Introduction to Machine Learning

Working Group "Computational Statistics" – Bernd Bischl et al.

Code demo for Random Forests

Variable Importance from mlr3

```
library(mlr3)
library(mlr3learners)
library(mlr3filters)
library(mlr3viz)
library(mlbench)
library(dplyr)
library(partykit)

data("Servo")

# transform ordered factors to numeric
servo <- Servo %>%
  mutate_at(c("Pgain", "Vgain"), as.character) %>%
  mutate_at(c("Pgain", "Vgain"), as.numeric)
rm(Servo)
str(servo)
```

```
## 'data.frame':    167 obs. of  5 variables:
##  $ Motor: Factor w/ 5 levels "A","B","C","D",..: 5 2 4 2 4 5 3 ..
##  $ Screw: Factor w/ 5 levels "A","B","C","D",..: 5 4 4 1 2 3 1 ..
##  $ Pgain: num  5 6 4 3 6 4 3 3 ...
##  $ Vgain: num  4 5 3 2 5 3 2 2 ...
##  $ Class: num  4 11 6 48 6 20 46 49 ...
```

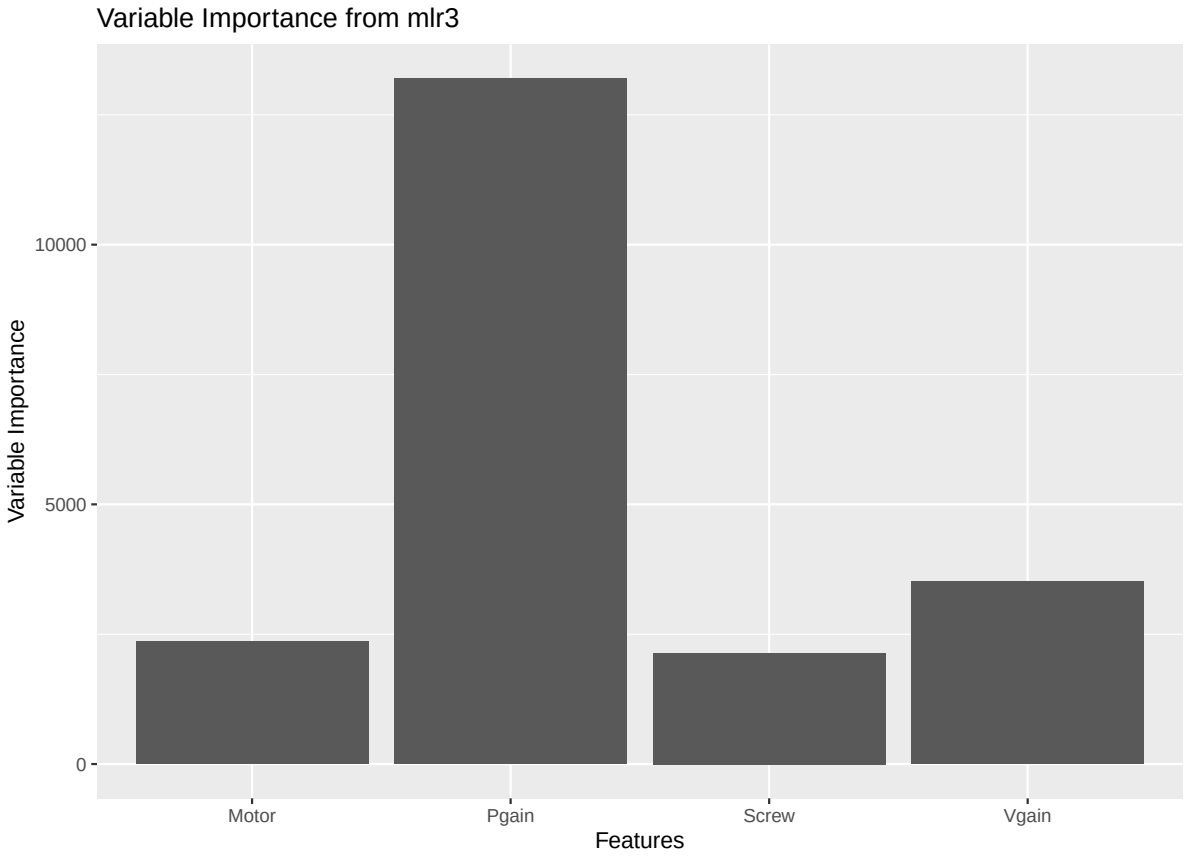
```
head(servo)
```

```
##   Motor Screw Pgain Vgain Class
## 1     E     E     5     4     4
## 2     B     D     6     5    11
## 3     D     D     4     3     6
## 4     B     A     3     2    48
## 5     D     B     6     5     6
## 6     E     C     4     3    20
```

```
# split in train and test with two different seeds to show data dependency
train_size <- 3 / 4

set.seed(1333)
train_indices <- sample(
  x = seq(1, nrow(servo), by = 1),
  size = ceiling(train_size * nrow(servo)), replace = FALSE
)
train_1 <- servo[ train_indices, ]
test_1 <- servo[ -train_indices, ]
```

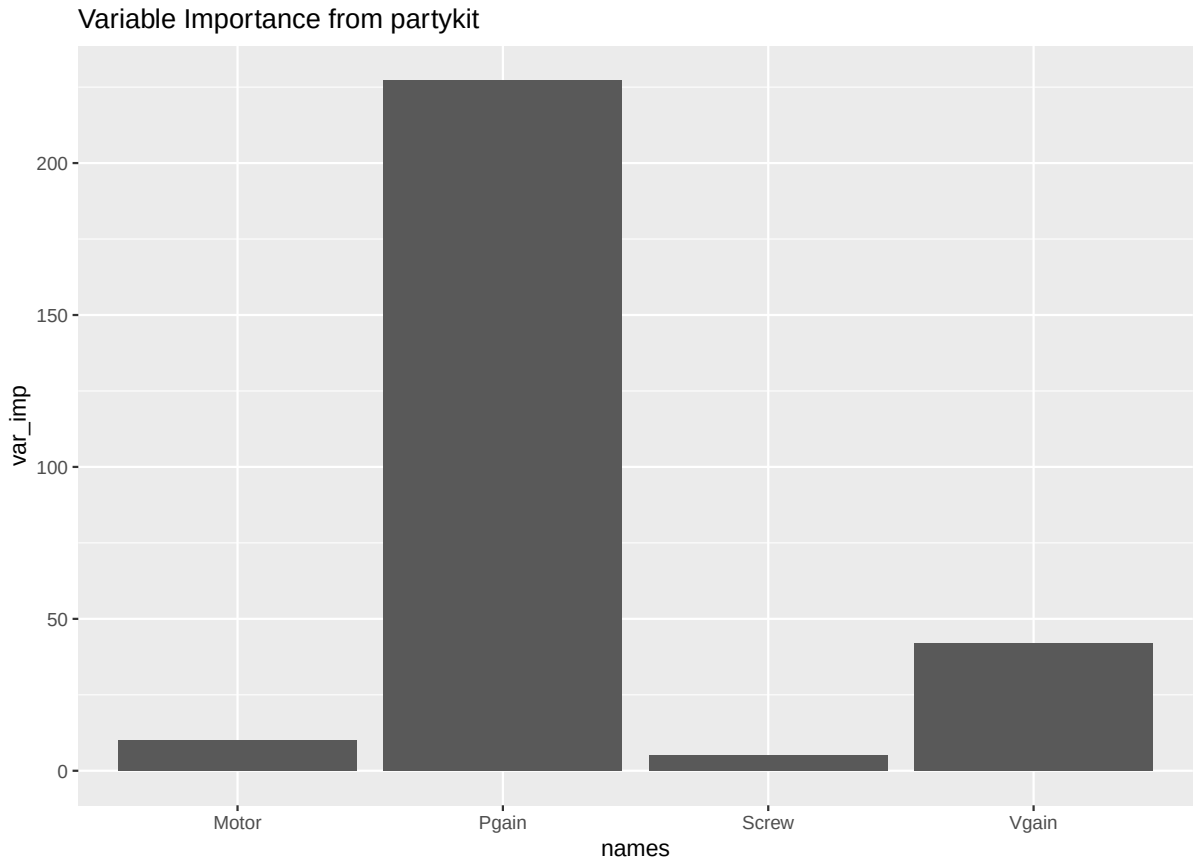
To access the variable importance we need to use a filter in mlr3. You can read more about this here.

```
library(ggplot2)

task <- TaskRegr$new(id = "servo", backend = train_1, target = "Class")
lrn1 <- lrn("regr.ranger", importance = "impurity")
lrn1$train(task = task)

filter <- flt("importance", learner = lrn1)
filter$calculate(task)
var <- as.data.table(filter)

ggplot(data = var, aes(x = feature, y = score)) + geom_bar(stat = "identity") +
  ggtitle(label = "Variable Importance from mlr3") +
  labs(x = "Features", y = "Variable Importance")
```



Variable importance from partykit

```
modForest <- partykit::cforest(Class ~ Motor + Screw + Pgain + Vgain, data = train_1)
var <- as.data.frame(varimp(modForest))
var$names <- rownames(var)
colnames(var) <- c("var_imp", "names")
ggplot(data = var, aes(x = names, y = var_imp)) +
  geom_bar(stat = "identity") +
  ggtitle(label = "Variable Importance from partykit")
```



Note that `{partykit}`'s `cforest` implementation of random forest is built on "conditional inference trees", which use a slightly different algorithm for finding optimal splits and deciding when to stop splitting the data than the CARTs we discussed in the lecture:

> Roughly, the algorithm works as follows: 1) Test the global null hypothesis of independence between any of the input variables and the response (which may be multivariate as well). Stop if this hypothesis cannot be rejected. Otherwise select the input variable with strongest association to the response. This association is measured by a p-value corresponding to a test for the partial null hypothesis of a single input variable and the response. 2) Implement a binary split in the selected input variable. 3) Recursively repeate steps 1) and 2)

(quoted from `?ctree`)

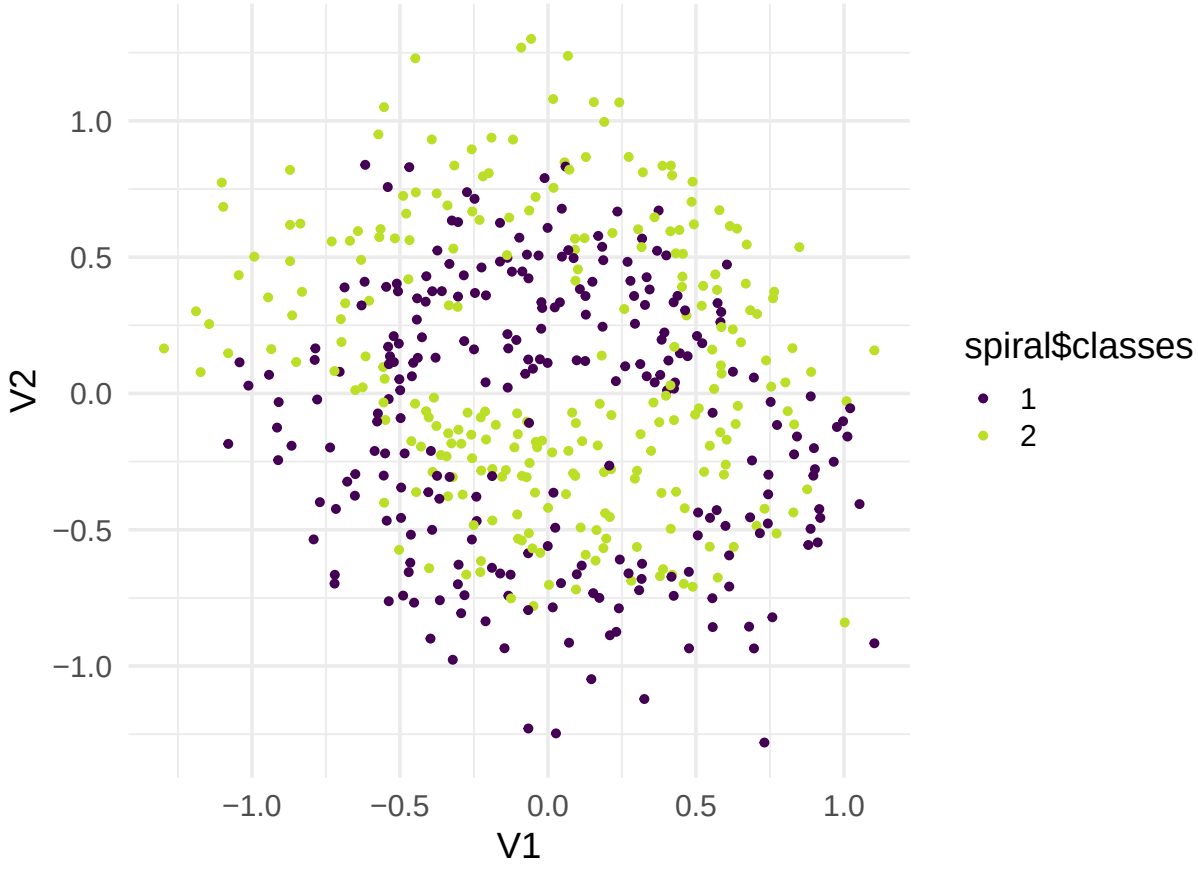
Decision Regions CART vs. Random Forest

Let's look at a simpler problem with just two features to visualize the way that CARTs and Random Forests divide the feature space into decision regions.

First, look a the data:

```
# get data
spiral <- mlbench::mlbench.spirals(500, cycles = 1, sd = 0.2)
p <- ggplot(data = as.data.frame(spiral$x), aes(
  x = V1, y = V2,
  colour = spiral$classes
```

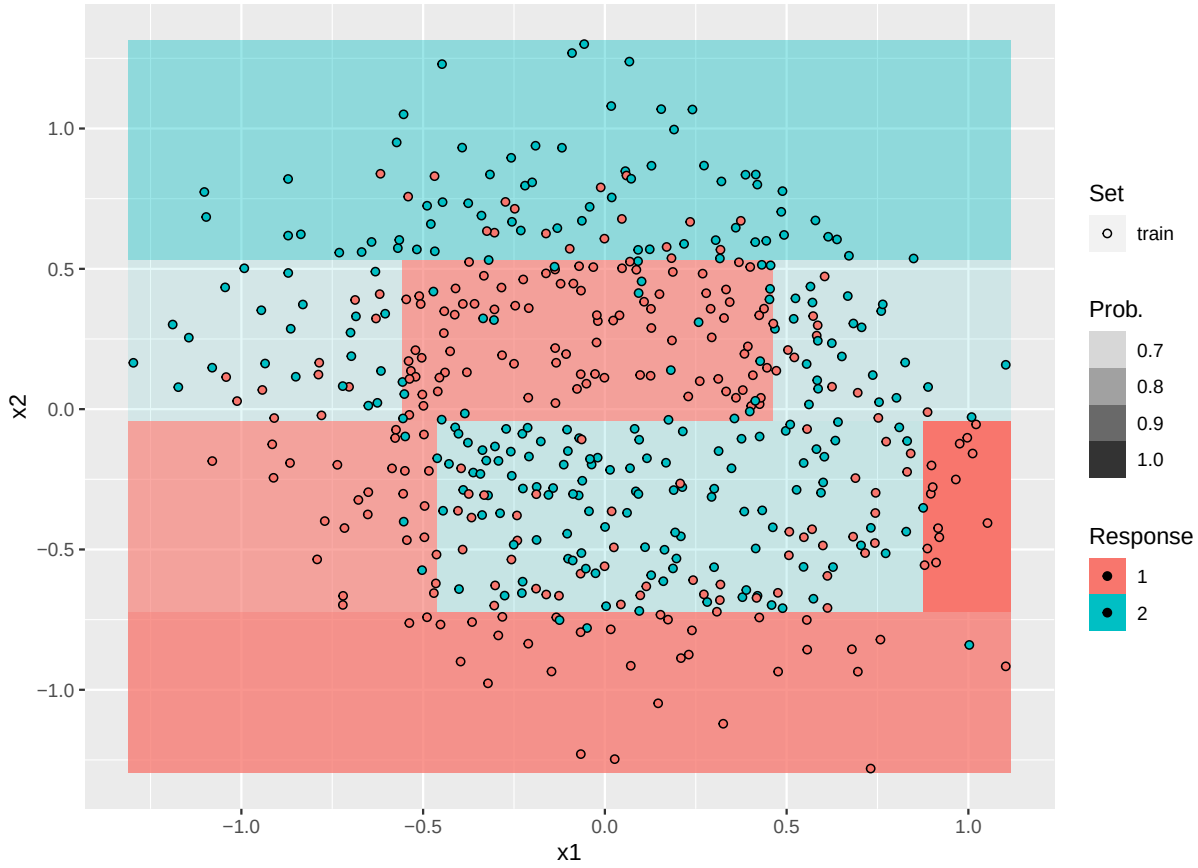
```
)) +
  geom_point()
p
```



Decision regions for the CART:

```
spiral_data <- data.frame(spiral$x, y = factor(spiral$classes))
colnames(spiral_data) <- c("x1", "x2", "y")
# set features that should be inspected within the regions
features <- c("x1", "x2")
spiral_task <- TaskClassif$new(
  id = "spirals", backend = spiral_data,
  target = "y"
)

plot_learner_prediction(
  lrn("classif.rpart", predict_type = "prob"),
  spiral_task
)
```

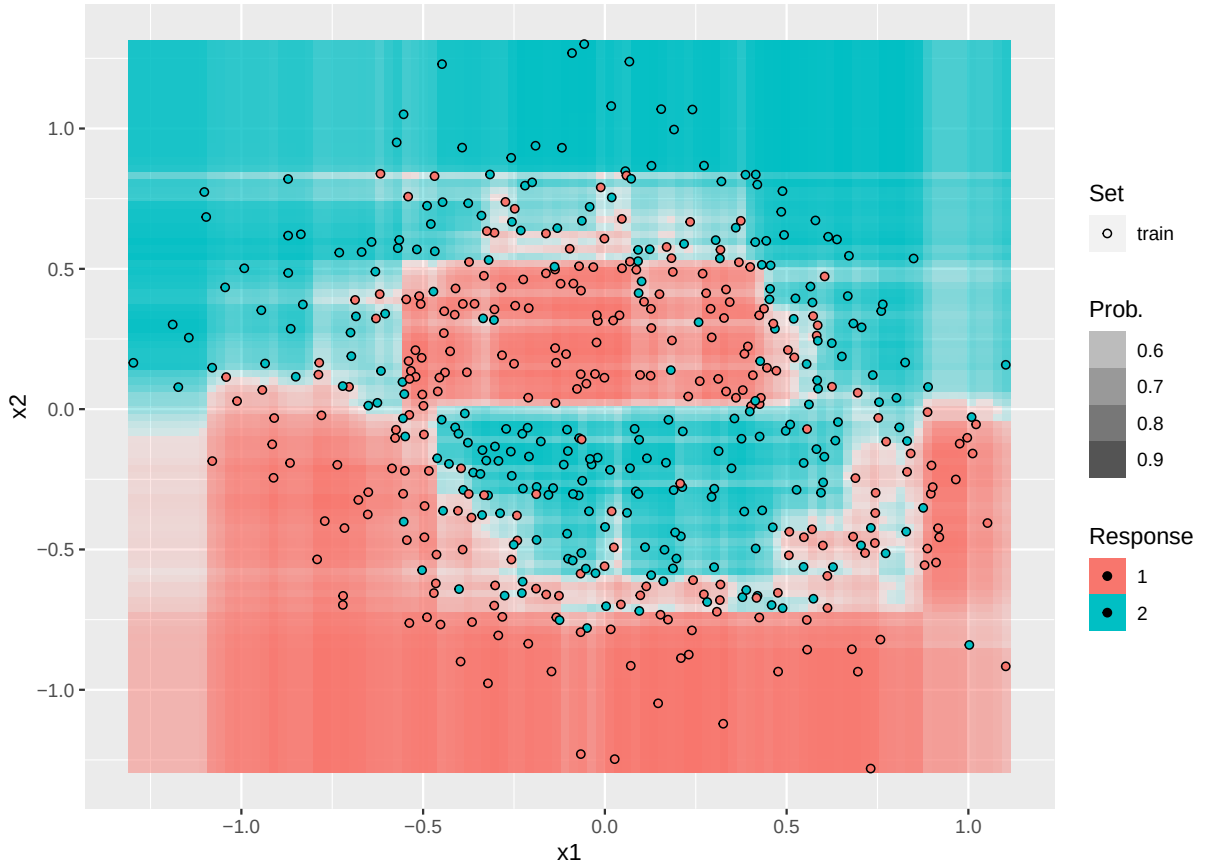
```
## INFO  [12:16:47.268] Applying learner 'classif.rpart' on task 'spirals' (iter 1/1)
```



Decision regions for the Random Forest:

```
plot_learner_prediction(
  lrn("classif.ranger", predict_type = "prob"),
  spiral_task
)
```

```
## INFO  [12:16:48.972] Applying learner 'classif.ranger' on task 'spirals' (iter 1/1)
```



Proximity measures in Random Forests

One neat feature of random forests is that they implicitly calculate a proximity measure (how close a data point is to another): relative frequencies of how often two (OOB-)observations end up in the same terminal node are easily calculated during the fit. If two observations are "close" in terms of the features that matter for the random forest, they have a "high proximity" in the sense that they will frequently be sorted into the same terminal node.

We apply this to the spiral data set and try to see, if we could also classify the points based on their proximity.

We fit a random forest and extract the proximity distance matrix:

```
library(randomForest)
# fit forest and extract proximity measure
set.seed(1337)
spiral_rf <- randomForest(
  x = spiral$x, y = spiral$classes,
  ntree = 1000,
  proximity = TRUE, oob.prox = TRUE,
)
spiral_proximity <- spiral_rf$proximity
spiral_proximity[1:5, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
```

```
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

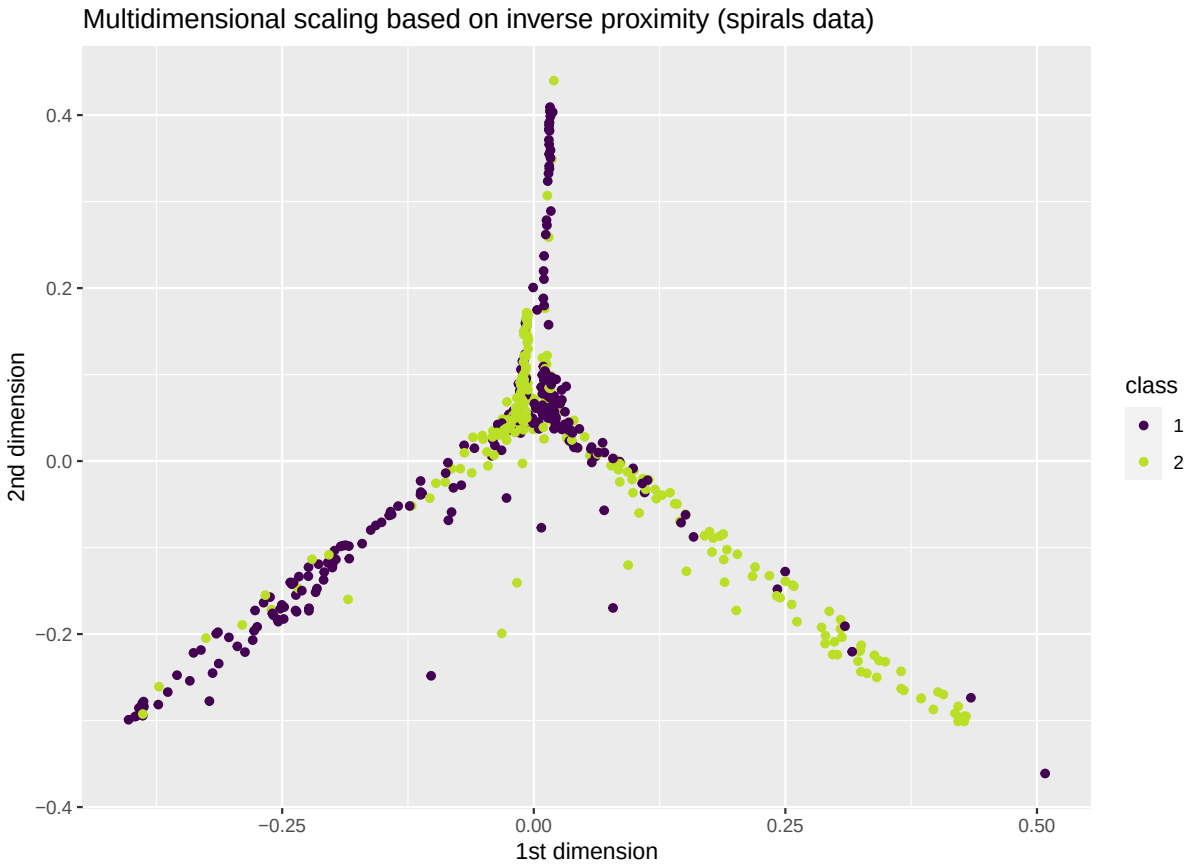
MDS (Multidimensional Scaling) is a method that takes a matrix containing distances between objects and tries to create a low-dimensional configuration of points (each corresponding to one of these objects) that has the same pairwise distances.
This low-dimensional configuration (typically in 2D) can be very useful to visualize the strcuture of similarity and dissimilarity implied by the distances between high-dimensional vectors or other complex objects. Check this blog post for a nice example of MDS for European capital cities.

We run MDS on the distance matrix derived from the (inverse) proximities and plot the data based on the two dimensions, this won't work all that well here because so many data points have mutual proximities of 0, i.e., an *"infinite"* distance – they never ended up in the same leaf of any tree, ever...

```r
# convert proximities into distances
proximity_to_dist <- function(proximity) {
  1 - proximity
}

spiral_dist <- proximity_to_dist(spiral_proximity)
spiral_dist[1:5, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    1    1    1    1
## [2,]    1    0    1    1    1
## [3,]    1    1    0    1    1
## [4,]    1    1    1    0    1
## [5,]    1    1    1    1    0
```

```r
spiral_mds <- as.data.frame(cmdscale(spiral_dist))
spiral_mds$class <- spiral$classes

# plot the result, sweet
p <- ggplot(data = spiral_mds, aes(x = V1, y = V2, colour = class)) +
  geom_point() +
  labs(
    x = "1st dimension", y = "2nd dimension",
    title = "Multidimensional scaling based on inverse proximity (spirals data)"
  )
p
```



Some structure seems visible, but this is really most useful for very high-dimensional feature vectors, so let's look at such a problem next...

MNIST example

In this exercise we want to train a random forest classifier, s.t. it can be used to identify the handwritten digits 8 and 9. For that, we'll use the iconic MNIST data set that contains 28×28 greyscale pixel images of handwritten digits.

```
library(readr)
library(dplyr)

mnist_raw <- read_csv("https://www.python-course.eu/data/mnist/mnist_train.csv",
  col_names = FALSE
)
```

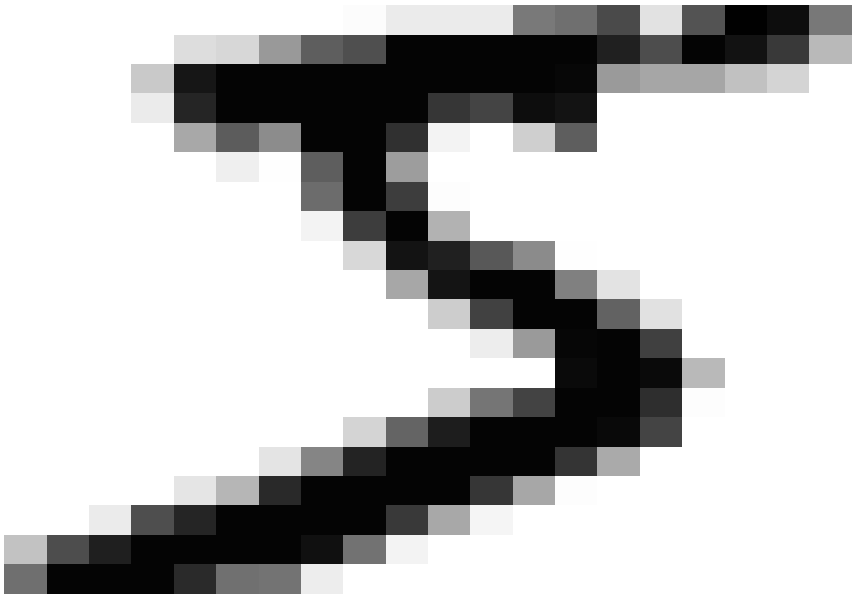
Every row of the MNIST data set contains the label (0-9) and the grayscale of the $28^2 = 784$ pixels of the corresponding image. For example:

```
rotate <- function(x) t(apply(x, 2, rev))

# function which plots a 28*28 raster image
plot_mnist_raster <- function(pixels, title = "") {
  pixels <- rotate(t(matrix(as.numeric(pixels), nrow = 28, byrow = F)))
```

```
  # transform matrix m[i,j] into a data.frame with rows (x = i, y = j, value = m[i,j])
  df <- data.frame(
    x = as.vector(row(pixels)),
    y = as.vector(col(pixels)),
    value = as.vector(pixels)
  )

  ggplot(df, aes(x, y)) + geom_raster(aes(fill = value)) +
    scale_fill_gradient(
      low = "white", high = "black",
      limits = c(0, 256), guide = "none"
    ) +
    ggtitle(title) + theme_void()
}

as.numeric(mnist_raw[1, 1])
```

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

```
plot_mnist_raster(mnist_raw[1, 2:ncol(mnist_raw)])
```



We are only interested in the digits 8 and 9:

```
mnist_8 <- mnist_raw[ mnist_raw[, 1] == 8, ]
mnist_9 <- mnist_raw[ mnist_raw[, 1] == 9, ]

train_size <- 150
set.seed(123)
train_8_indices <- sample(nrow(mnist_8), train_size / 2, replace = FALSE)
train_9_indices <- sample(nrow(mnist_9), train_size / 2, replace = FALSE)

mnist_89_train <- as.data.frame(rbind(
  mnist_8[train_8_indices, ],
  mnist_9[train_9_indices, ]
))

mnist_89_train$X1 <- as.factor(mnist_89_train$X1)

mnist_89_test <- as.data.frame(rbind(
  mnist_8[-train_8_indices, ],
  mnist_9[-train_9_indices, ]
))

mnist_89_test$X1 <- as.factor(mnist_89_test$X1)
```

Train a random forest model on the $28^2 = 784$ pixel intensity features to classify "8" versus "9":

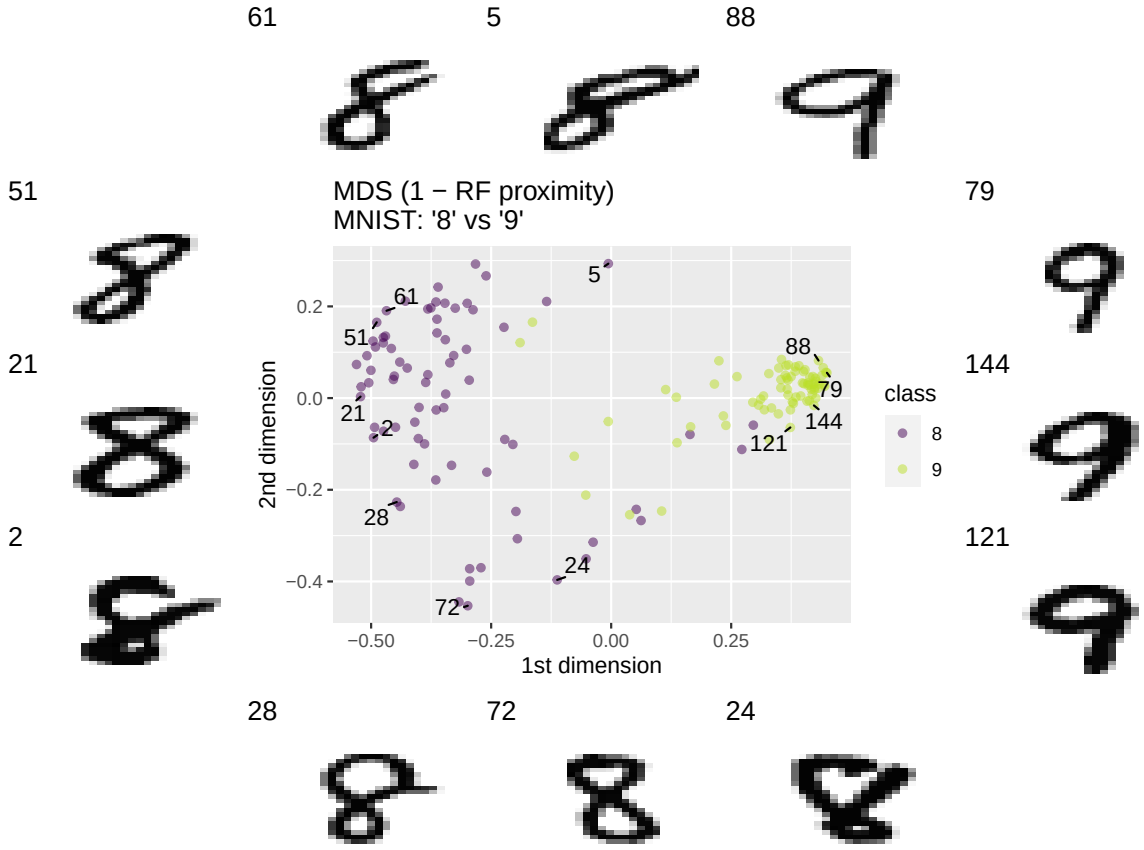
```
set.seed(123)
mnist_89_rf <- randomForest(
  x = mnist_89_train[, -1], y = mnist_89_train[, 1],
  ntree = 1000,
  proximity = TRUE, oob.prox = TRUE, var = "impurity"
)

# Determine mmce
mean(predict(mnist_89_rf, mnist_89_test) != mnist_89_test$X1)
```

```
## [1] 0.0448
```

With proximities we are able to explore how similar the images of our training set are:

```
library(ggrepel)
library(gridExtra)

# turn proximity matrix into dissimilarity matrix
# (proximity of 0 gets turned into a very small value)
mnist_89_dist <- proximity_to_dist(mnist_89_rf$proximity)

# apply mds on the proximity matrix
mnist_89_mds <- as.data.frame(cmdscale(mnist_89_dist))
mnist_89_mds$class <- mnist_89_train$X1
mnist_89_mds$id <- 1:nrow(mnist_89_mds)

# pick points on the outside of the point cloud for viz
library(dplyr)
```

```r
mnist_89_examples <- mnist_89_mds %>%
  # chull returns indices of points on the convex hull
  filter(row_number() %in% chull(mnist_89_mds[, c("V1", "V2")])) %>%
  mutate(
    angle = atan2(V1, V2),
  ) %>%
  arrange(angle) %>%
  #pick 12 examples, well spaced in terms of their angle
  filter(row_number() %in% round(seq(1, n(), l = 12)))


# plot the result
p <- ggplot(data = mnist_89_mds, aes(x = V1, y = V2, colour = class)) +
  geom_point(alpha = 0.5) +
  geom_text_repel(
    data = mnist_89_examples,
    aes(V1, V2, label = id), color = "black",
    min.segment.length = unit(0, "lines")
  ) +

  labs(
    x = "1st dimension", y = "2nd dimension", title =
      "MDS (1 - RF proximity)\nMNIST: '8' vs '9'"
  )

p_examples <- lapply(mnist_89_examples$id, function(id) {
  plot_mnist_raster(mnist_89_train[id, 2:ncol(mnist_raw)], as.character(id))
})

layout <-
  cbind(
    c(NA, 6:4, NA),
    rbind(
      c(7:9),
      matrix(13, 3, 3),
      c(3:1)),
    c(NA, 10:12, NA))


do.call(grid.arrange,
  c(p_examples, list(p),
    list(layout_matrix = layout)))
```

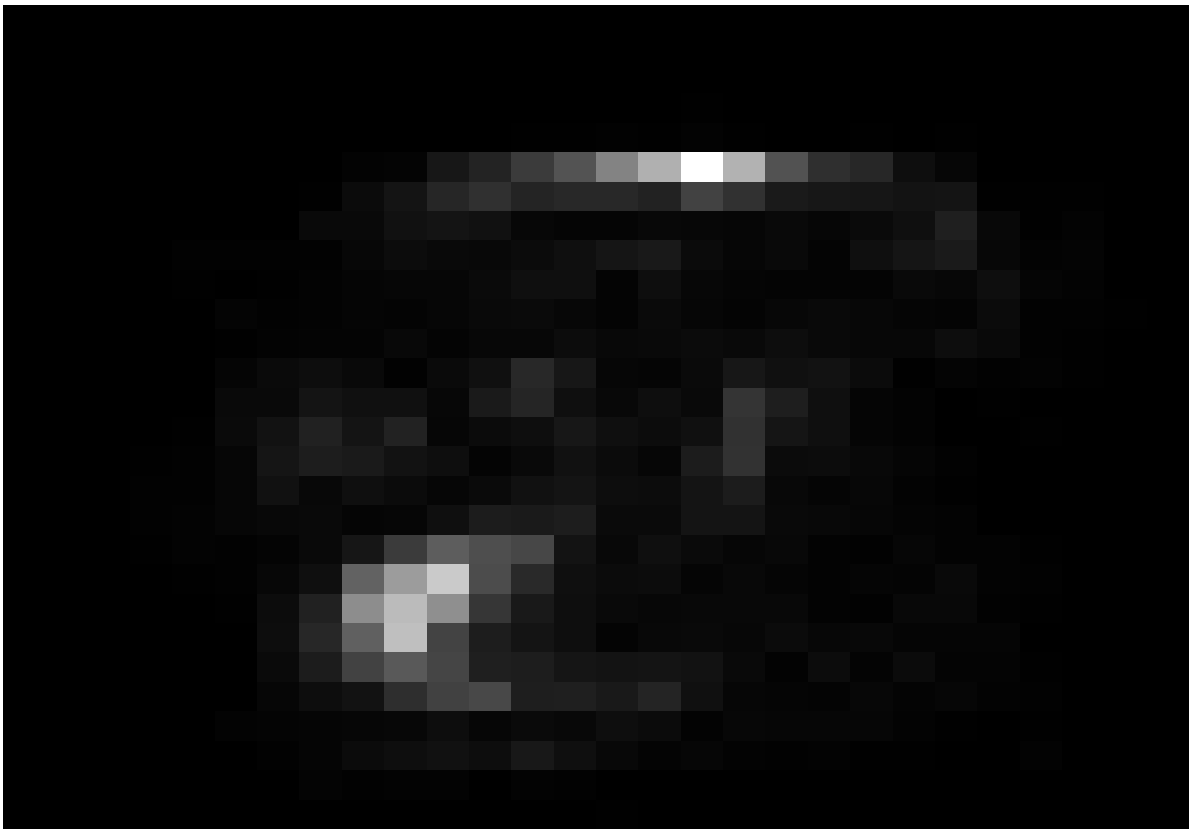


By looking at the variations of the images along the axes we see that while the first axis seems to represent mainly the shape of the lower half of the image, the second coordinate seems to vary with the size of the upper circle. This suggests that we were able to find a meaningful two dimensional representation of our data set, which gives us a good overview of the distribution of the occurring shapes and enables us to visualize the similarity of the digits.

(A further discussion of the problems of visualizing the MNIST data set and possible solutions can be found here)

With variable importance we can now investigate which pixels are the "most important" when we want to tell the digits 8 and 9 apart:

```
mnist_89_var_imp <- mnist_89_rf$importance
mnist_89_var_imp <- as.data.frame(t(mnist_89_var_imp))

# rescale importance to [0, 256] so they can be used as grayscale-image pixels:
mnist_89_var_imp_rescaled <- 256 - mnist_89_var_imp * 256 / max(mnist_89_var_imp)

plot_mnist_raster(mnist_89_var_imp_rescaled)
```



(white means high importance, black means low importance)

Can you think of why these regions are important?