Assignment 2 - BDBA

Loic Roldan Waals, 409763 & Iris Bominaar, 411495

March 28, 2018

Question 1

First the data was downloaded from the website, as the headers were missing, these were added to ensure understandability of the data set when importing it. Most of the variables are straight forward, there are two exceptions: *fnlwgt* (final weight) and *education-num*. The description explains that *"The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US."* and that "*People with similar demographic characteristics should have similar weights.*". It was also noted that these weights are only valid within their own state, given that we have no data as to which state people belong to, this lowers the validity of this measure in this research.

Finally it is still unclear what *education-num* means, for now it is assumed to be the total number of years that someone spent in the education system.

Question 2

We import both data sets and merge them with the following code:

```
#determine where files are read and written
setwd("C:/Users/Loic RW/Google Drive/Big Data and Business Analytics/Workshops/Session 2")



#read all the data
library(readr)
adult_data <- read_csv("C:/Users/Loic RW/Google Drive/Big Data and Business Analytics/Assignments/Assig
```

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   workclass = col_character(),
##   fnlwgt = col_integer(),
##   education = col_character(),
##   educationNum = col_integer(),
##   maritalStatus = col_character(),
##   occupation = col_character(),
##   relationship = col_character(),
##   race = col_character(),
##   sex = col_character(),
##   capitalGain = col_integer(),
##   capitalLoss = col_integer(),
##   hoursPerWeek = col_integer(),
##   nativeCountry = col_character(),
##   salary = col_character()
## )
```

```
adult_test <- read_csv("C:/Users/Loic RW/Google Drive/Big Data and Business Analytics/Assignments/Assig
```

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   workclass = col_character(),
##   fnlwgt = col_integer(),
##   education = col_character(),
##   educationNum = col_integer(),
##   maritalStatus = col_character(),
##   occupation = col_character(),
##   relationship = col_character(),
##   race = col_character(),
##   sex = col_character(),
##   capitalGain = col_integer(),
##   capitalLoss = col_integer(),
##   hoursPerWeek = col_integer(),
##   nativeCountry = col_character(),
##   salary = col_character()
## )
```

```
#merge datasets
alldata = rbind(adult_data, adult_test)
```

Question 3

The data is now inspected more thoroughly. We find many surprising problems in many variables. These are further discussed below.

Salary

It appears that in *adult.data*, salary has the factors *<=50K* and *>50K*; in *adult.test*, salary has the factors *<=50K.* and *>50K.*. Note that the *adult.test* set has a period at the end. We recode these all four of these factors to *1* and *0*, with *1* meaning that an observation earns more than $50K. We then rename the variable to *target* for ease of use. The code that is used is the following:

```
#transform the salary into a binomial variable
alldata$target[alldata$salary == "<=50K" | alldata$salary == "<=50K."] <- 0
```

```
## Warning: Unknown or uninitialised column: 'target'.
```

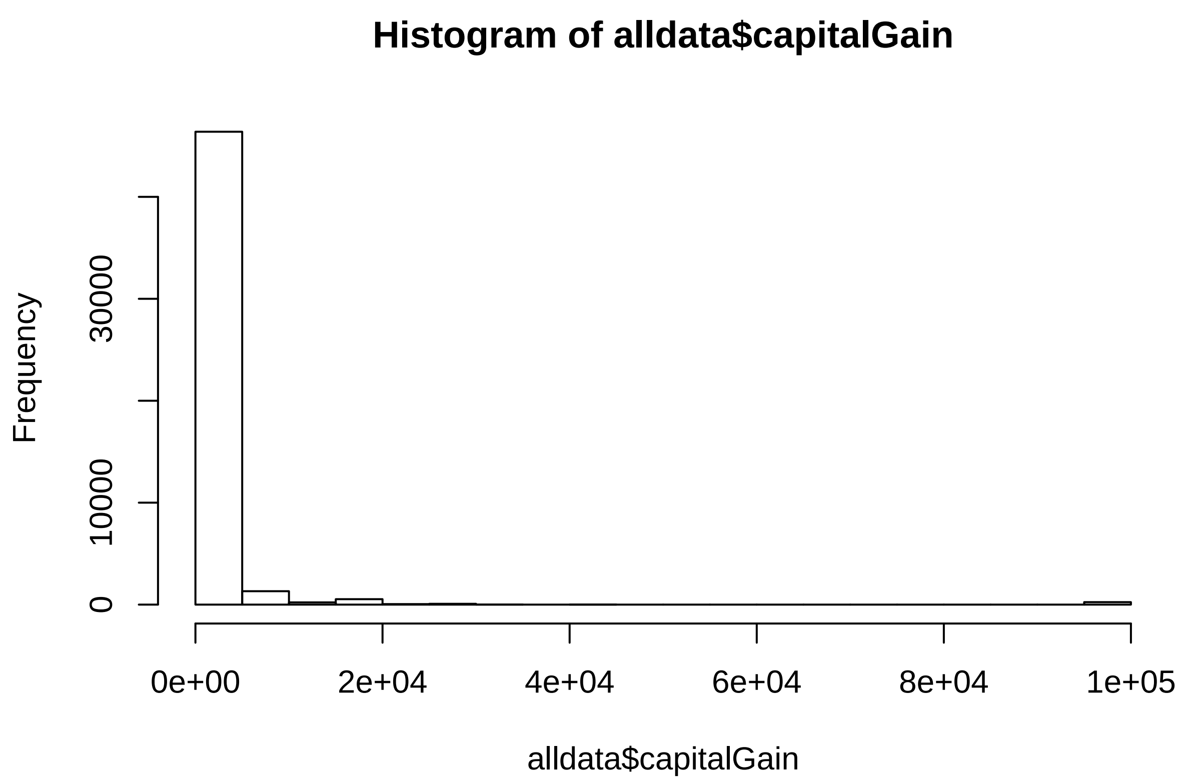
```
alldata$target[alldata$salary == ">50K" | alldata$salary == ">50K."] <- 1
alldata$salary <- NULL
alldata$target <- as.factor(alldata$target)
```

Continuous variables

There are two problems with several continuous variables, namely some unreasonable values and the fact that the variables have widely differing ranges.

With the variable *capitalGain* we find a positively skewed distribution. The problem is that there are 244 instances where the capital gain is exactly 99,999. This is peculiar as the next highest value for *capitalGain* is only 41,310. Whether this indicates a missing value or that the input could not handle numbers larger than 99,999 is unclear. Since these values only account for 0.5% of the total data set, these are set to missing so that they can be deleted afterwards. The code that is used is the following:

```r
#get rid of unreasonable values
hist(alldata$capitalGain)
```




```r
summary(alldata$capitalGain)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0    1079       0   99999
```

```r
summary(as.factor(alldata$capitalGain))
```

```
##      0  15024   7688   7298  99999   3103   5178   5013   4386
##  44807    513    410    364    244    152    146    117    108
##   8614   3325   2174  10520   4650  27828   4064    594   3137
##     82     81     74     64     63     58     54     52     51
##  14084  20051   2829   3908   6849  13550   1055   4787   3411
##     49     49     42     42     42     42     37     35     34
##  14344   3464   2176   2597   9386   2885   4101   2202   2407
##     34     33     31     31     31     30     29     28     25
##   4865   1506   4416   4508   3674   2354   2580  10605   2907
##     25     24     24     23     22     21     20     19     18
##   3942   5455   3781   6418   2105   2463   6497   7430   2635
##     18     18     16     16     15     15     15     15     14
##   2964  25236   1151   2653   2977   3471   3818    914   1409
##     14     14     13     11     11     11     11     10     10
##   1797   2290   2414   4934   6514  15020   1471   1831   1848
##     10     10     10     10     10     10      9      9      9
##    114   1086   2346   3418   3887  10566  15831   2329   3273
```

```
##       8       8       8       8       8       8       8       7       7
##    5721    7443     991    3456    5556    6767   25124   34095     401
##       7       7       6       6       6       6       6       6       5
##    1173    2036    2050    2228    2538    6723    9562    1424    1455
##       5       5       5       5       5       5       5       4       4
## (Other)
##      60
```

```r
alldata$capitalGain[alldata$capitalGain > 90000] <- NA
summary(alldata$capitalGain)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.0     0.0     0.0   582.4     0.0 41310.0     244
```

Furthermore we note that there are widely differing ranges for all the continuous variables (e.g. *educationNum* with a range of 15 and *fnlwgt* with a range of 1,478,115). To assure the models use these variables effectively they are scaled using the *scale* function. This subtracts the mean from the value and divides the result by the standard deviation of the variable. This is done after all missing values are removed, the code that is used is in the subsection **Categorical variables**.

```r
summary(alldata$educationNum)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    9.00   10.00   10.08   12.00   16.00
```

```r
summary(alldata$fnlwgt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   12285  117551  178145  189664  237642 1490400
```

Categorical variables

There are also several categorical variables that have some factors with very few observations (e.g. "Married-AF-spouse" in *maritalStatus*). To ensure no factors are present in a test set during cross validation that are not present in the training set of the same fold, these values are set to missing so that they can be deleted afterwards. It is also noted that these factors with very few observations are very prevalent in the *nativeCountry* variable, for this reason it will not be included in the model as it can present many problems when certain factors are in a training set, but not a test set. These uncommon factors are likely due to the fact that 91% of the people are from the US. Thus, we compute a variable that shows whether someone is originally from the us or not, with *1* representing a US native and *0* representing an immigrant.

Finally all missing values are deleted. After the deletion the continuous variables are scaled and *nativeCountry* is recoded to *USNative*. The total data set now contains 44,855 observations, or 91.8% of the original data set. The code that is used is the following:

```r
#get rid of variables with few observations
summary(as.factor(alldata$workclass))
```

```
##      Federal-gov        Local-gov     Never-worked          Private
##             1432             3136               10            33906
##     Self-emp-inc Self-emp-not-inc        State-gov      Without-pay
##             1695             3862             1981               21
##             NA's
##             2799
```

```r
summary(as.factor(alldata$education))
```

```
##         10th         11th         12th      1st-4th      5th-6th
```

```
##         1389         1812          657          247          509
##      7th-8th          9th   Assoc-acdm    Assoc-voc    Bachelors
##          955          756         1601         2061         8025
##    Doctorate      HS-grad      Masters    Preschool  Prof-school
##          594        15784         2657           83          834
## Some-college
##        10878
```

```
summary(as.factor(alldata$maritalStatus))
```

```
##              Divorced     Married-AF-spouse    Married-civ-spouse
##                  6633                    37                 22379
## Married-spouse-absent         Never-married             Separated
##                   628                 16117                  1530
##               Widowed
##                  1518
```

```
summary(as.factor(alldata$occupation))
```

```
##      Adm-clerical      Armed-Forces      Craft-repair   Exec-managerial
##              5611                15              6112              6086
##   Farming-fishing Handlers-cleaners Machine-op-inspct     Other-service
##              1490              2072              3022              4923
##   Priv-house-serv    Prof-specialty   Protective-serv             Sales
##               242              6172               983              5504
##      Tech-support  Transport-moving              NA's
##              1446              2355              2809
```

```
summary(as.factor(alldata$nativeCountry))
```

```
##                   Cambodia                     Canada
##                         28                        182
##                      China                   Columbia
##                        122                         85
##                       Cuba         Dominican-Republic
##                        138                        103
##                    Ecuador                El-Salvador
##                         45                        155
##                    England                     France
##                        127                         38
##                    Germany                     Greece
##                        206                         49
##                  Guatemala                      Haiti
##                         88                         75
##         Holand-Netherlands                   Honduras
##                          1                         20
##                       Hong                    Hungary
##                         30                         19
##                      India                       Iran
##                        151                         59
##                    Ireland                      Italy
##                         37                        105
##                    Jamaica                      Japan
##                        106                         92
##                       Laos                     Mexico
##                         23                        951
```

```
##                  Nicaragua Outlying-US(Guam-USVI-etc)
##                         49                         23
##                       Peru                Philippines
##                         46                        295
##                     Poland                   Portugal
##                         87                         67
##                Puerto-Rico                   Scotland
##                        184                         21
##                      South                     Taiwan
##                        115                         65
##                   Thailand            Trinadad&Tobago
##                         30                         27
##              United-States                    Vietnam
##                      43832                         86
##                 Yugoslavia                       NA's
##                         23                        857
```

```r
alldata$workclass[alldata$workclass == "Without-pay"] <- NA
alldata$workclass[alldata$workclass == "Never-worked"] <- NA
alldata$education[alldata$education == "Preschool"] <- NA
alldata$maritalStatus[alldata$maritalStatus == "Married-AF-spouse"] <- NA
alldata$occupation[alldata$occupation == "Armed-Forces"] <- NA


#delete all missing values
alldata <- alldata[complete.cases(alldata),]

#scale all int data
alldata$age <- scale(alldata$age)
alldata$fnlwgt <- scale(alldata$fnlwgt)
alldata$capitalGain <- scale(alldata$capitalGain)
alldata$capitalLoss <- scale(alldata$capitalLoss)
alldata$hoursPerWeek <- scale(alldata$hoursPerWeek)

#recode nativeCountry
alldata$USNative[alldata$nativeCountry == "United-States"] <- 1
```

```
## Warning: Unknown or uninitialised column: 'USNative'.
```

```r
alldata$USNative[alldata$nativeCountry != "United-States"] <- 0
alldata$USNative <- as.factor(alldata$USNative)
alldata$nativeCountry <- NULL
```

Question 4

In this question we check all the variables again after having cleaned them. If there was nothing noteworthy about a variable's values, it is not mentioned below. The highlights of this analysis include:

- Age has a slight positive skew
- No logical summary can be given of the *fnlwgt* variable
- The majority of the observations (92.1%) have a *capitalGain* value of 0
- The majority of the observations (95.2%) have a *capitalLoss* value of 0
- The most common hours per week number is 40, with 47.3% of all observations
- 73.8% works in the private sector

- 41.1% are husbands compared to only 4.8% who are wives. This imbalance is also reflected in the *sex* variable
- 67.4% of all cases are male, this imbalance could be due to the fact that the census data was from 1994, when women were not working as much as they are today.
- 86.0% of all observations were white
- As stated before 91.4% is originally from the US.
- The majority of all cases (75.6%), does not earn more than $50K, thus we are dealing with unbalanced classes. If we had a model that simply always predicts that an observation is earning less than $50K, it would have an expected accuracy of around 75%.

Question 5

We now define our model and split the data into 10 folds, in each fold we:

- Divide the data into a train and test set
- Train all three models (Logit, Support Vector Machine) on the train set
- Predict test set values using the trained models
- Measure the accuracy

The optimal number of iterations for the Neural Network were found by trying each value in 100 iteration increments and choosing the one with the lowest standard deviation. In our case this turned out to be 200 iterations.

The code that is used for this is the following:

```r
#load all packages
library("rpart")
library("rpart.plot")
library("caret")
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```r
library("e1071")
library(caTools)
library(C50)
library(nnet)
library(e1071)
library(stargazer)
```

```
##
## Please cite as:
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##  R package version 5.2.1. https://CRAN.R-project.org/package=stargazer
```

```r
library(caret)

#Set the seed
set.seed(1)

#define the model
modelA <- target ~ age + workclass + fnlwgt + education + educationNum + maritalStatus + occupation + re
  capitalGain + capitalLoss + hoursPerWeek + USNative
```

```r
#cross validation
nFolds = 10
myFolds <- cut(seq(1,nrow(alldata)),
               breaks = nFolds,
               labels = FALSE)

#initialise accuracy variables
accLogit <- rep(NA, nFolds)
accSVM <- rep(NA, nFolds)
accNN <- rep(NA, nFolds)

for (i in 1:nFolds) {
  cat("Analysis of fold", i, "\n")

  #define training and test set (print commands are to help troubleshoot
  #and determine where the programme aborted during errors)
  testIndex <- which(myFolds == i, arr.ind = TRUE)
  crossTest <- alldata[testIndex, ]
  crossTrain <- alldata[-testIndex, ]
  print("data allocated")

  #train the models
  rsltLogit <- glm(modelA, data = crossTrain, family = "binomial")
  print("logit trained")
  rsltSVM <- svm(modelA, data = crossTrain)
  print("svm trained")
  rsltNN <- nnet(modelA, data = crossTrain, maxit = 200, size = 10)
  print("nn trained")

  #predict values
  pdLogit = predict(rsltLogit, crossTest, type = "response")
  pdLogit[pdLogit > 0.5] <- 1
  pdLogit[pdLogit <= 0.5] <- 0
  print("logit predicted")

  pdSVM = predict(rsltSVM, crossTest)
  print("svm predicted")

  pdNN = predict(rsltNN, crossTest, type = "raw")
  pdNN[pdNN > 0.5] <- 1
  pdNN[pdNN <= 0.5] <- 0
  print("nn predicted")

  #measure accuracy
  accLogit[i] = mean(pdLogit == crossTest$target)
  print("Logit accuracy saved")
  accSVM[i] = mean(pdSVM == crossTest$target)
  print("SVM accuracy saved")
  accNN[i] = mean(pdNN == crossTest$target)
  print("NN accuracy saved")
}
```

```
## Analysis of fold 1
## [1] "data allocated"
```

```
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 42460.488139
## iter  10 value 16263.826036
## iter  20 value 14485.634310
## iter  30 value 13445.339692
## iter  40 value 12992.270966
## iter  50 value 12808.534022
## iter  60 value 12685.009177
## iter  70 value 12627.553935
## iter  80 value 12600.262605
## iter  90 value 12563.585141
## iter 100 value 12507.730169
## iter 110 value 12465.801908
## iter 120 value 12408.677656
## iter 130 value 12373.260575
## iter 140 value 12347.304346
## iter 150 value 12313.232039
## iter 160 value 12266.641940
## iter 170 value 12234.601428
## iter 180 value 12213.206135
## iter 190 value 12196.461895
## iter 200 value 12184.307649
## final  value 12184.307649
## stopped after 200 iterations
## [1] "nn trained"
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 2
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 32036.374693
## iter  10 value 21884.999999
## iter  20 value 17463.128225
## iter  30 value 14683.601160
## iter  40 value 14217.356000
## iter  50 value 13725.062325
## iter  60 value 13398.337530
## iter  70 value 13167.308780
## iter  80 value 13001.449847
## iter  90 value 12899.274823
## iter 100 value 12827.947371
## iter 110 value 12780.803237
## iter 120 value 12745.003541
```

```
## iter 130 value 12720.053116
## iter 140 value 12701.985829
## iter 150 value 12684.062850
## iter 160 value 12668.354857
## iter 170 value 12648.544067
## iter 180 value 12633.654053
## iter 190 value 12623.531880
## iter 200 value 12617.262774
## final  value 12617.262774
## stopped after 200 iterations
## [1] "nn trained"
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 3
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 26865.733663
## iter  10 value 15374.180083
## iter  20 value 13168.126076
## iter  30 value 12770.035159
## iter  40 value 12560.059349
## iter  50 value 12397.502530
## iter  60 value 12315.529288
## iter  70 value 12263.071189
## iter  80 value 12227.723295
## iter  90 value 12191.246693
## iter 100 value 12161.249969
## iter 110 value 12133.627623
## iter 120 value 12107.973202
## iter 130 value 12091.593385
## iter 140 value 12077.708783
## iter 150 value 12067.231089
## iter 160 value 12055.251736
## iter 170 value 12045.565309
## iter 180 value 12036.718134
## iter 190 value 12025.832368
## iter 200 value 12008.098766
## final  value 12008.098766
## stopped after 200 iterations
## [1] "nn trained"
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## [1] "logit predicted"
## [1] "svm predicted"
```

```
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 4
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 35755.858019
## iter  10 value 16825.356861
## iter  20 value 13474.449271
## iter  30 value 12842.596645
## iter  40 value 12549.631802
## iter  50 value 12340.653188
## iter  60 value 12245.744388
## iter  70 value 12187.537067
## iter  80 value 12135.744349
## iter  90 value 12093.287259
## iter 100 value 12055.576184
## iter 110 value 12025.956144
## iter 120 value 11995.032736
## iter 130 value 11962.089766
## iter 140 value 11942.334152
## iter 150 value 11927.007430
## iter 160 value 11908.610493
## iter 170 value 11874.094705
## iter 180 value 11850.670394
## iter 190 value 11824.172186
## iter 200 value 11791.483304
## final  value 11791.483304
## stopped after 200 iterations
## [1] "nn trained"
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 5
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 39722.390222
## iter  10 value 19048.436921
## iter  20 value 14303.168752
## iter  30 value 13134.743601
## iter  40 value 12830.469037
## iter  50 value 12625.311122
## iter  60 value 12480.553485
```

```
## iter  70 value 12427.501656
## iter  80 value 12369.355133
## iter  90 value 12312.322618
## iter 100 value 12243.348933
## iter 110 value 12191.296025
## iter 120 value 12153.938649
## iter 130 value 12118.543306
## iter 140 value 12092.555986
## iter 150 value 12078.824444
## iter 160 value 12063.078687
## iter 170 value 12044.033046
## iter 180 value 12025.633475
## iter 190 value 12008.801289
## iter 200 value 11992.360734
## final  value 11992.360734
## stopped after 200 iterations
## [1] "nn trained"
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 6
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 34212.534087
## iter  10 value 15409.302118
## iter  20 value 13734.606876
## iter  30 value 13426.328490
## iter  40 value 13311.712043
## iter  50 value 13205.604854
## iter  60 value 13081.355097
## iter  70 value 12962.765812
## iter  80 value 12866.025749
## iter  90 value 12790.972808
## iter 100 value 12743.447388
## iter 110 value 12703.171242
## iter 120 value 12665.188930
## iter 130 value 12634.182248
## iter 140 value 12600.005924
## iter 150 value 12575.185696
## iter 160 value 12554.697487
## iter 170 value 12539.788339
## iter 180 value 12523.494935
## iter 190 value 12512.581225
## iter 200 value 12502.193590
## final  value 12502.193590
## stopped after 200 iterations
```

```
## [1] "nn trained"

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 7
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 39225.861480
## iter  10 value 20224.351277
## iter  20 value 15398.942547
## iter  30 value 13749.586337
## iter  40 value 13391.925800
## iter  50 value 12932.681537
## iter  60 value 12790.081486
## iter  70 value 12708.778043
## iter  80 value 12579.714348
## iter  90 value 12514.018524
## iter 100 value 12472.943681
## iter 110 value 12440.800447
## iter 120 value 12400.160516
## iter 130 value 12368.439723
## iter 140 value 12346.749034
## iter 150 value 12327.138712
## iter 160 value 12308.093166
## iter 170 value 12281.311167
## iter 180 value 12267.236313
## iter 190 value 12245.317578
## iter 200 value 12233.369390
## final  value 12233.369390
## stopped after 200 iterations
## [1] "nn trained"

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 8
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 26067.699685
```

```
## iter  10 value 17635.512486
## iter  20 value 15744.176640
## iter  30 value 15152.845954
## iter  40 value 14665.663406
## iter  50 value 14053.188185
## iter  60 value 13558.967790
## iter  70 value 13236.577603
## iter  80 value 13047.776220
## iter  90 value 12915.697369
## iter 100 value 12859.371481
## iter 110 value 12805.623488
## iter 120 value 12778.420167
## iter 130 value 12718.609047
## iter 140 value 12658.740979
## iter 150 value 12617.839799
## iter 160 value 12592.692610
## iter 170 value 12564.906347
## iter 180 value 12534.529410
## iter 190 value 12502.152982
## iter 200 value 12473.142345
## final  value 12473.142345
## stopped after 200 iterations
## [1] "nn trained"

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 9
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 29450.384388
## iter  10 value 15941.259675
## iter  20 value 13516.607199
## iter  30 value 13144.780890
## iter  40 value 12885.744816
## iter  50 value 12776.860587
## iter  60 value 12647.088139
## iter  70 value 12571.462042
## iter  80 value 12505.861005
## iter  90 value 12457.260415
## iter 100 value 12417.093196
## iter 110 value 12394.654345
## iter 120 value 12360.312927
## iter 130 value 12333.423326
## iter 140 value 12320.465526
## iter 150 value 12303.978155
## iter 160 value 12282.642176
```

```
## iter 170 value 12253.668847
## iter 180 value 12226.681784
## iter 190 value 12211.831164
## iter 200 value 12198.170073
## final  value 12198.170073
## stopped after 200 iterations
## [1] "nn trained"

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
## Analysis of fold 10
## [1] "data allocated"
## [1] "logit trained"
## [1] "svm trained"
## # weights:  551
## initial  value 38493.281035
## iter  10 value 17917.652481
## iter  20 value 14378.511214
## iter  30 value 13133.224448
## iter  40 value 12817.222914
## iter  50 value 12734.906914
## iter  60 value 12679.887179
## iter  70 value 12646.752915
## iter  80 value 12601.996203
## iter  90 value 12547.223496
## iter 100 value 12508.497123
## iter 110 value 12484.714250
## iter 120 value 12465.545478
## iter 130 value 12448.169059
## iter 140 value 12433.325321
## iter 150 value 12422.667903
## iter 160 value 12414.671735
## iter 170 value 12406.615411
## iter 180 value 12398.800601
## iter 190 value 12384.901886
## iter 200 value 12374.665750
## final  value 12374.665750
## stopped after 200 iterations
## [1] "nn trained"

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "logit predicted"
## [1] "svm predicted"
## [1] "nn predicted"
## [1] "Logit accuracy saved"
## [1] "SVM accuracy saved"
## [1] "NN accuracy saved"
```

```r
#determine the average accuracy over the 10 folds for each model
avgAccLogit = mean(accLogit)
avgAccSVM = mean(accSVM)
avgAccNN = mean(accNN)

#determine the standard deviation of the accuracy over the 10 folds
#for each model
sdLogit = sd(accLogit)
sdSVM = sd(accSVM)
sdNN = sd(accNN)

#print all results
avgAccLogit
```

```
## [1] 0.8471297
```

```r
avgAccSVM
```

```
## [1] 0.8479547
```

```r
avgAccNN
```

```
## [1] 0.8520342
```

```r
sdLogit
```

```
## [1] 0.004806654
```

```r
sdSVM
```

```
## [1] 0.005657777
```

```r
sdNN
```

```
## [1] 0.003791826
```

Question 6

The best model appears to be the Neural Network with an accuracy of 85.2034239%, second comes the Support Vector Machine with an accuracy of 84.7954707% and last comes Logit with 84.712971%. It is to be noted that the difference between the best and worst performers was only about 0.4904529%, or about 200 observations. Thus the worst performance is not too different from the best.

Additionally, the standard deviations of the accuracy of the Logit, SVM and NN models were 0.4806654%, 0.5657777% and 0.3791826% respectively. Again there is not much difference between the most and least consistent model.

Overall it appears that the Neural Network performed the best in this problem with these parameters. Usually one would use the cross validation to determine the optimal parameters for a model and then calculate its actual performance against the test set. This was not done as we did not separate the train and test set before performing cross validation.