**Data Mining Assignment 2**

Jaideep Adusumelli

Kaushik Kompella

Shushant Srirangapathi

**Question 1**

**The dataset has many variables – some (most?) of them may not be useful for our purpose. Your first task is to clean and explore the data, determine missing values and how you might handle these, which variables you think need not be considered, which should be transformed, etc. This is a major task – and can take time, much more than the modeling step that comes next. You will find below a list of subset of variables that someone found useful.**

**Which variables will you consider for modeling (and why)?**

**Which attributes will you omit from the analyses and why.**

**How do you clean the data, handle missing values? What new attributes/values do you derive?**

**How do you approach data reduction? What methods for data reduction do you try? Data cleaning - certain variables have 'empty' values in many rows. Some of these may be actual missing values, while the empty values may carry information (e.g. for a variable like college Education, empty values may indicate no-college-education which can be coded as a specific value). Some variables carry separate information in different bytes. Outline the data cleaning steps that you perform (and why)**

**Data exploration: Import the data, and examine the different variables – distribution of values, mean and  
standard deviation, range of values. What do you observe?**

**What variable transformations do you make (and why)?**

**Perform Principal Components Analysis (PCA) – which variables do you include for PCA (give your  
reason). Do decision trees help determine which variables to include in a predictive model for donors? How?**

The process of selection of attributes followed by us consisted of the following steps:

1. Eliminating the useless attributes:

The first step that we felt to be very important was to eliminate the variables which may not at all be related to our outcome of prediction. For doing this we thoroughly went through the description of each and every attributes, and then eliminating the attributes which may not at all be related to our outcome.

We also removed the variables having redundant information like “wealth-1” and “wealth-2” had almost the same information, while “wealth-1” had higher number of missing values. Hence removing these variables brought down the number of attributes we were dealing with to 398.

Some variables which we felt not to related to the outcome are: OSOURCE, TCODE, etc.

Some attributes which we felt may have redundant data and hence removed are: DOB, MAIL CODES, etc.

1. Dealing with the missing values:

The next step undertaken by us after we eliminated the useless attributes in the data was to eliminate the attribute with more than 90% of the rows missing. After this step we were left with around 370 variables. We chose this route because working only with 10% of the data from these attributes would not give us a good model and as we are aiming at the class precision and class recall of the responders, and having only 10% of the data points in these attributes would even more decrease the chances of accurately predicting the responders.

Some of the attributes eliminated by us include: ODATEDW, MAILCODE, etc.

1. Replacing the missing values:

After we eliminated the attributes having more than 90% of the rows missing, we are left with the data that has attributes with less than 90% of the rows missing. We either replaced the missing values with the median of the class or we binned the values as missing by just insert “-1” in the empty rows for the numeric attributes. And for the categorical variables like “Gender” we created a new bin saying “U = Undefined”.

Attributes filled with median: AGE.

Attributes binned by inserting “-1”: RAMNT\_X, COLLECT\_1, VETERANS, etc.

1. Dealing with the coded Attributes:

For the coded attributes having multiple bytes of information we separated each byte into a new attribute as per the code book. For example:

Example: Domain, Wealth. (coded)

We used python to apply the split (Python code used for data cleaning is presented in the “appendix-1” Section of the report.).

1. Reduction of attributes (PCA):

We found the most of the attributes that we used had inter-related data sets, or had many missing values, or even had redundant data sets. Hence we created two new attributes in place of all these attributes which reduced our data to a great extent. The attributes created by us are: PCA\_NEIGH and PCA\_RFA.

PCA\_NEIGH: This variable contains PCA components of the combination of various neighborhood variables.

PCA\_RFA: This variable contains PCA components of the combination of various RFA variables containing information of number of donations and amount of donation.

1. Reduction of attributes (Random forests):

After the previous step we were only left with XXX number of data sets. We wanted to further reduce the number of variables by comparing their importance for the prediction of the outcome variable and we used “Random Forests” for this purpose. After generating the graph for the importance of the variables we selected only 60 variables based on the importance.

The importance graph obtained by generating random forests on the obtained attributes:

**Final Set of attributes selected by us are:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PEPSTRFL | PCA\_RFA\_1 | NUMPRM12 | PCA\_NEIGH\_58 | PCA\_NEIGH\_1 | RAMNT\_15 | RAMNT\_6 | PCA\_RFA\_16 |
| CARDGIFT | NUMPROM | DOMAIN\_economic\_status | PCA\_NEIGH\_2 | PCA\_NEIGH\_63 | COLLECT1 | PCA\_RFA\_18 | PCA\_NEIGH\_43 |
| NGIFTALL | RFA\_2\_F | RAMNT\_8 | PCA\_NEIGH\_75 | PCA\_RFA\_7 | PCA\_NEIGH\_25 | PCA\_RFA\_10 | PCA\_RFA\_6 |
| AVGGIFT | CARDPROM | MALEVET | PCA\_RFA\_3 | PCA\_NEIGH\_52 | WWIIVETS | PCA\_NEIGH\_23 |  |
| PCA\_RFA\_2 | RAMNTALL | PCA\_RFA\_15 | PCA\_NEIGH\_6 | PCA\_NEIGH\_60 | VIETVETS | PCA\_RFA\_9 |  |
| LASTGIFT | MINRAMNT | CARDPM12 | PCA\_RFA\_5 | PCA\_NEIGH\_18 | PCA\_RFA\_13 | PCA\_NEIGH\_36 |  |
| MAXRAMNT | RAMNT\_14 | PCA\_NEIGH\_71 | PUBNEWFN | TIMELAG | PCA\_NEIGH\_76 | CRAFTS |  |
| RFA\_2\_C | PCA\_RFA\_4 | RECP3 | PCA\_NEIGH\_34 | PCA\_NEIGH\_59 | PCA\_NEIGH\_27 | PCA\_NEIGH\_11 |  |

**Question 2**

**Modeling:  
Partitioning - Partition the dataset into 60% training and 40% validation (set the seed to 12345). [A specified seed ensures that we obtain the same random partitioning every time we run it. With no specified seed, the system clock is typically used to set the seed, and a different partitioning can result in different runs].**

**Consider the following classification techniques on the data:**

**• Decision Trees (you can use J48, or any other suitable type of decision tree)**

**• Logistic Regression**

**• Naïve-Bayes**

**• Random forest**

**• Boosted trees**

**Be sure to test different parameter values for each method, as you see suitable. What parameter values do you try for the different techniques, and what do you find to work best?**

**Run each method on a chosen subset of the variables - how do you select this subset?**

**(Be sure NOT to include “TARGET-D” in your analysis.)**

**Provide a comparative evaluation of performance of your best models from each technique.  
Does variable selection/PCA make a difference for the different models?**

**Answer 2**

1. **Decision Trees:**

**For normal Decision Trees:**

Accuracy of Training Data set for the best tree: 67.96

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 3555 | 1577 | 69.27 |
| Predicted Bad | 345 | 522 | 60.21 |
| Class recall | 91.15 | 24.87 |  |

Accuracy of Test Data set for the best tree: 64.98

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2309 | 1110 | 67.53 |
| Predicted Bad | 291 | 290 | 49.91 |
| Class recall | 88.81 | 20.71 |  |

Parameters Used:

Criterion: Gini Index

Tree Depth: 6

Pruning: Yes

Confidence: 0.35

Pre-pruning: no

Accuracy of Test Data set of the variables without PCAs for the best tree: 64.68

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2470 | 1283 | 65.81 |
| Predicted Bad | 130 | 117 | 47.37 |
| Class recall | 95.00 | 8.36 |  |

**For J-48 Decision Trees for the best tree:**

Accuracy of Training Data set for the best tree: 80.91

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 3431 | 676 | 83.54 |
| Predicted Bad | 469 | 1423 | 75.21 |
| Class recall | 87.97 | 67.79 |  |

Accuracy of Test Data set for the best tree: 58.30

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1857 | 925 | 66.75 |
| Predicted Bad | 743 | 475 | 39.00 |
| Class recall | 71.42 | 33.93 |  |

Simple decision trees had better accuracy than the J-48 decision trees, but both of these had a very low recall and precision, which is very important for this prediction because we are mainly concentrating on predicting the total number of true positives and are not very concerned about the prediction precision and recall of the negative responses. But we also need to keep the accuracy high enough so that this model may work well for the future unseen dataset.

Hence, the decision tree models are not worth using.

Accuracy of Test Data set of the variables without PCAs for the best tree: 57.12

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1740 | 855 | 67.05 |
| Predicted Bad | 860 | 545 | 38.79 |
| Class recall | 66.92 | 38.93 |  |

1. **For Naïve Bayes:**

Accuracy of Training Data set for the best tree: 63.01

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 3149 | 1468 | 68.20 |
| Predicted Bad | 751 | 631 | 45.66 |
| Class recall | 80.74 | 30.06 |  |

Accuracy of Test Data set for the best tree: 64.90

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2154 | 958 | 69.22 |
| Predicted Bad | 446 | 442 | 49.77 |
| Class recall | 82.85 | 31.57 |  |

Parameters Used:

Laplace Correction: Yes

We tried checking the output by using Laplace correction and without using Laplace correction. But as we didn’t have any empty attributes we found that the output for both the models was same.

Naïve Bayes model had a good accuracy and a good precision, but it had a very low recall. We are here aiming at maximizing the profits i.e. maximizing the number of true positives predicted, which is the recall.

Accuracy of Test Data set of the variables without PCAs for the best tree: 58.63

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1637 | 692 | 70.29 |
| Predicted Bad | 963 | 708 | 42.37 |
| Class recall | 62.96 | 50.57 |  |

1. **For Gradient Boosted Trees:**

Accuracy of Training Data set for the best tree: 64.88

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2314 | 521 | 81.62 |
| Predicted Bad | 1586 | 1578 | 49.87 |
| Class recall | 59.33 | 75.18 |  |

Accuracy of Test Data set for the best tree: 57.00

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1375 | 495 | 73.53 |
| Predicted Bad | 1225 | 905 | 42.49 |
| Class recall | 52.88 | 64.65 |  |

Parameters used:

Number of trees: 40

Maximal Depth: 5

Minimum Rows: 20

Minimum Split Improvement: 0.0025

Number of Bins: 20

Learning Rate:0.05

Sample Rate: 1

Distribution: Auto

Accuracy of Test Data set of the variables without PCAs for the best tree: 57.23

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1452 | 563 | 72.06 |
| Predicted Bad | 1148 | 837 | 42.17 |
| Class recall | 55.85 | 59.79 |  |

1. **For logistic Regression:**

Accuracy of Training Data set for the best tree: 50.09

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 1254 | 348 | 78.28 |
| Predicted Bad | 2646 | 1751 | 39.82 |
| Class recall | 32.15 | 83.42 |  |

Accuracy of Test Data set for the best tree: 48.52

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 791 | 250 | 75.98 |
| Predicted Bad | 1809 | 1150 | 38.86 |
| Class recall | 30.42 | 82.14 |  |

We tried applying logistic regression model by varying: Solver, regularization, standardization, non-negative coefficients, compute-p, remove collinear columns, add intercept, missing value handling, maximum iterations and max runtime and we found that for all the parametric changes the accuracy, recall and precision almost remained the same.

But we couldn’t select this model because even though it had a very high recall, this come up at the cost of accuracy and precision which were observed to be very low.

Accuracy of Test Data set of the variables without PCAs for the best tree: 50.22

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 932 | 323 | 74.26 |
| Predicted Bad | 1668 | 1077 | 39.23 |
| Class recall | 35.85 | 76.93 |  |

1. **For Random Forests:**

Accuracy of Training Data set for the best tree: 65.07

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction / Actual | Actual Good | Actual Bad | Class precision |
| Predicted Good | 2417 | 1242 | 66.06 |
| Predicted Bad | 155 | 185 | 54.41 |
| Class recall | 93.97 | 12.96 |  |

Parameters Used:

Number of trees: 5000

Depth of trees: 15

Statistical process (“R”) was used to generate random forests and test their accuracy.

We found that random forests had decent accuracy and precision but also a very bad recall. And this was true for all the parameter changes.

We tested random forests for number of trees ranging from 100 to 5000, and depth of trees ranging from 1 to 15.

**Question 3**

**Classification under asymmetric response and cost: What is the reasoning behind using weighted sampling to produce a training set with equal numbers of donors and non-donors? Why not use a simple random sample from the original dataset? (Hint: given the actual response rate of 5.1%, how do you think the classification models will behave under simple sampling)? In this case, is classification accuracy a good performance metric for our purposes of maximizing net profit? If not, how would you determine the best model? Explain your reasoning.**

**Answer 3**

The reason for producing a training set with equal numbers of donors and non-donors is to give us a more precise model for prediction given the relatively low ratio of number of responders to total number of potential responders mailed. (5.1%) In our case, we would only have 510 responders to build the model from the 10,000 potential responders. Using this simple random sample would increases our margin of error significantly.

In this case, since we have built our model on a 65/35% split of non-donors to donors, we cannot look at the confusion matrix in the manner in which we usually use it. Overall accuracy would take place at the sacrifice of profit maximization in this case.

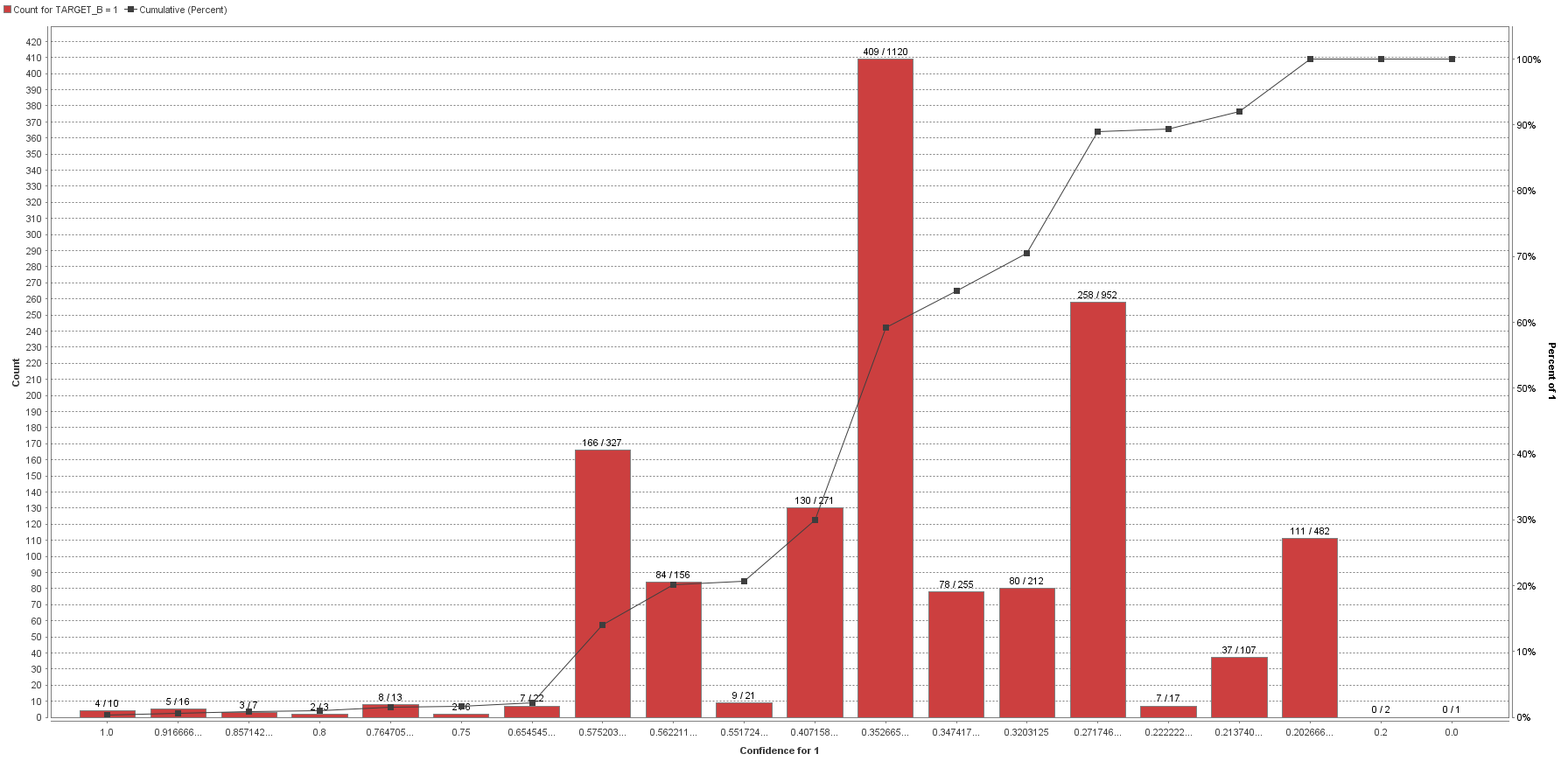
Very roughly speaking, the misclassification costs are very different. In choosing a model based on a 50/50 split, the profit realized would be much higher because there are very low misclassification costs. However, in reality, predicting a donor who is actually a donor would require greater class precision and class recall.

In practice, we would want to use a model that is not classification-centric, but is actually built to maximize overall profit. This is because it is more profitable to us to predict 1’s accurately given the profit received per mailer (~$13) rather than predicting the 0’s accurately given the cost per mailer (-.68). Ultimately, we are more concerned with the error rate and accuracy of 1’s than the accuracy and the error rate of 0’s.

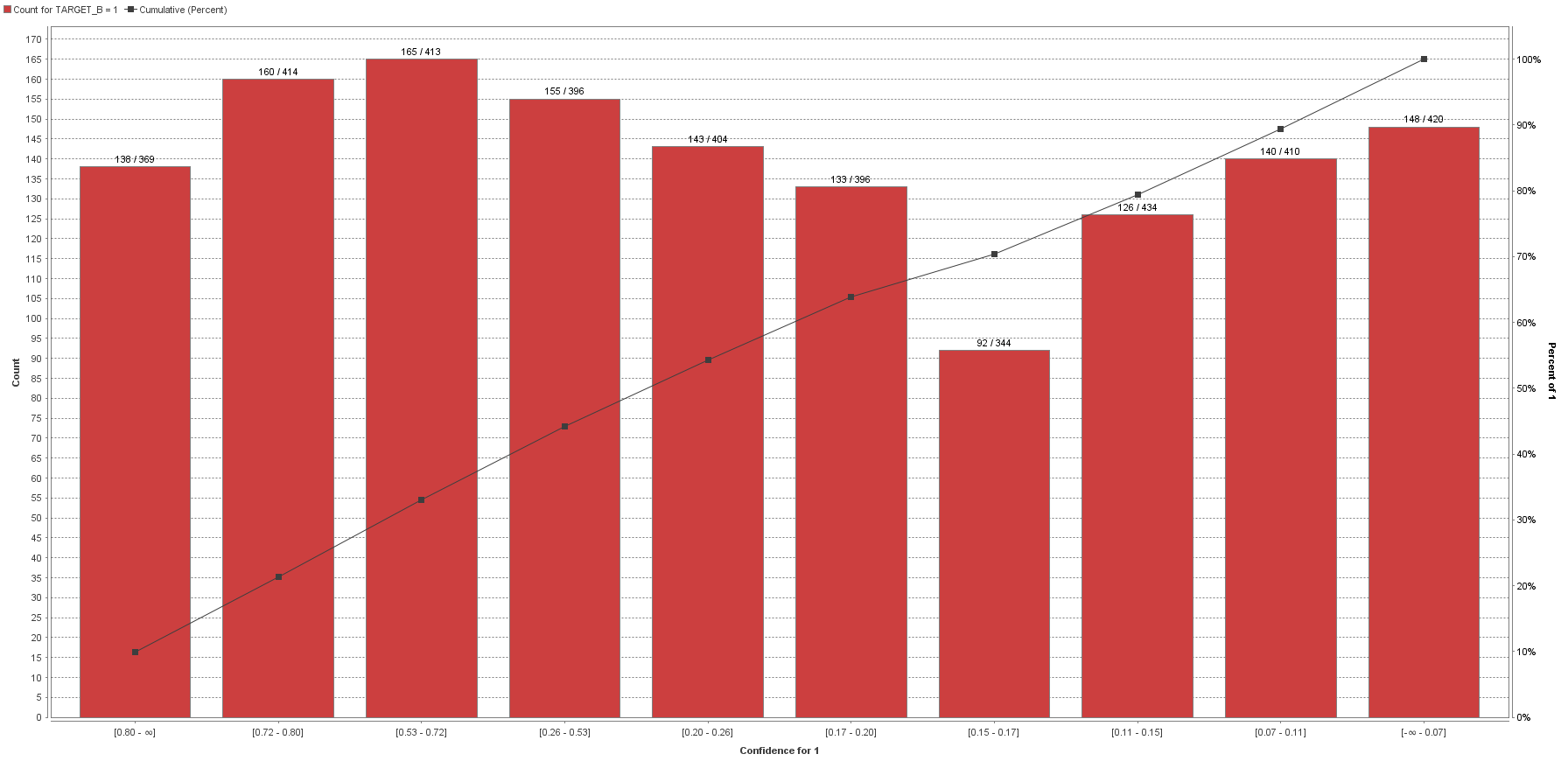
As seen in the above question the best model we chose in terms of accuracy was Gradient Boosted Trees did not have a high profit earnings.

These are the LIFT curves for our best models:

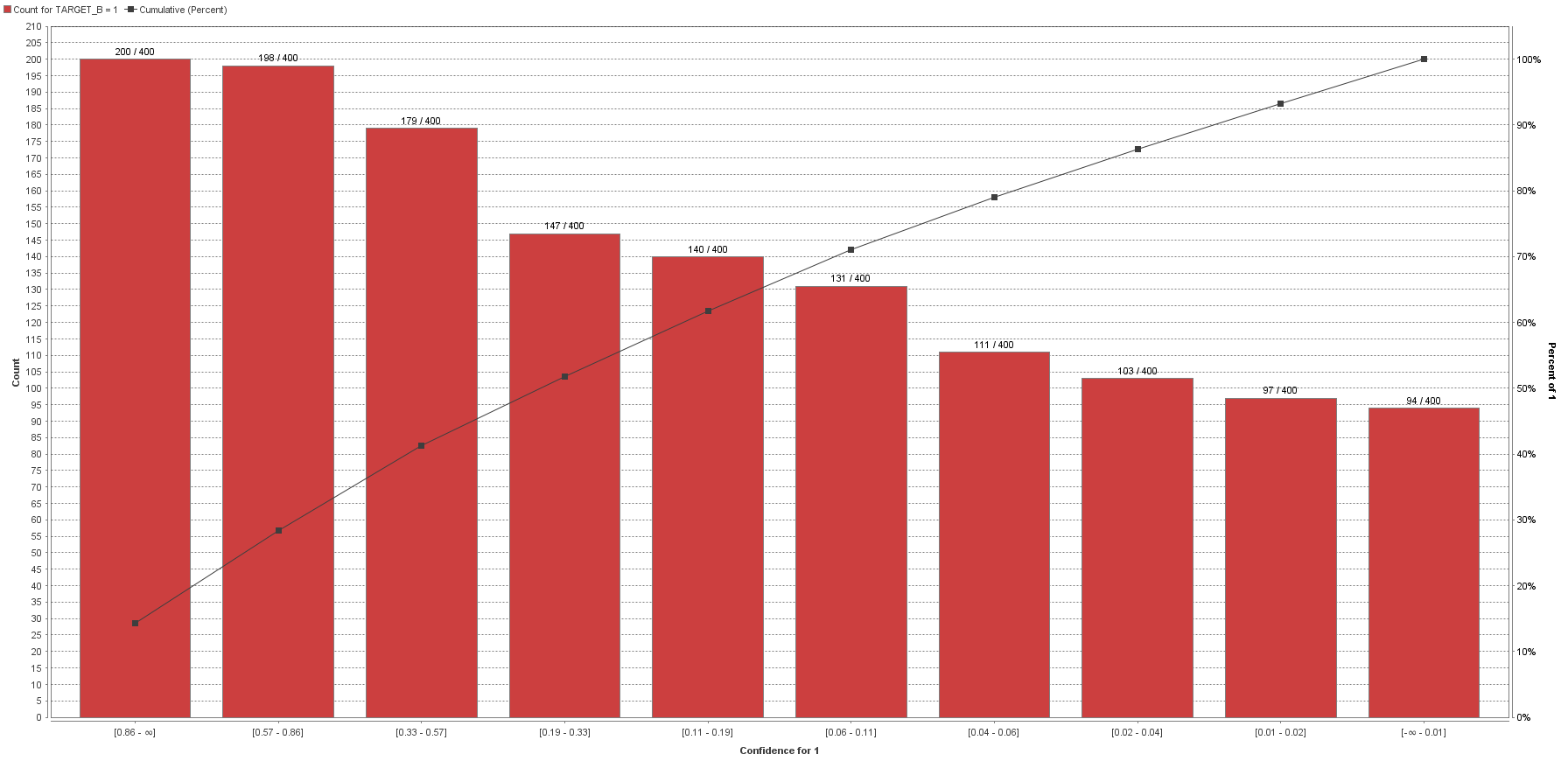
Decision Trees:



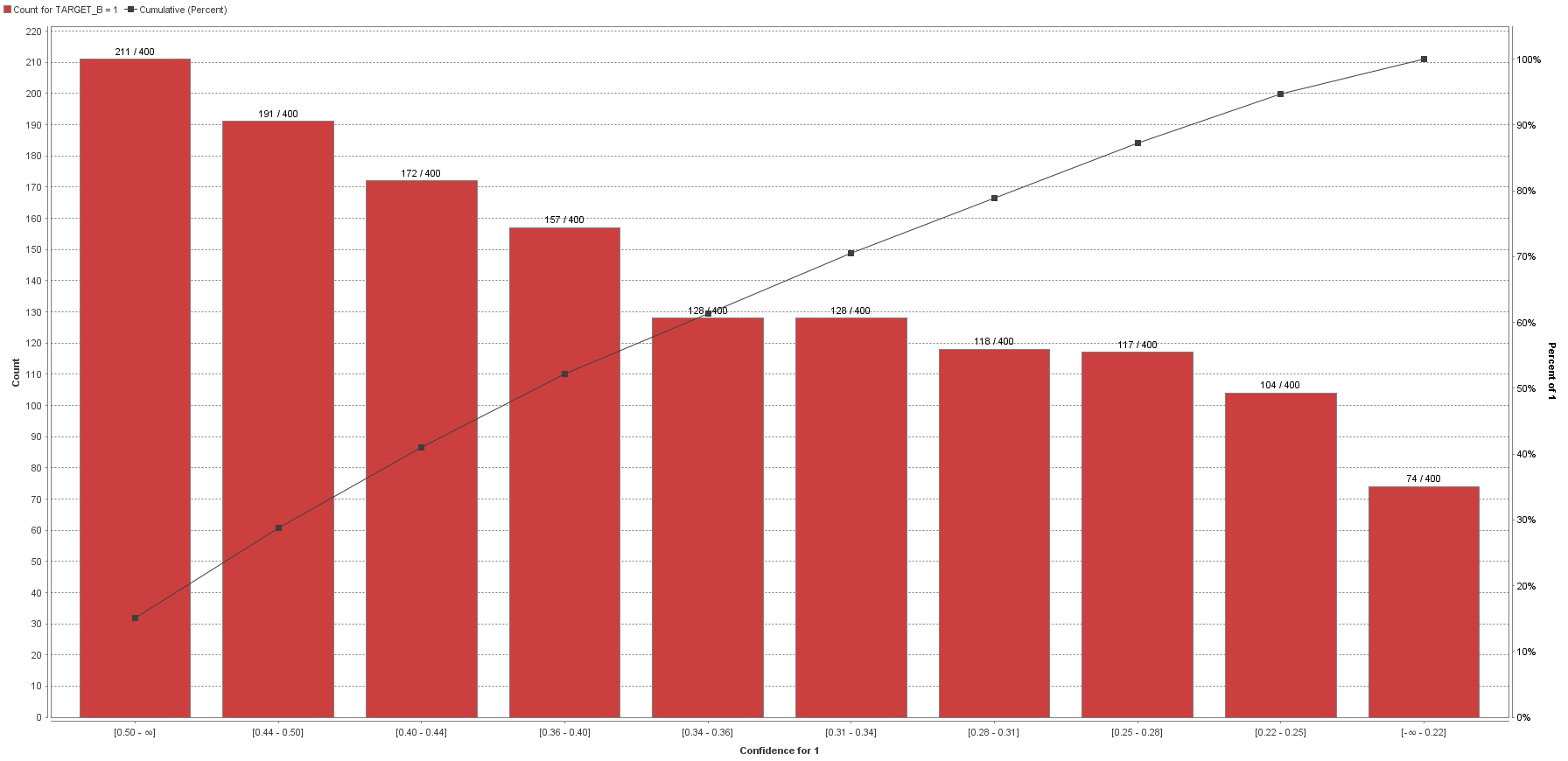
J-48 Decision Tree:



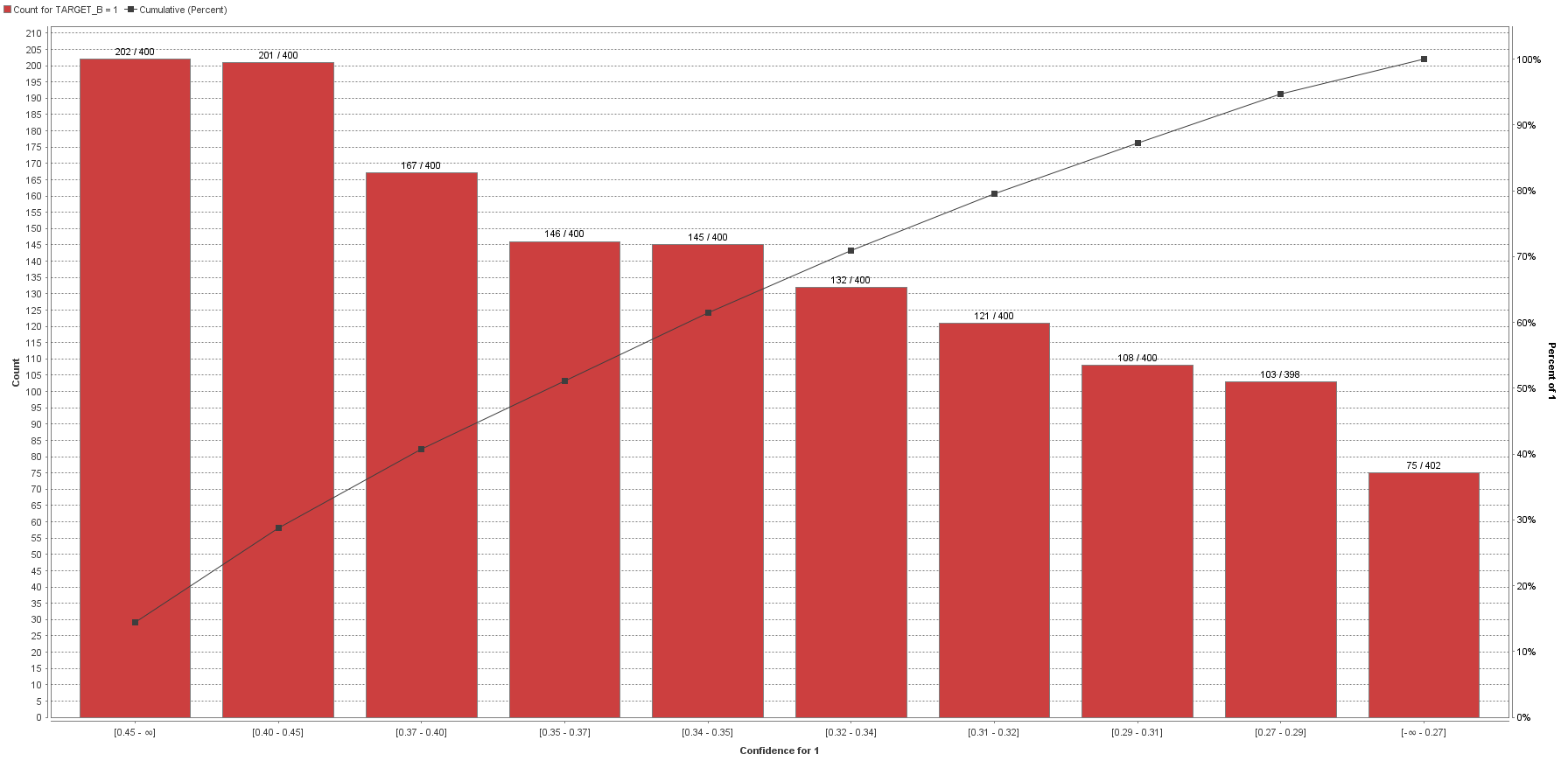
Naïve Bayes:



Logistic Regression:



Gradient Boosted Trees:



**Appendix 1**

Python Code 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 Line:

