# German Credit Scores using a Neural Network

Daniel Meister

10/23/2025

## Loading the Packages

For this example we use the `nnet` package (and `gamlss.add` to plot these networks) and `ggplot2` to create some plots.

```
library(nnet)
library(gamlss.add)
```

```
## Loading required package: gamlss.dist
## Loading required package: MASS
## Loading required package: gamlss
## Loading required package: splines
## Loading required package: gamlss.data
##
## Attaching package: 'gamlss.data'
## The following object is masked from 'package:datasets':
##
##     sleep
## Loading required package: nlme
## Loading required package: parallel
##  **********   GAMLSS Version 5.2-0  **********
## For more on GAMLSS look at https://www.gamlss.com/
## Type gamlssNews() to see new features/changes/bug fixes.
## Loading required package: mgcv
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
##
## Attaching package: 'mgcv'
## The following object is masked from 'package:nnet':
##
##     multinom
## Loading required package: rpart
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:nlme':
##
##     collapse

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
theme_set(theme_bw())
```

And we also need the `caret` package for some helper functions

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:gamlss':
##
##     calibration
```

## Loading the Data

Load the `GermanCredit` data set into the environment

```
data(GermanCredit)
```

and let's have a look at the structure.

```
str(GermanCredit)
```

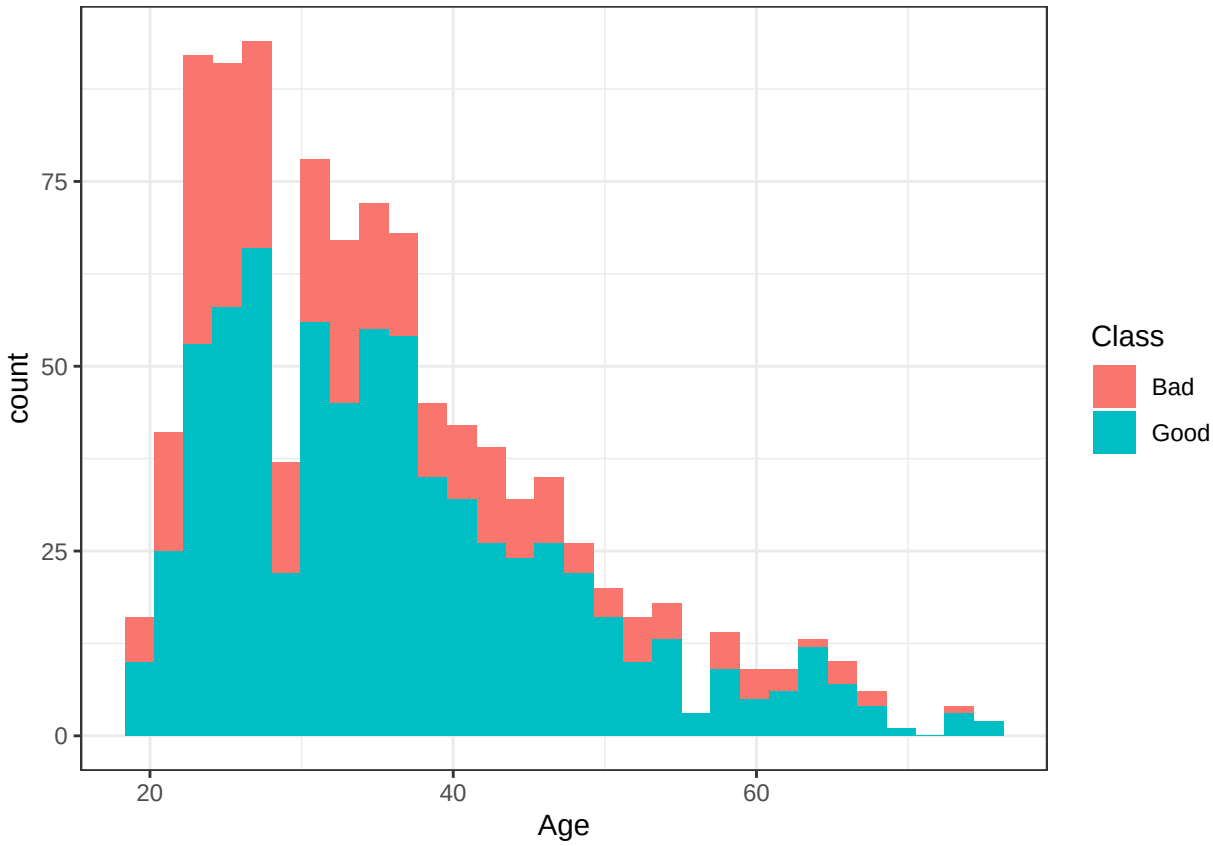
```
## 'data.frame':    1000 obs. of  62 variables:
##  $ Duration                              : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                                : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ..
##  $ InstallmentRatePercentage             : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration                     : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                                   : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits                 : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance               : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                             : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker                         : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                                 : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
##  $ CheckingAccountStatus.lt.0            : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ CheckingAccountStatus.0.to.200        : num  0 1 0 0 0 0 0 1 0 1 ...
##  $ CheckingAccountStatus.gt.200          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CheckingAccountStatus.none            : num  0 0 1 0 0 1 1 0 1 0 ...
##  $ CreditHistory.NoCredit.AllPaid        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.ThisBank.AllPaid        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.PaidDuly                : num  0 1 0 1 0 1 1 1 1 0 ...
```

```
##  $ CreditHistory.Delay                    : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ CreditHistory.Critical                 : num  1 0 1 0 0 0 0 0 0 1 ...
##  $ Purpose.NewCar                         : num  0 0 0 0 1 0 0 0 0 1 ...
##  $ Purpose.UsedCar                        : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Purpose.Furniture.Equipment            : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Purpose.Radio.Television               : num  1 1 0 0 0 0 0 0 1 0 ...
##  $ Purpose.DomesticAppliance              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Repairs                        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Education                      : num  0 0 1 0 0 1 0 0 0 0 ...
##  $ Purpose.Vacation                       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Retraining                     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Business                       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Other                          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.lt.100             : num  0 1 1 1 1 0 0 1 0 1 ...
##  $ SavingsAccountBonds.100.to.500         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.500.to.1000        : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ SavingsAccountBonds.gt.1000            : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ SavingsAccountBonds.Unknown            : num  1 0 0 0 0 1 0 0 0 0 ...
##  $ EmploymentDuration.lt.1                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ EmploymentDuration.1.to.4              : num  0 1 0 0 1 1 0 1 0 0 ...
##  $ EmploymentDuration.4.to.7              : num  0 0 1 1 0 0 0 0 1 0 ...
##  $ EmploymentDuration.gt.7                : num  1 0 0 0 0 0 1 0 0 0 ...
##  $ EmploymentDuration.Unemployed          : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Male.Divorced.Seperated       : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Personal.Female.NotSingle              : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Personal.Male.Single                   : num  1 0 1 1 1 1 1 1 0 0 ...
##  $ Personal.Male.Married.Widowed          : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Female.Single                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.None            : num  1 1 1 0 1 1 1 1 1 1 ...
##  $ OtherDebtorsGuarantors.CoApplicant     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.Guarantor       : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ Property.RealEstate                    : num  1 1 1 0 0 0 0 0 1 0 ...
##  $ Property.Insurance                     : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Property.CarOther                      : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Property.Unknown                       : num  0 0 0 0 1 1 0 0 0 0 ...
##  $ OtherInstallmentPlans.Bank             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.Stores           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.None             : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Housing.Rent                           : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Housing.Own                            : num  1 1 1 0 0 0 1 0 1 1 ...
##  $ Housing.ForFree                        : num  0 0 0 1 1 1 0 0 0 0 ...
##  $ Job.UnemployedUnskilled                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Job.UnskilledResident                  : num  0 0 1 0 0 1 0 0 1 0 ...
##  $ Job.SkilledEmployee                    : num  1 1 0 1 1 0 1 0 0 0 ...
##  $ Job.Management.SelfEmp.HighlyQualified: num  0 0 0 0 0 0 0 1 0 1 ...
```

## A first look at the Data

```
GermanCredit %>%
  ggplot(aes(x = Age, fill = Class)) +
  geom_histogram(position = "stack")
```

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




## Prepare the Data

Split the data into 80% to train and 20% to test.

```
set.seed(123)
is_train <- runif(nrow(GermanCredit)) < 0.8
mean(is_train)
```

```
## [1] 0.802
```

```
train <- GermanCredit[is_train, ]
test <- GermanCredit[!is_train, ]
```

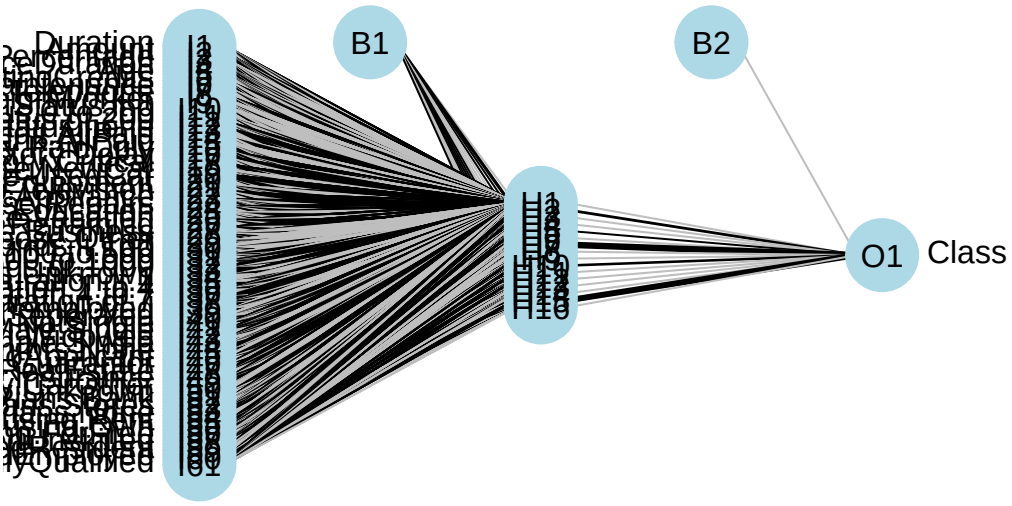
## Build the network

```
set.seed(412)
credit_net <- nnet(Class ~ ., data = train, size=16, maxit=10000, range=0.1, decay=5e-4, MaxNWts = 2000)
```

```
## # weights:  1009
## initial  value 501.568834
## iter  10 value 483.331514
## iter  20 value 483.008320
## iter  30 value 474.401684
## iter  40 value 410.688069
## iter  50 value 386.100062
## iter  60 value 376.841550
## iter  70 value 367.017968
## iter  80 value 362.937983
```

```
## iter  90 value 361.694879
## iter 100 value 357.033310
## iter 110 value 355.772473
## iter 120 value 355.241655
## iter 130 value 354.725773
## iter 140 value 351.553060
## iter 150 value 347.835241
## iter 160 value 341.347153
## iter 170 value 319.212289
## iter 180 value 306.060213
## iter 190 value 297.357488
## iter 200 value 292.878349
## iter 210 value 291.146559
## iter 220 value 287.704144
## iter 230 value 282.965247
## iter 240 value 281.841429
## iter 250 value 281.437552
## iter 260 value 281.106105
## iter 270 value 280.341136
## iter 280 value 279.827684
## iter 290 value 278.576682
## iter 300 value 277.749806
## iter 310 value 275.384433
## iter 320 value 274.188117
## iter 330 value 273.816287
## iter 340 value 273.727817
## iter 350 value 273.585570
## iter 360 value 273.489350
## iter 370 value 273.300678
## iter 380 value 272.999527
## iter 390 value 272.730002
## iter 400 value 272.326724
## iter 410 value 270.903382
## iter 420 value 268.938064
## iter 430 value 268.564586
## iter 440 value 268.319851
## iter 450 value 267.792483
## iter 460 value 267.584140
## iter 470 value 261.268878
## iter 480 value 250.566057
## iter 490 value 244.537311
## iter 500 value 240.875108
## iter 510 value 239.943155
## iter 520 value 238.738318
## iter 530 value 236.899955
## iter 540 value 236.469829
## iter 550 value 236.261427
## iter 560 value 236.168837
## final  value 236.167174
## converged
```

```
plot(credit_net)
```




```
credit_net
```

```
## a 61-16-1 network with 1009 weights
## inputs: Duration Amount InstallmentRatePercentage ResidenceDuration Age NumberExistingCredits Numberl
## output(s): Class
## options were - entropy fitting  decay=5e-04
```

### Make Predictions

```
pred <- predict(credit_net, test, type="class")
cm_nn <- table(pred=pred, true=test$Class)
cm_nn
```

```
##       true
## pred   Bad Good
##   Bad   43   37
##   Good  24   94
```

Calculating the accuracy

```
sum(diag(cm_nn))/sum(sum(cm_nn))
```

```
## [1] 0.6919192
```

And the precision

```
cm_nn[1, 1]/(cm_nn[1, 1] + cm_nn[1, 2])
```

```
## [1] 0.5375
```

Or do all of it using the `confusionMatrix` function

```
confusionMatrix(as.factor(pred), as.factor(test$Class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   43   37
##       Good  24   94
##
##                Accuracy : 0.6919
```

```
##                  95% CI : (0.6226, 0.7554)
##     No Information Rate : 0.6616
##     P-Value [Acc > NIR] : 0.2051
##
##                   Kappa : 0.3431
##
##  Mcnemar's Test P-Value : 0.1244
##
##             Sensitivity : 0.6418
##             Specificity : 0.7176
##          Pos Pred Value : 0.5375
##          Neg Pred Value : 0.7966
##              Prevalence : 0.3384
##          Detection Rate : 0.2172
##    Detection Prevalence : 0.4040
##       Balanced Accuracy : 0.6797
##
##        'Positive' Class : Bad
##
```

## ROC Curves*

The Receiver Operating Characteristic is a useful tool to indicate model quality. For this we plot sensitivity against 1-specificity

```
library(ROCR)
```

```
pred_raw <- predict(credit_net, test, decision.values=TRUE, type = "raw")
pred <- ROCR::prediction(pred_raw, test$Class)
perf <- ROCR::performance(pred, "tpr", "fpr")
plot(perf, lwd=2, col="blue")
abline(a=0, b=1)
```