CLassification using ML

Mahedi Hasan, PhD Candidate in Statistical Science, WSU

Email: mdmahedi.hasan@wsu.edu

10/26/2022

Todays Plan:

We have plan to cover:

- Common Techniques in ML for classification
  - SVM, KNN, DT, NN etc.
- Implement in R and compare the accuracy

A little about R

R is a system for statistical computation and graphics. - R is an open source language - R is an interpreted language - R can run on Windows, Mac, and Linux operating system. - R works based on its packages (mostly) - R is case sensitive

Installing R and RStudio

- You can download R from -https://cran.r-project.org
- RStudio is an integrated development environment (IDE) for R.

R Package and its Installation

- A Package is basically the compilation of a set of codes, data and instructions
- R has more than 17,000 packages (ref: r-cran) by now and adding…
- Packages are developed by the R users
- Different packages compiled different methos and has uses accordingly

```
# install.packages("name of the package")
# library("name of the package")
```

About the dataset

- Bank loan data
- Objective: "If you are a loan officer at a bank, then you want to be able to identify characteristics that are indicative of people who are likely to default on loans, and use those characteristics to identify good and bad credit risks"

Get the data

- This dataset is one of the example data sets used in SPSS

```
bankloan = read.csv("https://raw.githubusercontent.com/mahedihasanstat/data/main/bankloan.csv", header =
```

Lets look at the dataset closely

```r
dim(bankloan)
```

```
## [1] 700   9
```

```r
names(bankloan)
```

```
## [1] "Age"             "Education"       "EmployDuration"  "ResidDuration"
## [5] "Income"          "DebtIncomeRatio" "CreditDebt"      "OtherDebt"
## [9] "Default"
```

```r
head(bankloan)
```

```
##   Age Education EmployDuration ResidDuration Income DebtIncomeRatio CreditDebt
## 1  41         3             17            12    176             9.3  11.359392
## 2  27         1             10             6     31            17.3   1.362202
## 3  40         1             15            14     55             5.5   0.856075
## 4  41         1             15            14    120             2.9   2.658720
## 5  24         2              2             0     28            17.3   1.787436
## 6  41         2              5             5     25            10.2   0.392700
##   OtherDebt Default
## 1  5.008608       1
## 2  4.000798       0
## 3  2.168925       0
## 4  0.821280       0
## 5  3.056564       1
## 6  2.157300       0
```

```r
str(bankloan)
```

```
## 'data.frame':    700 obs. of  9 variables:
##  $ Age            : int  41 27 40 41 24 41 39 43 24 36 ...
##  $ Education      : int  3 1 1 1 2 2 1 1 1 1 ...
##  $ EmployDuration : int  17 10 15 15 2 5 20 12 3 0 ...
##  $ ResidDuration  : int  12 6 14 14 0 5 9 11 4 13 ...
##  $ Income         : int  176 31 55 120 28 25 67 38 19 25 ...
##  $ DebtIncomeRatio: num  9.3 17.3 5.5 2.9 17.3 10.2 30.6 3.6 24.4 19.7 ...
##  $ CreditDebt     : num  11.359 1.362 0.856 2.659 1.787 ...
##  $ OtherDebt      : num  5.009 4.001 2.169 0.821 3.057 ...
##  $ Default        : int  1 0 0 0 1 0 0 0 1 0 ...
```

```r
bankloan$Default = as.factor(bankloan$Default)
str(bankloan)
```

```
## 'data.frame':    700 obs. of  9 variables:
##  $ Age            : int  41 27 40 41 24 41 39 43 24 36 ...
##  $ Education      : int  3 1 1 1 2 2 1 1 1 1 ...
##  $ EmployDuration : int  17 10 15 15 2 5 20 12 3 0 ...
##  $ ResidDuration  : int  12 6 14 14 0 5 9 11 4 13 ...
##  $ Income         : int  176 31 55 120 28 25 67 38 19 25 ...
##  $ DebtIncomeRatio: num  9.3 17.3 5.5 2.9 17.3 10.2 30.6 3.6 24.4 19.7 ...
##  $ CreditDebt     : num  11.359 1.362 0.856 2.659 1.787 ...
##  $ OtherDebt      : num  5.009 4.001 2.169 0.821 3.057 ...
##  $ Default        : Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 2 1 ...
```

Let's look at the response variable

```r
table(bankloan$Default)
```

```
##
##   0   1
## 517 183
```

Installing required packages

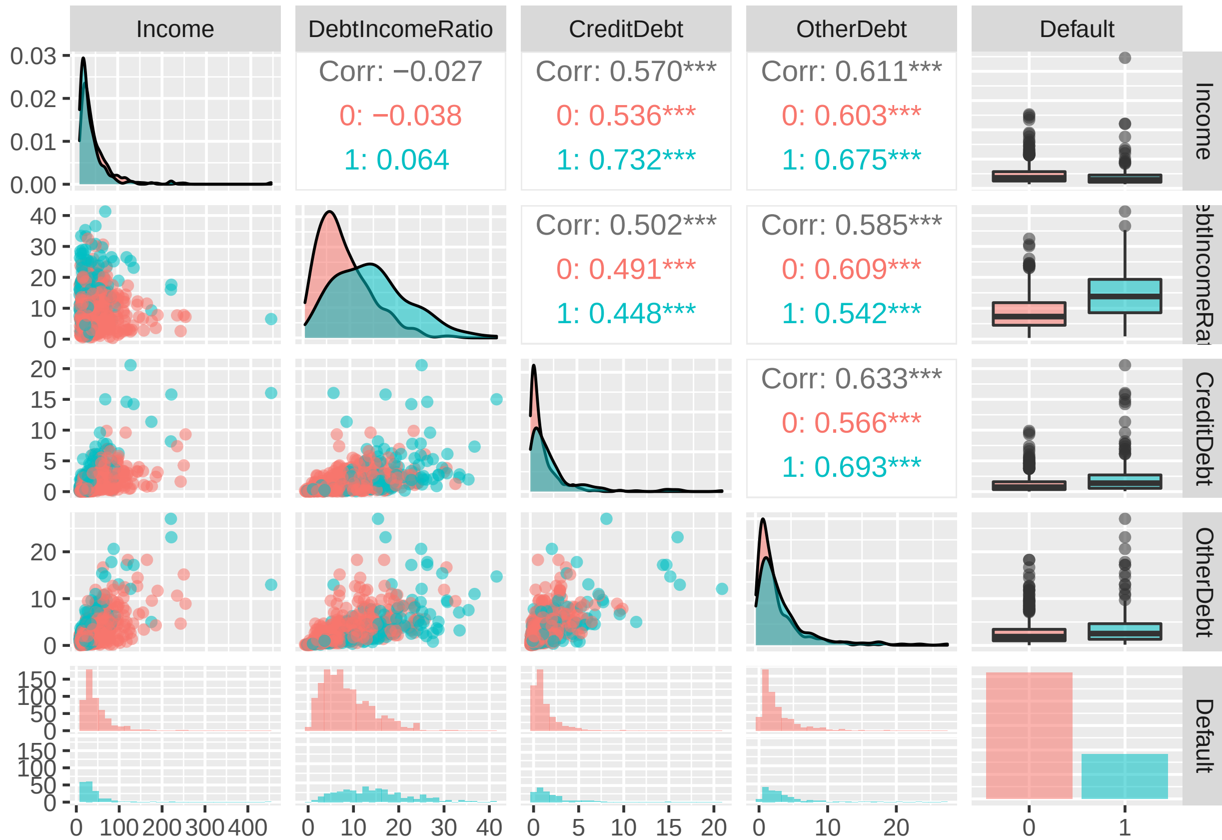
```r
#install.packages("rlang")
library(rlang)
library(ggplot2)
library(e1071)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

Vizualization

```r
bankloan1 = bankloan[, 5:9]
# ggpairs(bankloan)
ggpairs(bankloan1, ggplot2:: aes(colour = Default, alpha = 0.3))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Support Vector Machine (SVM)

A few deatils about SVM

- SVM is a supervised learning technique
- This is mostly used for classification purpose
- SVM finds a boundary in the hyperplane to classify the classes

Fit a SVM

```
fit.svm = svm(Default ~ ., data =bankloan, kernel = "radial")
```

Predict classes

```
pred.svm = predict(fit.svm, data = bankloan)
```

Confusion Matrix

```
conf.matrix = table(pred.svm, Actual = bankloan$Default)
conf.matrix
```

```
##           Actual
## pred.svm    0   1
##          0 491  95
```

```
##        1  26  88
```

Misclassification Rate

```
1 - sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.1728571
```

Improve the model

Parameter Tunning

```
set.seed(123)
fit.tune = tune(svm, Default ~ ., data = bankloan,
              ranges = list(epsilon = seq(0, 1, 0.1), cost = 2^(2:7)))
summary(fit.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  epsilon cost
##        0    4
##
## - best performance: 0.2142857
##
## - Detailed performance results:
##    epsilon cost     error dispersion
## 1      0.0    4 0.2142857 0.05345225
## 2      0.1    4 0.2142857 0.05345225
## 3      0.2    4 0.2142857 0.05345225
## 4      0.3    4 0.2142857 0.05345225
## 5      0.4    4 0.2142857 0.05345225
## 6      0.5    4 0.2142857 0.05345225
## 7      0.6    4 0.2142857 0.05345225
## 8      0.7    4 0.2142857 0.05345225
## 9      0.8    4 0.2142857 0.05345225
## 10     0.9    4 0.2142857 0.05345225
## 11     1.0    4 0.2142857 0.05345225
## 12     0.0    8 0.2257143 0.05585858
## 13     0.1    8 0.2257143 0.05585858
## 14     0.2    8 0.2257143 0.05585858
## 15     0.3    8 0.2257143 0.05585858
## 16     0.4    8 0.2257143 0.05585858
## 17     0.5    8 0.2257143 0.05585858
## 18     0.6    8 0.2257143 0.05585858
## 19     0.7    8 0.2257143 0.05585858
## 20     0.8    8 0.2257143 0.05585858
## 21     0.9    8 0.2257143 0.05585858
## 22     1.0    8 0.2257143 0.05585858
## 23     0.0   16 0.2171429 0.04301558
## 24     0.1   16 0.2171429 0.04301558
## 25     0.2   16 0.2171429 0.04301558
## 26     0.3   16 0.2171429 0.04301558
## 27     0.4   16 0.2171429 0.04301558
## 28     0.5   16 0.2171429 0.04301558
## 29     0.6   16 0.2171429 0.04301558
## 30     0.7   16 0.2171429 0.04301558
## 31     0.8   16 0.2171429 0.04301558
## 32     0.9   16 0.2171429 0.04301558
```

```
## 33     1.0   16 0.2171429 0.04301558
## 34     0.0   32 0.2214286 0.04054616
## 35     0.1   32 0.2214286 0.04054616
## 36     0.2   32 0.2214286 0.04054616
## 37     0.3   32 0.2214286 0.04054616
## 38     0.4   32 0.2214286 0.04054616
## 39     0.5   32 0.2214286 0.04054616
## 40     0.6   32 0.2214286 0.04054616
## 41     0.7   32 0.2214286 0.04054616
## 42     0.8   32 0.2214286 0.04054616
## 43     0.9   32 0.2214286 0.04054616
## 44     1.0   32 0.2214286 0.04054616
## 45     0.0   64 0.2442857 0.03716117
## 46     0.1   64 0.2442857 0.03716117
## 47     0.2   64 0.2442857 0.03716117
## 48     0.3   64 0.2442857 0.03716117
## 49     0.4   64 0.2442857 0.03716117
## 50     0.5   64 0.2442857 0.03716117
## 51     0.6   64 0.2442857 0.03716117
## 52     0.7   64 0.2442857 0.03716117
## 53     0.8   64 0.2442857 0.03716117
## 54     0.9   64 0.2442857 0.03716117
## 55     1.0   64 0.2442857 0.03716117
## 56     0.0  128 0.2614286 0.04716450
## 57     0.1  128 0.2614286 0.04716450
## 58     0.2  128 0.2614286 0.04716450
## 59     0.3  128 0.2614286 0.04716450
## 60     0.4  128 0.2614286 0.04716450
## 61     0.5  128 0.2614286 0.04716450
## 62     0.6  128 0.2614286 0.04716450
## 63     0.7  128 0.2614286 0.04716450
## 64     0.8  128 0.2614286 0.04716450
## 65     0.9  128 0.2614286 0.04716450
## 66     1.0  128 0.2614286 0.04716450
```

Best fitted model

```
fit.best = fit.tune$best.model
summary(fit.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = Default ~ ., data = bankloan, ranges = list(epsilon = seq(0,
##     1, 0.1), cost = 2^(2:7)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  4
##
## Number of Support Vectors:  348
##
##  ( 161 187 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Confusion Matrix for the best model

```
pred1 = predict(fit.best, bankloan)
conf.matrix1 = table(pred1, Actual = bankloan$Default)
conf.matrix1
```

```
##      Actual
## pred1   0   1
##     0 492  83
##     1  25 100
```

Misclassification Rate

```
1 - sum(diag(conf.matrix1)/sum(conf.matrix1))
```

```
## [1] 0.1542857
```

KNN

- Supervised Non-linear classifier
- Non-parametric algorithm
    - it does not hold any assumption about the data or underline distribution
- Deciding k (number of neighbor) is a key issue
- k = 1 produces underfit
- k = very high number will be computationally expensive

Spliting the data into training and test

```
library(caTools)
split = sample.split(bankloan$Default, SplitRatio = 0.8)
train = subset(bankloan, split == T)
dim(train)
```

```
## [1] 560   9
```

```
test = subset(bankloan, split == F)
dim(test)
```

```
## [1] 140   9
```

```
train.norm = scale(train[, 1:8])
test.norm = scale(test[, 1:8])
```

Fit KNN

```
library(class)
fit.knn = knn(train = train.norm,
              test = test.norm,
              cl = train$Default,
              k = 1)
```

Confusion Matrix

```
conf.knn = table(test$Default, fit.knn)
conf.knn
```

```
##    fit.knn
##      0  1
##   0 81 22
##   1 18 19
```

Misclassification Rate

```
1 - sum(diag(conf.knn)/sum(conf.knn))
```

```
## [1] 0.2857143
```

Optimam k

Choosing Optimal k

```
fit.knn = knn(train = train.norm,
              test = test.norm,
              cl = train$Default,
              k = round(sqrt(nrow(train))))
```

Confusion Matrix

```
conf.knn1 = table(test$Default, fit.knn)
conf.knn1
```

```
##    fit.knn
##      0  1
##   0 98  5
##   1 24 13
```

Misclassification rate

```
1 - sum(diag(conf.knn1)/sum(conf.knn1))
```

```
## [1] 0.2071429
```

Logistic Regression

Spliting the data into training and test

```
library(caTools)
split = sample.split(bankloan$Default, SplitRatio = 0.8)
train = subset(bankloan, split == T)
dim(train)
```

```
## [1] 560   9
```

```
test = subset(bankloan, split == F)
dim(test)
```

```
## [1] 140   9
```

```
fit.logit = glm(Default ~ ., data = train, family = "binomial"(link = "logit"))
```

Summary

```
summary(fit.logit)
```

```
##
## Call:
## glm(formula = Default ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2736  -0.6503  -0.2992   0.2440   2.9253
```

```
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.658079   0.683949  -2.424   0.0153 *
## Age               0.033815   0.018834   1.795   0.0726 .
## Education         0.016989   0.137596   0.123   0.9017
## EmployDuration   -0.280070   0.039053  -7.172 7.41e-13 ***
## ResidDuration    -0.102558   0.026129  -3.925 8.67e-05 ***
## Income            0.001352   0.012316   0.110   0.9126
## DebtIncomeRatio   0.065639   0.034985   1.876   0.0606 .
## CreditDebt        0.566521   0.122118   4.639 3.50e-06 ***
## OtherDebt         0.080541   0.086407   0.932   0.3513
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 642.66  on 559  degrees of freedom
## Residual deviance: 442.76  on 551  degrees of freedom
## AIC: 460.76
##
## Number of Fisher Scoring iterations: 6
```

Anova

```r
anova(fit.logit, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Default
##
## Terms added sequentially (first to last)
##
##
##                 Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                              559     642.66
## Age              1    7.635       558     635.02 0.0057259 **
## Education        1    9.432       557     625.59 0.0021324 **
## EmployDuration   1   35.600       556     589.99 2.423e-09 ***
## ResidDuration    1    6.130       555     583.86 0.0132921 *
## Income           1   14.587       554     569.28 0.0001338 ***
## DebtIncomeRatio  1   93.849       553     475.43 < 2.2e-16 ***
## CreditDebt       1   31.765       552     443.66 1.740e-08 ***
## OtherDebt        1    0.898       551     442.76 0.3434423
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Prediction

```r
pred.logit = predict(fit.logit, test, type = 'response')
```

Chaning Probabilities to classes

```
pred.logit1 = ifelse(pred.logit>0.5, 1, 0)
```

Confusion Matrix

```
Conf.logit = table(test$Default, pred.logit1)
Conf.logit
```

```
##    pred.logit1
##       0   1
##   0 100   3
##   1  21  16
```

Misclassification rate

```
1 - sum(diag(Conf.logit)/sum(Conf.logit))
```

```
## [1] 0.1714286
```

NN

A few details

- If the data is linearly separable perhaps no need of NN
- Every NN has three layers
  - Input, Hidden, and Output
- Input layer
  - Based on training data
  - No. of input neurons is equal to the number of feature (variables/columns) in training data
- Output Layer
  - One output layer
  - For regression, the output layer has a single node
  - For classification, output layer has one node per class label
- Hidden layers
- Pruning
  - Helps converging the NN faster and make the model smaller

Neural Network Architecture

- Input Layer
- Hidden Layer
- Learning rate
- Error function
- Activation function
- Output Layer

Spliting the data into training and test

```
library(caTools)
bankloan = data.frame(bankloan)
split = sample.split(bankloan$Default, SplitRatio = 0.8)
train = subset(bankloan, split == T)
dim(train)
```

```
## [1] 560   9
```

```r
test = subset(bankloan, split == F)
dim(test)
```

```
## [1] 140   9
```

Required packages

```r
#library(keras)
#library(mlbench)
# library(dplyr)
# library(magrittr)
library(neuralnet)
```

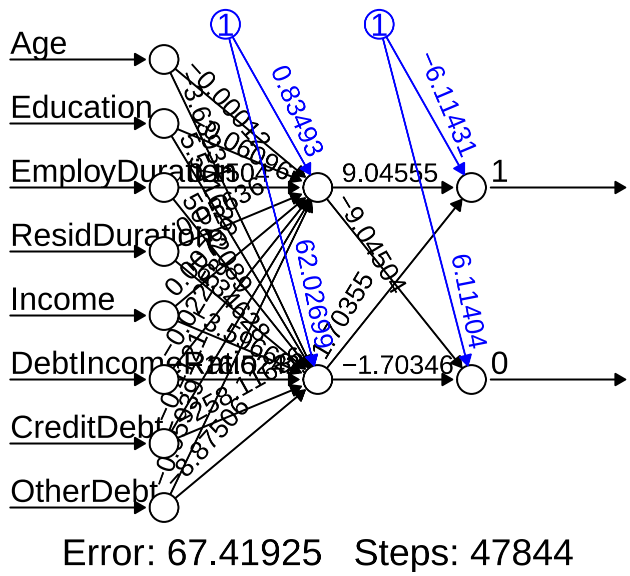
NN architecture

```r
fit.nn = neuralnet(Default ~ .,
                   data = train,
                   hidden = 2,
                   act.fct = "logistic",
                   linear.output = F)
summary(fit.nn)
```

```
##                     Length Class      Mode
## call                   6   -none-     call
## response            1120   -none-     logical
## covariate           4480   -none-     numeric
## model.list             2   -none-     list
## err.fct                1   -none-     function
## act.fct                1   -none-     function
## linear.output          1   -none-     logical
## data                   9   data.frame list
## exclude                0   -none-     NULL
## net.result             1   -none-     list
## weights                1   -none-     list
## generalized.weights    1   -none-     list
## startweights           1   -none-     list
## result.matrix         27   -none-     numeric
```

Plot the result

```r
plot(fit.nn, rep = "best")
```




```r
pred.nn = compute(fit.nn, train[, 1:8])
pred.nn.result = pred.nn$net.result
pred.nn.result
```

```
##             [,1]        [,2]
## 1   0.202258981 0.797741449
## 2   0.817456352 0.182568172
## 3   0.987932400 0.012071462
## 4   0.935937184 0.064076663
## 5   0.146885092 0.853112511
## 6   0.763540994 0.236485352
## 7   0.795581725 0.204443716
## 8   0.986739473 0.013264694
## 9   0.282681576 0.717323999
## 10  0.164436040 0.835562365
## 12  0.723421826 0.276604997
## 13  0.937606941 0.062406661
## 14  0.682871907 0.317154753
## 15  0.937392397 0.062621237
## 16  0.870760134 0.129260949
## 17  0.789492305 0.210533348
## 20  0.970600817 0.029406966
## 21  0.988848574 0.011155047
## 22  0.989253440 0.010750074
## 24  0.972436268 0.027571143
## 25  0.350855246 0.649155067
## 27  0.987172707 0.012831350
## 28  0.989710755 0.010292636
## 29  0.727094151 0.272932656
## 30  0.430224741 0.569790973
## 31  0.974948030 0.025058858
## 33  0.649997847 0.350028265
## 35  0.975801735 0.024204971
```

```
## 37  0.792302242 0.207723316
## 38  0.657167868 0.342858393
## 40  0.989383070 0.010620409
## 41  0.965966122 0.034042565
## 42  0.984459575 0.015545160
## 44  0.984952598 0.015052016
## 45  0.403671792 0.596342159
## 48  0.715005067 0.285021772
## 52  0.616664335 0.383360886
## 53  0.919418433 0.080597644
## 54  0.973817891 0.026189235
## 55  0.591105691 0.408918641
## 56  0.199424490 0.800575777
## 57  0.974552057 0.025454915
## 59  0.988219372 0.011784415
## 60  0.882669949 0.117350096
## 61  0.983956638 0.016048219
## 62  0.887953735 0.112065814
## 63  0.531019152 0.469002494
## 65  0.761387874 0.238638516
## 67  0.648609740 0.351416341
## 68  0.985883650 0.014120733
## 69  0.774898357 0.225127725
## 70  0.309100085 0.690907312
## 74  0.279562097 0.720443266
## 75  0.987599877 0.012404070
## 76  0.860231669 0.139790243
## 77  0.932670078 0.067343982
## 78  0.726704288 0.273322522
## 79  0.985852736 0.014151655
## 81  0.371221323 0.628790407
## 82  0.814786023 0.185238626
## 83  0.371677046 0.628334715
## 84  0.709111325 0.290915510
## 85  0.758756858 0.241269582
## 86  0.844535585 0.155487420
## 87  0.241401182 0.758601648
## 88  0.667439900 0.332586547
## 90  0.040703078 0.959293096
## 91  0.963285040 0.036724149
## 93  0.671788899 0.328237616
## 95  0.069366529 0.930629193
## 96  0.987798688 0.012205209
## 97  0.907282771 0.092734756
## 99  0.786380979 0.213644774
## 100 0.615286885 0.384738292
## 101 0.989731350 0.010272035
## 102 0.973732788 0.026274355
## 103 0.974294617 0.025712409
## 104 0.989558513 0.010444919
## 105 0.384483202 0.615529442
## 106 0.984141620 0.015863191
## 107 0.886917861 0.113101786
## 108 0.623880490 0.376144950
```

```
## 109 0.912059631 0.087957343
## 110 0.888767441 0.111252029
## 111 0.682171698 0.317854954
## 112 0.779763440 0.220262511
## 114 0.807740343 0.192284620
## 115 0.988546357 0.011457345
## 116 0.782421517 0.217604357
## 117 0.860606218 0.139415666
## 118 0.988436087 0.011567643
## 119 0.272902559 0.727102352
## 120 0.474361388 0.525657099
## 121 0.985555998 0.014448467
## 122 0.618405748 0.381619527
## 123 0.117550958 0.882445559
## 124 0.449262763 0.550754175
## 125 0.973628023 0.026379143
## 126 0.978785873 0.021220182
## 127 0.711100236 0.288926602
## 128 0.979995221 0.020010562
## 129 0.733819732 0.266207032
## 130 0.986882817 0.013121315
## 131 0.671941525 0.328084991
## 132 0.771464272 0.228561897
## 133 0.988343381 0.011660373
## 134 0.930881250 0.069133314
## 135 0.962226907 0.037782476
## 136 0.976206862 0.023799758
## 138 0.980953027 0.019052537
## 139 0.978601689 0.021404407
## 140 0.651414546 0.348611596
## 141 0.219299867 0.780701577
## 142 0.595677430 0.404347074
## 143 0.989358488 0.010644998
## 144 0.982993110 0.017011978
## 145 0.988994395 0.011009187
## 146 0.817576778 0.182447739
## 147 0.386548725 0.613464060
## 150 0.794451676 0.205573807
## 152 0.934292641 0.065721442
## 153 0.646395753 0.353630278
## 154 0.973348200 0.026659023
## 156 0.703776548 0.296250272
## 157 0.886894212 0.113125438
## 158 0.737995963 0.262030765
## 159 0.794922157 0.205103309
## 160 0.978850479 0.021155561
## 161 0.502949092 0.497071054
## 163 0.585604070 0.414420050
## 164 0.977485926 0.022520416
## 166 0.299009649 0.700997048
## 167 0.394248852 0.605764460
## 168 0.776211939 0.223814109
## 169 0.985338544 0.014665975
## 170 0.987251191 0.012752846
```

```
## 172 0.806666033 0.193358975
## 173 0.533501973 0.466519798
## 174 0.858671470 0.141350558
## 175 0.975982975 0.024023693
## 176 0.707141444 0.292885387
## 177 0.376538561 0.623473536
## 178 0.963071619 0.036937609
## 179 0.982745036 0.017260110
## 182 0.984046049 0.015958786
## 185 0.103581773 0.896414366
## 186 0.983939293 0.016065568
## 190 0.827987005 0.172036981
## 191 0.532904665 0.467117076
## 192 0.571572179 0.428451367
## 193 0.834145616 0.165878025
## 194 0.711526199 0.288500640
## 196 0.724370032 0.275656788
## 198 0.980168089 0.019837655
## 199 0.987088858 0.012915220
## 200 0.191303206 0.808696602
## 201 0.954094244 0.045916561
## 202 0.907356855 0.092660664
## 203 0.847899773 0.152123011
## 205 0.827520429 0.172503581
## 206 0.931430680 0.068583807
## 207 0.970889253 0.029118472
## 208 0.974592653 0.025414310
## 209 0.924402097 0.075613341
## 210 0.965393671 0.034615124
## 211 0.911319118 0.088697944
## 213 0.736322179 0.263704565
## 215 0.846010138 0.154012772
## 217 0.987923296 0.012080568
## 218 0.938923327 0.061090081
## 222 0.814451168 0.185573497
## 223 0.931539481 0.068474991
## 224 0.139980846 0.860016470
## 225 0.246386353 0.753616799
## 227 0.545035265 0.454987075
## 228 0.843819452 0.156203599
## 230 0.746098136 0.253928501
## 231 0.801135311 0.198889922
## 232 0.801751496 0.198273712
## 233 0.985755977 0.014248438
## 234 0.958829252 0.041180740
## 235 0.985795392 0.014209013
## 236 0.955697123 0.044313411
## 237 0.715005904 0.285020936
## 238 0.763594460 0.236431885
## 241 0.573594163 0.426429468
## 242 0.869418945 0.130602249
## 243 0.977507219 0.022499118
## 244 0.681512707 0.318513937
## 245 0.363089125 0.636922040
```

```
## 246 0.744689394 0.255337261
## 248 0.599866460 0.400158196
## 249 0.982948351 0.017056747
## 250 0.663277384 0.336748991
## 251 0.938413107 0.061600376
## 252 0.702522942 0.297503873
## 253 0.924579133 0.075436282
## 254 0.470347665 0.529670580
## 255 0.970852110 0.029155623
## 256 0.925244968 0.074770358
## 257 0.989355239 0.010648247
## 259 0.988483772 0.011519945
## 260 0.971894113 0.028113409
## 261 0.793764497 0.206261010
## 262 0.829362725 0.170661186
## 263 0.969885176 0.030122750
## 265 0.980365384 0.019640315
## 266 0.793260672 0.206764853
## 268 0.591749091 0.408275265
## 269 0.583772246 0.416251802
## 270 0.989261911 0.010741600
## 271 0.941764865 0.058248114
## 273 0.717806897 0.282219940
## 274 0.984333330 0.015671435
## 276 0.397468557 0.602544974
## 277 0.373109303 0.626902558
## 278 0.985075646 0.014928937
## 279 0.713634370 0.286392470
## 280 0.865938145 0.134083327
## 281 0.982916152 0.017088953
## 282 0.520875599 0.479145521
## 284 0.561986008 0.438037119
## 285 0.764747479 0.235278842
## 286 0.982834655 0.017170469
## 289 0.988868771 0.011134845
## 291 0.752107582 0.247918970
## 292 0.977403193 0.022603167
## 293 0.990041870 0.009961431
## 294 0.953497870 0.046513035
## 295 0.821518264 0.178506059
## 296 0.984395789 0.015608961
## 297 0.602510493 0.397514256
## 298 0.234625934 0.765376464
## 299 0.763767210 0.236259131
## 300 0.988775204 0.011228437
## 301 0.004146141 0.995852914
## 303 0.989709935 0.010293456
## 304 0.963619807 0.036389320
## 305 0.557763851 0.442259086
## 306 0.980222742 0.019782989
## 307 0.980552332 0.019453324
## 309 0.839277254 0.160746081
## 310 0.851073976 0.148948594
## 311 0.166055452 0.833943032
```

```
## 312 0.872620884 0.127400044
## 313 0.397108549 0.602904957
## 314 0.801860303 0.198164901
## 315 0.376295482 0.623716599
## 316 0.804733981 0.195291108
## 317 0.723750550 0.276276273
## 319 0.973350571 0.026656651
## 320 0.753334931 0.246691603
## 321 0.974006999 0.026000087
## 323 0.989462438 0.010541020
## 324 0.528635030 0.471386493
## 325 0.679031174 0.320995441
## 326 0.011802617 0.988195408
## 327 0.739443180 0.260583534
## 328 0.660467989 0.339558335
## 329 0.709969013 0.290057824
## 330 0.837410972 0.162612476
## 331 0.651940265 0.348085889
## 333 0.684773313 0.315253368
## 334 0.370872549 0.629139156
## 335 0.780271948 0.219753988
## 336 0.983561859 0.016443093
## 337 0.739097823 0.260928895
## 339 0.052883088 0.947112778
## 340 0.621067944 0.378957413
## 341 0.836952144 0.163071332
## 342 0.955024946 0.044985702
## 343 0.988591214 0.011412475
## 345 0.830095187 0.169928683
## 346 0.427145412 0.572870100
## 347 0.638908776 0.361117075
## 348 0.979544674 0.020461211
## 349 0.957404905 0.042605335
## 350 0.657360243 0.342666022
## 351 0.987071087 0.012932997
## 352 0.301062769 0.698944070
## 353 0.984886870 0.015117760
## 354 0.854483265 0.145539067
## 355 0.590700106 0.409324211
## 356 0.792405520 0.207620034
## 357 0.427865016 0.572150544
## 358 0.295528389 0.704478067
## 359 0.536135979 0.463885925
## 361 0.986455452 0.013548787
## 362 0.987221794 0.012782251
## 363 0.701025672 0.299001136
## 365 0.596362079 0.403662450
## 366 0.972544149 0.027463241
## 367 0.955920715 0.044089782
## 368 0.988778897 0.011224743
## 369 0.861118042 0.138903804
## 370 0.989383476 0.010620003
## 371 0.908551334 0.091466048
## 373 0.981647718 0.018357685
```

```
## 374 0.002558164 0.997441183
## 375 0.989922652 0.010080681
## 377 0.976115815 0.023890824
## 378 0.483814461 0.516204588
## 379 0.985625275 0.014379173
## 380 0.982132496 0.017872794
## 382 0.373053344 0.626958512
## 383 0.124815297 0.875181455
## 384 0.745458755 0.254567891
## 385 0.748493053 0.251533552
## 386 0.501157880 0.498862166
## 387 0.969854698 0.030153234
## 388 0.986850988 0.013153152
## 389 0.978550364 0.021455744
## 391 0.989445692 0.010557771
## 392 0.646060207 0.353965817
## 393 0.849214596 0.150808101
## 394 0.155530394 0.844467592
## 396 0.772437597 0.227588548
## 397 0.980870033 0.019135550
## 398 0.820289467 0.179734918
## 399 0.388969055 0.611043896
## 400 0.492763949 0.507255620
## 401 0.451203619 0.548813442
## 402 0.975752332 0.024254389
## 403 0.663358761 0.336667616
## 404 0.010335382 0.989662808
## 406 0.985515598 0.014488878
## 408 0.686546835 0.313479864
## 409 0.986943654 0.013060462
## 410 0.980897600 0.019107977
## 411 0.989891008 0.010112335
## 412 0.989668607 0.010334795
## 413 0.986712216 0.013291959
## 414 0.206603602 0.793397081
## 418 0.785544410 0.214481370
## 420 0.827025803 0.172998235
## 421 0.590858013 0.409166310
## 422 0.988622674 0.011381007
## 423 0.723217657 0.276809167
## 424 0.979462443 0.020543461
## 425 0.174161158 0.825837733
## 427 0.507885748 0.492134671
## 429 0.667990487 0.332035968
## 430 0.535536370 0.464485504
## 433 0.882176145 0.117843946
## 434 0.386327293 0.613685478
## 435 0.475498534 0.524520021
## 436 0.503477767 0.496542409
## 437 0.661615893 0.338410452
## 439 0.846580625 0.153442247
## 440 0.988235523 0.011768260
## 441 0.843120354 0.156902742
## 442 0.888116295 0.111903238
```

```
## 443 0.904780379 0.095237429
## 444 0.076968713 0.923027028
## 445 0.211053334 0.788947613
## 446 0.784352034 0.215673782
## 450 0.226689942 0.773311957
## 452 0.814703813 0.185320840
## 453 0.872682829 0.127338094
## 454 0.528903413 0.471118125
## 455 0.226278089 0.773723785
## 456 0.633161840 0.366863862
## 457 0.927681717 0.072333284
## 459 0.756583537 0.243442942
## 461 0.969540350 0.030467644
## 463 0.976503063 0.023503493
## 464 0.969829838 0.030178099
## 465 0.363423579 0.636587610
## 466 0.336912506 0.663096833
## 467 0.006492056 0.993506633
## 468 0.960806350 0.039203290
## 470 0.584550991 0.415473088
## 471 0.707189359 0.292837472
## 472 0.390530208 0.609482850
## 475 0.984348207 0.015656554
## 478 0.770707538 0.229318648
## 479 0.990060548 0.009942749
## 480 0.915286424 0.084730164
## 481 0.985870803 0.014133584
## 482 0.485325746 0.514693392
## 483 0.750550254 0.249476322
## 484 0.963756743 0.036252359
## 485 0.626273086 0.373752423
## 486 0.814699660 0.185324993
## 487 0.699207814 0.300818984
## 488 0.191163868 0.808835933
## 489 0.409948776 0.590065597
## 490 0.989766452 0.010236923
## 491 0.987631294 0.012372645
## 492 0.733997314 0.266029449
## 493 0.024378567 0.975618393
## 494 0.516444665 0.483576219
## 495 0.398979507 0.601034127
## 496 0.746868641 0.253157986
## 498 0.864815454 0.135206107
## 499 0.249426865 0.750576485
## 500 0.056915188 0.943080617
## 501 0.344047173 0.655962664
## 502 0.465691835 0.534326126
## 503 0.872019072 0.128001906
## 504 0.813468529 0.186556181
## 505 0.258579384 0.741424568
## 506 0.214525637 0.785475517
## 507 0.989925050 0.010078283
## 508 0.977333465 0.022672910
## 509 0.401122418 0.598891360
```

```
## 510 0.988842263 0.011161360
## 511 0.150801146 0.849196628
## 512 0.897889317 0.102129234
## 513 0.971065203 0.028942487
## 515 0.897754094 0.102264471
## 516 0.844502498 0.155520509
## 517 0.921603495 0.078412305
## 519 0.986218373 0.013785927
## 520 0.977779785 0.022226492
## 521 0.668172019 0.331854439
## 524 0.989254844 0.010748670
## 525 0.666342478 0.333683950
## 526 0.979594052 0.020411821
## 527 0.983794827 0.016210068
## 528 0.744367176 0.255659484
## 529 0.947187842 0.052824287
## 532 0.155288524 0.844709451
## 533 0.692598846 0.307427908
## 535 0.729825599 0.270201193
## 536 0.762739939 0.237286423
## 538 0.496230552 0.503789215
## 539 0.988820022 0.011183607
## 540 0.401988986 0.598024850
## 541 0.986158964 0.013845350
## 542 0.979857158 0.020148656
## 543 0.727258105 0.272768702
## 544 0.351066096 0.648944232
## 546 0.986543627 0.013460591
## 547 0.836496134 0.163527369
## 548 0.749202634 0.250823961
## 549 0.724067940 0.275958881
## 550 0.184443278 0.815556155
## 551 0.604656884 0.395367940
## 552 0.322678117 0.677330227
## 553 0.217122175 0.782879136
## 555 0.989676663 0.010326738
## 556 0.967420859 0.032587550
## 557 0.965693458 0.034315281
## 558 0.786545286 0.213480462
## 559 0.686818818 0.313207884
## 560 0.841804589 0.158218590
## 562 0.885037974 0.114981852
## 563 0.989820773 0.010182588
## 564 0.985551047 0.014453419
## 565 0.027371989 0.972624787
## 566 0.636142383 0.363883397
## 568 0.987519809 0.012484159
## 569 0.189382273 0.810617428
## 570 0.943663250 0.056349436
## 571 0.117506361 0.882490155
## 572 0.622510082 0.377515317
## 573 0.988339887 0.011663869
## 574 0.711931592 0.288095247
## 576 0.365405236 0.634606090
```

```
## 577 0.963324717 0.036684465
## 578 0.989112671 0.010890880
## 579 0.618432503 0.381592773
## 580 0.978261624 0.021744548
## 581 0.735167609 0.264859144
## 582 0.975641473 0.024365267
## 584 0.103395517 0.896600617
## 585 0.014863070 0.985134642
## 586 0.973557620 0.026449559
## 587 0.972299512 0.027707927
## 588 0.871885234 0.128135756
## 589 0.887044984 0.112974652
## 590 0.725313344 0.274713472
## 592 0.330706593 0.669302312
## 594 0.959147093 0.040862843
## 595 0.181837912 0.818161380
## 599 0.663938830 0.336087557
## 601 0.813456448 0.186568263
## 602 0.897553181 0.102465405
## 603 0.833559196 0.166464478
## 604 0.982585656 0.017419528
## 605 0.119795902 0.880200685
## 606 0.989409015 0.010594457
## 607 0.090951118 0.909044773
## 609 0.806701476 0.193323531
## 611 0.645294144 0.354731861
## 612 0.905826675 0.094191016
## 614 0.988418569 0.011585166
## 615 0.889197995 0.110821433
## 616 0.989299553 0.010703948
## 619 0.457529871 0.542487586
## 622 0.967906627 0.032101694
## 624 0.988059132 0.011944696
## 625 0.800724117 0.199301132
## 626 0.934213587 0.065800508
## 628 0.829652257 0.170371638
## 629 0.392779298 0.607233913
## 631 0.807740956 0.192284008
## 632 0.902979924 0.097038083
## 633 0.079649792 0.920345966
## 634 0.316814672 0.683193262
## 635 0.986158232 0.013846082
## 636 0.931373130 0.068641365
## 637 0.185194670 0.814804803
## 638 0.273332414 0.726672526
## 639 0.955457974 0.044552600
## 640 0.869159370 0.130861844
## 641 0.963828699 0.036180390
## 643 0.550900868 0.449121752
## 644 0.965106046 0.034902803
## 645 0.989532954 0.010470485
## 646 0.410269621 0.589744773
## 647 0.982558102 0.017447088
## 648 0.839746121 0.160277185
```

```
## 649 0.987672500 0.012331429
## 651 0.842051052 0.157972112
## 652 0.673237975 0.326788560
## 653 0.985298406 0.014706123
## 654 0.484008953 0.516010107
## 655 0.615013395 0.385011773
## 657 0.984505951 0.015498773
## 658 0.986349375 0.013654892
## 659 0.722560082 0.277466744
## 660 0.828837216 0.171186723
## 661 0.669147927 0.330878547
## 662 0.806249174 0.193775851
## 663 0.975416722 0.024590067
## 664 0.982600232 0.017404948
## 665 0.635539205 0.364486559
## 666 0.989068619 0.010934944
## 667 0.930400233 0.069614397
## 668 0.955936128 0.044074365
## 669 0.981704014 0.018301376
## 670 0.831847563 0.168176209
## 671 0.681118432 0.318908208
## 672 0.471184413 0.528833882
## 673 0.967279989 0.032728447
## 674 0.315418356 0.684589481
## 675 0.948978609 0.051033033
## 676 0.989968589 0.010034732
## 677 0.801148706 0.198876526
## 678 0.977276496 0.022729892
## 679 0.978876901 0.021129134
## 680 0.988097814 0.011906005
## 681 0.491037953 0.508981517
## 682 0.667419346 0.332607100
## 684 0.557362594 0.442660325
## 685 0.987801564 0.012202331
## 686 0.654019133 0.346007065
## 688 0.722860620 0.277166205
## 689 0.910876428 0.089140686
## 690 0.695290828 0.304735945
## 692 0.932878122 0.067136162
## 693 0.313509717 0.686497986
## 695 0.446500660 0.553516103
## 696 0.979842469 0.020163348
## 697 0.775998304 0.224027750
## 698 0.985075051 0.014929533
## 699 0.988866306 0.011137311
## 700 0.857789804 0.142232289
```

Convert the probability like outputs to classes

```
idx = apply(pred.nn$net.result, 1, which.max)
pred.nn1 = c(0, 1)[idx]
pred.nn1
```

```
##   [1] 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
```

```
##  [38] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0
##  [75] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
## [149] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
## [186] 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
## [223] 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0
## [260] 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
## [297] 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 1
## [334] 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1
## [371] 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0
## [408] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0
## [445] 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0
## [482] 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
## [519] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1
## [556] 0 0 0 0 0
```

Confusion Matrix

```r
conf.nn = table(pred.nn1, train$Default)
conf.nn
```

```
##
## pred.nn1   0   1
##        0 385  66
##        1  29  80
```

Misclassification rate

```r
1 - sum(diag(conf.nn)/sum(conf.nn))
```

```
## [1] 0.1696429
```

Can we improve the NN?

- Yes, we can; make changes of NN architecture (carelly)
- No of Hidden layers
- Work with the activation functions
  - sigmoid
  - softmax
- Custom activation function?
  - Yes, we can use our own AF
- Learning rate (for optimization)
- Error function (Cost function) etc.