,'Z': 7}, regex=False)

new\_attri.append(r)

new\_attri.append(f)

new\_attri.append(c)

del df[val]

val = "DOMAIN"

df[val].fillna("Z5", inplace=True)

domain\_att\_1 = val + '\_urban\_level'

domain\_att\_2 = val + "\_economic\_status"

df[domain\_att\_1] = df[val].str.extract('([UCSTRZ])',expand = True)

df[domain\_att\_2] = df[val].str.extract('(\d)', expand=True)

df[domain\_att\_1] = df[domain\_att\_1].replace({'U': 0,'C': 1,'S': 2,'T': 3,'R': 4,'Z': 5}, regex=False)

# new\_attri.append(domain\_att\_1)

# new\_attri.append(domain\_att\_2)

del df[val]

# exporting the dataframe to csv

df.to\_csv('cleaned\_PVA\_data.csv')

# NORMALIZE DATA

print np.count\_nonzero(df.columns.values)

pca\_variables = ["CHILD03", "CHILD07", "CHILD12", "CHILD18", "MBCRAFT", "MBGARDEN", "MBBOOKS", "MBCOLECT", "MAGFAML",

"MAGFEM", "MAGMALE", "PUBGARDN", "PUBCULIN", "PUBHLTH", "PUBDOITY", "PUBNEWFN", "PUBPHOTO", "PUBOPP",

"COLLECT1", "VETERANS", "BIBLE", "CATLG", "HOMEE", "PETS", "CDPLAY", "STEREO", "PCOWNERS", "PHOTO",

"CRAFTS", "FISHER", "GARDENIN", "BOATS", "WALKER", "KIDSTUFF", "CARDS", "PLATES", "LIFESRC",

"PEPSTRFL", "POP901", "POP902", "POP903", "POP90C1", "POP90C2", "POP90C3", "POP90C4", "POP90C5",

"ETH1", "ETH2", "ETH3", "ETH4", "ETH5", "ETH6", "ETH7", "ETH8", "ETH9", "ETH10", "ETH11", "ETH12",

"ETH13", "ETH14", "ETH15", "ETH16", "AGE901", "AGE902", "AGE903", "AGE904", "AGE905", "AGE906",

"AGE907", "CHIL1", "CHIL2", "CHIL3", "AGEC1", "AGEC2", "AGEC4", "AGEC5", "AGEC6", "AGEC7", "CHILC1",

"CHILC2", "CHILC3", "CHILC4", "CHILC5", "HHAGE1", "HHAGE2", "HHAGE3", "HHN1", "HHN2", "HHN3", "HHN3",

"HHN4", "HHN5", "HHN6", "MARR1", "MARR2", "MARR3", "MARR4", "ETHC1", "ETHC2", "ETHC3", "ETHC4",

"ETHC5", "ETHC6", "HVP1", "HVP2", "HVP3", "HVP4", "HVP5", "HVP6", "IC1", "IC2", "IC3", "IC4", "IC5",

"IC6", "IC7", "IC8", "IC9", "IC9", "IC9", "IC10", "IC11", "IC12", "IC13", "IC14", "IC15", "IC16",

"IC17", "IC18", "IC19", "IC20", "IC21", "IC22", "IC23", "HHAS1", "HHAS2", "HHAS3", "HHAS4", "LFC1",

"LFC2", "LFC3", "LFC4", "LFC5", "LFC6", "LFC7", "LFC8", "LFC9", "LFC10", "OCC1", "OCC2", "OCC3",

"OCC4", "OCC5", "OCC6", "OCC7", "OCC8", "OCC9", "OCC10", "OCC11", "OCC12", "OCC13", "EIC1", "EIC2",

"EIC3", "EIC4", "EIC5", "EIC6", "EIC7", "EIC8", "EIC9", "EIC10", "EIC11", "EIC12", "EIC13", "EIC14",

"EIC15", "EIC16", "OEDC1", "OEDC2", "OEDC3", "OEDC4", "OEDC5", "OEDC6", "OEDC7", "EC1", "EC2", "EC3",

"EC4", "EC5", "EC6", "EC7", "EC8", "SEC1", "SEC2", "SEC3", "SEC4", "AFC1", "AFC2", "AFC3", "AFC4",

"AFC5", "AFC6", "VC1", "VC2", "VC3", "VC4", "HC1", "HC2", "HC3", "HC4", "HC5", "HC6", "HC7", "HC8",

"HC9", "HC10", "HC11", "HC12", "HC13", "HC14", "HC15", "HC16", "HC17", "HC18", "HC19", "HC20", "HC21",

"MHUC1", "MHUC2", "AC1", "AC2", "RAMNT\_3", "RAMNT\_4", "RAMNT\_5", "RAMNT\_6", "RAMNT\_7", "RAMNT\_8",

"RAMNT\_9", "RAMNT\_10", "RAMNT\_11", "RAMNT\_12", "RAMNT\_13", "RAMNT\_14", "RAMNT\_15", "RAMNT\_16",

"RAMNT\_17", "RAMNT\_18", "RAMNT\_19", "RAMNT\_20", "RAMNT\_21", "RAMNT\_22", "RAMNT\_23", "RAMNT\_24",

"RAMNTALL", "NGIFTALL", "CARDGIFT", "MINRAMNT", "MAXRAMNT", "TIMELAG", "AVGGIFT", new\_attri]

print len(pca\_variables)

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

# std\_scale = preprocessing.StandardScaler().fit(df[pca\_variables])

# df\_std = std\_scale.transform(df[pca\_variables])

# minmax\_scale = preprocessing.MinMaxScaler().fit(df[pca\_variables])

# df\_minmax = minmax\_scale.transform(df[pca\_variables])

"""

df1 = pd.DataFrame(index=range(0, 99999), columns=['A'], dtype='int')

for nd\_series in df.columns:

print type(nd\_series)

if nd\_series in pca\_variables:

df1.append(df[nd\_series])

print df1.column

X\_std = StandardScaler().fit\_transform(df1)

mean\_vec = np.mean(X\_std, axis=0)

cov\_mat = (X\_std - mean\_vec).T.dot((X\_std - mean\_vec)) / (X\_std.shape[0]-1)

print('Covariance matrix \n%s' %cov\_mat)

cov\_mat = np.cov(X\_std.T)

eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)

print('Eigenvectors \n%s' %eig\_vecs)

print('\nEigenvalues \n%s' %eig\_vals)"""

**Appendix-2**

Process chart:

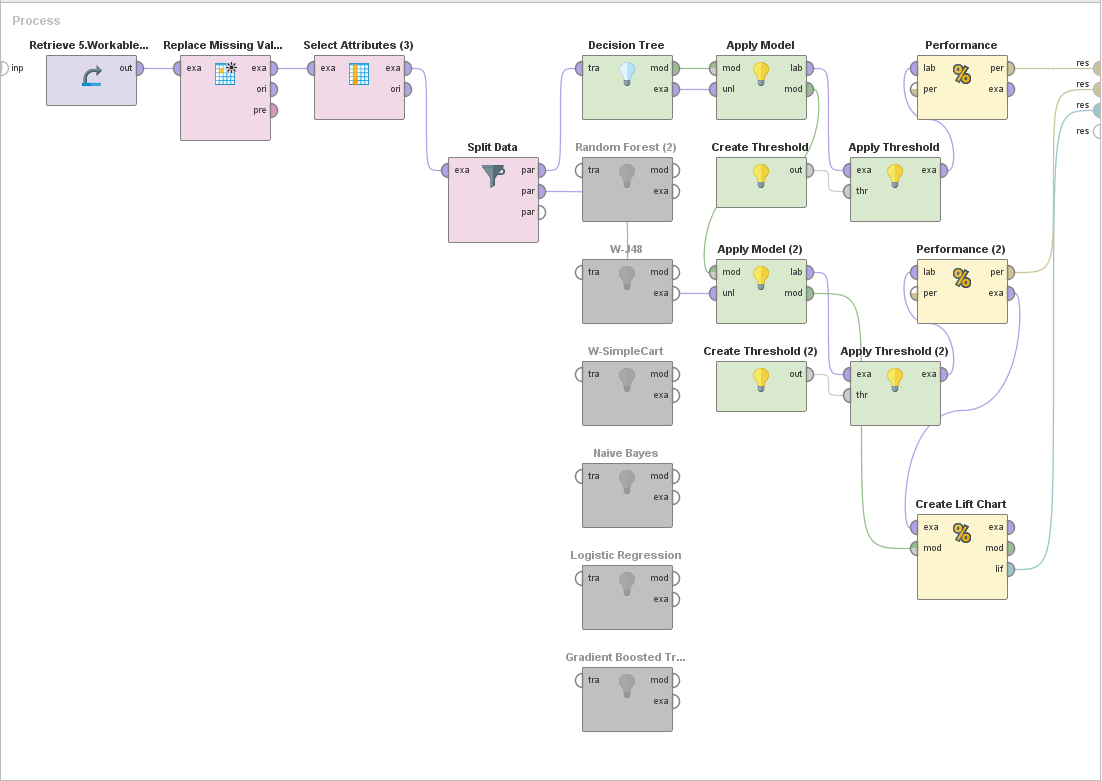


Fig.1 Classification modelling process Rapidminer.

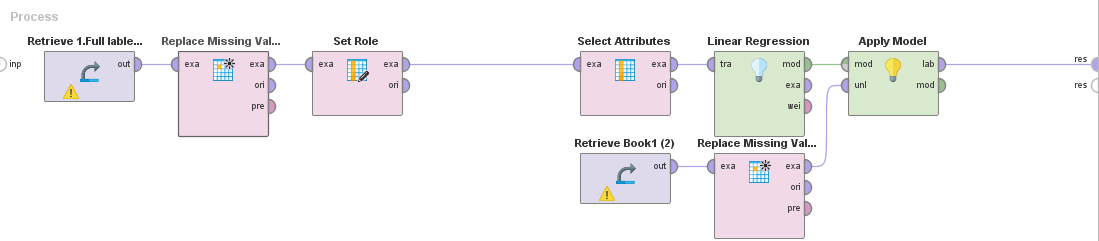


Fig.2 Regression Modelling Process Rapidminer.