

Iris Data Classification using a Neural Network

Daniel Meister

10/23/2025

Loading the Packages

First make sure we have all the functionality from the `tidyverse` available:

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.0 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     vforcats 0.5.0

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

theme_set(theme_bw())
```

We are going to use the `neuralnet` package for this second example and also some helper functions from `caret`

```
library(neuralnet)

##
## Attaching package: 'neuralnet'
## The following object is masked from 'package:dplyr':
## 
##     compute
library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
## 
##     lift
```

Loading the Data

Load the `iris` data set into the environment

```
data(iris)
```

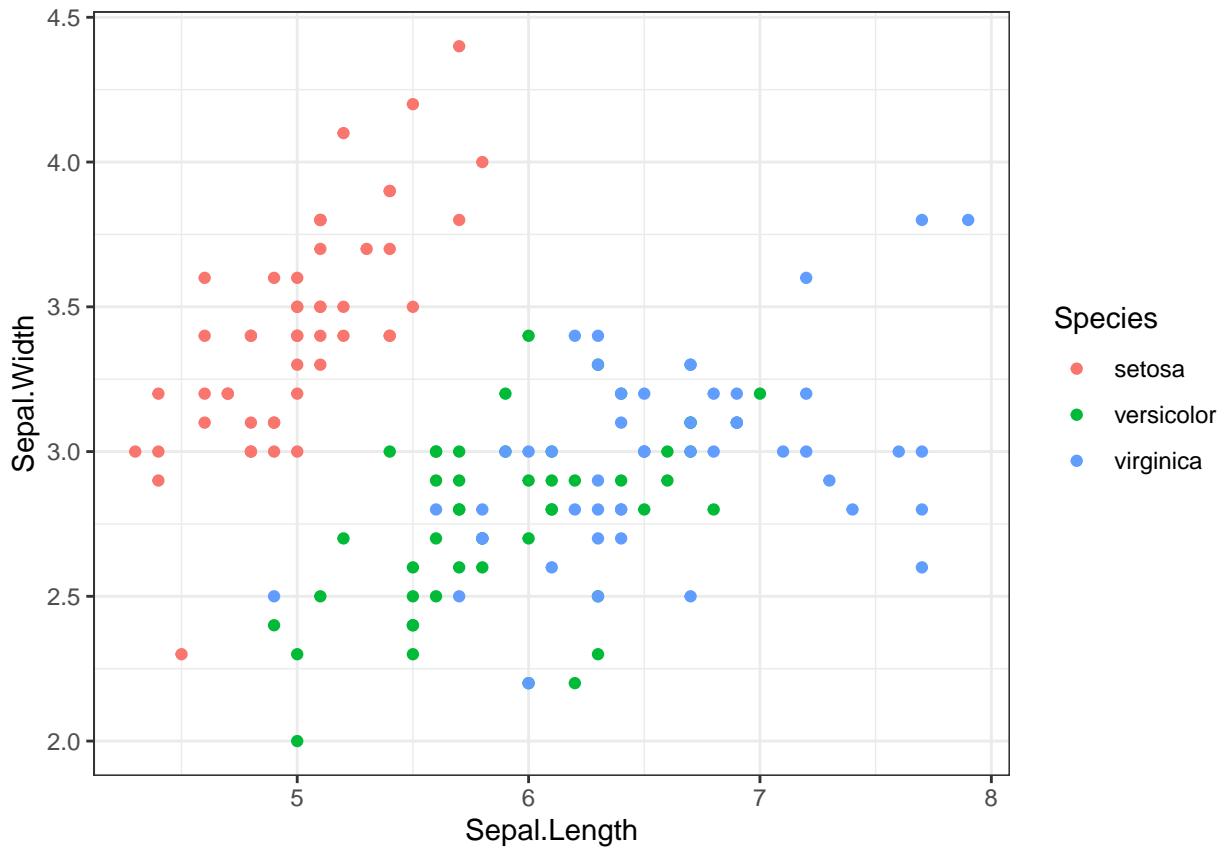
and let's have a look at the structure.

```
str(iris)
```

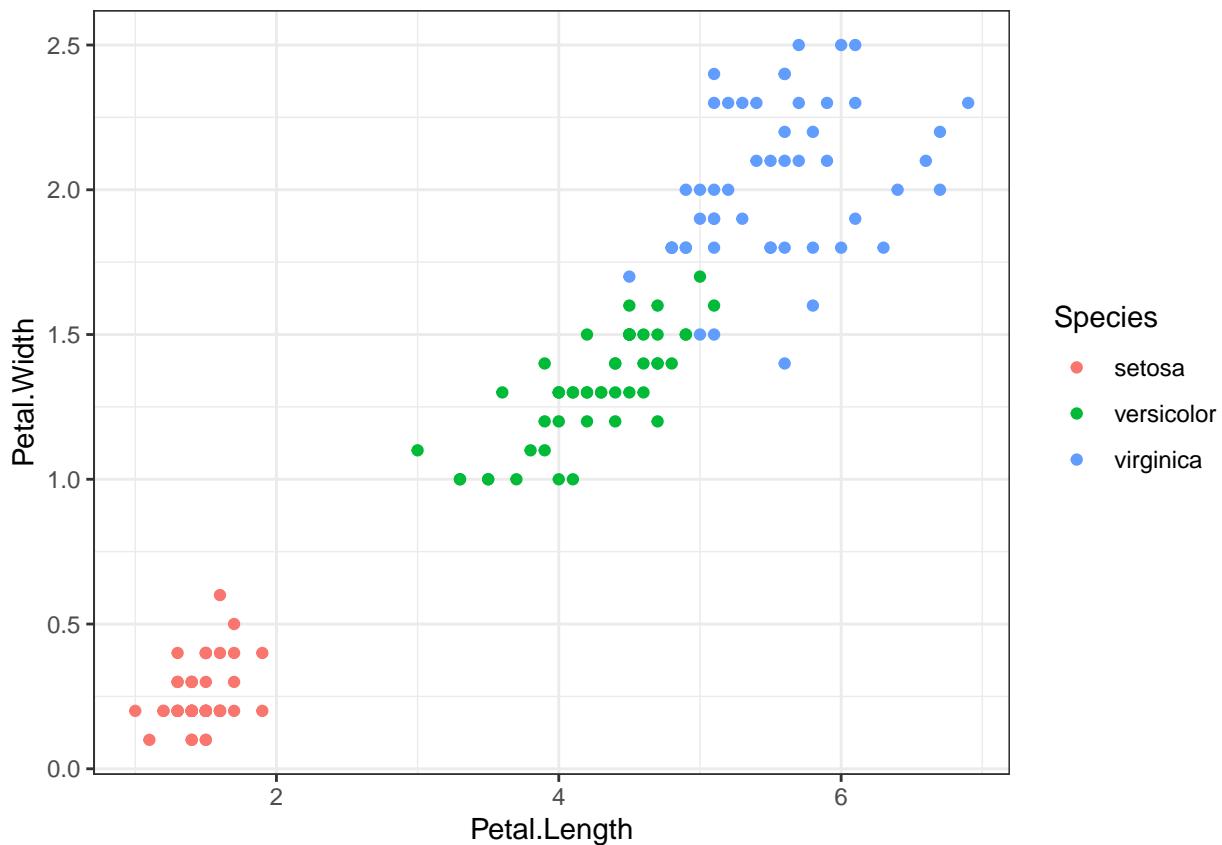
```
## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Have a quick look at the Data

```
iris %>%  
  ggplot(aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +  
  geom_point()
```



```
iris %>%  
  ggplot(aes(x = Petal.Length, y = Petal.Width, color = Species)) +  
  geom_point()
```



Prepare the Data for Training

```
set.seed(123)
indices <- createDataPartition(iris$Species, p=.85, list = F)
```

Why do we use the caret function

Look at the distribution of our different prediction classes in the train and test datasets:

```
iris %>%
  mutate(train = row_number() %in% indices) %>%
  select(Species, train) %>%
  table()

##          train
## Species      FALSE TRUE
##   setosa       7    43
##   versicolor   7    43
##   virginica    7    43
```

Now compare this to the “random” approach:

```
iris %>%
  mutate(train = runif(nrow(iris)) < .85) %>%
  select(Species, train) %>%
  table()
```

```
##          train
```

```

## Species      FALSE TRUE
##   setosa       8   42
##   versicolor   9   41
##   virginica    4   46

```

So using the `createDataPartition` function makes sure that no class is over- or underrepresented relative to the total occurrence in the two sets

Create some easy Variables to access Data

```

train <- iris %>%
  slice(indices)
test_in <- iris %>%
  slice(-indices) %>%
  select(-Species)
test_truth <- iris %>%
  slice(-indices) %>%
  pull(Species)

```

Train the Neural Network

Call the `neuralnet` function creating a network with two hidden layers containing 4 and 3 neurons (probably way too complex for our problem here).

```

set.seed(123)
iris_net <- neuralnet(Species ~ ., train, hidden = c(4,3))

```

Plot the resulting network including the weights

```
plot(iris_net)
```

Make Predictions

```

test_results <- neuralnet::compute(iris_net, test_in)
test_results$net.result

```

```

##           [,1]          [,2]          [,3]
## [1,] 1.000245e+00 7.184849e-05 -0.002387740
## [2,] 9.987040e-01 1.628771e-03 -0.002490078
## [3,] 1.001452e+00 -2.101939e-03  0.004385866
## [4,] 1.002581e+00 -2.842763e-03  0.001656209
## [5,] 1.001264e+00 -8.321009e-04 -0.003200581
## [6,] 1.002004e+00 -1.808872e-03 -0.001544580
## [7,] 1.000806e+00 -1.897685e-03  0.007483295
## [8,] -1.763162e-03 1.002726e+00 -0.001031508
## [9,] 7.994995e-04 9.993755e-01  0.004476715
## [10,] 7.846437e-03 9.911943e-01  0.012387450
## [11,] 4.638568e-04 9.997797e-01  0.003998233
## [12,] 2.440913e-03 8.415458e-01  0.162336378
## [13,] -1.673082e-04 1.001323e+00 -0.002393173
## [14,] -2.909598e-03 1.004421e+00 -0.004871084
## [15,] 9.969198e-04 2.038949e-02  0.982788747
## [16,] 7.272816e-05 2.187307e-02  0.978872481
## [17,] 1.301818e-03 2.005262e-02  0.983011077
## [18,] 1.560035e-03 1.924179e-02  0.986883733

```

```

## [19,] -1.171284e-03 2.355382e-02 0.975818020
## [20,] 1.444380e-03 2.028897e-02 0.980353882
## [21,] -2.215089e-03 2.523489e-02 0.971356514

```

Find class (i.e. output neuron) with the highest probability and convert this back into a factor

```

test_pred <- apply(test_results$net.result, 1, which.max)
test_pred <- factor(levels(test_truth)[test_pred], levels = levels(test_truth))
test_pred

```

```

## [1] setosa      setosa      setosa      setosa      setosa      setosa
## [7] setosa      versicolor  versicolor  versicolor  versicolor  versicolor
## [13] versicolor  versicolor  virginica  virginica  virginica  virginica
## [19] virginica   virginica   virginica
## Levels: setosa versicolor virginica

```

Evaluate the Results

```

conf_matrix <- confusionMatrix(test_truth, test_pred)
conf_matrix

```

```

## Confusion Matrix and Statistics
##
##          Reference
## Prediction    setosa versicolor virginica
##   setosa        7       0       0
##   versicolor    0       7       0
##   virginica     0       0       7
##
## Overall Statistics
##
##          Accuracy : 1
##                 95% CI : (0.8389, 1)
##   No Information Rate : 0.3333
##   P-Value [Acc > NIR] : 9.56e-11
##
##          Kappa : 1
##
## Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##          Class: setosa Class: versicolor Class: virginica
## Sensitivity           1.0000           1.0000           1.0000
## Specificity            1.0000           1.0000           1.0000
## Pos Pred Value         1.0000           1.0000           1.0000
## Neg Pred Value         1.0000           1.0000           1.0000
## Prevalence              0.3333           0.3333           0.3333
## Detection Rate          0.3333           0.3333           0.3333
## Detection Prevalence    0.3333           0.3333           0.3333
## Balanced Accuracy        1.0000           1.0000           1.0000

```

Optimize Network Structure

First we need to remodel the data due to some limitations of caret

```

xor <- model.matrix(~ Species, iris)
xor[,1] <- ifelse(xor[,2] %in% 0 & xor[,3] %in% 0, 1, 0)
colnames(xor)[1] <- 'Speciesetosa'
head(xor)

##   Speciesetosa Speciesversicolor Speciesvirginica
## 1           1              0                  0
## 2           1              0                  0
## 3           1              0                  0
## 4           1              0                  0
## 5           1              0                  0
## 6           1              0                  0

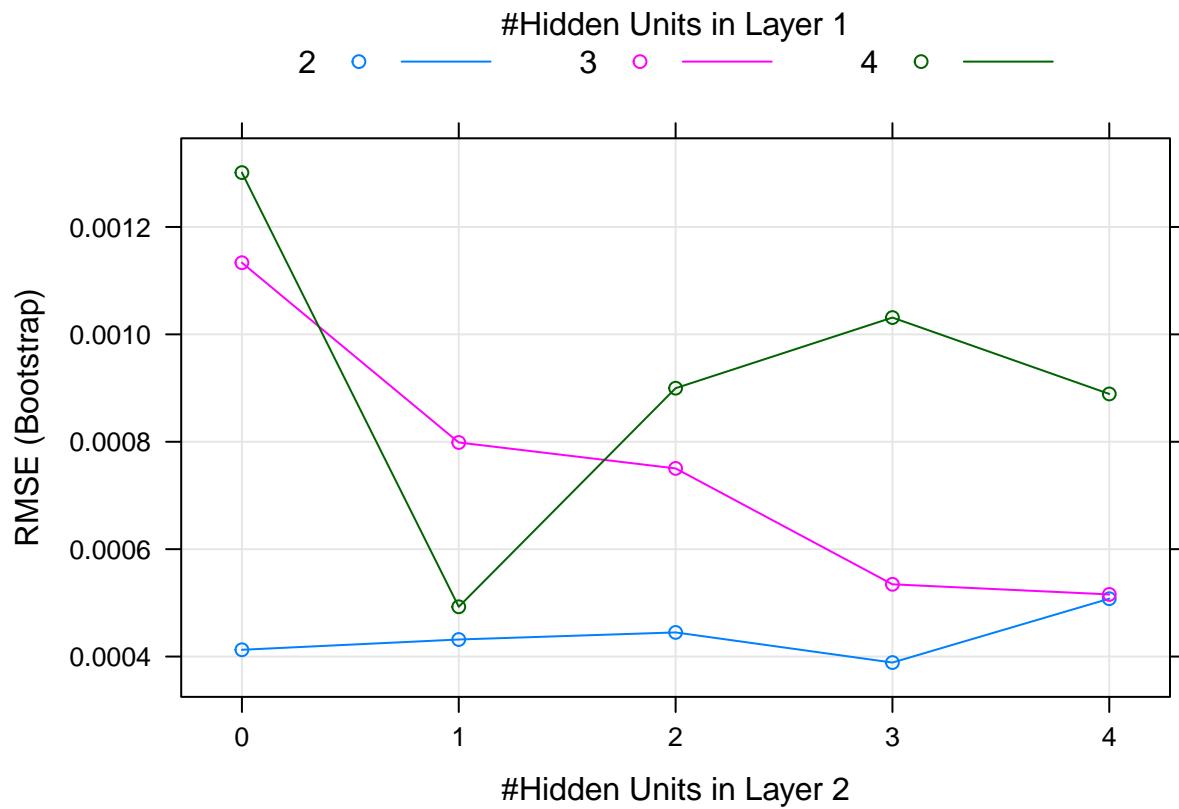
iris_mod <- cbind(xor, iris %>% select(-Species))
train_mod <- iris_mod %>% slice(indices)

set.seed(142)
formula = Speciesetosa + Speciesversicolor + Speciesvirginica ~ Sepal.Length + Sepal.Width + Petal.Length
models <- train(formula, train_mod,
                 method="neuralnet",
                 ### Parameters for layers
                 tuneGrid = expand.grid(.layer1=c(2:4), .layer2=c(0:4), .layer3=c(0)),
                 ### Parameters for optimisation
                 learningrate = 0.01,
                 threshold = 0.001,
                 stepmax = 50000
)

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

And have a look at the different models
plot(models)

```



And try out the best model

```
set.seed(42)
best_model <- neuralnet(
  formula,
  iris_mod %>% slice(indices),
  hidden = c(2,3),
  learningrate = 0.01,
  threshold = 0.01,
  stepmax = 50000
)
plot(best_model)
```

A word of caution: the `iris` dataset is almost too easy to separate, so it is very easy to get stuck in local minima i.e. your results may depend a lot on the random seeds chosen.

Do we need to Scale and Center the Inputs?*

While it is advised to do so (for potentially faster training times) neural networks can also work with unscaled data (and adjust the weights accordingly)

```
tuGrid <- expand.grid(size=1:8, decay=3**(-6:1))

trCtrl <- trainControl(
  method = 'repeatedcv',
  number = 10,
  repeats = 10,
  returnResamp = 'final'
)
```

```

model_noscale <- train(
  x = iris[,1:4], y = iris[,5],
  method = 'nnet', metric = 'Kappa',
  trace = FALSE,
  tuneGrid = tuGrid,
  trControl = trCtrl
)

model_scaled <- train(
  x = (iris[,1:4]), y = iris[,5],
  method = 'nnet', metric = 'Kappa',
  preProcess = c('center', 'scale'),
  trace = FALSE,
  tuneGrid = tuGrid,
  trControl = trCtrl
)

model_noscale$finalModel

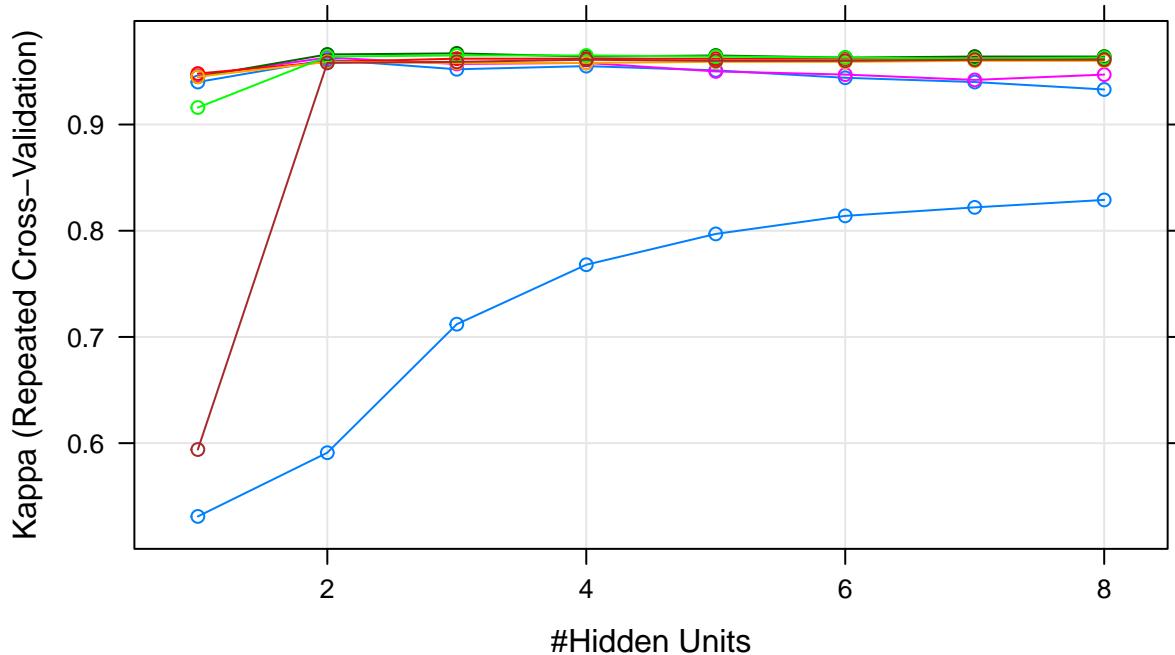
## a 4-3-3 network with 27 weights
## inputs: Sepal.Length Sepal.Width Petal.Length Petal.Width
## output(s): .outcome
## options were - softmax modelling decay=0.01234568
model_scaled$finalModel

## a 4-2-3 network with 19 weights
## inputs: Sepal.Length Sepal.Width Petal.Length Petal.Width
## output(s): .outcome
## options were - softmax modelling decay=0.01234568
plot(model_noscale)

```

Weight Decay

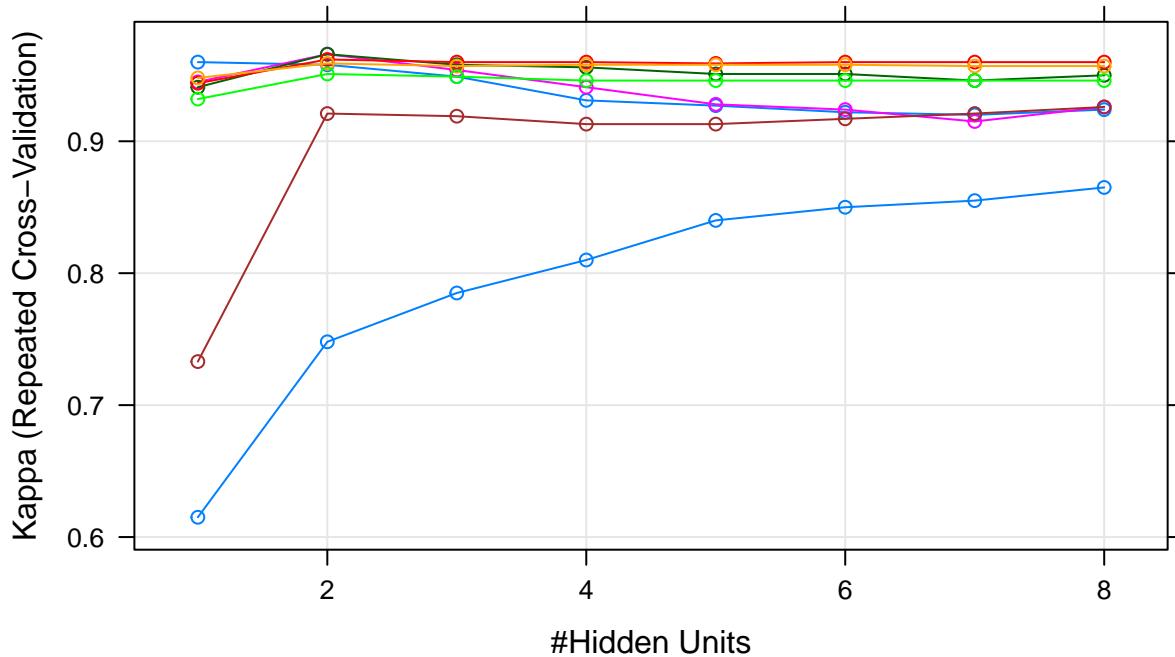
0.0123456790123457	0.037037037037037	0.1111111111111111	0.3333333333333333
○ —	○ —	○ —	○ -
○ —	○ —	○ —	○ -



```
plot(model_scaled)
```

Weight Decay

0.0123456790123457	0.037037037037037	0.1111111111111111	0.3333333333333333
○ —	○ —	○ —	○ -
○ —	○ —	○ —	○ -



```
boxplot(data.frame(model_noscale$resample$Kappa, model_scaled$resample$Kappa))
```

