UNIVERSITY OF WATERLOO
Faculty of Mathematics
STAT 441, Fall 2022

Analysis of Factors Affected by Gender Differences in Speed Dating

Prepared by
Clarissa Diana
Haoqi Zhang
Jae Ho Jung
Jianglong Xing

Table of Contents

1 Introduction

With the amount of time spent on classes and coursework in university, it often leaves students with little to no free time and naturally, makes it all the more challenging when looking for a partner. The vast number of factors that come into play and not knowing where to start may also feel overwhelming.

1.1 Problem of Interest

In hopes of better understanding which attributes matter the most in dating, this report performs analysis on a data set obtained from a speed dating experiment. The aim is to assess whether or not each attribute holds the same degree of importance in obtaining a successful match for males and females, and for each case determine which attributes are most significant.

2 Analysis

2.1 Data

The data used in the report is extracted from Kaggle, which initially includes up to 195 attributes, ranging from individual traits, preferences, and occurrences during the date, with one final binary variable representing match outcome. Table 11 shows a few instances of the data, where 1 in the gender column represents male and 0 represents female, and the next 3 columns are few of the many other attributes presented in the data set.

Match	Gender	Age	Field	From
0	0	21	Law	Chicago
0	0	21	Law	Chicago
1	0	21	Law	Chicago

Table 1: First 3 columns of the data set with 5 attributes

Upon conducting brief exploratory analysis, the data was cleaned in the following ways: a number of columns were dropped due to excessive replication, and attributes outside of personal characteristics, e.g. evaluation score given by the other party on the date, were also removed. This leaves us with 49 attributes, including 'match'.

Prior to model-fitting, further exploratory analysis by graphical summaries had been done to find out whether gender affects results. With the use of the below ROC curves, the results have helped establish that when gender is used as a factor to predict the decision of a person about the match, a more accurate result is obtained. This is because when the data is split to be male or female only, gender is no longer to be considered as a factor or variable as all the data under variable gender in that data set will now have the same value. Therefore, this implies that the data can be split by gender to first perform analyses based on different gender groups, and used to assess the differences and similarities in which different gender groups will behave.

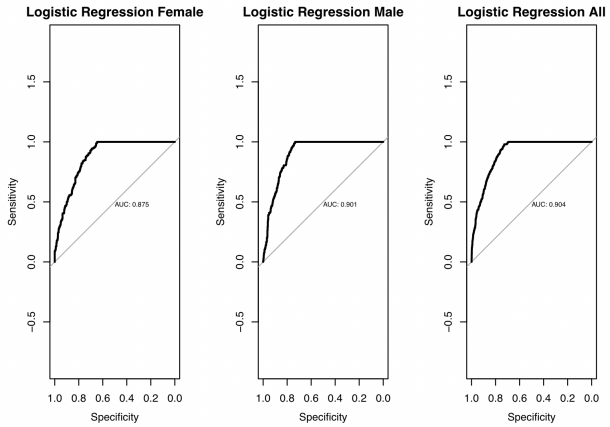


Figure 1: ROC curve for logistic regression on male, female and all data

2.2 Modelling

The methods used for modelling can be divided into five categories: Regression, Naive Bayes, Linear Discriminant Analysis (LDA), Random Forest and Single-Layer Neural Networks. This section will apply each technique individually, select an optimal set of weights that contribute to match outcome when using each technique, and determine to what extent this set of weights attached to each attribute for females can also be used for males. Afterwards, further analysis on exactly which variables are of larger importance is also made.

To test the predictive power of each model, half the data with gender female is first split and used to train the model (2082 in total), and the remaining half is further split into two: half for testing, and half for cross-validation testing (1041 each). Following fitting or training of the models, said model

is then tested on the male data set to analyze whether males and females share the same characteristics. Here, each method is analyzed through their respective confusion matrices, test errors and errors obtained from testing on the male data set as a whole. The model that is deemed the most well-fitting will be further used into the report's analysis and conclusion. The following five methods have been selected as they cover the scope of the course well, have relatively low training duration, and each carry their own pros and cons that will hopefully allow further logical deductions about the data set to be made.

2.2.1 Naive Bayes

Test on Test set		
	Matched	Not Matched
Matched	576	9
Not Matched	303	153

Table 2: Confusion matrix for test set with naive Bayes

Test on Male set		
	Matched	Not Matched
Matched	2703	55
Not Matched	736	625

Table 3: Confusion matrix for male data set with naive Bayes

The naive Bayes model fitted onto the 4164 training samples resulted in a test error rate of 29.97%. This is undeniably very large and several reasons are considered as to why this might be the case. It could be that the strong assumption made in naive Bayes that predictors are independent within each

class does not hold true, or the Bayes classifier in general is just not well-fitted for the model etc. Furthermore when using this model to predict on the overall male data, the error rate was found to be 19.20%. Hence, more models are fitted in hopes of finding one that does better at analyzing our classification problem.

2.2.2 LDA

Test on Test set		
	Matched	Not Matched
Matched	836	114
Not Matched	43	48

Table 4: Confusion matrix for test set with LDA

Test on Male set		
	Matched	Not Matched
Matched	3362	537
Not Matched	77	143

Table 5: Confusion matrix for male data set with LDA

With LDA, the test set error rate seems to have dropped to 15.08%, which is significantly better than what was obtained with naive Bayes. Since the LDA also uses a form of the Bayes classifier, this could mean that it wasn't the Bayes classifier that was unsuited for the data, but possibly the previous independence assumption that had been too strong. The error rate from the male data set is 14.91%, which is also noted to be lower than the test set's.

Test on Test set		
	Matched	Not Matched
Matched	836	117
Not Matched	43	15

Table 6: Confusion matrix for test set with regression

Test on Male set		
	Matched	Not Matched
Matched	3349	517
Not Matched	90	163

Table 7: Confusion matrix for male data set with regression

2.2.3 Regression

Here, the initial idea was to first perform feature selection by analyzing each attribute and comparing the match and no match boxplots of each attribute. Attributes that have boxplots that are similar but not too alike are to be be added to the model. The model is scaled before performing logistic regression and predicting on the test and male data set to assess error rates.

However as the selection process had been too tedious from the manual work of going through boxplots of all 48 columns, and whether or not an attribute was deemed appropriate to add to the model was also subjective as it was only up to the checker to judge, this step was omitted and logistic regression was performed on all the attributes, like with all the other methods.

The test set error rate is found to be 15.37% and the male set error rate 14.74%. This is similar to the results found with LDA, so for the next two

analyses, more complex methods have been chosen.

2.2.4 Random Forest

Test on Test set		
	Matched	Not Matched
Matched	829	50
Not Matched	114	48

Table 8: Confusion matrix for test set with random forest

Test on Male set		
	Matched	Not Matched
Matched	3392	47
Not Matched	586	94

Table 9: Confusion matrix for male data set with random forest

Although the tree-based model is different from previous ones, the results obtained are not much different. The test set error rate for this method is 15.75% and the error rate on the male set is 15.37%.

2.2.5 Neural network

Test on Test set		
	Matched	Not Matched
Matched	872	153
Not Matched	7	9

Table 10: Confusion matrix for test set with neural network

The final method was to conduct a single-layer neural network on the data. The explanatory variables are first rescaled and standardized to have mean zero and standard deviation one in order for all inputs to be treated equally

Test on Male set		
	Matched	Not Matched
Matched	3369	638
Not Matched	70	42

Table 11: Confusion matrix for male data set with neural network

in the regularization process. Categorical variables have been assigned an integer value beforehand in order for this step to work.

The neural network is then run and several size/decay combinations are attempted until one that performed best with the data is found. With size 4 and decay 0.01, the test set error rate obtained for this is 15.37% which is similar to past results. Here, the error rate on the male set is 17.19%, and is the only time it came out higher than the test error rate.

Although a lot of tuning is required for this method, empirically it still proves to be the best prediction tool even for this data set. It is also fortunately stable as the size of the data dealt with here is large enough.

3 Conclusion

Comparing the confusion matrices and test errors from the above five methods, it can be concluded that training the model with Single Hidden Layer Neural Networks works best for this classification problem. The plot below is the plot of importance of the variables. Based on this, attributes education, intelligence and region of living seem to be the most important factors

affecting whether someone is able to find a partner. In order to make better judgment, we generate two new Neural Network models for both men and women to see the differences between them. The following plots seem to indicate that men's matching success rate is related to personal achievement and financial status, whereas women's is more related to academic background.

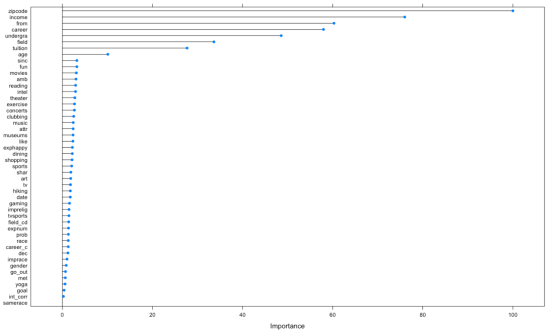


Figure 2: Plot of Importance for Neural Network Model

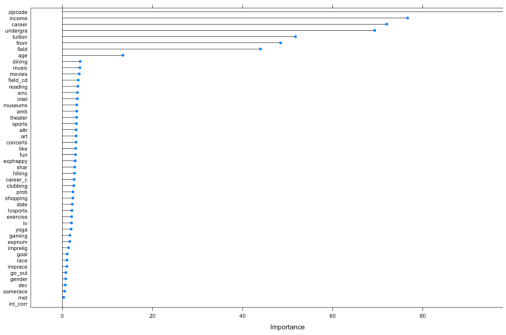


Figure 3: Plot of Importance for Neural Network Model for Male

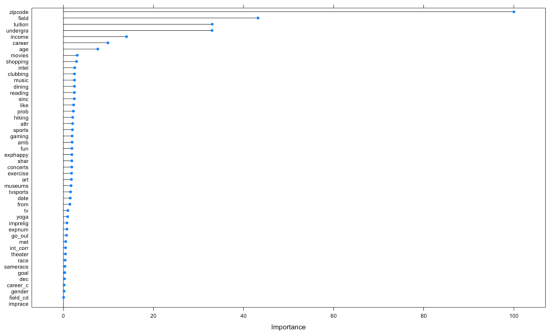


Figure 4: Plot of Importance for Neural Network Model for Female

4 Contribution

Haoqi formulated potential research questions and Jae narrowed it down to the current one, and Haoqi and Jae got started with the data cleaning. Each member then fitted a number of different models and the ones displayed on the report were done by the following members as follows: Clarissa did the Naive Bayes, LDA and Neural Network model, Haoqi did the Random Forest model and Jae did the Regression model. Jianglong generated the report in LaTeX, and arranged and modified the R code with Haoqi. Clarissa then wrote up the report and worked with Jianglong who helped with formatting. Haoqi then started the slides and Clarissa filled and tidied it up. Jae then made the video with the contents prepared.

5 Appendix

```r
Dating = read.csv("Dating.csv")
Dating = Dating[complete.cases(Dating$gender),]
Dating = Dating[complete.cases(Dating$age),]
rownames(Dating) = 1:nrow(Dating)
```

Data Cleaning

```r
allNA = c()
fa = c()
ind = c()

too.many.miss = 0

for (i in 1:dim(Dating)[2]) {
  ind = c(ind, i)
  column = Dating[,i]

  if (is.numeric(column)) {
    check.col = na.omit(column)
    # we check if numeric value
    max = max(check.col)
    min = min(check.col)

    if ((!is.na(max) & !is.na(min)) & (max == 1 & min == 0)) {
      fa = c(fa, "0/1")
      # it is categorical with 0/1
      new.col = Dating[,i]
      new.col[is.na(new.col) | new.col == ""] = "Other"
      Dating[,i] = as.integer(factor(new.col))

    } else {
      fa = c(fa, "number")
      # it is continuous number
      new.col = Dating[,i]

      # 0 - dim(t(na.omit(new.col)))[2] + dim(Dating)[1] > too.many.miss
      if (FALSE) {
        Dating = subset(Dating, select=-c(i))
      } else {
        # may change here if you like
        mean.val = median(check.col)
        new.col[is.na(new.col)] = mean.val
        Dating[,i] = new.col
      }

    }
  } else {

    fa = c(fa, "cat")
    # we check if categorical value
    new.col = Dating[,i]
    new.col[is.na(new.col) | new.col == ""] = "Other"
    Dating[,i] = as.integer(factor(new.col))
  }
```

```
}

mistake = c()

for (i in 1:dim(Dating)[2]) {
  if (dim(t(Dating[,i]))[2] != dim(t(na.omit(Dating[,i])))[2]) {
    mistake = c(mistake, i)
  }
}

femaleDating = Dating[Dating$gender == "2",]
maleDating = Dating[Dating$gender == "1",]
```

The following variables have the same value hence the columns will be dropped.

```
delete_col = c("amb5_3", "fun5_3", "intel5_3", "sinc5_3", "attr5_3", "shar2_3", "amb2_3", "fun2_3", "in
               "sinc4_3", "attr4_3", "shar7_3", "amb7_3", "fun7_3", "intel7_3",  "sinc7_3",  "attr7_3",
               "attr5_2", "shar2_2", "amb2_2", "fun2_2", "intel2_2", "sinc2_2",  "attr2_2", "shar4_2", "
               "intel7_2", "sinc7_2", "attr7_2", "amb3_s", "fun3_s", "intel3_s", "sinc3_s", "attr3_s", "
               "fun5_1", "intel5_1", "sinc5_1", "attr5_1", "shar4_1", "amb4_1", "fun4_1", "intel4_1", "s

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col]
maleDating = maleDating[, !names(maleDating) %in% delete_col]
```

We hope to study the influencing factors of the successful match, and the other half's evaluation data will affect the result of the match, so it is removed

```
delete_col2 = c("age_o","race_o","pf_o_att","pf_o_sin","pf_o_int","pf_o_fun","pf_o_amb","pf_o_sha","dec

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col2]
maleDating = maleDating[, !names(maleDating) %in% delete_col2]
```

Obviously, a high evaluation of both parties leads to a high matching success rate, but we want to study which characteristics of ourselves lead to a higher matching rate, so we proceed to drop these columns

```
delete_col3 = c("attr1_1",  "sinc1_1",  "intel1_1", "fun1_1",   "amb1_1"  ,
 "shar1_1" , "attr2_1",  "sinc2_1" , "intel2_1","fun2_1",   "amb2_1" ,  "shar2_1" ,
"attr3_1" , "sinc3_1" , "fun3_1" ,  "intel3_1", "amb3_1","numdat_2" ,"amb7_2"  , "attr1_2" ,
"sinc1_2",  "intel1_2" ,"fun1_2" ,  "amb1_2" ,  "shar1_2" , "attr3_2" , "sinc3_2",
"intel3_2", "fun3_2" ,  "amb3_2"  , "you_call", "them_cal" ,"date_3" ,  "attr1_3" ,
 "sinc1_3" , "intel1_3","fun1_3"  , "amb1_3"  , "shar1_3" , "attr3_3" , "sinc3_3" ,
 "intel3_3", "fun3_3"  , "amb3_3" , "satis_2","iid",   "id","idg", "condtn" ,  "wave","round"  , "posit

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col3]
maleDating = maleDating[, !names(maleDating) %in% delete_col3]
```

```
colnames(femaleDating)
```

```
##  [1] "gender"   "match"    "int_corr" "samerace" "age"      "field"
##  [7] "field_cd" "undergra" "tuition"  "race"     "imprace"  "imprelig"
## [13] "from"     "zipcode"  "income"   "goal"     "date"     "go_out"
## [19] "career"   "career_c" "sports"   "tvsports" "exercise" "dining"
## [25] "museums"  "art"      "hiking"   "gaming"   "clubbing" "reading"
## [31] "tv"       "theater"  "movies"   "concerts" "music"    "shopping"
## [37] "yoga"     "exphappy" "expnum"   "dec"      "attr"     "sinc"
## [43] "intel"    "fun"      "amb"      "shar"     "like"     "prob"
```

```
## [49] "met"
```

Now we use a single hidden layer neural network as an attempt to solve our classification problem.

```
library(nnet)
set.seed(1)
perm <- sample(x=nrow(femaleDating))
set1 <- femaleDating[which(perm<=nrow(femaleDating)/2),] # training set
set2 <- femaleDating[which(nrow(femaleDating)/2<perm & perm<=3*nrow(femaleDating)/4),] # validation set
set3 <- femaleDating[which(perm>3*nrow(femaleDating)/4),] # test set
setMale <- maleDating
```

```
# Function to rescale
rescale.set1 <- function(x1,x2){
  minx <- apply(X=x1, MARGIN=2, FUN=min)
  maxx <- apply(X=x1, MARGIN=2, FUN=max)
  x3 <- matrix (nrow=nrow(x2), ncol=ncol(x2))
  for(i in c(1:ncol(x2))){
    x3[,i] <- (x2[,i] - minx[i])/(maxx[i] - minx[i])
  }
  x3
}

x.1.unscaled <- as.matrix(set1[,-2])
x.1 <- rescale.set1(x.1.unscaled, x.1.unscaled)
x.2.unscaled <- as.matrix(set2[,-2])
x.2 <- rescale.set1(x.1.unscaled, x.2.unscaled)
x.3.unscaled <- as.matrix(set3[,-2])
x.3 <- rescale.set1(x.1.unscaled, x.3.unscaled)
x.male.unscaled <- as.matrix(setMale[,-2])
x.male <- rescale.set1(x.1.unscaled, x.male.unscaled)

x.1 <- as.matrix(set1[,-2])
x.2 <- as.matrix(set2[,-2])
x.3 <- as.matrix(set3[,-2])
x.male <- as.matrix(setMale[,-2])

y.1 <- class.ind(set1[,2])
y.2 <- class.ind(set2[,2])
y.3 <- class.ind(set3[,2])
y.male <- class.ind(setMale[,2])
```

```
### Fitting onto Set 1
nn.10.0 <- nnet(x=x.1, y=y.1, size=10, maxit=1000, softmax=TRUE, maxNWT=10000, print=FALSE)
```

```
## # weights:  512
## initial  value 2562.395417
## iter  10 value 955.714515
## iter  20 value 945.313161
## iter  30 value 939.101511
## iter  40 value 932.310500
## iter  50 value 929.047388
## iter  60 value 919.122929
## iter  70 value 898.827621
## iter  80 value 872.043400
## iter  90 value 832.155299
```

```
## iter 100 value 772.239273
## iter 110 value 720.163632
## iter 120 value 692.915290
## iter 130 value 661.718525
## iter 140 value 647.584404
## iter 150 value 633.178787
## iter 160 value 624.546034
## iter 170 value 619.244037
## iter 180 value 611.393081
## iter 190 value 599.237061
## iter 200 value 589.079906
## iter 210 value 586.329839
## iter 220 value 585.978397
## iter 230 value 585.811462
## iter 240 value 585.431578
## iter 250 value 584.694331
## iter 260 value 584.017519
## iter 270 value 583.948529
## iter 280 value 583.851773
## iter 290 value 583.585844
## iter 300 value 583.361159
## iter 310 value 583.095305
## iter 320 value 582.704428
## iter 330 value 582.485118
## iter 340 value 582.461468
## iter 350 value 582.422478
## iter 360 value 582.173576
## iter 370 value 581.955337
## iter 380 value 581.862012
## iter 390 value 581.643469
## iter 400 value 581.611776
## iter 410 value 581.561410
## iter 420 value 581.452753
## iter 430 value 581.230851
## iter 440 value 581.054614
## iter 450 value 580.808406
## iter 460 value 580.202364
## iter 470 value 580.106385
## iter 480 value 580.064193
## iter 490 value 579.986232
## iter 500 value 579.661252
## iter 510 value 579.607495
## iter 520 value 579.456751
## iter 530 value 579.296114
## iter 540 value 579.184081
## iter 550 value 579.053009
## iter 560 value 578.910584
## iter 570 value 578.278658
## iter 580 value 578.191485
## iter 590 value 578.069735
## iter 600 value 577.791223
## iter 610 value 577.720290
## iter 620 value 577.689930
## iter 630 value 577.616096
```

```
## iter 640 value 577.562470
## iter 650 value 577.516994
## iter 660 value 577.337309
## iter 670 value 575.863918
## iter 680 value 575.646471
## iter 690 value 575.488267
## iter 700 value 575.227837
## iter 710 value 575.116030
## iter 720 value 575.047166
## iter 730 value 575.009339
## iter 740 value 574.999437
## iter 750 value 574.972769
## iter 760 value 574.932692
## iter 770 value 574.782040
## iter 780 value 574.655502
## iter 790 value 574.588804
## iter 800 value 574.535175
## iter 810 value 574.489115
## iter 820 value 574.433899
## iter 830 value 574.382321
## iter 840 value 574.214900
## iter 850 value 573.949394
## iter 860 value 573.819653
## iter 870 value 573.744795
## iter 880 value 573.646759
## iter 890 value 573.550528
## iter 900 value 573.435942
## iter 910 value 573.383853
## iter 920 value 573.377302
## iter 930 value 573.366125
## iter 940 value 573.341794
## iter 950 value 573.339160
## iter 960 value 573.333052
## iter 970 value 573.323800
## iter 980 value 573.307921
## iter 990 value 573.259810
## iter1000 value 573.208381
## final  value 573.208381
## stopped after 1000 iterations
```

```
# Training set error
p1.nn.10.0 <-predict(nn.10.0, newdata=x.1, type="class")
table(p1.nn.10.0, as.factor(set1$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted    1    2
##         1 1637  168
##         2   85  192
```

```
(misclass1.10.0 <- mean(ifelse(p1.nn.10.0 == as.factor(set1$match), yes=0, no=1)))
```

```
## [1] 0.1215178
```

```
# Validation set error
p2.nn.10.0 <-predict(nn.10.0, newdata=x.2, type="class")
table(p2.nn.10.0, as.factor(set2$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted   1   2
##         1 826 118
##         2  55  42
```

```
(misclass2.10.0 <- mean(ifelse(p2.nn.10.0 == set2$match, yes=0, no=1)))
```

```
## [1] 0.1661864
```

```
# Test set error
p3.nn.10.0 <-predict(nn.10.0, newdata=x.3, type="class")
table(p3.nn.10.0, as.factor(set3$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted   1   2
##         1 815 126
##         2  64  36
```

```
(misclass3.10.0 <- mean(ifelse(p3.nn.10.0 == as.factor(set3$match), yes=0, no=1)))
```

```
## [1] 0.1825168
```

```
### Testing on male dataset
male.nn.10.0 <- predict(nn.10.0, newdata=x.male, type="class")
table(male.nn.10.0, as.factor(setMale$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted    1    2
##         1 3334  566
##         2  105  114
```

```
(misclassMale.10.0 <- mean(ifelse(male.nn.10.0 == as.factor(setMale$match), yes=0, no=1)))
```

```
## [1] 0.1629036
```

```
## Trying a different size/decay combination of 4/0.01
nn.4.01 <- nnet(x=x.1, y=y.1, size=4, maxit=1000, decay=.01, softmax=TRUE, MaxNWTs= 10000, print=FALSE)
```

```
## # weights:  206
## initial  value 1293.986209
## iter  10 value 958.704920
## iter  20 value 956.269922
## iter  30 value 954.920332
## iter  40 value 954.180896
## iter  50 value 949.928316
## iter  60 value 946.050506
## iter  70 value 937.774013
## iter  80 value 927.814753
## iter  90 value 924.229903
## iter 100 value 923.359014
## iter 110 value 921.237485
## iter 120 value 917.332807
## iter 130 value 916.695803
## iter 140 value 912.473955
## iter 150 value 911.053139
## iter 160 value 908.755196
## iter 170 value 897.822336
## iter 180 value 886.878104
## iter 190 value 880.471825
```

```
## iter 200 value 852.596570
## iter 210 value 832.283297
## iter 220 value 800.877320
## iter 230 value 701.515269
## iter 240 value 637.804301
## iter 250 value 622.839224
## iter 260 value 618.758366
## iter 270 value 610.639563
## iter 280 value 598.657565
## iter 290 value 588.080317
## iter 300 value 580.222927
## iter 310 value 575.883680
## iter 320 value 571.480619
## iter 330 value 569.135978
## iter 340 value 568.547188
## iter 350 value 568.079832
## iter 360 value 567.733910
## iter 370 value 566.918821
## iter 380 value 566.420445
## iter 390 value 565.804189
## iter 400 value 563.879624
## iter 410 value 562.030485
## iter 420 value 560.924886
## iter 430 value 560.490176
## iter 440 value 560.248723
## iter 450 value 559.818838
## iter 460 value 559.528828
## iter 470 value 559.281699
## iter 480 value 558.958881
## iter 490 value 558.595499
## iter 500 value 558.357554
## iter 510 value 558.324417
## iter 520 value 558.290850
## iter 530 value 557.898282
## iter 540 value 557.837066
## iter 550 value 557.410020
## iter 560 value 557.145841
## iter 570 value 556.776827
## iter 580 value 555.671556
## iter 590 value 554.710854
## iter 600 value 553.938310
## iter 610 value 553.680157
## iter 620 value 553.017306
## iter 630 value 548.756484
## iter 640 value 532.986253
## iter 650 value 526.290772
## iter 660 value 522.815137
## iter 670 value 522.498510
## iter 680 value 522.124840
## iter 690 value 521.595103
## iter 700 value 520.854994
## iter 710 value 520.202085
## iter 720 value 520.119017
## iter 730 value 520.007286
```

```
## iter 740 value 519.597312
## iter 750 value 519.077443
## iter 760 value 518.568013
## iter 770 value 518.085202
## iter 780 value 516.775025
## iter 790 value 516.308676
## iter 800 value 516.100896
## iter 810 value 516.019475
## iter 820 value 515.856114
## iter 830 value 515.691089
## iter 840 value 515.516322
## iter 850 value 515.204461
## iter 860 value 515.147917
## iter 870 value 514.911762
## iter 880 value 514.744524
## iter 890 value 514.441046
## iter 900 value 514.232452
## iter 910 value 513.744004
## iter 920 value 513.099261
## iter 930 value 512.766760
## iter 940 value 512.348674
## iter 950 value 511.824890
## iter 960 value 511.674778
## iter 970 value 511.395156
## iter 980 value 510.436586
## iter 990 value 509.059615
## iter1000 value 507.595522
## final  value 507.595522
## stopped after 1000 iterations
```

```
# Test set error on females
p3.nn.4.01 <-predict(nn.4.01, newdata=x.3, type="class")
table(p3.nn.4.01, as.factor(set3$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted   1   2
##         1 872 153
##         2   7   9
```

```
(misclass3.4.01 <- mean(ifelse(p3.nn.4.01 == as.factor(set3$match), yes=0, no=1)))
```

```
## [1] 0.1536984
```

```
# Test set error on males
male.nn.4.01 <- predict(nn.4.01, newdata=x.male, type="class")
table(male.nn.4.01, as.factor(setMale$match),  dnn=c("Predicted","Observed"))
```

```
##          Observed
## Predicted    1    2
##         1 3369  638
##         2   70   42
```

```
(misclassMale.4.01 <- mean(ifelse(male.nn.4.01 == as.factor(setMale$match), yes=0, no=1)))
```

```
## [1] 0.1718864
```

```
library(mlbench)
library(caret)
```

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

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

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)
model.nn <- train(match~., data=set1, method="nnet", trControl=control, print=FALSE)
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to do
## classification? If so, use a 2 level factor as your outcome column.
```

```
## # weights:  51
## initial  value 1179.802879
## final  value 318.000000
## converged
## # weights:  151
## initial  value 1007.495646
## final  value 318.000000
## converged
## # weights:  251
## initial  value 2042.095488
## final  value 318.000000
## converged
## # weights:  51
## initial  value 907.282119
## iter  10 value 322.456226
## iter  20 value 321.044810
## iter  30 value 321.021485
## final  value 321.021475
## converged
## # weights:  151
## initial  value 894.034793
## iter  10 value 324.110650
## iter  20 value 319.810530
## final  value 319.758231
## converged
## # weights:  251
## initial  value 802.270675
## iter  10 value 320.842045
## iter  20 value 319.492425
## iter  30 value 319.311138
## iter  40 value 319.281847
## iter  50 value 319.276050
## iter  60 value 319.275520
## final  value 319.275452
## converged
## # weights:  51
## initial  value 1024.502280
## iter  10 value 321.106631
## iter  20 value 318.035817
## final  value 318.009884
## converged
```

```
## # weights:  151
## initial  value 1080.775988
## iter  10 value 321.504880
## iter  20 value 318.040990
## iter  30 value 318.018400
## iter  40 value 318.015723
## iter  50 value 318.013998
## iter  60 value 318.011655
## final  value 318.005414
## converged
## # weights:  251
## initial  value 1172.846263
## iter  10 value 325.705269
## iter  20 value 318.088836
## iter  30 value 318.012015
## final  value 318.005859
## converged
## # weights:  51
## initial  value 1230.517704
## final  value 324.000000
## converged
## # weights:  151
## initial  value 1436.739875
## final  value 324.000000
## converged
## # weights:  251
## initial  value 1573.583697
## final  value 324.000000
## converged
## # weights:  51
## initial  value 1743.073222
## iter  10 value 329.217321
## iter  20 value 327.630235
## iter  30 value 327.340327
## iter  40 value 327.034262
## final  value 327.034209
## converged
## # weights:  151
## initial  value 885.811416
## iter  10 value 332.154145
## iter  20 value 325.861610
## iter  30 value 325.775173
## iter  40 value 325.765568
## iter  50 value 325.765266
## iter  60 value 325.765199
## final  value 325.765180
## converged
## # weights:  251
## initial  value 1537.660829
## iter  10 value 327.727964
## iter  20 value 325.303089
## iter  30 value 325.281719
## iter  40 value 325.280321
## final  value 325.280311
```

```
## converged
## # weights:  51
## initial  value 1422.855869
## iter  10 value 326.131725
## iter  20 value 324.024577
## iter  30 value 324.012983
## final  value 324.010895
## converged
## # weights:  151
## initial  value 1305.220336
## iter  10 value 330.662488
## iter  20 value 324.076813
## iter  30 value 324.006528
## iter  40 value 324.005695
## final  value 324.005690
## converged
## # weights:  251
## initial  value 772.817162
## iter  10 value 326.993193
## iter  20 value 324.034509
## iter  30 value 324.006235
## iter  30 value 324.006232
## iter  30 value 324.006232
## final  value 324.006232
## converged
## # weights:  51
## initial  value 1110.893455
## final  value 321.000000
## converged
## # weights:  151
## initial  value 2047.074035
## final  value 321.000000
## converged
## # weights:  251
## initial  value 607.439341
## final  value 321.000000
## converged
## # weights:  51
## initial  value 1344.132984
## iter  10 value 324.095315
## final  value 324.027849
## converged
## # weights:  151
## initial  value 1046.843824
## iter  10 value 323.274639
## iter  20 value 322.764947
## final  value 322.761683
## converged
## # weights:  251
## initial  value 1185.135218
## iter  10 value 331.882395
## iter  20 value 322.570631
## iter  30 value 322.489391
## iter  40 value 322.358201
```

```
## iter  50 value 322.278113
## iter  60 value 322.277901
## final  value 322.277844
## converged
## # weights:  51
## initial  value 1546.299430
## iter  10 value 323.457478
## iter  20 value 321.028333
## iter  30 value 321.011868
## iter  40 value 321.009904
## final  value 321.009898
## converged
## # weights:  151
## initial  value 569.639843
## iter  10 value 321.844631
## iter  20 value 321.010110
## final  value 321.009409
## converged
## # weights:  251
## initial  value 1207.901798
## iter  10 value 331.928793
## iter  20 value 321.126000
## iter  30 value 321.013286
## iter  40 value 321.007054
## iter  40 value 321.007053
## iter  40 value 321.007052
## final  value 321.007052
## converged
## # weights:  51
## initial  value 1192.459064
## final  value 322.000000
## converged
## # weights:  151
## initial  value 1464.614271
## final  value 322.000000
## converged
## # weights:  251
## initial  value 1633.708965
## final  value 322.000000
## converged
## # weights:  51
## initial  value 1216.772652
## iter  10 value 325.092155
## iter  20 value 325.029998
## final  value 325.029985
## converged
## # weights:  151
## initial  value 1338.038355
## iter  10 value 327.328415
## iter  20 value 325.177221
## iter  30 value 323.774039
## iter  40 value 323.765817
## iter  50 value 323.763381
## iter  60 value 323.763038
```

```
## final  value 323.762872
## converged
## # weights:  251
## initial  value 804.448105
## iter  10 value 334.457013
## iter  20 value 323.651478
## iter  30 value 323.529292
## iter  40 value 323.281925
## iter  50 value 323.279420
## iter  60 value 323.279190
## iter  70 value 323.279139
## iter  80 value 323.278966
## iter  90 value 323.278779
## final  value 323.278696
## converged
## # weights:  51
## initial  value 1049.126267
## iter  10 value 325.206802
## iter  20 value 322.036972
## final  value 322.009909
## converged
## # weights:  151
## initial  value 1384.668286
## iter  10 value 326.967800
## iter  20 value 322.057275
## final  value 322.011170
## converged
## # weights:  251
## initial  value 1622.765022
## iter  10 value 327.180959
## iter  20 value 322.059732
## iter  30 value 322.010716
## final  value 322.009724
## converged
## # weights:  51
## initial  value 943.326220
## final  value 325.000000
## converged
## # weights:  151
## initial  value 1287.519101
## final  value 325.000000
## converged
## # weights:  251
## initial  value 1435.170212
## final  value 325.000000
## converged
## # weights:  51
## initial  value 1091.160096
## iter  10 value 328.129688
## iter  20 value 328.036314
## iter  20 value 328.036312
## iter  20 value 328.036311
## final  value 328.036311
## converged
```

```
## # weights:  151
## initial  value 874.687342
## iter  10 value 328.281234
## iter  20 value 326.773331
## final  value 326.766327
## converged
## # weights:  251
## initial  value 1059.371704
## iter  10 value 328.107458
## iter  20 value 326.496728
## iter  30 value 326.285074
## iter  40 value 326.281696
## iter  50 value 326.281149
## final  value 326.281115
## converged
## # weights:  51
## initial  value 1253.344297
## iter  10 value 327.022048
## iter  20 value 325.023313
## final  value 325.018344
## converged
## # weights:  151
## initial  value 1264.494139
## iter  10 value 328.583419
## iter  20 value 325.041314
## final  value 325.010970
## converged
## # weights:  251
## initial  value 1640.836184
## iter  10 value 333.533611
## iter  20 value 325.098386
## iter  30 value 325.017413
## iter  40 value 325.008748
## final  value 325.008237
## converged
## # weights:  51
## initial  value 1018.548034
## final  value 331.000000
## converged
## # weights:  151
## initial  value 1255.434620
## final  value 331.000000
## converged
## # weights:  251
## initial  value 947.110838
## final  value 331.000000
## converged
## # weights:  51
## initial  value 1535.055780
## iter  10 value 334.169656
## iter  20 value 334.049095
## final  value 334.048809
## converged
## # weights:  151
```

```
## initial  value 1545.047124
## iter  10 value 334.488818
## iter  20 value 333.011126
## iter  30 value 332.779642
## iter  40 value 332.774489
## iter  50 value 332.773161
## final  value 332.773140
## converged
## # weights:  251
## initial  value 1454.629918
## iter  10 value 333.853809
## iter  20 value 332.301877
## iter  30 value 332.286223
## final  value 332.285875
## converged
## # weights:  51
## initial  value 1500.460160
## iter  10 value 333.452548
## iter  20 value 331.028276
## final  value 331.013713
## converged
## # weights:  151
## initial  value 812.372733
## iter  10 value 333.557786
## iter  20 value 331.029985
## iter  30 value 331.012499
## iter  40 value 331.006975
## iter  50 value 331.006720
## final  value 331.005751
## converged
## # weights:  251
## initial  value 1362.027011
## iter  10 value 337.267702
## iter  20 value 331.072262
## iter  30 value 331.012395
## iter  40 value 331.007812
## iter  50 value 331.005104
## final  value 331.004497
## converged
## # weights:  51
## initial  value 932.091774
## final  value 327.000000
## converged
## # weights:  151
## initial  value 1406.383027
## final  value 327.000000
## converged
## # weights:  251
## initial  value 1728.936072
## final  value 327.000000
## converged
## # weights:  51
## initial  value 785.440487
## iter  10 value 332.165387
```

```
## iter  20 value 330.040503
## final  value 330.040497
## converged
## # weights:  151
## initial  value 1034.081482
## iter  10 value 333.080314
## iter  20 value 328.812740
## iter  30 value 328.768964
## final  value 328.768607
## converged
## # weights:  251
## initial  value 1468.340154
## iter  10 value 329.518711
## iter  20 value 328.428100
## iter  30 value 328.291630
## iter  40 value 328.282895
## final  value 328.282705
## converged
## # weights:  51
## initial  value 1089.021590
## iter  10 value 328.711424
## iter  20 value 327.020312
## final  value 327.018314
## converged
## # weights:  151
## initial  value 1307.190514
## iter  10 value 333.109975
## iter  20 value 327.070443
## iter  30 value 327.013948
## iter  40 value 327.009836
## final  value 327.007008
## converged
## # weights:  251
## initial  value 615.143015
## iter  10 value 328.574686
## iter  20 value 327.018155
## final  value 327.005159
## converged
## # weights:  51
## initial  value 1195.105372
## final  value 325.000000
## converged
## # weights:  151
## initial  value 705.937005
## final  value 325.000000
## converged
## # weights:  251
## initial  value 967.525700
## final  value 325.000000
## converged
## # weights:  51
## initial  value 988.047604
## iter  10 value 336.737681
## iter  20 value 328.888578
```

```
## iter  30 value 328.147467
## iter  40 value 328.036410
## final  value 328.036303
## converged
## # weights:  151
## initial  value 1070.129727
## iter  10 value 330.651470
## iter  20 value 326.907847
## iter  30 value 326.767932
## iter  40 value 326.766312
## final  value 326.766305
## converged
## # weights:  251
## initial  value 1016.743213
## iter  10 value 327.698283
## iter  20 value 327.048280
## iter  30 value 326.348926
## iter  40 value 326.281529
## iter  50 value 326.281213
## iter  60 value 326.281097
## final  value 326.281083
## converged
## # weights:  51
## initial  value 858.014620
## iter  10 value 326.405614
## iter  20 value 325.022604
## final  value 325.022476
## converged
## # weights:  151
## initial  value 949.942227
## iter  10 value 329.053945
## iter  20 value 325.046739
## final  value 325.006974
## converged
## # weights:  251
## initial  value 1324.391078
## iter  10 value 331.105565
## iter  20 value 325.070392
## iter  30 value 325.011669
## final  value 325.005848
## converged
## # weights:  51
## initial  value 1150.267911
## final  value 318.000000
## converged
## # weights:  151
## initial  value 1479.623911
## final  value 318.000000
## converged
## # weights:  251
## initial  value 1217.385639
## final  value 318.000000
## converged
## # weights:  51
```

```
## initial  value 1330.041281
## iter  10 value 321.525470
## iter  20 value 321.027595
## final  value 321.021467
## converged
## # weights:  151
## initial  value 1127.836756
## iter  10 value 320.777488
## iter  20 value 319.781581
## iter  30 value 319.759400
## iter  40 value 319.758287
## final  value 319.758215
## converged
## # weights:  251
## initial  value 1574.721682
## iter  10 value 322.651416
## iter  20 value 319.363725
## iter  30 value 319.281645
## iter  40 value 319.275432
## iter  40 value 319.275429
## iter  40 value 319.275429
## final  value 319.275429
## converged
## # weights:  51
## initial  value 994.034933
## iter  10 value 319.726464
## iter  20 value 318.021257
## final  value 318.020542
## converged
## # weights:  151
## initial  value 875.187558
## iter  10 value 319.914483
## iter  20 value 318.022072
## final  value 318.009709
## converged
## # weights:  251
## initial  value 1037.669288
## iter  10 value 322.171127
## iter  20 value 318.048090
## iter  30 value 318.009967
## iter  40 value 318.005120
## iter  40 value 318.005119
## iter  40 value 318.005119
## final  value 318.005119
## converged
## # weights:  51
## initial  value 793.443129
## final  value 329.000000
## converged
## # weights:  151
## initial  value 678.719023
## final  value 329.000000
## converged
## # weights:  251
```

```
## initial  value 1736.096998
## final  value 329.000000
## converged
## # weights:  51
## initial  value 1385.690806
## iter  10 value 334.312393
## iter  20 value 332.055093
## iter  30 value 332.045541
## iter  40 value 332.044671
## final  value 332.044666
## converged
## # weights:  151
## initial  value 816.429720
## iter  10 value 332.752198
## iter  20 value 330.839488
## iter  30 value 330.772300
## iter  40 value 330.770897
## final  value 330.770884
## converged
## # weights:  251
## initial  value 1764.665733
## iter  10 value 332.334219
## iter  20 value 331.723136
## iter  30 value 330.833293
## iter  40 value 330.440244
## iter  50 value 330.367829
## iter  60 value 330.287522
## iter  70 value 330.285338
## iter  80 value 330.284950
## iter  90 value 330.284671
## iter 100 value 330.284467
## final  value 330.284467
## stopped after 100 iterations
## # weights:  51
## initial  value 940.536839
## iter  10 value 331.619658
## iter  20 value 329.030203
## iter  30 value 329.009930
## iter  30 value 329.009928
## iter  30 value 329.009928
## final  value 329.009928
## converged
## # weights:  151
## initial  value 1150.700340
## iter  10 value 333.101891
## iter  20 value 329.047292
## final  value 329.013155
## converged
## # weights:  251
## initial  value 875.066220
## iter  10 value 332.129008
## iter  20 value 329.036075
## final  value 329.007034
## converged
```

```
## # weights:  51
## initial  value 1708.593621
## final  value 310.000000
## converged
## # weights:  151
## initial  value 859.828261
## final  value 310.000000
## converged
## # weights:  251
## initial  value 854.774873
## final  value 310.000000
## converged
## # weights:  51
## initial  value 1724.534934
## iter  10 value 315.199906
## iter  20 value 313.005353
## iter  30 value 313.004207
## final  value 313.004167
## converged
## # weights:  151
## initial  value 875.949891
## iter  10 value 313.720107
## iter  20 value 311.922041
## iter  30 value 311.749362
## final  value 311.748817
## converged
## # weights:  251
## initial  value 959.191920
## iter  10 value 321.762486
## iter  20 value 311.750128
## iter  30 value 311.648584
## iter  40 value 311.294303
## iter  50 value 311.273272
## iter  60 value 311.271429
## iter  70 value 311.270801
## iter  80 value 311.270430
## iter  90 value 311.269139
## final  value 311.268842
## converged
## # weights:  51
## initial  value 980.578319
## iter  10 value 311.463197
## iter  20 value 310.018130
## iter  30 value 310.013066
## iter  40 value 310.009856
## final  value 310.009849
## converged
## # weights:  151
## initial  value 1947.268712
## iter  10 value 316.773793
## iter  20 value 310.078097
## iter  30 value 310.007647
## final  value 310.007593
## converged
```

```
## # weights:  251
## initial  value 1260.411966
## iter  10 value 317.421742
## iter  20 value 310.085567
## iter  30 value 310.011172
## iter  40 value 310.005407
## final  value 310.005392
## converged
## # weights:  51
## initial  value 1142.975071
## final  value 323.000000
## converged
## # weights:  151
## initial  value 1540.532281
## final  value 323.000000
## converged
## # weights:  251
## initial  value 1702.255133
## final  value 323.000000
## converged
## # weights:  51
## initial  value 1434.362134
## iter  10 value 326.592167
## iter  20 value 326.032270
## final  value 326.032095
## converged
## # weights:  151
## initial  value 871.571931
## iter  10 value 333.089655
## iter  20 value 325.797205
## iter  30 value 325.214615
## iter  40 value 324.832388
## iter  50 value 324.766825
## iter  60 value 324.764185
## final  value 324.764021
## converged
## # weights:  251
## initial  value 677.453117
## iter  10 value 327.758385
## iter  20 value 324.726996
## iter  30 value 324.285525
## iter  40 value 324.280252
## iter  50 value 324.279717
## final  value 324.279496
## converged
## # weights:  51
## initial  value 1179.778501
## iter  10 value 326.660088
## iter  20 value 323.042489
## iter  30 value 323.027273
## iter  40 value 323.018030
## iter  50 value 323.016231
## iter  60 value 323.011101
## final  value 323.010838
```

```
## converged
## # weights:  151
## initial  value 1231.372827
## iter  10 value 328.978544
## iter  20 value 323.068928
## iter  30 value 323.014624
## final  value 323.006959
## converged
## # weights:  251
## initial  value 1640.786303
## iter  10 value 331.816115
## iter  20 value 323.101643
## iter  30 value 323.018668
## iter  40 value 323.009362
## iter  50 value 323.006514
## final  value 323.005231
## converged
## # weights:  51
## initial  value 1082.788649
## final  value 332.000000
## converged
## # weights:  151
## initial  value 991.541315
## final  value 332.000000
## converged
## # weights:  251
## initial  value 1263.473248
## final  value 332.000000
## converged
## # weights:  51
## initial  value 1028.091224
## iter  10 value 335.697347
## iter  20 value 335.050882
## final  value 335.050872
## converged
## # weights:  151
## initial  value 585.463962
## iter  10 value 337.238131
## iter  20 value 334.054700
## iter  30 value 333.800802
## iter  40 value 333.774714
## iter  50 value 333.774356
## final  value 333.774265
## converged
## # weights:  251
## initial  value 1211.864128
## iter  10 value 336.268526
## iter  20 value 333.305532
## iter  30 value 333.293671
## iter  40 value 333.291133
## iter  50 value 333.288592
## iter  60 value 333.287222
## iter  60 value 333.287220
## final  value 333.286661
```

```
## converged
## # weights:  51
## initial  value 1125.701111
## iter  10 value 333.755895
## iter  20 value 332.020684
## final  value 332.018100
## converged
## # weights:  151
## initial  value 1235.783193
## iter  10 value 340.762804
## iter  20 value 332.101028
## iter  30 value 332.013976
## final  value 332.007449
## converged
## # weights:  251
## initial  value 1487.839420
## iter  10 value 339.910277
## iter  20 value 332.091199
## iter  30 value 332.009859
## iter  40 value 332.007002
## final  value 332.006938
## converged
## # weights:  51
## initial  value 882.186421
## final  value 322.000000
## converged
## # weights:  151
## initial  value 1451.677887
## final  value 322.000000
## converged
## # weights:  251
## initial  value 1394.242963
## final  value 322.000000
## converged
## # weights:  51
## initial  value 1301.193765
## iter  10 value 327.181575
## iter  20 value 325.108622
## iter  30 value 325.030502
## iter  40 value 325.030019
## iter  40 value 325.030017
## final  value 325.029975
## converged
## # weights:  151
## initial  value 897.137699
## iter  10 value 325.075913
## iter  20 value 323.790353
## iter  30 value 323.775016
## iter  40 value 323.763126
## iter  50 value 323.762887
## final  value 323.762853
## converged
## # weights:  251
## initial  value 494.265688
```

```
## iter  10 value 324.408487
## iter  20 value 323.528596
## iter  30 value 323.291493
## iter  40 value 323.280136
## final  value 323.278672
## converged
## # weights:  51
## initial  value 930.431331
## iter  10 value 323.993898
## iter  20 value 322.022988
## iter  30 value 322.010080
## final  value 322.009903
## converged
## # weights:  151
## initial  value 1216.314208
## iter  10 value 327.708698
## iter  20 value 322.065817
## iter  30 value 322.015288
## final  value 322.010044
## converged
## # weights:  251
## initial  value 560.080830
## iter  10 value 322.927980
## iter  20 value 322.010700
## iter  30 value 322.006890
## iter  40 value 322.004482
## final  value 322.004446
## converged
## # weights:  51
## initial  value 1206.189490
## final  value 318.000000
## converged
## # weights:  151
## initial  value 1349.959411
## final  value 318.000000
## converged
## # weights:  251
## initial  value 662.734707
## final  value 318.000000
## converged
## # weights:  51
## initial  value 895.399570
## iter  10 value 323.267692
## iter  20 value 321.211637
## iter  30 value 321.047395
## iter  40 value 321.021654
## final  value 321.021475
## converged
## # weights:  151
## initial  value 657.869207
## iter  10 value 320.782692
## iter  20 value 320.219430
## iter  30 value 319.758688
## iter  40 value 319.758249
```

```
## final  value 319.758231
## converged
## # weights:  251
## initial  value 1210.708088
## iter  10 value 321.259590
## iter  20 value 319.354258
## iter  30 value 319.276120
## final  value 319.275452
## converged
## # weights:  51
## initial  value 1590.272271
## iter  10 value 333.631545
## iter  20 value 318.180219
## iter  30 value 318.028071
## iter  40 value 318.017912
## iter  50 value 318.016045
## final  value 318.011822
## converged
## # weights:  151
## initial  value 1018.410996
## iter  10 value 321.194473
## iter  20 value 318.037842
## iter  30 value 318.010782
## final  value 318.009882
## converged
## # weights:  251
## initial  value 985.013867
## iter  10 value 324.927428
## iter  20 value 318.079868
## iter  30 value 318.006904
## final  value 318.006457
## converged
## # weights:  51
## initial  value 1082.229272
## final  value 331.000000
## converged
## # weights:  151
## initial  value 1836.796764
## final  value 331.000000
## converged
## # weights:  251
## initial  value 1759.550990
## final  value 331.000000
## converged
## # weights:  51
## initial  value 1150.178348
## iter  10 value 334.328645
## final  value 334.048808
## converged
## # weights:  151
## initial  value 831.341239
## iter  10 value 333.453002
## iter  20 value 332.781706
## iter  30 value 332.773176
```

```
## final  value 332.773141
## converged
## # weights:  251
## initial  value 2153.426146
## iter  10 value 336.006067
## iter  20 value 332.319617
## iter  30 value 332.286726
## iter  40 value 332.286123
## iter  50 value 332.285977
## iter  60 value 332.285889
## final  value 332.285879
## converged
## # weights:  51
## initial  value 1642.923849
## iter  10 value 335.342522
## iter  20 value 331.050066
## iter  30 value 331.028045
## final  value 331.018092
## converged
## # weights:  151
## initial  value 973.892158
## iter  10 value 333.486528
## iter  20 value 331.031159
## iter  30 value 331.016725
## iter  40 value 331.013972
## final  value 331.013701
## converged
## # weights:  251
## initial  value 1042.742275
## iter  10 value 338.690916
## iter  20 value 331.088670
## iter  30 value 331.008889
## final  value 331.006782
## converged
## # weights:  51
## initial  value 1040.554716
## final  value 333.000000
## converged
## # weights:  151
## initial  value 1276.233030
## final  value 333.000000
## converged
## # weights:  251
## initial  value 1365.556195
## final  value 333.000000
## converged
## # weights:  51
## initial  value 1343.311558
## iter  10 value 344.389605
## iter  20 value 336.236643
## iter  30 value 336.052935
## final  value 336.052927
## converged
## # weights:  151
```

```
## initial  value 1231.607197
## iter  10 value 338.873500
## iter  20 value 335.648174
## iter  30 value 334.834701
## iter  40 value 334.776078
## iter  50 value 334.775480
## iter  60 value 334.775379
## iter  60 value 334.775378
## iter  60 value 334.775378
## final  value 334.775378
## converged
## # weights:  251
## initial  value 1864.306664
## iter  10 value 341.652638
## iter  20 value 334.498949
## iter  30 value 334.406564
## iter  40 value 334.291197
## iter  50 value 334.288127
## iter  60 value 334.287718
## iter  70 value 334.287510
## final  value 334.287432
## converged
## # weights:  51
## initial  value 1140.826268
## iter  10 value 336.691195
## iter  20 value 333.042557
## iter  30 value 333.015638
## iter  40 value 333.009960
## final  value 333.009943
## converged
## # weights:  151
## initial  value 1278.533022
## iter  10 value 335.641811
## iter  20 value 333.030458
## iter  30 value 333.007838
## final  value 333.006997
## converged
## # weights:  251
## initial  value 1319.660094
## iter  10 value 342.732474
## iter  20 value 333.112208
## iter  30 value 333.010714
## final  value 333.006392
## converged
## # weights:  51
## initial  value 748.181265
## final  value 323.000000
## converged
## # weights:  151
## initial  value 1311.548226
## final  value 323.000000
## converged
## # weights:  251
## initial  value 1309.450559
```

```
## final  value 323.000000
## converged
## # weights:  51
## initial  value 759.184630
## iter  10 value 326.953901
## iter  20 value 326.032253
## iter  30 value 326.032124
## final  value 326.032095
## converged
## # weights:  151
## initial  value 1271.455589
## iter  10 value 328.459551
## iter  20 value 325.478416
## iter  30 value 324.950742
## iter  40 value 324.766429
## iter  50 value 324.764018
## iter  50 value 324.764016
## iter  50 value 324.764016
## final  value 324.764016
## converged
## # weights:  251
## initial  value 1737.159642
## iter  10 value 325.617076
## iter  20 value 324.540660
## iter  30 value 324.481264
## iter  40 value 324.355118
## iter  50 value 324.279896
## iter  60 value 324.279498
## final  value 324.279488
## converged
## # weights:  51
## initial  value 1029.772736
## iter  10 value 326.000529
## iter  20 value 323.035587
## iter  30 value 323.023789
## iter  40 value 323.012992
## final  value 323.009916
## converged
## # weights:  151
## initial  value 889.684033
## iter  10 value 323.890515
## iter  20 value 323.067244
## iter  30 value 323.012557
## iter  40 value 323.009265
## final  value 323.009053
## converged
## # weights:  251
## initial  value 1533.673181
## iter  10 value 328.949874
## iter  20 value 323.068597
## iter  30 value 323.009228
## iter  40 value 323.006973
## iter  40 value 323.006970
## iter  40 value 323.006969
```

```
## final  value 323.006969
## converged
## # weights:  51
## initial  value 848.218049
## final  value 328.000000
## converged
## # weights:  151
## initial  value 1180.151783
## final  value 328.000000
## converged
## # weights:  251
## initial  value 674.638359
## final  value 328.000000
## converged
## # weights:  51
## initial  value 1001.461929
## iter  10 value 335.312853
## iter  20 value 331.042608
## final  value 331.042585
## converged
## # weights:  151
## initial  value 1920.946250
## iter  10 value 332.771819
## iter  20 value 330.488698
## iter  30 value 329.821549
## iter  40 value 329.771909
## iter  50 value 329.769912
## final  value 329.769781
## converged
## # weights:  251
## initial  value 1952.811347
## iter  10 value 343.144165
## iter  20 value 329.910118
## iter  30 value 329.776259
## iter  40 value 329.291486
## iter  50 value 329.283518
## iter  50 value 329.283515
## iter  50 value 329.283512
## final  value 329.283512
## converged
## # weights:  51
## initial  value 1005.159586
## iter  10 value 329.485142
## iter  20 value 328.018484
## final  value 328.018070
## converged
## # weights:  151
## initial  value 1422.178440
## iter  10 value 334.659061
## iter  20 value 328.076774
## iter  30 value 328.009496
## final  value 328.007631
## converged
## # weights:  251
```

```
## initial  value 847.670669
## iter  10 value 330.648074
## iter  20 value 328.030530
## iter  30 value 328.006324
## final  value 328.005441
## converged
## # weights:  51
## initial  value 889.173926
## final  value 320.000000
## converged
## # weights:  151
## initial  value 922.951777
## final  value 320.000000
## converged
## # weights:  251
## initial  value 1315.355155
## final  value 320.000000
## converged
## # weights:  51
## initial  value 1525.868953
## iter  10 value 325.255681
## iter  20 value 323.029896
## final  value 323.025740
## converged
## # weights:  151
## initial  value 1084.235834
## iter  10 value 324.438784
## iter  20 value 322.101320
## iter  30 value 321.806244
## iter  40 value 321.776671
## iter  50 value 321.760935
## iter  60 value 321.760588
## final  value 321.760555
## converged
## # weights:  251
## initial  value 1321.493772
## iter  10 value 358.426025
## iter  20 value 322.084239
## iter  30 value 321.297933
## iter  40 value 321.279627
## iter  50 value 321.277529
## iter  60 value 321.277211
## final  value 321.277073
## converged
## # weights:  51
## initial  value 1199.981711
## iter  10 value 322.767196
## iter  20 value 320.031904
## final  value 320.010393
## converged
## # weights:  151
## initial  value 782.024981
## iter  10 value 321.948754
## iter  20 value 320.022468
```

```
## final  value 320.011635
## converged
## # weights:  251
## initial  value 1315.633008
## iter  10 value 323.989702
## iter  20 value 320.045998
## final  value 320.008975
## converged
## # weights:  51
## initial  value 958.634562
## final  value 330.000000
## converged
## # weights:  151
## initial  value 943.337451
## final  value 330.000000
## converged
## # weights:  251
## initial  value 1142.690579
## final  value 330.000000
## converged
## # weights:  51
## initial  value 1107.065558
## iter  10 value 337.929746
## iter  20 value 333.119479
## final  value 333.046725
## converged
## # weights:  151
## initial  value 1658.754601
## iter  10 value 335.888495
## iter  20 value 332.214649
## iter  30 value 331.945041
## iter  40 value 331.773785
## iter  50 value 331.771993
## final  value 331.771982
## converged
## # weights:  251
## initial  value 1149.360078
## iter  10 value 335.112203
## iter  20 value 331.827672
## iter  30 value 331.657718
## iter  40 value 331.474376
## iter  50 value 331.367537
## iter  60 value 331.285214
## final  value 331.285050
## converged
## # weights:  51
## initial  value 1585.834901
## iter  10 value 332.168518
## iter  20 value 330.025001
## final  value 330.018085
## converged
## # weights:  151
## initial  value 1441.433603
## iter  10 value 336.520700
```

```
## iter  20 value 330.075179
## final  value 330.010823
## converged
## # weights:  251
## initial  value 772.647260
## iter  10 value 335.828215
## iter  20 value 330.067195
## iter  30 value 330.014703
## iter  40 value 330.009275
## iter  50 value 330.005895
## final  value 330.005668
## converged
## # weights:  51
## initial  value 1525.913580
## final  value 314.000000
## converged
## # weights:  151
## initial  value 1597.797184
## final  value 314.000000
## converged
## # weights:  251
## initial  value 932.994169
## final  value 314.000000
## converged
## # weights:  51
## initial  value 1473.116990
## iter  10 value 324.720709
## iter  20 value 317.028492
## final  value 317.012867
## converged
## # weights:  151
## initial  value 1016.376104
## iter  10 value 317.563489
## iter  20 value 316.436234
## iter  30 value 316.203549
## iter  40 value 315.755314
## iter  50 value 315.753531
## final  value 315.753525
## converged
## # weights:  251
## initial  value 1451.503824
## iter  10 value 320.456958
## iter  20 value 315.674279
## iter  30 value 315.570375
## iter  40 value 315.532468
## iter  50 value 315.398563
## iter  60 value 315.296186
## iter  70 value 315.274663
## iter  80 value 315.272919
## iter  90 value 315.272587
## iter 100 value 315.272425
## final  value 315.272425
## stopped after 100 iterations
## # weights:  51
```

```
## initial  value 1509.279465
## iter  10 value 316.185095
## iter  20 value 314.025192
## final  value 314.018348
## converged
## # weights:  151
## initial  value 696.620988
## iter  10 value 315.821417
## iter  20 value 314.021000
## final  value 314.007135
## converged
## # weights:  251
## initial  value 906.443153
## iter  10 value 320.339784
## iter  20 value 314.073093
## iter  30 value 314.007948
## final  value 314.004694
## converged
## # weights:  51
## initial  value 960.817626
## final  value 329.000000
## converged
## # weights:  151
## initial  value 1581.456009
## final  value 329.000000
## converged
## # weights:  251
## initial  value 748.748491
## final  value 329.000000
## converged
## # weights:  51
## initial  value 1156.002769
## iter  10 value 332.639075
## iter  20 value 332.045043
## final  value 332.044667
## converged
## # weights:  151
## initial  value 892.939173
## iter  10 value 341.759880
## iter  20 value 330.915128
## iter  30 value 330.792058
## iter  40 value 330.771137
## final  value 330.770882
## converged
## # weights:  251
## initial  value 1422.756426
## iter  10 value 331.817044
## iter  20 value 330.639264
## iter  30 value 330.376804
## iter  40 value 330.286309
## final  value 330.284407
## converged
## # weights:  51
## initial  value 1053.125678
```

```
## iter  10 value 330.942538
## iter  20 value 329.023248
## final  value 329.021425
## converged
## # weights:  151
## initial  value 1643.585088
## iter  10 value 333.424434
## iter  20 value 329.051010
## iter  30 value 329.010284
## iter  40 value 329.007782
## final  value 329.007239
## converged
## # weights:  251
## initial  value 881.659191
## iter  10 value 332.322021
## iter  20 value 329.038300
## iter  30 value 329.007423
## iter  30 value 329.007421
## iter  30 value 329.007420
## final  value 329.007420
## converged
## # weights:  51
## initial  value 794.354514
## final  value 318.000000
## converged
## # weights:  151
## initial  value 1406.730063
## final  value 318.000000
## converged
## # weights:  251
## initial  value 423.991374
## final  value 318.000000
## converged
## # weights:  51
## initial  value 1000.011481
## iter  10 value 323.614833
## iter  20 value 321.021469
## iter  20 value 321.021466
## iter  20 value 321.021466
## final  value 321.021466
## converged
## # weights:  151
## initial  value 668.108351
## iter  10 value 320.273499
## iter  20 value 320.180205
## iter  30 value 319.762110
## iter  40 value 319.758478
## final  value 319.758227
## converged
## # weights:  251
## initial  value 1407.432669
## iter  10 value 339.029209
## iter  20 value 319.843759
## iter  30 value 319.313936
```

```
## iter  40 value 319.285642
## iter  50 value 319.276475
## iter  60 value 319.275920
## iter  70 value 319.275447
## iter  70 value 319.275445
## final  value 319.275422
## converged
## # weights:  51
## initial  value 636.623753
## iter  10 value 318.975109
## iter  20 value 318.011500
## final  value 318.010230
## converged
## # weights:  151
## initial  value 1445.041614
## iter  10 value 326.284228
## iter  20 value 318.095511
## iter  30 value 318.005428
## final  value 318.005413
## converged
## # weights:  251
## initial  value 1311.330977
## iter  10 value 329.910219
## iter  20 value 318.137315
## iter  30 value 318.004849
## iter  40 value 318.004315
## final  value 318.004170
## converged
## # weights:  51
## initial  value 1815.549630
## final  value 327.000000
## converged
## # weights:  151
## initial  value 651.638813
## final  value 327.000000
## converged
## # weights:  251
## initial  value 811.334831
## final  value 327.000000
## converged
## # weights:  51
## initial  value 1207.584856
## iter  10 value 330.920126
## iter  20 value 330.096761
## iter  30 value 330.041447
## iter  40 value 330.040714
## iter  40 value 330.040711
## final  value 330.040497
## converged
## # weights:  151
## initial  value 912.886453
## iter  10 value 331.067503
## iter  20 value 328.916900
## iter  30 value 328.775154
```

```
## iter  40 value 328.769275
## iter  50 value 328.768785
## iter  60 value 328.768631
## final  value 328.768605
## converged
## # weights:  251
## initial  value 561.194971
## iter  10 value 330.566219
## iter  20 value 328.330472
## iter  30 value 328.290894
## iter  40 value 328.285029
## iter  50 value 328.283523
## iter  60 value 328.283076
## iter  70 value 328.282708
## final  value 328.282703
## converged
## # weights:  51
## initial  value 1548.178337
## iter  10 value 329.333485
## iter  20 value 327.026903
## final  value 327.019280
## converged
## # weights:  151
## initial  value 1457.452954
## iter  10 value 335.102121
## iter  20 value 327.093411
## iter  30 value 327.016738
## iter  40 value 327.010001
## iter  50 value 327.005842
## final  value 327.005467
## converged
## # weights:  251
## initial  value 1510.605533
## iter  10 value 329.762810
## iter  20 value 327.031853
## final  value 327.013165
## converged
## # weights:  51
## initial  value 1214.631909
## final  value 331.000000
## converged
## # weights:  151
## initial  value 951.953259
## final  value 331.000000
## converged
## # weights:  251
## initial  value 485.975191
## final  value 331.000000
## converged
## # weights:  51
## initial  value 1468.849952
## iter  10 value 335.297690
## iter  20 value 334.052963
## iter  30 value 334.049244
```

```
## iter  40 value 334.048914
## final  value 334.048809
## converged
## # weights:  151
## initial  value 837.129438
## iter  10 value 335.593302
## iter  20 value 334.215042
## iter  30 value 332.944287
## iter  40 value 332.794361
## iter  50 value 332.788097
## iter  60 value 332.780390
## iter  70 value 332.775370
## iter  80 value 332.774090
## iter  90 value 332.773806
## iter 100 value 332.773683
## final  value 332.773683
## stopped after 100 iterations
## # weights:  251
## initial  value 598.343027
## iter  10 value 332.419710
## iter  20 value 332.286046
## final  value 332.285879
## converged
## # weights:  51
## initial  value 1584.295597
## iter  10 value 334.318513
## iter  20 value 331.038260
## final  value 331.010164
## converged
## # weights:  151
## initial  value 984.738372
## iter  10 value 336.654876
## iter  20 value 331.065196
## iter  30 value 331.008933
## final  value 331.007384
## converged
## # weights:  251
## initial  value 1336.430531
## iter  10 value 339.324644
## iter  20 value 331.095977
## iter  30 value 331.011927
## final  value 331.006988
## converged
## # weights:  51
## initial  value 1343.511046
## final  value 325.000000
## converged
## # weights:  151
## initial  value 2104.985436
## final  value 325.000000
## converged
## # weights:  251
## initial  value 903.936213
## final  value 325.000000
```

```
## converged
## # weights:  51
## initial  value 615.091912
## iter  10 value 329.139688
## iter  20 value 328.036394
## final  value 328.036307
## converged
## # weights:  151
## initial  value 855.193975
## iter  10 value 331.178795
## iter  20 value 326.842371
## iter  30 value 326.769893
## iter  40 value 326.766436
## iter  50 value 326.766325
## iter  50 value 326.766322
## iter  50 value 326.766320
## final  value 326.766320
## converged
## # weights:  251
## initial  value 831.435654
## iter  10 value 331.012178
## iter  20 value 326.520090
## iter  30 value 326.482138
## iter  40 value 326.481516
## iter  50 value 326.416701
## iter  60 value 326.283086
## iter  70 value 326.281115
## final  value 326.281104
## converged
## # weights:  51
## initial  value 666.247038
## iter  10 value 325.821549
## iter  20 value 325.010309
## final  value 325.010166
## converged
## # weights:  151
## initial  value 1412.212934
## iter  10 value 331.154837
## iter  20 value 325.070960
## final  value 325.006718
## converged
## # weights:  251
## initial  value 1400.230408
## iter  10 value 334.431977
## iter  20 value 325.108743
## iter  30 value 325.018538
## iter  40 value 325.008107
## final  value 325.006140
## converged
## # weights:  51
## initial  value 1073.463538
## final  value 320.000000
## converged
## # weights:  151
```

```
## initial  value 902.773997
## final  value 320.000000
## converged
## # weights:  251
## initial  value 1149.067140
## final  value 320.000000
## converged
## # weights:  51
## initial  value 1173.845376
## iter  10 value 324.884978
## iter  20 value 323.028144
## final  value 323.025743
## converged
## # weights:  151
## initial  value 1271.128072
## iter  10 value 325.384590
## iter  20 value 322.211032
## iter  30 value 322.121346
## iter  40 value 321.773764
## iter  50 value 321.767897
## iter  60 value 321.760564
## iter  60 value 321.760562
## iter  60 value 321.760562
## final  value 321.760562
## converged
## # weights:  251
## initial  value 1419.232371
## iter  10 value 328.109213
## iter  20 value 321.590455
## iter  30 value 321.422550
## iter  40 value 321.277212
## final  value 321.277083
## converged
## # weights:  51
## initial  value 1524.717494
## iter  10 value 323.152047
## iter  20 value 320.036341
## iter  30 value 320.022952
## iter  40 value 320.017795
## final  value 320.017249
## converged
## # weights:  151
## initial  value 1239.669821
## iter  10 value 325.215922
## iter  20 value 320.060135
## final  value 320.013420
## converged
## # weights:  251
## initial  value 1220.965944
## iter  10 value 326.913801
## iter  20 value 320.079711
## iter  30 value 320.007940
## final  value 320.005875
## converged
```

```
## # weights:  51
## initial  value 1468.474875
## final  value 326.000000
## converged
## # weights:  151
## initial  value 1454.587525
## final  value 326.000000
## converged
## # weights:  251
## initial  value 1174.144905
## final  value 326.000000
## converged
## # weights:  51
## initial  value 1559.553069
## iter  10 value 329.306213
## final  value 329.038407
## converged
## # weights:  151
## initial  value 804.418645
## iter  10 value 339.607635
## iter  20 value 327.927053
## iter  30 value 327.771426
## iter  40 value 327.767947
## iter  50 value 327.767642
## iter  60 value 327.767575
## iter  70 value 327.767476
## iter  70 value 327.767472
## iter  70 value 327.767472
## final  value 327.767472
## converged
## # weights:  251
## initial  value 1190.171338
## iter  10 value 328.733240
## iter  20 value 327.439709
## iter  30 value 327.298268
## iter  40 value 327.282817
## iter  50 value 327.282083
## final  value 327.281925
## converged
## # weights:  51
## initial  value 1517.968833
## iter  10 value 328.757344
## iter  20 value 326.031790
## iter  30 value 326.011491
## iter  40 value 326.009922
## iter  40 value 326.009919
## iter  40 value 326.009919
## final  value 326.009919
## converged
## # weights:  151
## initial  value 1323.430415
## iter  10 value 330.639673
## iter  20 value 326.053492
## iter  30 value 326.013479
```

```
## final  value 326.010018
## converged
## # weights:  251
## initial  value 986.898653
## iter  10 value 330.690994
## iter  20 value 326.054083
## iter  30 value 326.016017
## iter  40 value 326.005673
## final  value 326.005634
## converged
## # weights:  51
## initial  value 1085.069016
## final  value 320.000000
## converged
## # weights:  151
## initial  value 1377.997285
## final  value 320.000000
## converged
## # weights:  251
## initial  value 921.629608
## final  value 320.000000
## converged
## # weights:  51
## initial  value 1518.178627
## iter  10 value 324.564757
## iter  20 value 323.027411
## final  value 323.025743
## converged
## # weights:  151
## initial  value 1357.573840
## iter  10 value 323.669212
## iter  20 value 322.233077
## iter  30 value 322.204347
## iter  40 value 321.762066
## iter  50 value 321.761356
## iter  60 value 321.760569
## final  value 321.760562
## converged
## # weights:  251
## initial  value 1124.651771
## iter  10 value 344.285489
## iter  20 value 321.776107
## iter  30 value 321.646170
## iter  40 value 321.398620
## iter  50 value 321.285437
## iter  60 value 321.278512
## iter  70 value 321.277992
## iter  80 value 321.277721
## iter  90 value 321.277493
## iter 100 value 321.277393
## final  value 321.277393
## stopped after 100 iterations
## # weights:  51
## initial  value 807.557267
```

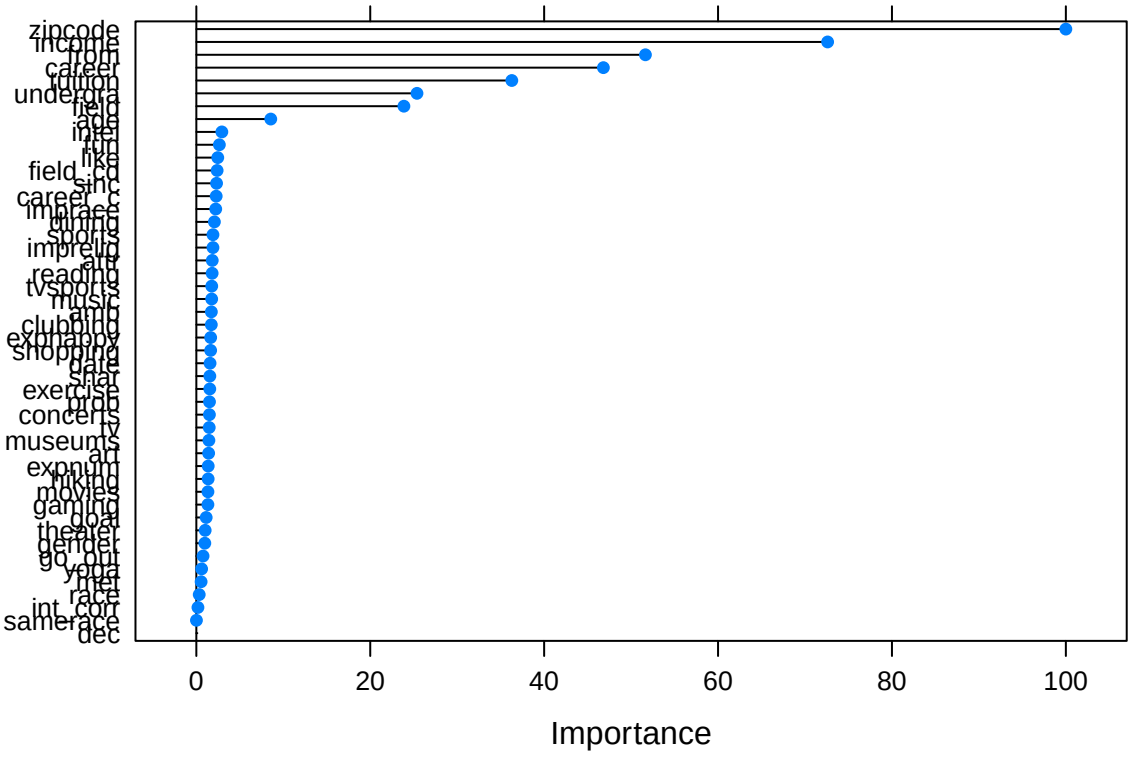
```
## iter  10 value 321.043016
## iter  20 value 320.013438
## final  value 320.013376
## converged
## # weights:  151
## initial  value 1713.586610
## iter  10 value 324.103166
## iter  20 value 320.047306
## iter  30 value 320.009970
## iter  30 value 320.009967
## iter  30 value 320.009967
## final  value 320.009967
## converged
## # weights:  251
## initial  value 549.779087
## iter  10 value 321.029680
## iter  20 value 320.011871
## final  value 320.005067
## converged
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
## # weights:  51
## initial  value 717.694399
## final  value 360.000000
## converged
```

```
importance.nn <- varImp(model.nn)
```

```
plot(importance.nn)
```



Also tried to fit the model using LDA and QDA. The results for LDA are as below, whereas obtained rank deficiency error with QDA.

```
library(MASS)
set1 = set1[, !names(set1) %in% c("gender")]
set2 = set2[, !names(set2) %in% c("gender")]
set3 = set3[, !names(set3) %in% c("gender")]
setMale = setMale[, !names(setMale) %in% c("gender")]

## LDA
# Fitting onto the training set
lda.fit=lda(match~., data=set1)
# Test Set Error
lda.predict=predict(lda.fit,set3)
lda.class=lda.predict$class
mean(lda.class!=set3$match)
```

```
## [1] 0.1508165
```

```
table(lda.class, set3$match)
```

```
##
## lda.class   1   2
##         1 836 114
##         2  43  48
```

```
# Testing on male dataset
lda.predict=predict(lda.fit,setMale)
lda.class=lda.predict$class
mean(lda.class!=setMale$match)
```

```
## [1] 0.1490653
```

```
table(lda.class, setMale$match)
```

```
##
## lda.class    1    2
##         1 3362  537
##         2   77  143
```

Using Naive Bayes to fit, worst so far

```
library(e1071)

# Fitting onto the training set
nb.fit=naiveBayes(match~., data=set1)
# Test Set Error
nb.class=predict(nb.fit,set3)
mean(nb.class!=set3$match)
```

```
## [1] 0.2997118
```

```
table(nb.class, set3$match)
```

```
##
## nb.class   1   2
##        1 576   9
##        2 303 153
```

```
# Testing on male dataset
nb.class=predict(nb.fit,setMale)
mean(nb.class!=setMale$match)
```

```
## [1] 0.1920369
```

```
table(nb.class, setMale$match)
```

```
##
## nb.class    1    2
##        1 2703   55
##        2  736  625
```

```
library(tree)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

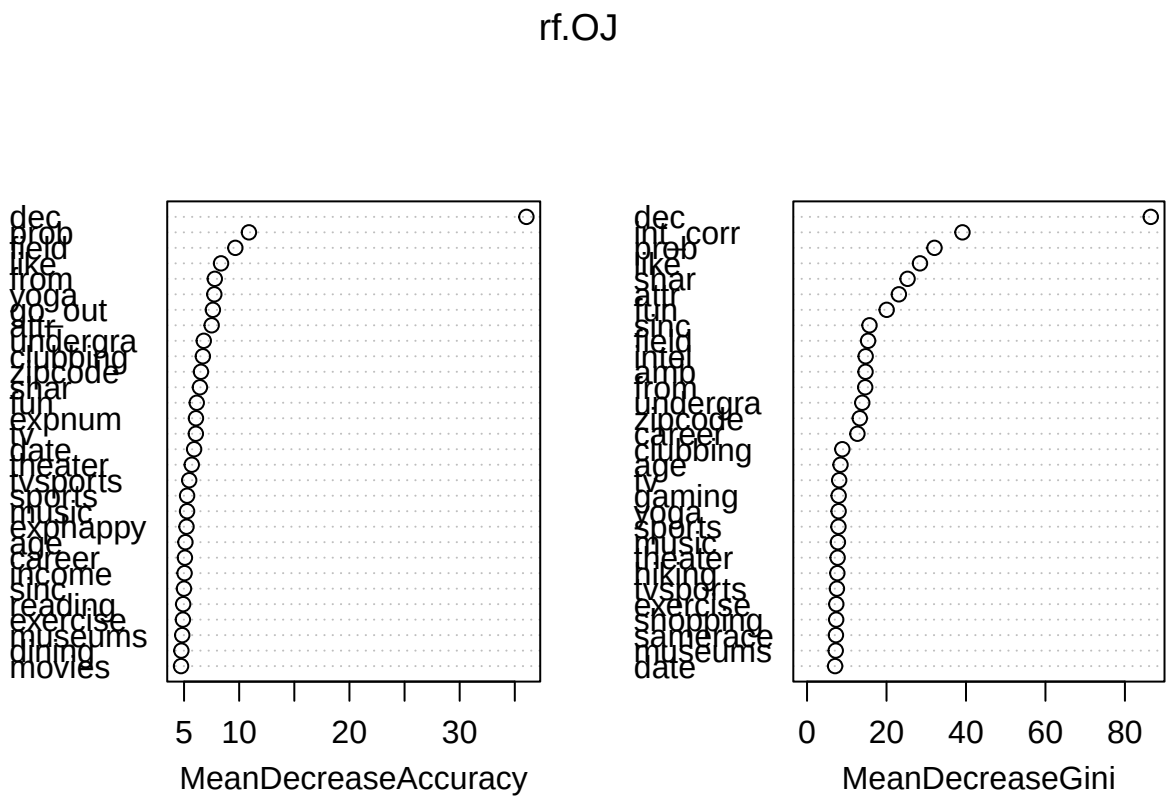
```
OJ.train = set1
OJ.test = set3
num.tree=200
rf.OJ = randomForest(as.factor(match) ~ ., data=OJ.train,
                     mtry=10,
                     ntree  = num.tree, importance=TRUE, xtest=OJ.test[, -1],
                     ytest=as.factor(OJ.test$match), keep.forest=TRUE)
```

```
rf.OJ
```

```
##
## Call:
##  randomForest(formula = as.factor(match) ~ ., data = OJ.train,      mtry = 10, ntree = num.tree, imp
##                Type of random forest: classification
##                      Number of trees: 200
```

```
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 15.47%
## Confusion matrix:
##      1   2 class.error
## 1 1625  97  0.05632985
## 2  225 135  0.62500000
##                 Test set error rate: 15.75%
## Confusion matrix:
##     1  2 class.error
## 1 829 50  0.05688282
## 2 114 48  0.70370370
```

```
varImpPlot(rf.OJ)
```



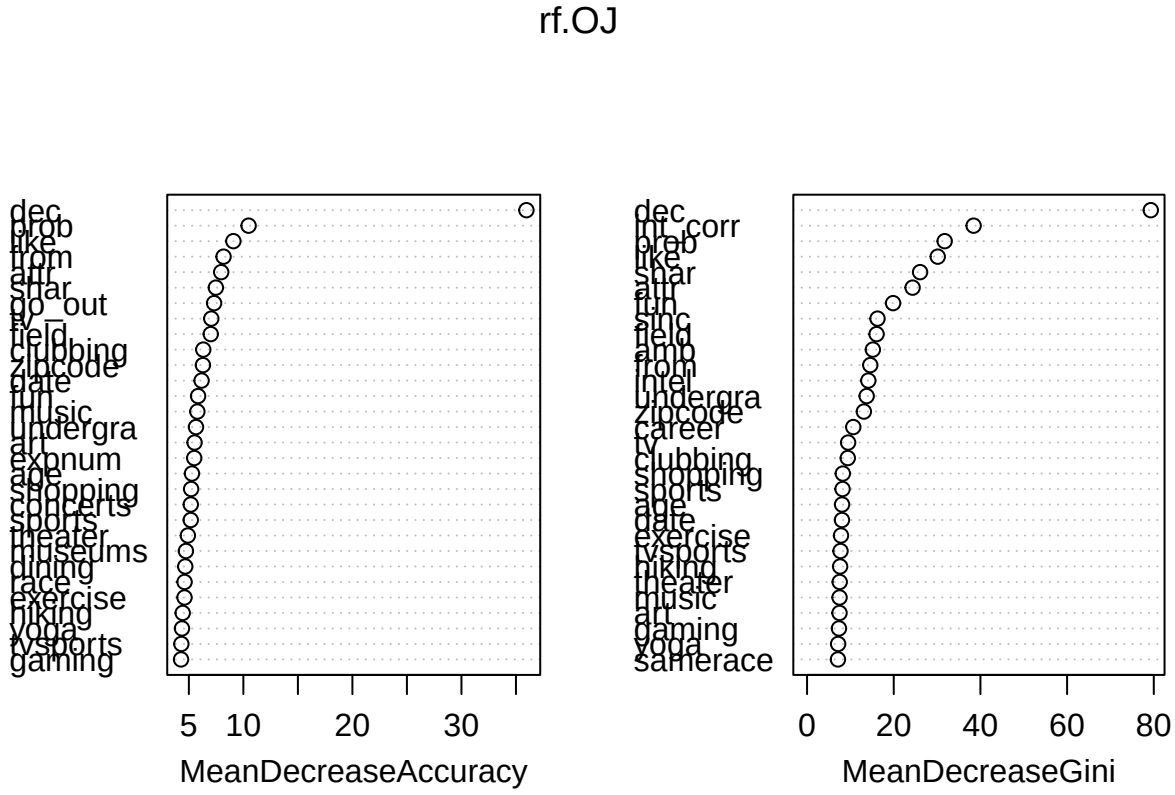

```
rf.OJ = randomForest(as.factor(match) ~ ., data=OJ.train,
                     mtry=10,
                     ntree  = num.tree, importance=TRUE, xtest=setMale[,-1],
                     ytest=as.factor(setMale$match), keep.forest=TRUE)
```

```
rf.OJ
```

```
##
## Call:
##  randomForest(formula = as.factor(match) ~ ., data = OJ.train,      mtry = 10, ntree = num.tree, imp
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 15.95%
```

```
## Confusion matrix:
##      1   2 class.error
## 1 1616 106  0.06155633
## 2  226 134  0.62777778
##                 Test set error rate: 15.37%
## Confusion matrix:
##      1  2 class.error
## 1 3392 47  0.01366676
## 2  586 94  0.86176471
```

```
varImpPlot(rf.OJ)
```



simple linear regression model

```
simp.reg = glm(as.factor(match) ~ ., data=set1, family = binomial)
summary(simp.reg)
```

```
##
## Call:
## glm(formula = as.factor(match) ~ ., family = binomial, data = set1)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.81915  -0.62222  -0.00005  -0.00003   2.38151
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.160e+01  1.024e+03  -0.041 0.967596
```

```
## int_corr     2.228e-01  2.877e-01   0.774 0.438692
## samerace     1.586e-01  1.571e-01   1.010 0.312644
## age         -1.649e-02  2.446e-02  -0.674 0.500103
## field       -4.277e-03  1.201e-03  -3.561 0.000369 ***
## field_cd    -1.707e-02  2.540e-02  -0.672 0.501485
## undergra     2.821e-04  1.607e-03   0.176 0.860604
## tuition     -4.148e-03  2.726e-03  -1.522 0.128034
## race        -9.663e-02  7.048e-02  -1.371 0.170373
## imprace      6.492e-03  3.638e-02   0.178 0.858373
## imprelig     1.735e-02  3.370e-02   0.515 0.606575
## from        -4.783e-04  1.134e-03  -0.422 0.673132
## zipcode      9.899e-04  5.826e-04   1.699 0.089319 .
## income      -3.669e-05  1.056e-03  -0.035 0.972292
## goal         7.549e-02  5.597e-02   1.349 0.177428
## date        -5.828e-02  6.153e-02  -0.947 0.343496
## go_out      -1.621e-01  8.330e-02  -1.946 0.051713 .
## career      -4.454e-04  7.803e-04  -0.571 0.568145
## career_c     1.439e-02  2.795e-02   0.515 0.606570
## sports       4.392e-02  4.031e-02   1.090 0.275836
## tvsports    -4.298e-02  3.485e-02  -1.233 0.217539
## exercise     3.261e-03  3.417e-02   0.095 0.923969
## dining       5.243e-02  5.406e-02   0.970 0.332047
## museums     -1.248e-01  7.911e-02  -1.578 0.114571
## art          5.933e-02  6.974e-02   0.851 0.394898
## hiking      -1.424e-03  3.307e-02  -0.043 0.965658
## gaming       7.920e-03  3.659e-02   0.216 0.828646
## clubbing     2.960e-02  3.452e-02   0.857 0.391181
## reading      2.011e-02  4.413e-02   0.456 0.648635
## tv          -5.665e-02  4.137e-02  -1.369 0.170867
## theater     -5.695e-02  4.781e-02  -1.191 0.233549
## movies       6.693e-02  5.352e-02   1.250 0.211127
## concerts     6.648e-03  4.778e-02   0.139 0.889333
## music        6.655e-02  5.098e-02   1.305 0.191784
## shopping    -1.398e-02  4.058e-02  -0.345 0.730470
## yoga         7.405e-03  3.248e-02   0.228 0.819681
## exphappy    -5.146e-02  5.278e-02  -0.975 0.329552
## expnum       4.838e-02  3.054e-02   1.584 0.113179
## dec          1.975e+01  5.120e+02   0.039 0.969221
## attr         1.034e-02  7.212e-02   0.143 0.885934
## sinc         5.183e-02  7.790e-02   0.665 0.505813
## intel        1.848e-01  8.547e-02   2.163 0.030576 *
## fun          1.958e-02  6.977e-02   0.281 0.778983
## amb         -1.748e-01  7.050e-02  -2.479 0.013166 *
## shar         1.115e-01  5.677e-02   1.964 0.049542 *
## like        -2.003e-02  9.247e-02  -0.217 0.828545
## prob         1.757e-01  4.856e-02   3.618 0.000297 ***
## met         -2.101e-04  8.616e-02  -0.002 0.998055
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1917.4  on 2081  degrees of freedom
## Residual deviance: 1173.5  on 2034  degrees of freedom
```

```
## AIC: 1269.5
##
## Number of Fisher Scoring iterations: 19
```

```
pred.linear = predict(simp.reg, newdata = set3)
pred.prob = exp(pred.linear)/(1+exp(pred.linear))
pred.class = ifelse(pred.prob >= 0.5, 2, 1)
table(factor(pred.class), true.class = as.factor(set3$match))
```

```
##    true.class
##       1   2
##   1 836 117
##   2  43  45
```

```
mean(pred.class != factor(set3$match))
```

```
## [1] 0.1536984
```

```
pred.linear = predict(simp.reg, newdata = setMale)
pred.prob = exp(pred.linear)/(1+exp(pred.linear))
pred.class = ifelse(pred.prob >= 0.5, 2, 1)
table(factor(pred.class), true.class = as.factor(setMale$match))
```

```
##    true.class
##        1    2
##   1 3349  517
##   2   90  163
```

```
mean(pred.class != factor(setMale$match))
```

```
## [1] 0.1473659
```

Project Code 2

2022-12-06

```
Dating = read.csv("Dating.csv")
Dating = Dating[complete.cases(Dating$gender),]
Dating = Dating[complete.cases(Dating$age),]
rownames(Dating) = 1:nrow(Dating)
```

Data Cleaning

```
allNA = c()
fa = c()
ind = c()

too.many.miss = 0

for (i in 1:dim(Dating)[2]) {
  ind = c(ind, i)
  column = Dating[,i]

  if (is.numeric(column)) {
    check.col = na.omit(column)
    # we check if numeric value
    max = max(check.col)
    min = min(check.col)

    if ((!is.na(max) & !is.na(min)) & (max == 1 & min == 0)) {
      fa = c(fa, "0/1")
      # it is categorical with 0/1
      new.col = Dating[,i]
      new.col[is.na(new.col) | new.col == ""] = "Other"
      Dating[,i] = as.integer(factor(new.col))

    } else {
      fa = c(fa, "number")
      # it is continuous number
      new.col = Dating[,i]

      # 0 - dim(t(na.omit(new.col)))[2] + dim(Dating)[1] > too.many.miss
      if (FALSE) {
        Dating = subset(Dating, select=-c(i))
      } else {
        # may change here if you like
        mean.val = median(check.col)
        new.col[is.na(new.col)] = mean.val
        Dating[,i] = new.col
      }
```

```
    }
  } else {

    fa = c(fa, "cat")
    # we check if categorical value
    new.col = Dating[,i]
    new.col[is.na(new.col) | new.col == ""] = "Other"
    Dating[,i] = as.integer(factor(new.col))
  }
}

mistake = c()

for (i in 1:dim(Dating)[2]) {
  if (dim(t(Dating[,i]))[2] != dim(t(na.omit(Dating[,i])))[2]) {
    mistake = c(mistake, i)
  }
}

femaleDating = Dating[Dating$gender == "2",]
maleDating = Dating[Dating$gender == "1",]
```

The following variables have the same value hence the columns will be dropped.

```
delete_col = c("amb5_3", "fun5_3", "intel5_3", "sinc5_3", "attr5_3", "shar2_3", "amb2_3", "fun2_3", "in
               "sinc4_3", "attr4_3", "shar7_3", "amb7_3", "fun7_3", "intel7_3",  "sinc7_3",  "attr7_3",
               "attr5_2", "shar2_2", "amb2_2", "fun2_2", "intel2_2", "sinc2_2",  "attr2_2", "shar4_2", "
               "intel7_2", "sinc7_2", "attr7_2", "amb3_s", "fun3_s", "intel3_s", "sinc3_s", "attr3_s", "
               "fun5_1", "intel5_1", "sinc5_1", "attr5_1", "shar4_1", "amb4_1", "fun4_1", "intel4_1", "s

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col]
maleDating = maleDating[, !names(maleDating) %in% delete_col]
```

We hope to study the influencing factors of the successful match, and the other half's evaluation data will affect the result of the match, so it is removed

```
delete_col2 = c("age_o","race_o","pf_o_att","pf_o_sin","pf_o_int","pf_o_fun","pf_o_amb","pf_o_sha","dec

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col2]
maleDating = maleDating[, !names(maleDating) %in% delete_col2]
```

The following variables have the same value hence the columns will be dropped.

```
delete_col = c("amb5_3", "fun5_3", "intel5_3", "sinc5_3", "attr5_3", "shar2_3", "amb2_3", "fun2_3", "in
               "sinc4_3", "attr4_3", "shar7_3", "amb7_3", "fun7_3", "intel7_3",  "sinc7_3",  "attr7_3",
               "attr5_2", "shar2_2", "amb2_2", "fun2_2", "intel2_2", "sinc2_2",  "attr2_2", "shar4_2", "
               "intel7_2", "sinc7_2", "attr7_2", "amb3_s", "fun3_s", "intel3_s", "sinc3_s", "attr3_s", "
               "fun5_1", "intel5_1", "sinc5_1", "attr5_1", "shar4_1", "amb4_1", "fun4_1", "intel4_1", "s

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col]
maleDating = maleDating[, !names(maleDating) %in% delete_col]
Dating = Dating[, !names(Dating) %in% delete_col]
```

We hope to study the influencing factors of the successful match, and the other half's evaluation data will affect the result of the match, so it is removed

```
delete_col2 = c("age_o","race_o","pf_o_att","pf_o_sin","pf_o_int","pf_o_fun","pf_o_amb","pf_o_sha","dec

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col2]
maleDating = maleDating[, !names(maleDating) %in% delete_col2]
Dating = Dating[, !names(Dating) %in% delete_col2]
```

Obviously, a high evaluation of both parties leads to a high matching success rate, but we want to study which characteristics of ourselves lead to a higher matching rate, so we proceed to drop these columns

```
delete_col3 = c("attr1_1",  "sinc1_1",  "intel1_1", "fun1_1",   "amb1_1"  ,
 "shar1_1" , "attr2_1",  "sinc2_1" , "intel2_1","fun2_1",   "amb2_1" ,  "shar2_1" ,
"attr3_1" , "sinc3_1" , "fun3_1" ,  "intel3_1", "amb3_1","numdat_2" ,"amb7_2"  , "attr1_2" ,
"sinc1_2",  "intel1_2" ,"fun1_2" ,  "amb1_2" ,  "shar1_2" , "attr3_2" , "sinc3_2",
"intel3_2", "fun3_2" ,  "amb3_2"  , "you_call", "them_cal" ,"date_3" ,  "attr1_3" ,
 "sinc1_3" , "intel1_3","fun1_3"  , "amb1_3"  , "shar1_3" , "attr3_3" , "sinc3_3" ,
 "intel3_3", "fun3_3"  , "amb3_3" , "satis_2","iid",   "id","idg", "condtn" ,  "wave","round"  , "posit

femaleDating = femaleDating[, !names(femaleDating) %in% delete_col3]
maleDating = maleDating[, !names(maleDating) %in% delete_col3]
Dating = Dating[, !names(Dating) %in% delete_col3]
```

```
set.seed(11)
train = sample(1:nrow(maleDating), nrow(maleDating)*3/4)

setmale1 = maleDating[train,]
setmale3 = maleDating[-train,]

train.1 = sample(1:nrow(femaleDating), nrow(femaleDating)*3/4)

setfemale1 = femaleDating[train.1,]
setfemale3 = femaleDating[-train.1,]

simp.reg = glm(as.factor(match-1) ~ ., data=setfemale1, family = binomial)
#summary(simp.reg)
pred.linear.female = predict(simp.reg, newdata = setfemale3)
```

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

```
pred.prob = exp(pred.linear.female)/(1+exp(pred.linear.female))
pred.class = ifelse(pred.prob >= 0.5, 1, 0)
table(factor(pred.class), true.class = as.factor(setfemale3$match-1))
```

```
##    true.class
##       0   1
##   0 829 127
##   1  34  51
```

```
mean(pred.class != factor(setfemale3$match-1))
```

```
## [1] 0.154659
```

```
simp.reg.male= glm(as.factor(match-1) ~ ., data=setmale1, family = binomial)
pred.linear.male = predict(simp.reg.male, newdata = setmale3)
```

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

```r
pred.prob = exp(pred.linear.male)/(1+exp(pred.linear.male))
pred.class = ifelse(pred.prob >= 0.5, 1, 0)
table(factor(pred.class), true.class = as.factor(setmale3$match-1))
```

```
##    true.class
##       0   1
##   0 797  83
##   1  69  81
```

```r
mean(pred.class != factor(setmale3$match-1))
```

```
## [1] 0.1475728
```

```r
train.all = sample(1:nrow(Dating), nrow(Dating)*3/4)
setall1 = Dating[train.all,]
setall3 = Dating[-train.all,]

simp.reg.all= glm(as.factor(match-1) ~ ., data=setall1, family = binomial)
pred.linear.all = predict(simp.reg.all, newdata = setall3)
pred.prob = exp(pred.linear.all)/(1+exp(pred.linear.all))
pred.class = ifelse(pred.prob >= 0.5, 1, 0)
table(factor(pred.class), true.class = as.factor(setall3$match-1))
```

```
##    true.class
##       0    1
##   0 1653  206
##   1   73  139
```

```r
mean(pred.class != factor(setall3$match-1))
```

```
## [1] 0.1347175
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
par(mfrow=c(1,3))
plot.roc(femaleDating$match[-train.1], pred.linear.female, print.auc= T,
         main = "Logistic Regression Female")
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```r
plot.roc(maleDating$match[-train], pred.linear.male, print.auc= T,
         main = "Logistic Regression Male")
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

```r
plot.roc(Dating$match[-train.all], pred.linear.all, print.auc= T,
         main = "Logistic Regression All")
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```