Classification Methods: Implementations in R

Verena Koeck, Sourav Adhikari

30 5 2022

```
knitr::opts_chunk$set(echo = TRUE)
```

The IRIS dataset

The iris dataset is a built-in dataset in R that contains measurements on 4 different attributes (in centimeters) for 50 flowers from 3 different species.

https://www.kaggle.com/code/vinayshaw/iris-species-100-accuracy-using-naive-bayes/notebook

```
data(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

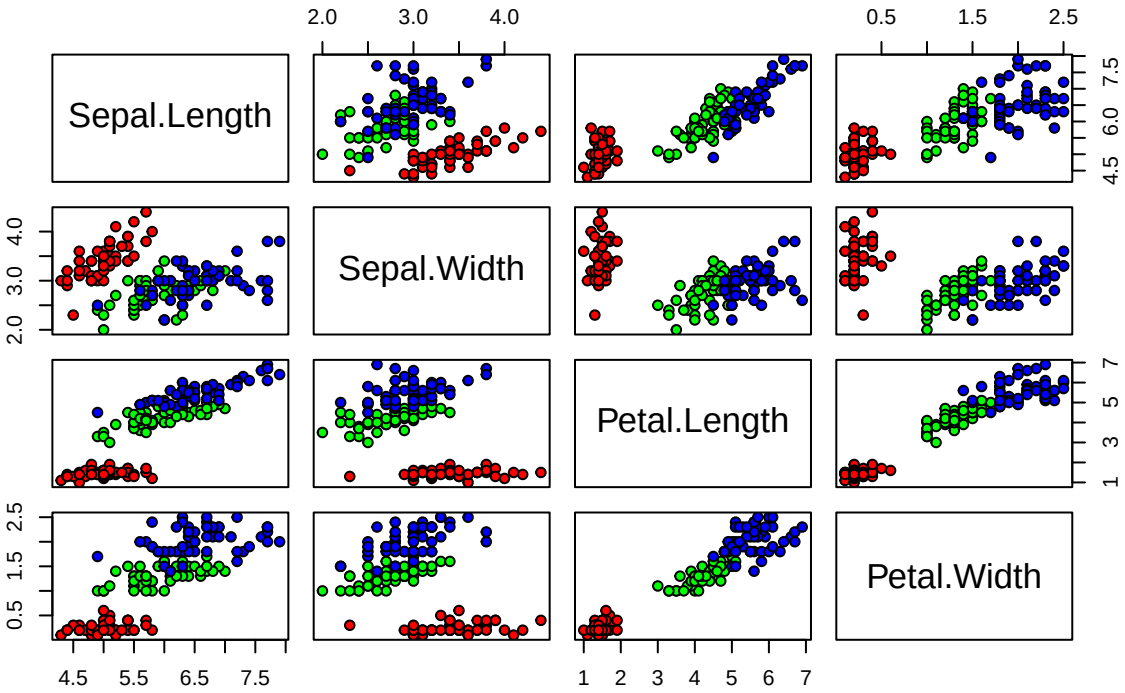
```
dim(iris)
```

```
## [1] 150   5
```

```
summary(iris)
```

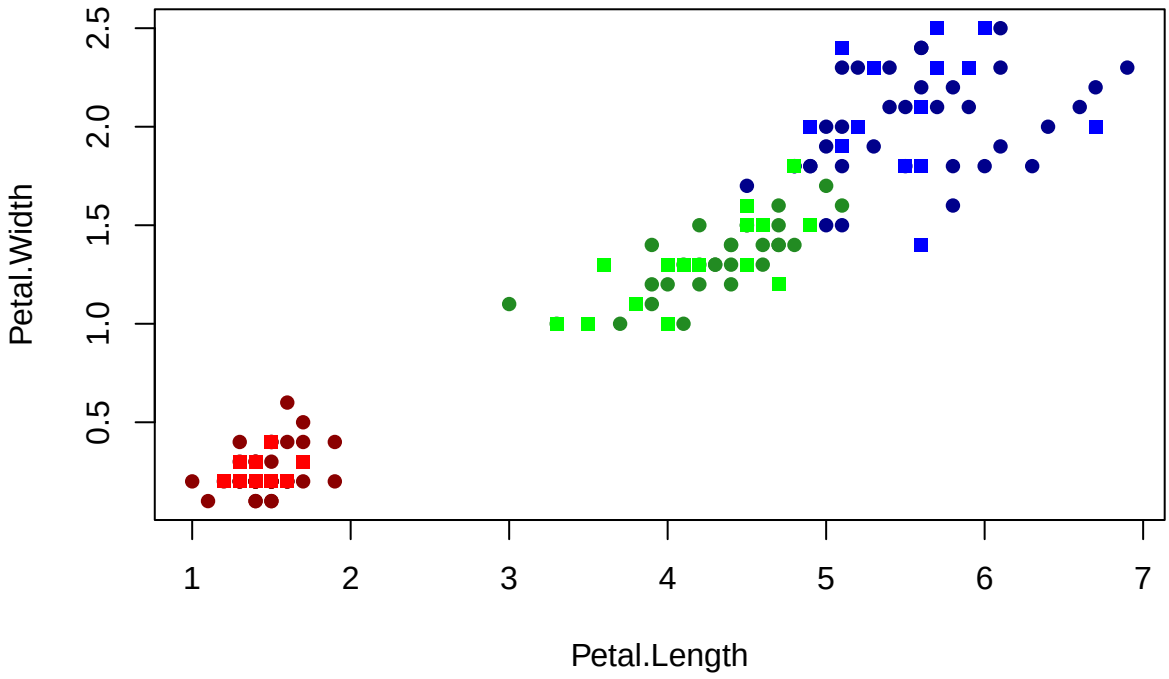
```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
pairs(iris[1:4],main="Iris Data(red=setosa,green=versicolor,blue=virginica)", pch=21,bg=c("red","green"
```




```
# Determine sample size
set.seed(3)
ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.67, 0.33))

# Split the `iris` data
iris.training <- iris[ind==1, 1:4]
iris.test <- iris[ind==2, 1:4]

# Split the class attribute
iris.trainingtarget <- iris[ind==1, 5]
iris.testtarget <- iris[ind==2, 5]

make.grid<-function(x,n=200){
  grange=apply(x,2,range)
  x1=seq(from=grange[1,1],to=grange[2,1],length=n)
  x2=seq(from=grange[1,2],to=grange[2,2],length=n)
  expand.grid(X1=x1,X2=x2)
}

xtrain<-iris.training[,3:4]
xtest<-iris.test[,3:4]
ytest<-iris.testtarget
ytrain<-iris.trainingtarget
xgrid = make.grid(xtest, n = 150)
colnames(xgrid)<-c("Petal.Length", "Petal.Width")

plot(xtrain[,1],xtrain[,2],col=c("darkred","forestgreen","darkblue")[unclass(ytrain)],pch=16,xlab="Petal
```

```
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```



Naive Bayes Estimation

```
library("e1071")
model_nb<-naiveBayes(iris.training, iris.trainingtarget)
table(predict(model_nb, iris.test), iris.testtarget,  dnn=list('predicted','actual'))
```

```
##             actual
## predicted    setosa versicolor virginica
##   setosa         11          0         0
##   versicolor      0         20         1
##   virginica       0          1        14
```

```
model_nb$apriori
```

```
## iris.trainingtarget
##     setosa versicolor  virginica
##         39         29         35
```

```
model_nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = iris.training, y = iris.trainingtarget)
##
## A-priori probabilities:
## iris.trainingtarget
##     setosa versicolor  virginica
##  0.3786408  0.2815534  0.3398058
```

```
##
## Conditional probabilities:
##                     Sepal.Length
## iris.trainingtarget     [,1]      [,2]
##          setosa     4.953846 0.3135922
##          versicolor 6.041379 0.5401833
##          virginica  6.680000 0.6596791
##
##                     Sepal.Width
## iris.trainingtarget     [,1]      [,2]
##          setosa     3.433333 0.3206353
##          versicolor 2.810345 0.2845357
##          virginica  2.982857 0.3560285
##
##                     Petal.Length
## iris.trainingtarget     [,1]      [,2]
##          setosa     1.466667 0.1811271
##          versicolor 4.296552 0.4648216
##          virginica  5.560000 0.5941875
##
##                     Petal.Width
## iris.trainingtarget      [,1]      [,2]
##          setosa     0.2461538 0.1143544
##          versicolor 1.3275862 0.1830233
##          virginica  2.0028571 0.2617652
```

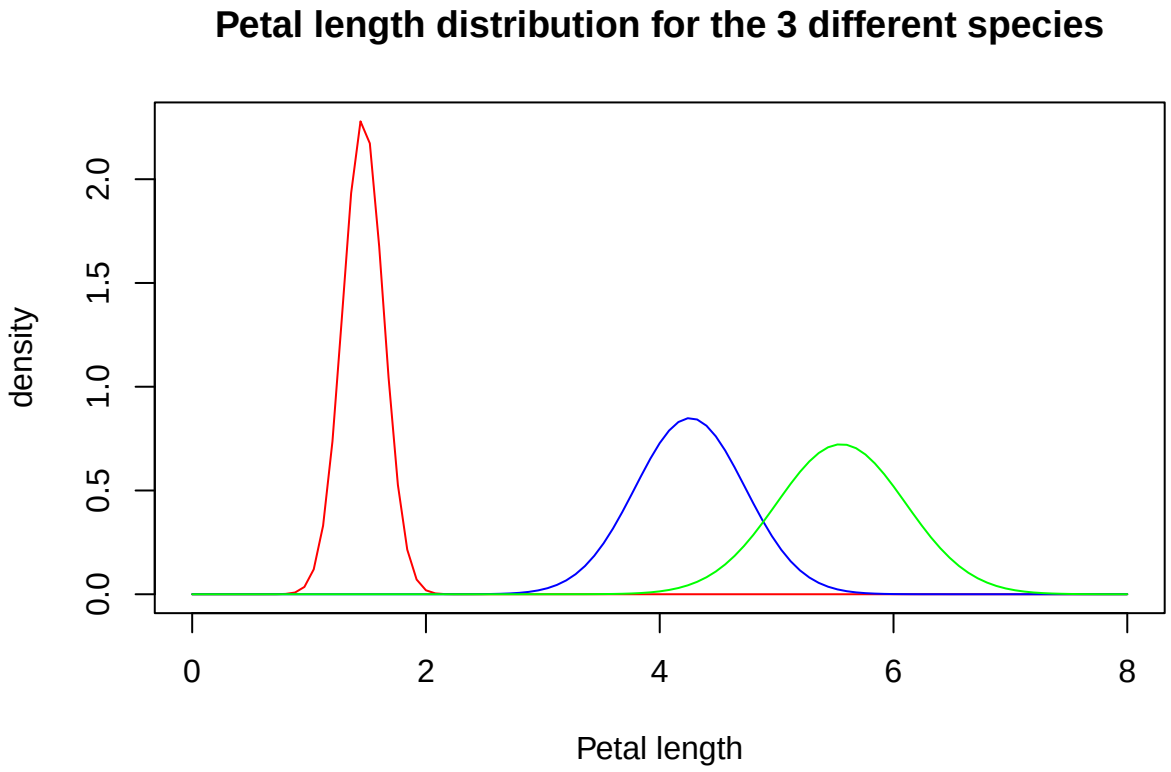
```
model_nb$tables$Petal.Length
```

```
##                     Petal.Length
## iris.trainingtarget     [,1]      [,2]
##          setosa     1.466667 0.1811271
##          versicolor 4.296552 0.4648216
##          virginica  5.560000 0.5941875
```

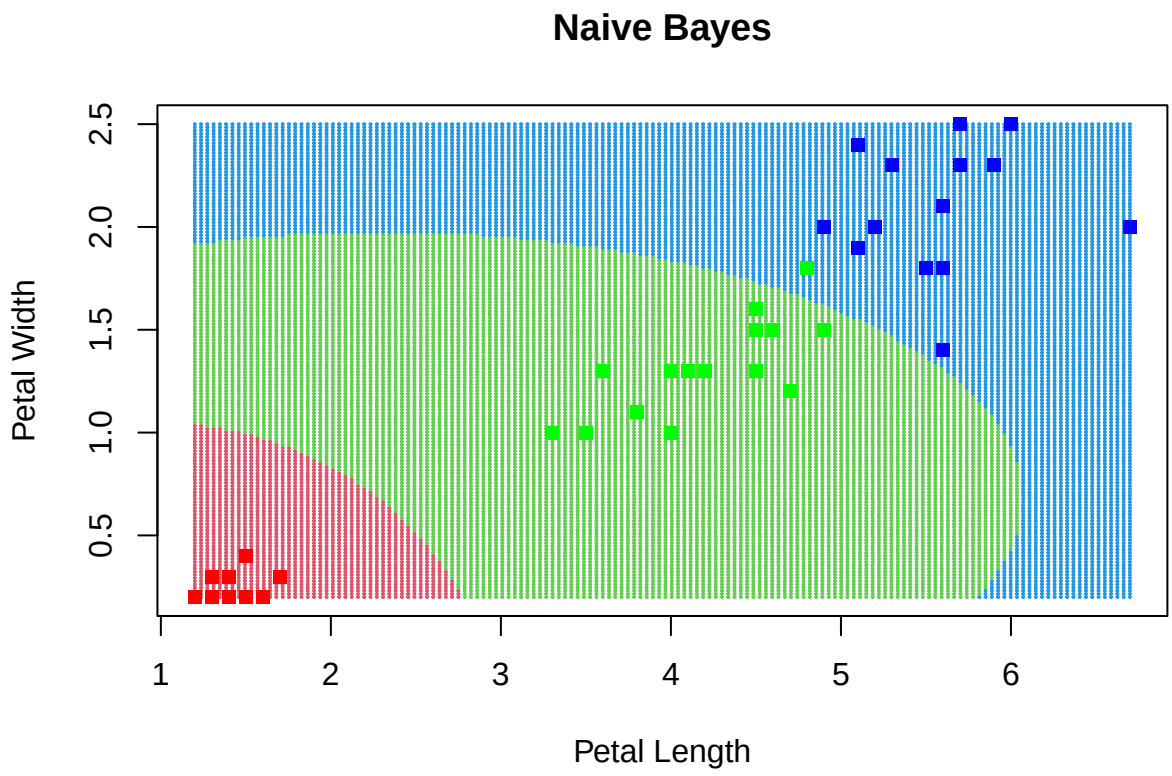
```
score_nb<-mean(predict(model_nb, iris.test)== iris.testtarget)
score_nb
```

```
## [1] 0.9574468
```

```
plot(function(x) dnorm(x, 1.462, 0.1736640), 0, 8,  col="red",xlab="Petal length", ylab="density", main=
curve(dnorm(x, 4.260, 0.4699110), add=TRUE, col="blue")
curve(dnorm(x, 5.552, 0.5518947), add=TRUE, col="green")
```

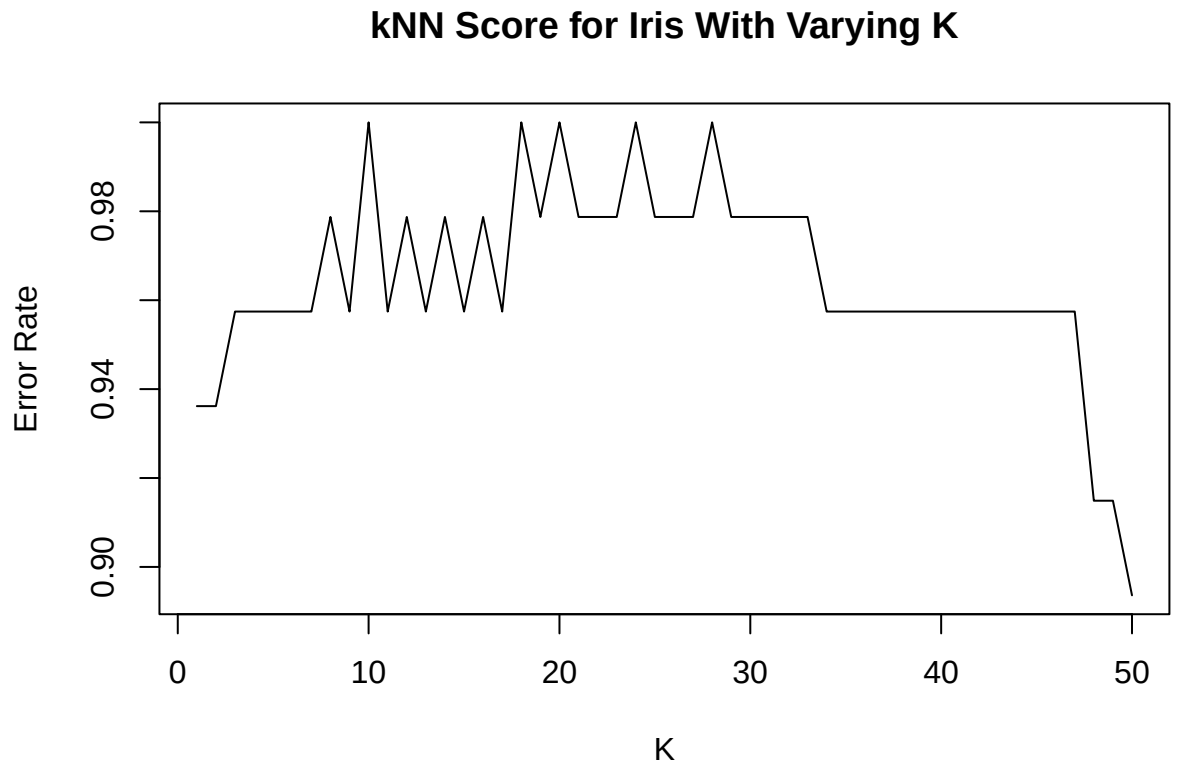



```
#Illustration
model_nb<-naiveBayes(xtrain, ytrain)
nb_x <- predict(model_nb,newdata=list("Petal.Length"=xgrid[,1], "Petal.Width"= xgrid[,2]))
plot(xgrid,col=c(2,3,4)[as.numeric(nb_x)],pch = 20,cex = .2, main = "Naive Bayes",xlab="Petal Length",y
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```

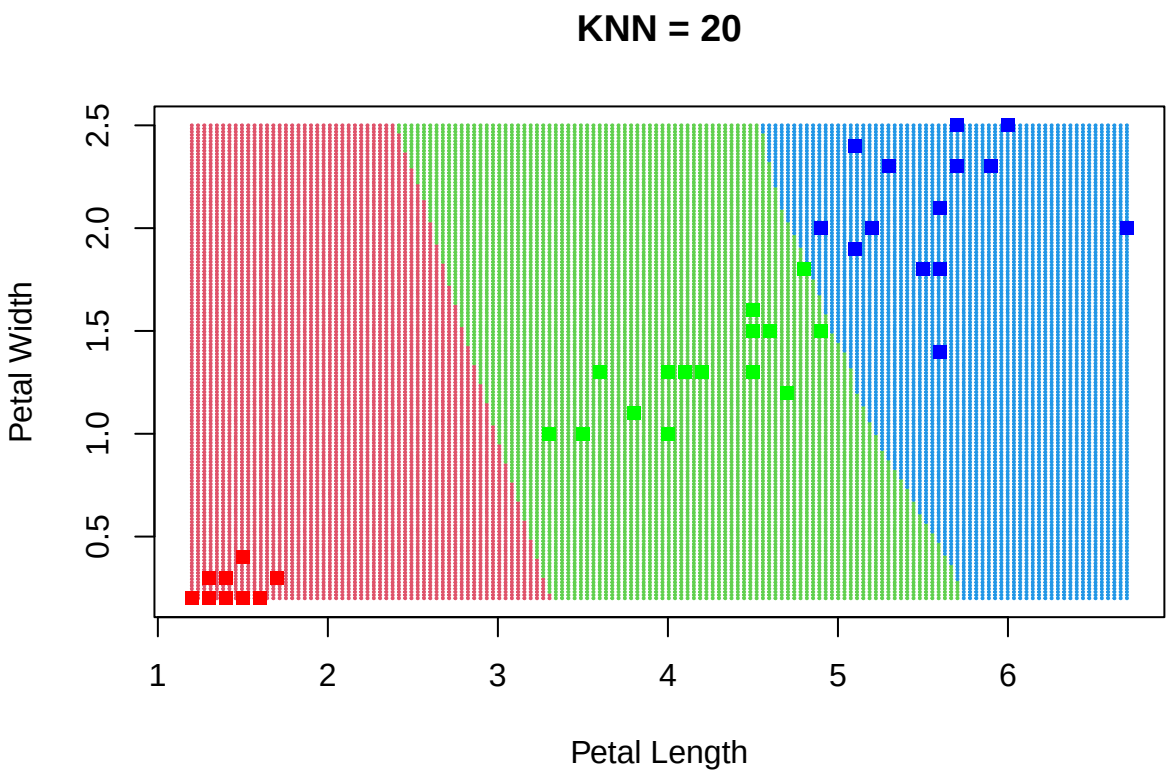


K-Nearest-Neighbors estimation

```
library(FNN) #knn regression

#First Attempt to Determine Right K####
iris_acc<-numeric() #Holding variable

for(i in 1:50){
  #Apply knn with k = i
  predict<-knn(iris.training,iris.test,
               iris.trainingtarget,k=i)
  iris_acc<-c(iris_acc,
              mean(predict==iris.testtarget))
}
#Plot k= 1 through 30
plot(iris_acc,type="l",ylab="Error Rate",
     xlab="K",main="kNN Score for Iris With Varying K")
```




```
#Illustration




k<-20
knn_x <- knn(train=xtrain,  test=xgrid , cl=factor(ytrain), k=i)
plot(xgrid,col=c(2,3,4)[as.numeric(knn_x)],pch = 20,cex = .2, main = paste("KNN =", k),xlab="Petal Leng
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```



Logistic Regression

```
library(nnet)
fit <- multinom(iris.trainingtarget ~  ., data=cbind(iris.training,iris.trainingtarget))
```

```
## # weights:  18 (10 variable)
## initial  value 113.157066
## iter  10 value 8.067276
## iter  20 value 4.933856
## iter  30 value 4.747766
## iter  40 value 4.722652
## iter  50 value 4.719503
## iter  60 value 4.719117
## iter  70 value 4.719039
## iter  80 value 4.718826
## iter  90 value 4.718567
## iter 100 value 4.718444
## final  value 4.718444
## stopped after 100 iterations
```

```
summary(fit)
```

```
## Call:
## multinom(formula = iris.trainingtarget ~ ., data = cbind(iris.training,
##     iris.trainingtarget))
##
## Coefficients:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    12.53520    -2.363754   -8.352525     9.143761   0.6613419
```

```
## virginica    -17.63227    -5.417008  -11.838608    16.440940  14.9045856
##
## Std. Errors:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    65.45560     130.2007    150.4919     154.9567    105.1227
## virginica     65.72887     130.2188    150.5600     155.0391    105.2455
##
## Residual Deviance: 9.436889
## AIC: 29.43689
```

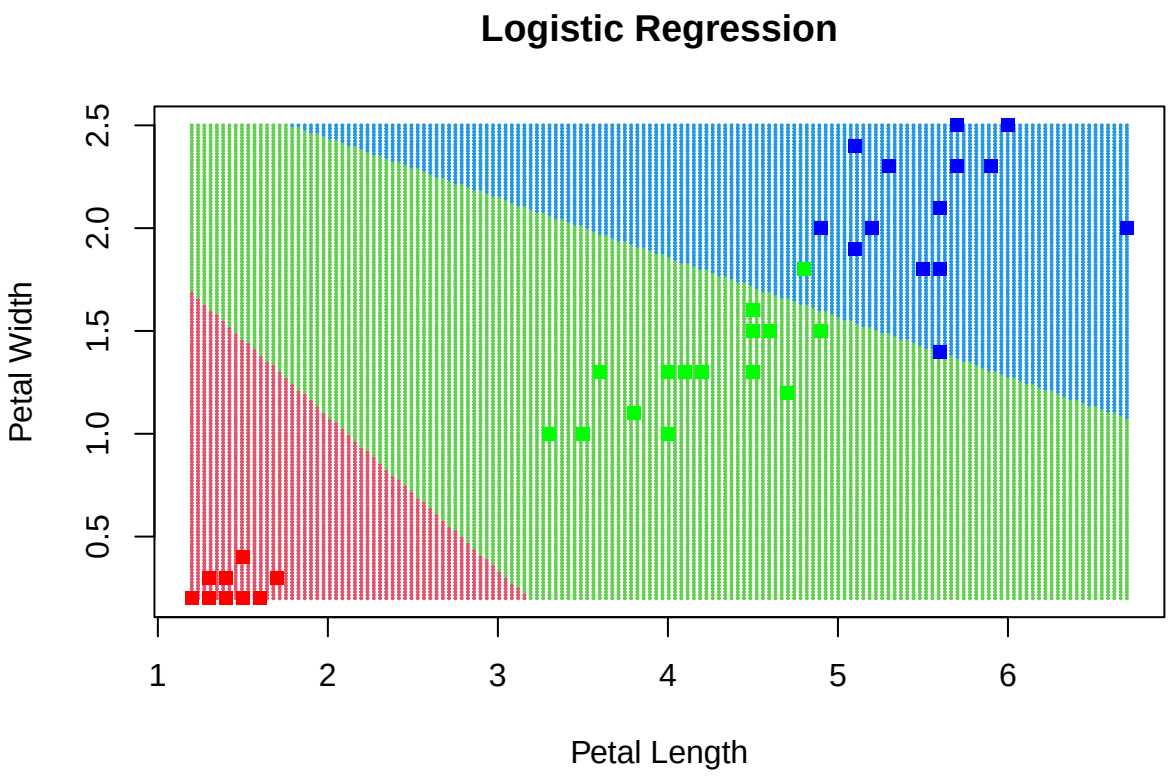
```
predicted.classes <-  predict(fit,iris.test)
head(predicted.classes)
```

```
## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
# Model accuracy
scorelog<-mean(predicted.classes == iris.testtarget)


log_model <- multinom(ytrain ~ ., data=cbind(xtrain))
```

```
## # weights:  12 (6 variable)
## initial  value 113.157066
## iter  10 value 9.675025
## iter  20 value 7.426201
## iter  30 value 7.384976
## iter  40 value 7.356171
## iter  50 value 7.346781
## iter  60 value 7.342964
## iter  70 value 7.340718
## iter  80 value 7.338835
## iter  90 value 7.336898
## iter 100 value 7.335650
## final  value 7.335650
## stopped after 100 iterations
```

```
log_x <- predict(log_model,newdata=xgrid)
plot(xgrid,col=c(2,3,4)[as.numeric(log_x)],pch = 20,cex = .2, main = "Logistic Regression",xlab="Petal
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```

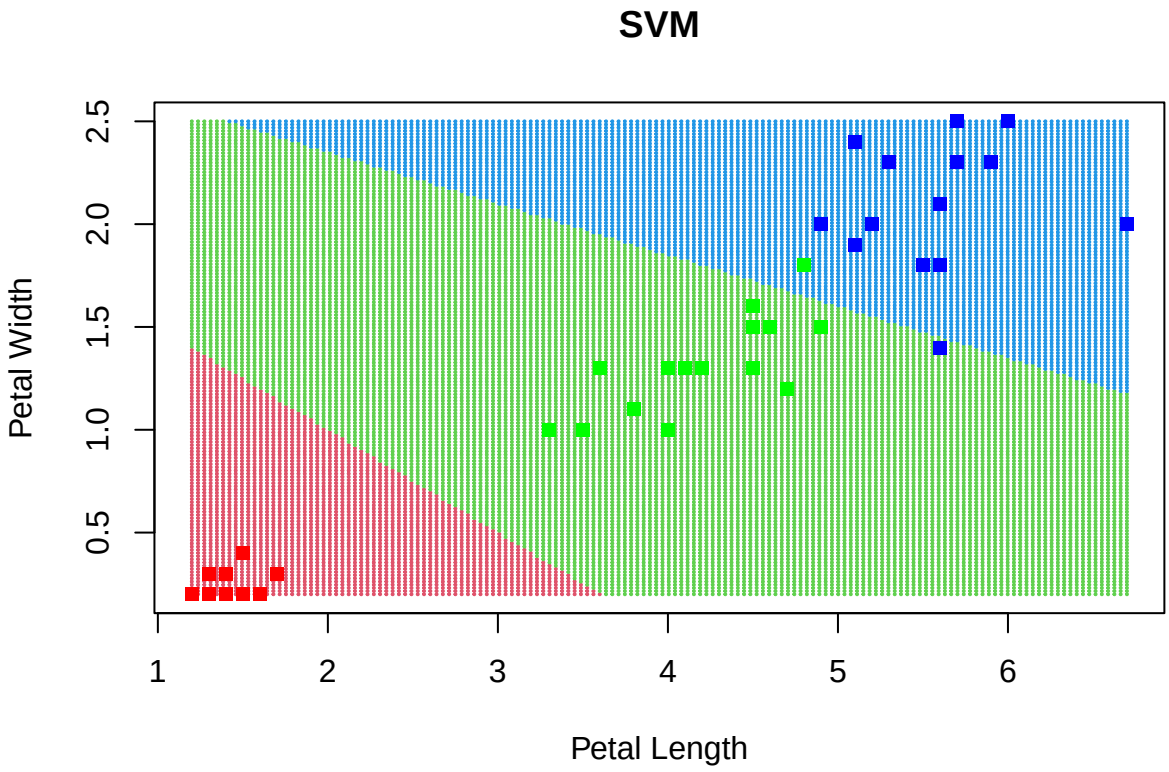


Support Vector Machine

```
library("e1071")

svm_model <- svm(iris.trainingtarget ~  ., data=cbind(iris.training,iris.trainingtarget), kernel="linea
#plot(svm_model, data=cbind(iris.training,iris.trainingtarget),Petal.Width~Petal.Length, slice = list(S
pred = predict(svm_model,iris.test)
tab<-table(iris.testtarget,pred)
score_svm<-sum(diag(tab)/sum(tab))
score_svm
```

```
## [1] 0.9787234
```

```
#Illustration
svm_model <- svm(ytrain ~ ., data=cbind(xtrain),   kernel="linear")
svm_x <- predict(svm_model,newdata=cbind(xgrid[,1],xgrid[,2]))
plot(xgrid,col=c(2,3,4)[as.numeric(svm_x)],pch = 20,cex = .2, main = "SVM",xlab="Petal Length",ylab="Pe
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```



Neural Network

```
library(neuralnet)


iris$setosa <- iris$Species=="setosa"
iris$virginica <- iris$Species == "virginica"
iris$versicolor <- iris$Species == "versicolor"
iris.training_nn <- iris[ind==1,]
iris.test_nn <- iris[(ind==2),]



library(neuralnet)
iris.net <- neuralnet(setosa+versicolor+virginica ~
                        Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                      data=iris.training_nn, hidden=c(10,10), rep = 5, err.fct = "ce", act.fct = "logis
                      linear.output = F, lifesign = "minimal", stepmax = 1000000,
                      threshold = 0.01)


#plot(iris.net, rep="best")

iris.prediction <- compute(iris.net, iris.test_nn)
idx <- apply(iris.prediction$net.result, 1, which.max)
predicted <- c('setosa', 'versicolor', 'virginica')[idx]
tab<-table(predicted, iris.test_nn$Species)
```

```
score_net<-sum(diag(tab)/sum(tab))
score_net


#Illustration
library(neuralnet)
iris.net <- neuralnet(setosa+versicolor+virginica ~ Petal.Length + Petal.Width,
                      data=iris.training_nn, hidden=c(10,10), rep = 5, err.fct = "ce", act.fct = "logis
                      linear.output = F, lifesign = "minimal", stepmax = 1000000,
                      threshold = 0.01)

iris.prediction <- compute(iris.net, cbind(xgrid[,1],xgrid[,2]))
idx <- apply(iris.prediction$net.result, 1, which.max)
predicted_x <- c(1,2, 3)[idx]

#plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Neural Network",xlab="Peta
#points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)

plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Neural Network",xlab="Petal

#points(xtrain[,1],xtrain[,2],col=c("darkred","forestgreen","darkblue")[unclass(ytrain)],pch=".",cex=5)
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```

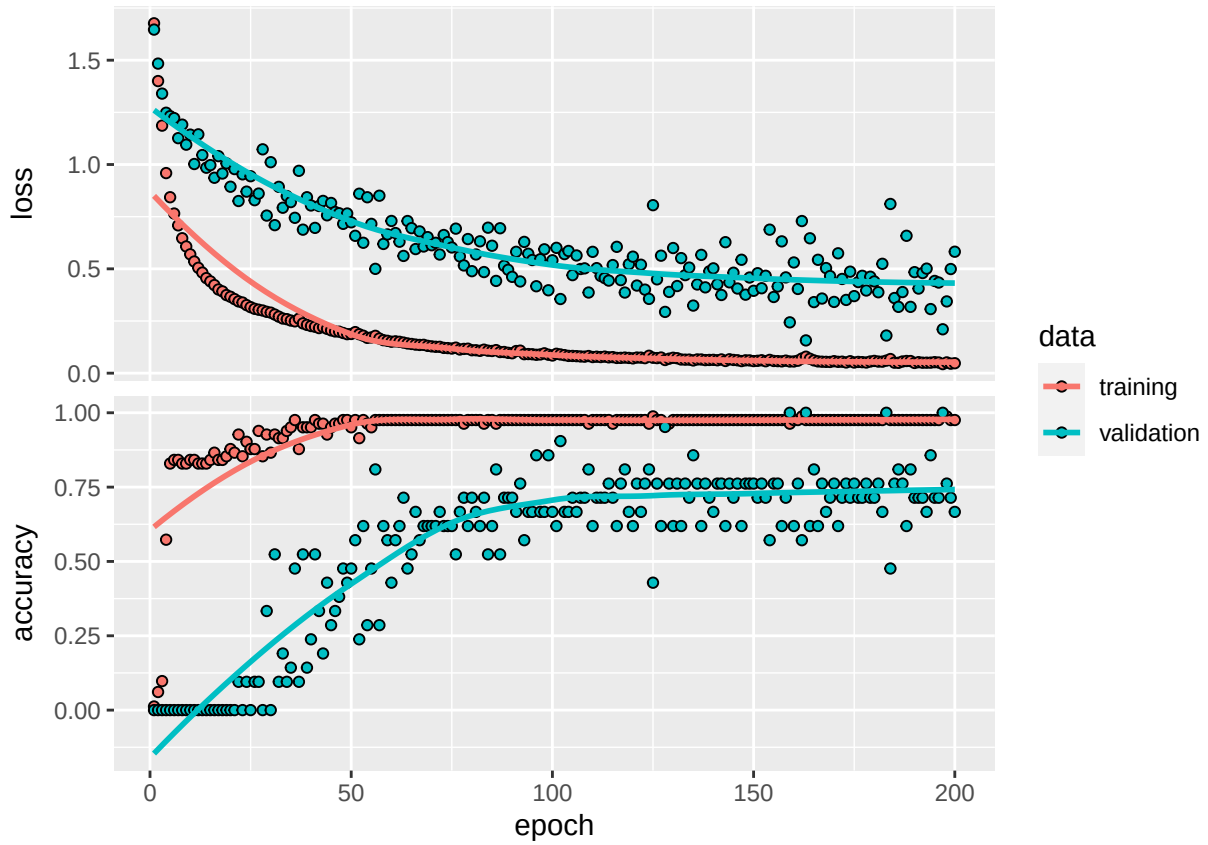
```
library(keras)
library(tensorflow)
library(datasets)

data(iris)

#iris[,5] <- as.numeric(iris[,5]) -1
#iris <- as.matrix(iris)
#dimnames(iris) <- NULL


# Split the `iris` data

#iris.training <- iris[ind==1, 1:4]
#iris.test <- iris[ind==2, 1:4]

# Split the class attribute

iris.training_keras <- as.matrix(iris.training)
iris.test_keras <-as.matrix(iris.test)

iris.trainingtarget_keras <- as.numeric(iris.trainingtarget)-1
iris.testtarget_keras <- as.numeric(iris.testtarget) -1
iris.trainLabels <- to_categorical(iris.trainingtarget_keras)
```

```
## Loaded Tensorflow version 2.3.1
```

```
iris.testLabels <- to_categorical(iris.testtarget_keras)


# Initialize a sequential model
```

```r
model3 <- keras_model_sequential()

# Add layers to the model
model3 %>%
  layer_dense(units = 20, activation = 'relu', input_shape = c(4)) %>%
  layer_dense(units = 10, activation = 'relu') %>%
  layer_dense(units = 3, activation = 'softmax')

# Compile the model
model3 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = 'accuracy'
)

x = as.matrix(apply(iris.training_keras, 2, function(x) (x-min(x))/(max(x) - min(x))))


# Fit the model to the data
fit=model3 %>% fit(
  iris.training_keras, iris.trainLabels,
  epochs = 200, batch_size = 5,
  validation_split = 0.2
)

plot(fit)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



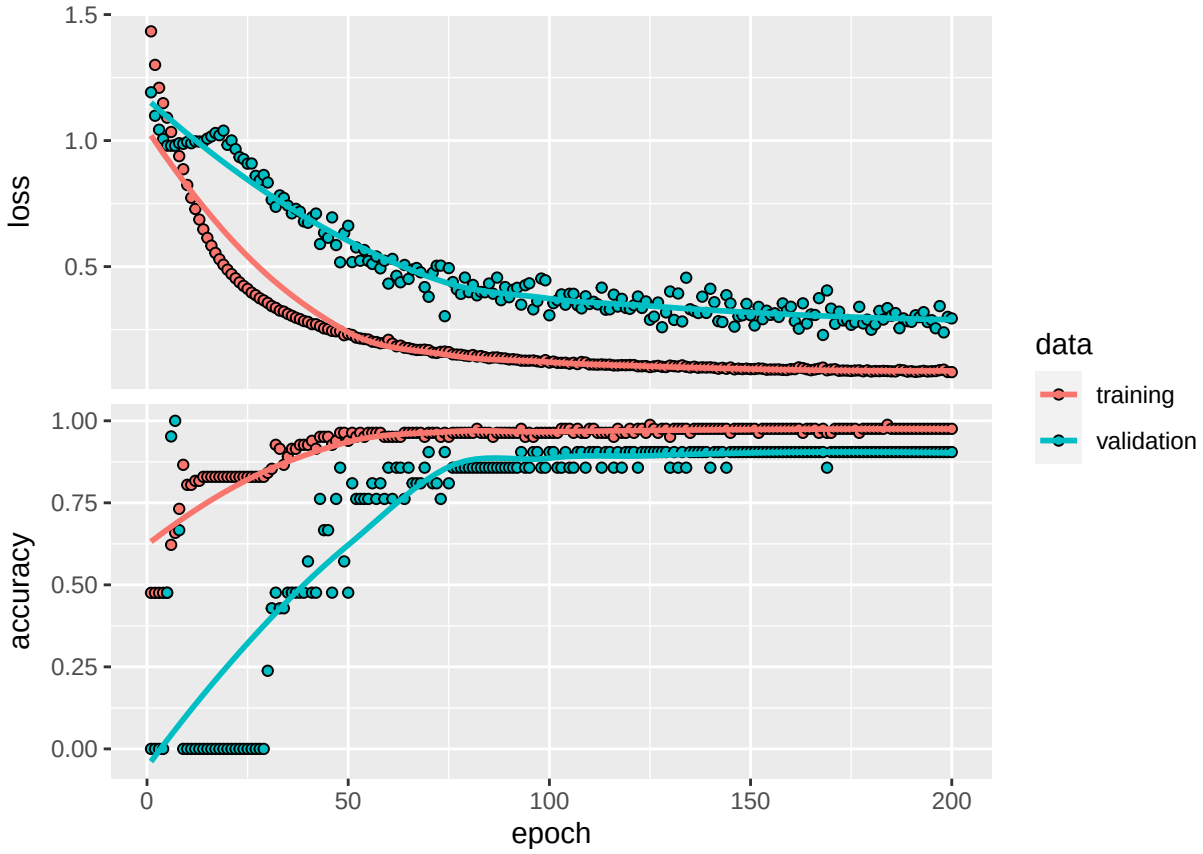

```
# Evaluate the model
score <- model3 %>% evaluate(iris.test_keras, iris.testLabels, batch_size = 128 )

# Print the score
score_keras<-score['acc']
print(score_keras)
```

```
## <NA>
##   NA
```

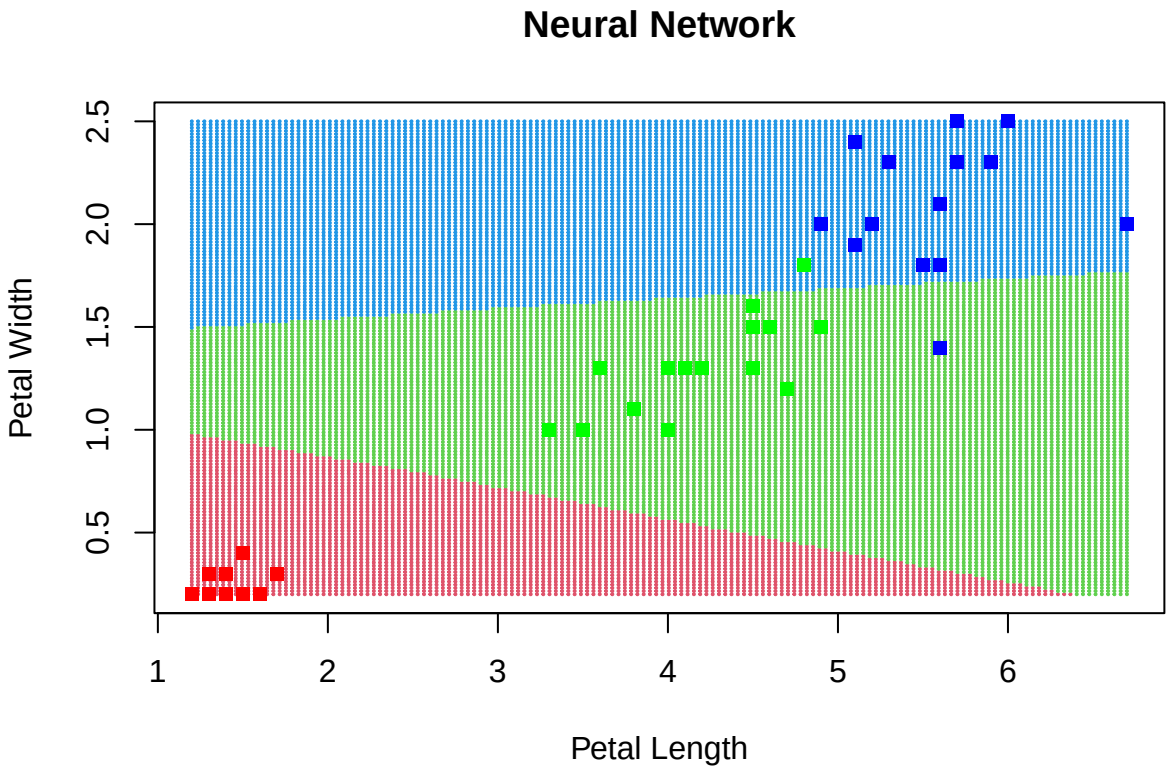
```
library(keras)
library(tensorflow)
library(datasets)
#Illustration
# Initialize a sequential model
model4 <- keras_model_sequential()

# Add layers to the model
model4 %>%
  layer_dense(units = 20, activation = 'relu', input_shape = c(2)) %>%
  layer_dense(units = 10, activation = 'relu') %>%
  layer_dense(units = 3, activation = 'softmax')

# Compile the model
model4 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = 'accuracy'
```

```
)


# Fit the model to the data
fit=model4 %>% fit(
  as.matrix(xtrain), iris.trainLabels,
  epochs = 200, batch_size = 5,
  validation_split = 0.2
)

plot(fit)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Evaluate the model
score4<- model4 %>% evaluate(as.matrix(xtest), iris.testLabels, batch_size = 128)



iris.prediction <- predict(model4,as.matrix(xgrid))
idx <- apply(iris.prediction, 1, which.max)
predicted_x <- c(1,2, 3)[idx]

plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Neural Network",xlab="Petal
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```




```
#plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Neural Network",xlab="Peta

#points(xtrain[,1],xtrain[,2],col=c("darkred","forestgreen","darkblue")[unclass(ytrain)],pch=".",cex=5)
#points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)


# Print the score
print(score4)
```

```
##      loss  accuracy
## 0.1610491 0.9574468
```

Cross Validation

```
data(iris)
library(caret)
```

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

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:tensorflow':
##
##     train
```

```
tc <- trainControl(method = "cv", number = 10)

fit <- train(Species ~.,
```

```
              data = iris,
              method = "nnet", #Neural Net
              trControl = tc,
              metric = "Accuracy")
```

```
## # weights:  11
## initial  value 157.855101
## iter  10 value 67.322206
## iter  20 value 61.328759
## iter  30 value 50.600777
## iter  40 value 9.222800
## iter  50 value 2.936057
## iter  60 value 2.683190
## iter  70 value 2.610204
## iter  80 value 2.572262
## iter  90 value 2.482828
## iter 100 value 2.470988
## final  value 2.470988
## stopped after 100 iterations
## # weights:  27
## initial  value 177.971581
## iter  10 value 72.338533
## iter  20 value 62.391110
## iter  30 value 62.383196
## iter  40 value 5.590406
## iter  50 value 2.792144
## iter  60 value 2.017566
## iter  70 value 0.002986
## iter  80 value 0.000242
## final  value 0.000041
## converged
## # weights:  43
## initial  value 161.022930
## iter  10 value 61.785407
## iter  20 value 6.062519
## iter  30 value 4.331151
## iter  40 value 1.828782
## iter  50 value 1.507838
## iter  60 value 1.395189
## iter  70 value 1.054451
## iter  80 value 0.149004
## iter  90 value 0.000483
## final  value 0.000056
## converged
## # weights:  11
## initial  value 169.087860
## iter  10 value 146.301072
## iter  20 value 66.963213
## iter  30 value 49.908192
## final  value 49.907864
## converged
## # weights:  27
## initial  value 148.256102
## iter  10 value 78.681519
```

```
## iter  20 value 27.887979
## iter  30 value 23.294805
## iter  40 value 21.702381
## iter  50 value 21.701277
## iter  50 value 21.701277
## iter  50 value 21.701277
## final  value 21.701277
## converged
## # weights:  43
## initial  value 156.259643
## iter  10 value 78.378918
## iter  20 value 29.802194
## iter  30 value 22.789688
## iter  40 value 22.175111
## iter  50 value 22.013082
## iter  60 value 21.928108
## iter  70 value 21.778704
## iter  80 value 21.671687
## iter  90 value 21.106940
## iter 100 value 20.646114
## final  value 20.646114
## stopped after 100 iterations
## # weights:  11
## initial  value 154.578608
## final  value 148.314032
## converged
## # weights:  27
## initial  value 152.885438
## iter  10 value 62.962818
## iter  20 value 62.107253
## iter  30 value 61.527755
## iter  40 value 61.327245
## iter  50 value 59.281764
## iter  60 value 36.282737
## iter  70 value 9.640402
## iter  80 value 4.986948
## iter  90 value 4.334787
## iter 100 value 3.900679
## final  value 3.900679
## stopped after 100 iterations
## # weights:  43
## initial  value 151.619936
## iter  10 value 42.104170
## iter  20 value 3.272971
## iter  30 value 2.257780
## iter  40 value 2.205699
## iter  50 value 2.151806
## iter  60 value 2.017196
## iter  70 value 1.953702
## iter  80 value 1.887951
## iter  90 value 1.884019
## iter 100 value 1.844327
## final  value 1.844327
## stopped after 100 iterations
```

```
## # weights:  11
## initial  value 170.831695
## final  value 148.312657
## converged
## # weights:  27
## initial  value 156.223822
## final  value 62.383242
## converged
## # weights:  43
## initial  value 156.619269
## iter  10 value 126.756909
## iter  20 value 28.403796
## iter  30 value 7.397986
## iter  40 value 5.299451
## iter  50 value 4.872142
## iter  60 value 4.817941
## iter  70 value 4.817695
## iter  70 value 4.817695
## final  value 4.817692
## converged
## # weights:  11
## initial  value 161.175762
## iter  10 value 84.682835
## iter  20 value 69.450494
## iter  30 value 52.223669
## final  value 51.685167
## converged
## # weights:  27
## initial  value 149.318756
## iter  10 value 71.139898
## iter  20 value 45.426010
## iter  30 value 25.675738
## iter  40 value 25.366165
## iter  50 value 25.253960
## iter  60 value 25.240939
## final  value 25.239439
## converged
## # weights:  43
## initial  value 168.065845
## iter  10 value 64.594920
## iter  20 value 25.683857
## iter  30 value 23.748018
## iter  40 value 22.528031
## iter  50 value 22.141233
## iter  60 value 22.008777
## iter  70 value 21.973367
## iter  80 value 21.893148
## iter  90 value 21.804484
## iter 100 value 21.777784
## final  value 21.777784
## stopped after 100 iterations
## # weights:  11
## initial  value 150.156857
## iter  10 value 61.244738
```

```
## iter  20 value 13.155798
## iter  30 value 6.769873
## iter  40 value 6.702958
## iter  50 value 6.687399
## iter  60 value 6.680287
## iter  70 value 6.676872
## iter  80 value 6.670945
## iter  90 value 6.670634
## iter 100 value 6.670576
## final  value 6.670576
## stopped after 100 iterations
## # weights:  27
## initial  value 156.239742
## iter  10 value 148.268516
## iter  20 value 63.453366
## iter  30 value 62.452469
## iter  40 value 61.912235
## iter  50 value 25.841718
## iter  60 value 8.526543
## iter  70 value 6.739524
## iter  80 value 6.708426
## iter  90 value 6.684311
## iter 100 value 6.671875
## final  value 6.671875
## stopped after 100 iterations
## # weights:  43
## initial  value 241.824719
## iter  10 value 70.765646
## iter  20 value 6.926973
## iter  30 value 5.692833
## iter  40 value 5.392456
## iter  50 value 5.315200
## iter  60 value 5.251612
## iter  70 value 5.239315
## iter  80 value 5.194925
## iter  90 value 5.160030
## iter 100 value 5.145413
## final  value 5.145413
## stopped after 100 iterations
## # weights:  11
## initial  value 155.684513
## iter  10 value 63.080745
## iter  20 value 62.383985
## final  value 62.383293
## converged
## # weights:  27
## initial  value 149.929359
## iter  10 value 80.495239
## iter  20 value 7.261181
## iter  30 value 5.532523
## iter  40 value 5.509366
## iter  50 value 5.502101
## iter  60 value 5.502058
## iter  70 value 5.501813
```

```
## iter  80 value 5.498348
## iter  90 value 5.491766
## iter 100 value 5.488787
## final  value 5.488787
## stopped after 100 iterations
## # weights:  43
## initial  value 173.328902
## iter  10 value 13.730050
## iter  20 value 5.793582
## iter  30 value 5.567133
## iter  40 value 5.522563
## iter  50 value 5.512597
## iter  60 value 5.492759
## iter  70 value 5.485778
## iter  80 value 5.483855
## iter  90 value 5.483233
## iter 100 value 5.482875
## final  value 5.482875
## stopped after 100 iterations
## # weights:  11
## initial  value 168.966178
## iter  10 value 76.922204
## iter  20 value 71.673915
## iter  30 value 65.457051
## iter  40 value 50.815828
## final  value 50.809677
## converged
## # weights:  27
## initial  value 175.655761
## iter  10 value 104.701038
## iter  20 value 65.132680
## iter  30 value 35.531349
## iter  40 value 23.168206
## iter  50 value 23.069402
## iter  60 value 23.054336
## iter  70 value 23.048413
## iter  70 value 23.048413
## final  value 23.048413
## converged
## # weights:  43
## initial  value 160.549679
## iter  10 value 53.320217
## iter  20 value 24.544001
## iter  30 value 23.385801
## iter  40 value 21.838541
## iter  50 value 21.571021
## iter  60 value 21.395337
## iter  70 value 21.370800
## iter  80 value 21.253571
## iter  90 value 20.816539
## iter 100 value 20.792121
## final  value 20.792121
## stopped after 100 iterations
## # weights:  11
```

```
## initial  value 142.892112
## iter  10 value 62.516320
## iter  20 value 59.496264
## iter  30 value 26.108602
## iter  40 value 6.888228
## iter  50 value 6.246627
## iter  60 value 6.220018
## iter  70 value 6.183690
## iter  80 value 6.182709
## iter  90 value 6.178518
## iter 100 value 6.178241
## final  value 6.178241
## stopped after 100 iterations
## # weights:  27
## initial  value 171.489004
## iter  10 value 141.587358
## iter  20 value 56.433850
## iter  30 value 31.804158
## iter  40 value 9.004031
## iter  50 value 7.117762
## iter  60 value 7.065055
## iter  70 value 6.850478
## iter  80 value 6.661148
## iter  90 value 6.559592
## iter 100 value 6.358955
## final  value 6.358955
## stopped after 100 iterations
## # weights:  43
## initial  value 157.348141
## iter  10 value 66.607053
## iter  20 value 57.201760
## iter  30 value 6.963104
## iter  40 value 5.943316
## iter  50 value 5.761124
## iter  60 value 5.676935
## iter  70 value 5.518538
## iter  80 value 4.413884
## iter  90 value 3.150696
## iter 100 value 0.956104
## final  value 0.956104
## stopped after 100 iterations
## # weights:  11
## initial  value 158.047633
## iter  10 value 148.316273
## iter  20 value 148.312662
## final  value 148.312660
## converged
## # weights:  27
## initial  value 166.583749
## iter  10 value 62.460126
## iter  20 value 62.383572
## final  value 62.383247
## converged
## # weights:  43
```

```
## initial  value 162.527260
## iter  10 value 62.522434
## iter  20 value 44.934024
## iter  30 value 6.517710
## iter  40 value 6.078195
## iter  50 value 5.990058
## iter  60 value 5.969730
## iter  70 value 5.954916
## iter  80 value 5.948454
## iter  90 value 5.947429
## iter 100 value 5.942532
## final  value 5.942532
## stopped after 100 iterations
## # weights:  11
## initial  value 170.215020
## iter  10 value 135.513426
## iter  20 value 71.729560
## iter  30 value 63.142951
## iter  40 value 51.456015
## final  value 51.448879
## converged
## # weights:  27
## initial  value 166.011916
## iter  10 value 96.505186
## iter  20 value 53.051262
## iter  30 value 49.900267
## iter  40 value 49.659772
## iter  50 value 38.033361
## iter  60 value 24.698429
## iter  70 value 23.756189
## iter  80 value 23.666458
## iter  90 value 23.615028
## final  value 23.615022
## converged
## # weights:  43
## initial  value 192.577166
## iter  10 value 89.075589
## iter  20 value 35.500454
## iter  30 value 27.671755
## iter  40 value 22.903052
## iter  50 value 22.456769
## iter  60 value 22.429316
## iter  70 value 22.426422
## iter  80 value 22.425673
## final  value 22.425571
## converged
## # weights:  11
## initial  value 168.890510
## final  value 148.313816
## converged
## # weights:  27
## initial  value 157.899269
## iter  10 value 65.217527
## iter  20 value 14.451353
```

```
## iter  30 value 6.449894
## iter  40 value 6.001787
## iter  50 value 5.794981
## iter  60 value 5.789013
## iter  70 value 5.778847
## iter  80 value 5.775968
## iter  90 value 5.775050
## iter 100 value 5.773472
## final  value 5.773472
## stopped after 100 iterations
## # weights:  43
## initial  value 151.297575
## iter  10 value 50.998536
## iter  20 value 6.908164
## iter  30 value 6.310996
## iter  40 value 6.211857
## iter  50 value 6.083751
## iter  60 value 6.073738
## iter  70 value 6.057472
## iter  80 value 6.031784
## iter  90 value 5.861853
## iter 100 value 5.195532
## final  value 5.195532
## stopped after 100 iterations
## # weights:  11
## initial  value 140.800855
## final  value 62.383348
## converged
## # weights:  27
## initial  value 154.373642
## iter  10 value 25.507474
## iter  20 value 7.819655
## iter  30 value 6.387731
## iter  40 value 5.013931
## iter  50 value 4.818283
## iter  60 value 4.817720
## final  value 4.817705
## converged
## # weights:  43
## initial  value 173.349183
## iter  10 value 104.867954
## iter  20 value 21.160878
## iter  30 value 6.335586
## iter  40 value 5.918124
## iter  50 value 3.330545
## iter  60 value 3.250727
## iter  70 value 3.238487
## iter  80 value 3.197671
## iter  90 value 3.049267
## iter 100 value 3.047272
## final  value 3.047272
## stopped after 100 iterations
## # weights:  11
## initial  value 155.702308
```

```
## iter  10 value 73.731979
## iter  20 value 70.948546
## iter  30 value 62.725593
## iter  40 value 51.765636
## iter  50 value 51.763713
## iter  50 value 51.763712
## iter  50 value 51.763712
## final  value 51.763712
## converged
## # weights:  27
## initial  value 157.259747
## iter  10 value 91.380507
## iter  20 value 27.584170
## iter  30 value 25.470154
## iter  40 value 23.409892
## iter  50 value 23.313334
## final  value 23.312534
## converged
## # weights:  43
## initial  value 158.254970
## iter  10 value 74.135278
## iter  20 value 29.475047
## iter  30 value 22.798203
## iter  40 value 21.833140
## iter  50 value 21.683589
## iter  60 value 21.623551
## iter  70 value 21.622900
## iter  70 value 21.622900
## iter  70 value 21.622900
## final  value 21.622900
## converged
## # weights:  11
## initial  value 160.450101
## iter  10 value 71.518381
## iter  20 value 10.375326
## iter  30 value 6.730262
## iter  40 value 6.683714
## iter  50 value 6.682322
## iter  60 value 6.681107
## iter  70 value 6.680011
## iter  80 value 6.679321
## iter  90 value 6.679297
## final  value 6.679291
## converged
## # weights:  27
## initial  value 165.197104
## iter  10 value 62.386634
## iter  20 value 10.080327
## iter  30 value 7.220223
## iter  40 value 6.847537
## iter  50 value 6.744182
## iter  60 value 6.654835
## iter  70 value 6.651311
## iter  80 value 6.638115
```

```
## iter  90 value 6.594853
## iter 100 value 6.489803
## final  value 6.489803
## stopped after 100 iterations
## # weights:  43
## initial  value 153.169925
## iter  10 value 71.258096
## iter  20 value 8.309399
## iter  30 value 6.432536
## iter  40 value 6.281078
## iter  50 value 6.075718
## iter  60 value 6.040231
## iter  70 value 5.580874
## iter  80 value 4.842727
## iter  90 value 3.719107
## iter 100 value 3.495746
## final  value 3.495746
## stopped after 100 iterations
## # weights:  11
## initial  value 154.010243
## iter  10 value 56.177675
## iter  20 value 17.160964
## iter  30 value 6.115906
## iter  40 value 5.842782
## iter  50 value 5.800509
## iter  60 value 5.798295
## iter  70 value 5.791926
## iter  80 value 5.786806
## iter  90 value 5.777748
## iter 100 value 5.774619
## final  value 5.774619
## stopped after 100 iterations
## # weights:  27
## initial  value 157.820136
## iter  10 value 89.469701
## iter  20 value 48.267988
## iter  30 value 7.262869
## iter  40 value 5.823968
## iter  50 value 5.799413
## iter  60 value 5.798161
## iter  70 value 5.780371
## iter  80 value 5.777929
## iter  90 value 5.762245
## iter 100 value 5.758646
## final  value 5.758646
## stopped after 100 iterations
## # weights:  43
## initial  value 149.208997
## iter  10 value 53.107583
## iter  20 value 12.224029
## iter  30 value 6.629907
## iter  40 value 5.196983
## iter  50 value 4.961638
## iter  60 value 4.867750
```

```
## iter  70 value 4.849247
## iter  80 value 4.838940
## iter  90 value 4.817707
## final  value 4.817693
## converged
## # weights:  11
## initial  value 150.584249
## iter  10 value 124.079727
## iter  20 value 68.786520
## iter  30 value 50.139421
## final  value 50.109018
## converged
## # weights:  27
## initial  value 150.237818
## iter  10 value 92.830566
## iter  20 value 69.714700
## iter  30 value 25.044413
## iter  40 value 23.593748
## iter  50 value 23.581526
## final  value 23.581525
## converged
## # weights:  43
## initial  value 162.257620
## iter  10 value 72.138600
## iter  20 value 40.072277
## iter  30 value 23.085312
## iter  40 value 21.311315
## iter  50 value 20.618322
## iter  60 value 20.491990
## iter  70 value 20.466454
## iter  80 value 20.451082
## iter  90 value 20.440974
## iter 100 value 20.437719
## final  value 20.437719
## stopped after 100 iterations
## # weights:  11
## initial  value 152.646007
## iter  10 value 50.205654
## iter  20 value 12.728405
## iter  30 value 6.475367
## iter  40 value 6.427969
## iter  50 value 6.420409
## iter  60 value 6.414626
## iter  70 value 6.411720
## iter  80 value 6.409281
## iter  90 value 6.409168
## iter 100 value 6.409096
## final  value 6.409096
## stopped after 100 iterations
## # weights:  27
## initial  value 152.508928
## iter  10 value 30.607046
## iter  20 value 7.958785
## iter  30 value 6.983448
```

```
## iter  40 value 5.589261
## iter  50 value 4.536635
## iter  60 value 3.743041
## iter  70 value 3.702723
## iter  80 value 3.644327
## iter  90 value 3.523073
## iter 100 value 3.426943
## final  value 3.426943
## stopped after 100 iterations
## # weights:  43
## initial  value 150.920396
## iter  10 value 64.263910
## iter  20 value 62.524017
## iter  30 value 52.304435
## iter  40 value 7.831218
## iter  50 value 5.982190
## iter  60 value 5.832033
## iter  70 value 5.819524
## iter  80 value 5.788687
## iter  90 value 5.751464
## iter 100 value 5.400536
## final  value 5.400536
## stopped after 100 iterations
## # weights:  11
## initial  value 149.446376
## iter  10 value 72.377211
## iter  20 value 62.392054
## final  value 62.383799
## converged
## # weights:  27
## initial  value 151.015921
## iter  10 value 62.381012
## iter  20 value 37.305718
## iter  30 value 7.889347
## iter  40 value 2.838844
## iter  50 value 0.043402
## iter  60 value 0.000876
## final  value 0.000058
## converged
## # weights:  43
## initial  value 200.626034
## iter  10 value 61.668273
## iter  20 value 8.834953
## iter  30 value 0.043802
## iter  40 value 0.000323
## final  value 0.000013
## converged
## # weights:  11
## initial  value 155.495770
## iter  10 value 131.002376
## iter  20 value 56.588943
## iter  30 value 49.473098
## final  value 49.443349
## converged
```

```
## # weights:  27
## initial  value 152.634129
## iter  10 value 49.229950
## iter  20 value 24.748265
## iter  30 value 20.594403
## iter  40 value 20.073717
## final  value 20.073714
## converged
## # weights:  43
## initial  value 164.020048
## iter  10 value 78.954463
## iter  20 value 28.567049
## iter  30 value 21.671119
## iter  40 value 19.703820
## iter  50 value 18.663076
## iter  60 value 18.457684
## iter  70 value 18.080553
## iter  80 value 17.946828
## iter  90 value 17.876587
## iter 100 value 17.786359
## final  value 17.786359
## stopped after 100 iterations
## # weights:  11
## initial  value 158.289135
## iter  10 value 63.828268
## iter  20 value 62.458562
## final  value 62.453664
## converged
## # weights:  27
## initial  value 165.828998
## iter  10 value 64.226952
## iter  20 value 62.463324
## iter  30 value 61.414534
## iter  40 value 47.813687
## iter  50 value 5.930392
## iter  60 value 2.408984
## iter  70 value 2.261501
## iter  80 value 2.064711
## iter  90 value 1.169594
## iter 100 value 0.988486
## final  value 0.988486
## stopped after 100 iterations
## # weights:  43
## initial  value 193.811371
## iter  10 value 64.582764
## iter  20 value 62.493783
## iter  30 value 62.471277
## iter  40 value 62.436548
## iter  50 value 9.286572
## iter  60 value 0.252337
## iter  70 value 0.201222
## iter  80 value 0.197808
## iter  90 value 0.180694
## iter 100 value 0.175585
```

```
## final  value 0.175585
## stopped after 100 iterations
## # weights:  11
## initial  value 150.595115
## iter  10 value 75.832261
## iter  20 value 62.395933
## final  value 62.384050
## converged
## # weights:  27
## initial  value 152.803312
## iter  10 value 55.874991
## iter  20 value 8.954665
## iter  30 value 5.892206
## iter  40 value 5.879371
## iter  50 value 5.868508
## iter  60 value 5.861386
## iter  70 value 5.858420
## iter  80 value 5.853399
## iter  90 value 5.794527
## iter 100 value 5.248756
## final  value 5.248756
## stopped after 100 iterations
## # weights:  43
## initial  value 210.101457
## iter  10 value 62.476866
## iter  20 value 12.512845
## iter  30 value 6.045528
## iter  40 value 5.964842
## iter  50 value 5.869996
## iter  60 value 5.857240
## iter  70 value 5.852960
## iter  80 value 5.851494
## iter  90 value 5.851020
## iter 100 value 5.850580
## final  value 5.850580
## stopped after 100 iterations
## # weights:  11
## initial  value 157.086841
## iter  10 value 73.578895
## iter  20 value 65.076052
## iter  30 value 51.322026
## final  value 51.033951
## converged
## # weights:  27
## initial  value 154.224009
## iter  10 value 70.646907
## iter  20 value 31.490417
## iter  30 value 24.374541
## iter  40 value 23.619112
## iter  50 value 23.498519
## iter  60 value 23.497805
## final  value 23.497582
## converged
## # weights:  43
```

```
## initial  value 176.773938
## iter  10 value 52.534772
## iter  20 value 26.639676
## iter  30 value 24.509985
## iter  40 value 23.730933
## iter  50 value 22.978944
## iter  60 value 22.552767
## iter  70 value 22.142523
## iter  80 value 21.827055
## iter  90 value 21.692537
## iter 100 value 21.647613
## final  value 21.647613
## stopped after 100 iterations
## # weights:  11
## initial  value 161.214191
## iter  10 value 148.315906
## final  value 148.313423
## converged
## # weights:  27
## initial  value 148.778710
## iter  10 value 37.322415
## iter  20 value 6.426856
## iter  30 value 6.139142
## iter  40 value 6.035669
## iter  50 value 6.029995
## iter  60 value 6.011612
## iter  70 value 5.994037
## iter  80 value 5.971723
## iter  90 value 5.946334
## iter 100 value 5.540241
## final  value 5.540241
## stopped after 100 iterations
## # weights:  43
## initial  value 229.143950
## iter  10 value 146.790406
## iter  20 value 53.108884
## iter  30 value 12.212356
## iter  40 value 7.308836
## iter  50 value 7.001653
## iter  60 value 6.863598
## iter  70 value 6.853069
## iter  80 value 6.799003
## iter  90 value 6.669624
## iter 100 value 6.640148
## final  value 6.640148
## stopped after 100 iterations
## # weights:  11
## initial  value 171.748199
## final  value 148.312585
## converged
## # weights:  27
## initial  value 152.781980
## iter  10 value 113.778479
## iter  20 value 9.945421
```

```
## iter  30 value 6.537383
## iter  40 value 5.895609
## iter  50 value 4.942395
## iter  60 value 4.819812
## iter  70 value 4.817721
## final  value 4.817693
## converged
## # weights:  43
## initial  value 168.685009
## iter  10 value 25.745372
## iter  20 value 6.979424
## iter  30 value 5.661126
## iter  40 value 4.837472
## iter  50 value 4.818101
## iter  60 value 4.817695
## final  value 4.817692
## converged
## # weights:  11
## initial  value 154.372074
## iter  10 value 93.358094
## iter  20 value 71.078694
## iter  30 value 55.987572
## iter  40 value 51.615800
## final  value 51.615698
## converged
## # weights:  27
## initial  value 153.084044
## iter  10 value 77.092193
## iter  20 value 54.204211
## iter  30 value 27.039392
## iter  40 value 25.336916
## iter  50 value 24.459629
## final  value 24.458622
## converged
## # weights:  43
## initial  value 163.550624
## iter  10 value 71.383294
## iter  20 value 51.211531
## iter  30 value 27.716925
## iter  40 value 25.148366
## iter  50 value 24.261904
## iter  60 value 22.747763
## iter  70 value 22.144160
## iter  80 value 21.903598
## iter  90 value 21.783562
## iter 100 value 21.744273
## final  value 21.744273
## stopped after 100 iterations
## # weights:  11
## initial  value 148.931707
## iter  10 value 62.115473
## iter  20 value 48.481804
## iter  30 value 15.358251
## iter  40 value 7.547856
```

```
## iter  50 value 6.788357
## iter  60 value 6.719131
## iter  70 value 6.681741
## iter  80 value 6.681007
## iter  90 value 6.680842
## iter  90 value 6.680842
## final  value 6.680842
## converged
## # weights:  27
## initial  value 166.461062
## iter  10 value 63.962247
## iter  20 value 62.485338
## iter  30 value 62.481863
## iter  40 value 62.470335
## iter  50 value 62.460785
## iter  60 value 11.859210
## iter  70 value 6.780942
## iter  80 value 6.277415
## iter  90 value 6.150454
## iter 100 value 6.039339
## final  value 6.039339
## stopped after 100 iterations
## # weights:  43
## initial  value 153.589647
## iter  10 value 67.549944
## iter  20 value 8.628325
## iter  30 value 6.745716
## iter  40 value 6.045744
## iter  50 value 5.992078
## iter  60 value 5.959327
## iter  70 value 5.897379
## iter  80 value 5.772929
## iter  90 value 5.769958
## iter 100 value 5.764130
## final  value 5.764130
## stopped after 100 iterations
## # weights:  11
## initial  value 157.408378
## iter  10 value 65.539265
## iter  20 value 62.386824
## final  value 62.383478
## converged
## # weights:  27
## initial  value 209.989664
## iter  10 value 62.584426
## iter  20 value 62.385135
## iter  30 value 62.383534
## final  value 62.383263
## converged
## # weights:  43
## initial  value 157.050073
## iter  10 value 60.877937
## iter  20 value 9.298657
## iter  30 value 6.456301
```

```
## iter  40 value 4.852606
## iter  50 value 4.818690
## iter  60 value 4.817698
## final  value 4.817692
## converged
## # weights:  11
## initial  value 160.408203
## iter  10 value 93.724895
## iter  20 value 51.085904
## iter  30 value 50.650390
## final  value 50.650389
## converged
## # weights:  27
## initial  value 155.407281
## iter  10 value 80.628658
## iter  20 value 56.684135
## iter  30 value 25.118502
## iter  40 value 24.027073
## iter  50 value 24.009791
## final  value 24.009790
## converged
## # weights:  43
## initial  value 170.286077
## iter  10 value 52.038782
## iter  20 value 25.662211
## iter  30 value 23.400468
## iter  40 value 21.713634
## iter  50 value 21.226583
## iter  60 value 21.120687
## iter  70 value 21.114693
## iter  80 value 21.112703
## iter  90 value 21.112607
## final  value 21.112570
## converged
## # weights:  11
## initial  value 150.136639
## iter  10 value 62.928080
## iter  20 value 60.971384
## iter  30 value 16.823235
## iter  40 value 6.416685
## iter  50 value 6.274394
## iter  60 value 6.244692
## iter  70 value 6.207748
## iter  80 value 6.204669
## iter  90 value 6.197787
## iter 100 value 6.197620
## final  value 6.197620
## stopped after 100 iterations
## # weights:  27
## initial  value 171.440220
## iter  10 value 63.615776
## iter  20 value 62.554968
## iter  30 value 61.974165
## iter  40 value 7.977148
```

```
## iter  50 value 6.582684
## iter  60 value 5.813266
## iter  70 value 5.763730
## iter  80 value 5.721426
## iter  90 value 5.691078
## iter 100 value 5.688882
## final  value 5.688882
## stopped after 100 iterations
## # weights:  43
## initial  value 173.090939
## iter  10 value 61.624648
## iter  20 value 11.741481
## iter  30 value 6.302311
## iter  40 value 6.193287
## iter  50 value 6.043955
## iter  60 value 5.903094
## iter  70 value 5.813435
## iter  80 value 5.703692
## iter  90 value 5.677162
## iter 100 value 5.661234
## final  value 5.661234
## stopped after 100 iterations
## # weights:  27
## initial  value 173.323174
## iter  10 value 64.192527
## iter  20 value 7.278484
## iter  30 value 6.360244
## iter  40 value 6.159287
## iter  50 value 6.090705
## iter  60 value 5.959928
## iter  70 value 5.939869
## iter  80 value 5.885079
## iter  90 value 5.879687
## iter 100 value 5.869233
## final  value 5.869233
## stopped after 100 iterations
```

```
print(fit)
```

```
## Neural Network
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.5800000  0.37
##   1     1e-04  0.7733333  0.66
##   1     1e-01  0.9666667  0.95
##   3     0e+00  0.8600000  0.79
```

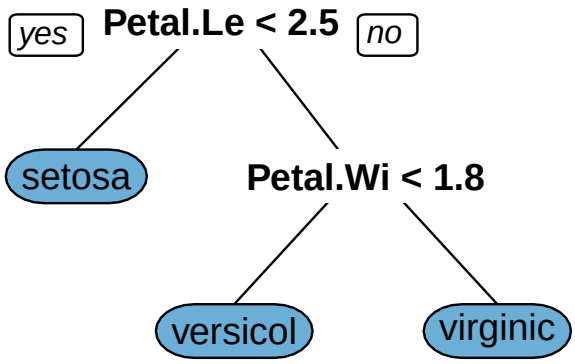
```
##   3     1e-04  0.9800000  0.97
##   3     1e-01  0.9733333  0.96
##   5     0e+00  0.9733333  0.96
##   5     1e-04  0.9733333  0.96
##   5     1e-01  0.9733333  0.96
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 1e-04.
```

```
pred <- predict(fit, iris[ ,-5])
confusionMatrix(iris[ ,5], pred)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         1
##   virginica       0          0        50
##
## Overall Statistics
##
##                Accuracy : 0.9933
##                  95% CI : (0.9634, 0.9998)
##     No Information Rate : 0.34
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.99
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9804
## Specificity                 1.0000            0.9901           1.0000
## Pos Pred Value              1.0000            0.9800           1.0000
## Neg Pred Value              1.0000            1.0000           0.9900
## Prevalence                  0.3333            0.3267           0.3400
## Detection Rate              0.3333            0.3267           0.3333
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            0.9950           0.9902
```

Decision Trees

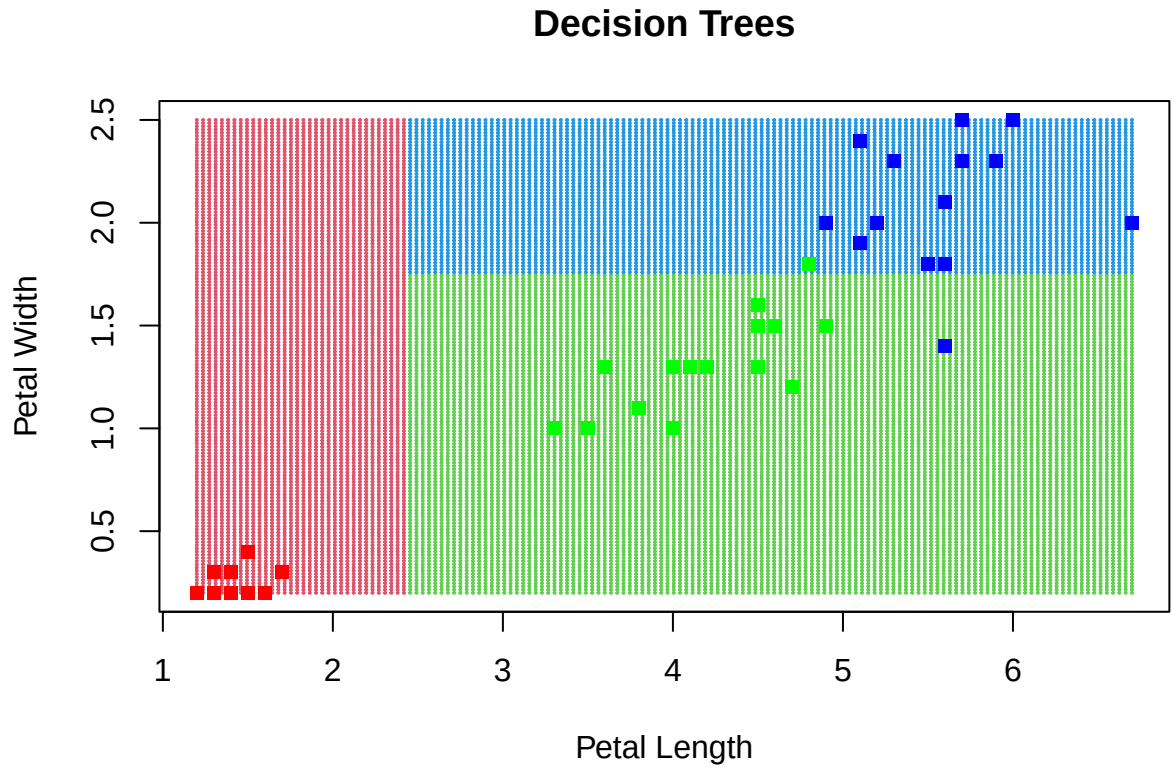
```
library(rpart.plot)
library(caret)


dtree_fit_gini <- train(iris.trainingtarget ~ ., data=cbind(iris.training,iris.trainingtarget), method =
                    parms = list(split = "gini"),
                   tuneLength = 10)
prp(dtree_fit_gini$finalModel,box.palette = "Blues", tweak = 1.2)
```




```
test_pred_gini<-predict(dtree_fit_gini,newdata = as.matrix(iris.test))
con_mat<-confusionMatrix(test_pred_gini, iris.testtarget)
score_dt_gini<-sum(diag(con_mat$table))/(sum(con_mat$table))
score_dt_gini
```

```
## [1] 0.9574468
```

```
predicted_x<-predict(dtree_fit_info,newdata = list("Sepal.Length"=xgrid[,1], "Sepal.Width"=xgrid[,2],

plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Decision Trees",xlab="Petal

#points(xtrain[,1],xtrain[,2],col=c("darkred","forestgreen","darkblue")[unclass(ytrain)],pch=".",cex=5)
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```



#

Random Forest

```
library(randomForest)
```

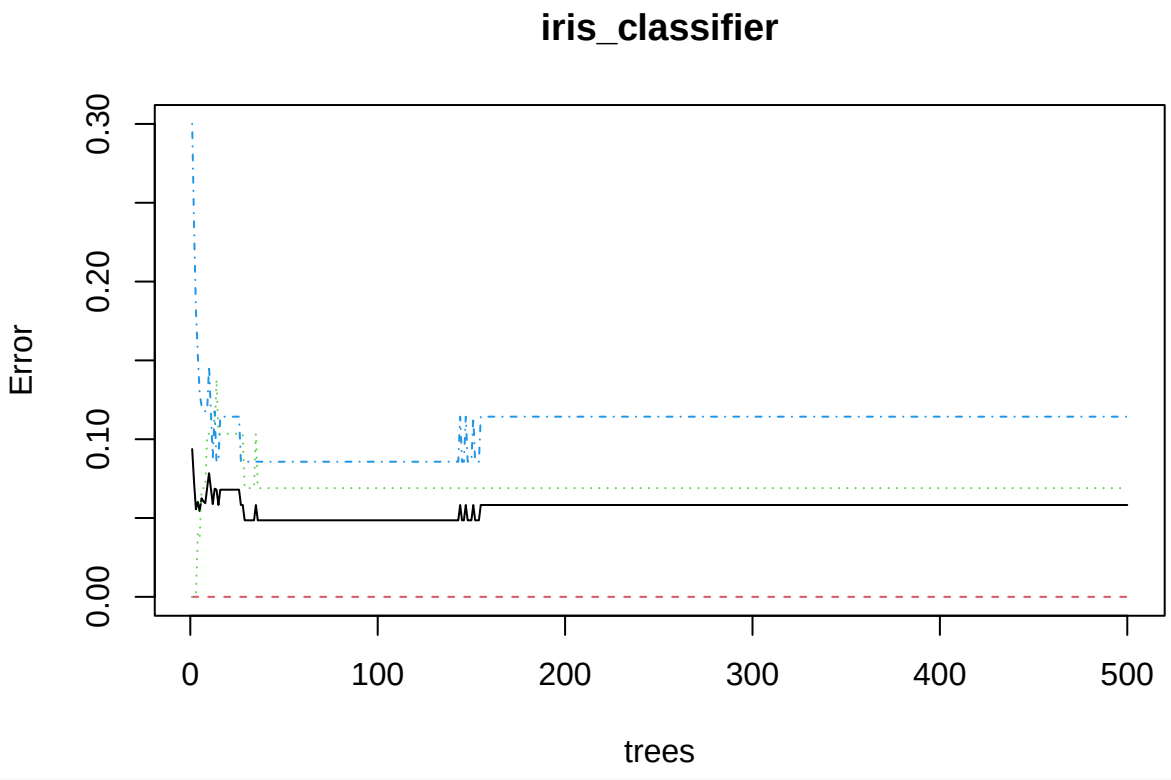
```
## randomForest 4.7-1
```

```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
iris_classifier <- randomForest(iris.trainingtarget ~ ., data=cbind(iris.training,iris.trainingtarget),
                                importance = T)
iris_classifier
```

```
##
## Call:
##  randomForest(formula = iris.trainingtarget ~ ., data = cbind(iris.training,      iris.trainingtarget
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 5.83%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         39          0         0  0.00000000
## versicolor      0         27         2  0.06896552
## virginica       0          4        31  0.11428571
```

```
plot(iris_classifier)
```




```
predicted_table <- predict(iris_classifier, iris.test)
```

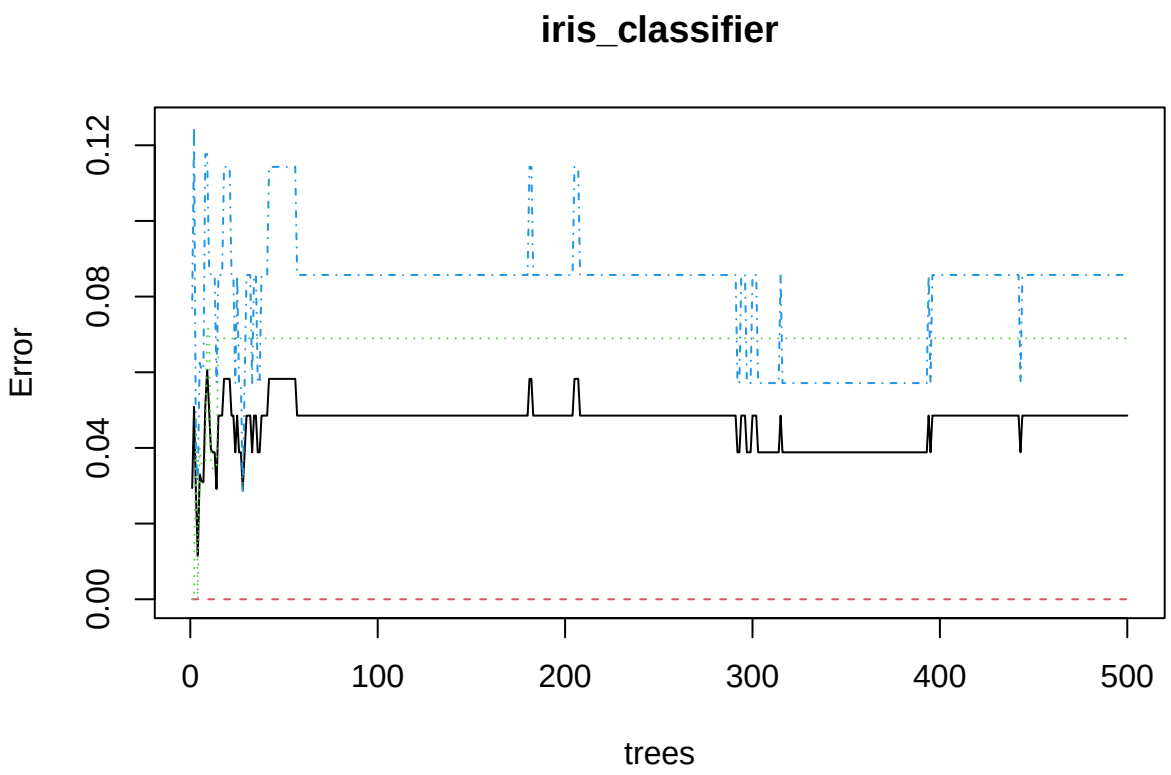
```
tab<-table(observed = iris.testtarget, predicted = predicted_table)
score_rf<-sum(diag(tab))/(sum(tab))
score_rf
```

```
## [1] 0.9574468
```

```
#Illustration

iris_classifier <- randomForest(iris.trainingtarget ~ ., data=cbind(iris.training[,3:4],iris.trainingta
                                importance = T)
iris_classifier
```

```
##
## Call:
##  randomForest(formula = iris.trainingtarget ~ ., data = cbind(iris.training[,      3:4], iris.trainin
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 4.85%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         39          0         0  0.00000000
## versicolor      0         27         2  0.06896552
## virginica       0          3        32  0.08571429
```

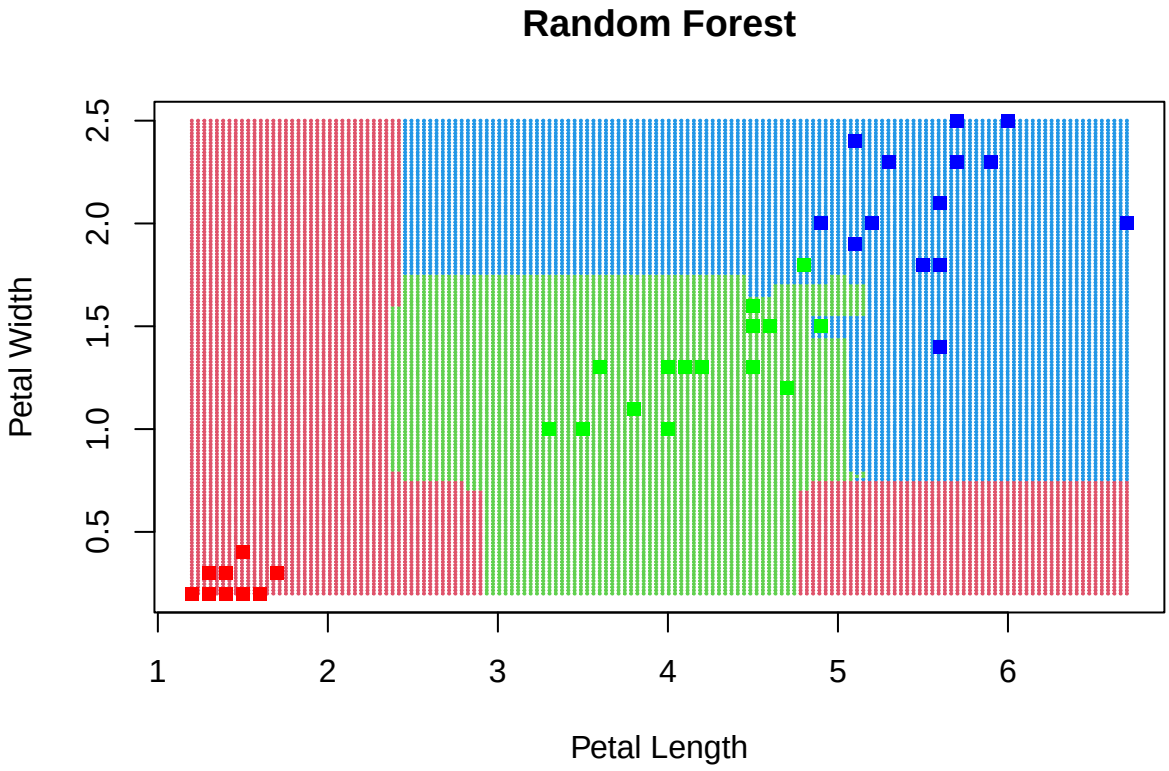
```
plot(iris_classifier)
```




```
predicted_table <- predict(iris_classifier, iris.test)
table(observed = iris.testtarget, predicted = predicted_table)
```

```
##             predicted
## observed     setosa versicolor virginica
##   setosa         11          0         0
##   versicolor      0         18         3
##   virginica       0          0        15
```

```
predicted_x<-predict(iris_classifier,newdata = list("Petal.Length"=xgrid[,1], "Petal.Width"=xgrid[,2] ))

plot(xgrid,col=c(2,3,4)[as.numeric(predicted_x)],pch = 20,cex = .2, main = "Random Forest",xlab="Petal L

#points(xtrain[,1],xtrain[,2],col=c("darkred","forestgreen","darkblue")[unclass(ytrain)],pch=".",cex=5)
points(xtest[,1],xtest[,2],col=c("red","green","blue")[unclass(ytest)],pch=".",cex=7)
```



Cross Validation

```
data(iris)
library(caret)

tc <- trainControl(method = "cv", number = 10)

fit <- train(Species ~.,
             data = iris,
             method = "rf",
             trControl = tc,
             metric = "Accuracy")

print(fit)
```

```
## Random Forest
```

```
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9533333  0.93
##   3     0.9533333  0.93
##   4     0.9466667  0.92
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
pred <- predict(fit, iris[ ,-5])
confusionMatrix(iris[ ,5], pred)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         50         0
##   virginica       0          0        50
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9757, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   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
```

Summary

```
#Illustration

summary<-c("NB"= score_nb, "kNN"= iris_acc[20], "logistic"=scorelog,"SVM"=score_svm, "NN"=score_keras,
print(summary)
```

```
##        NB       kNN  logistic       SVM     NN.NA        DT        RF
## 0.9574468 1.0000000 0.9787234 0.9787234        NA 0.9574468 0.9574468
```