6 Walkthrough with examples in R

6.1 Getting the variables

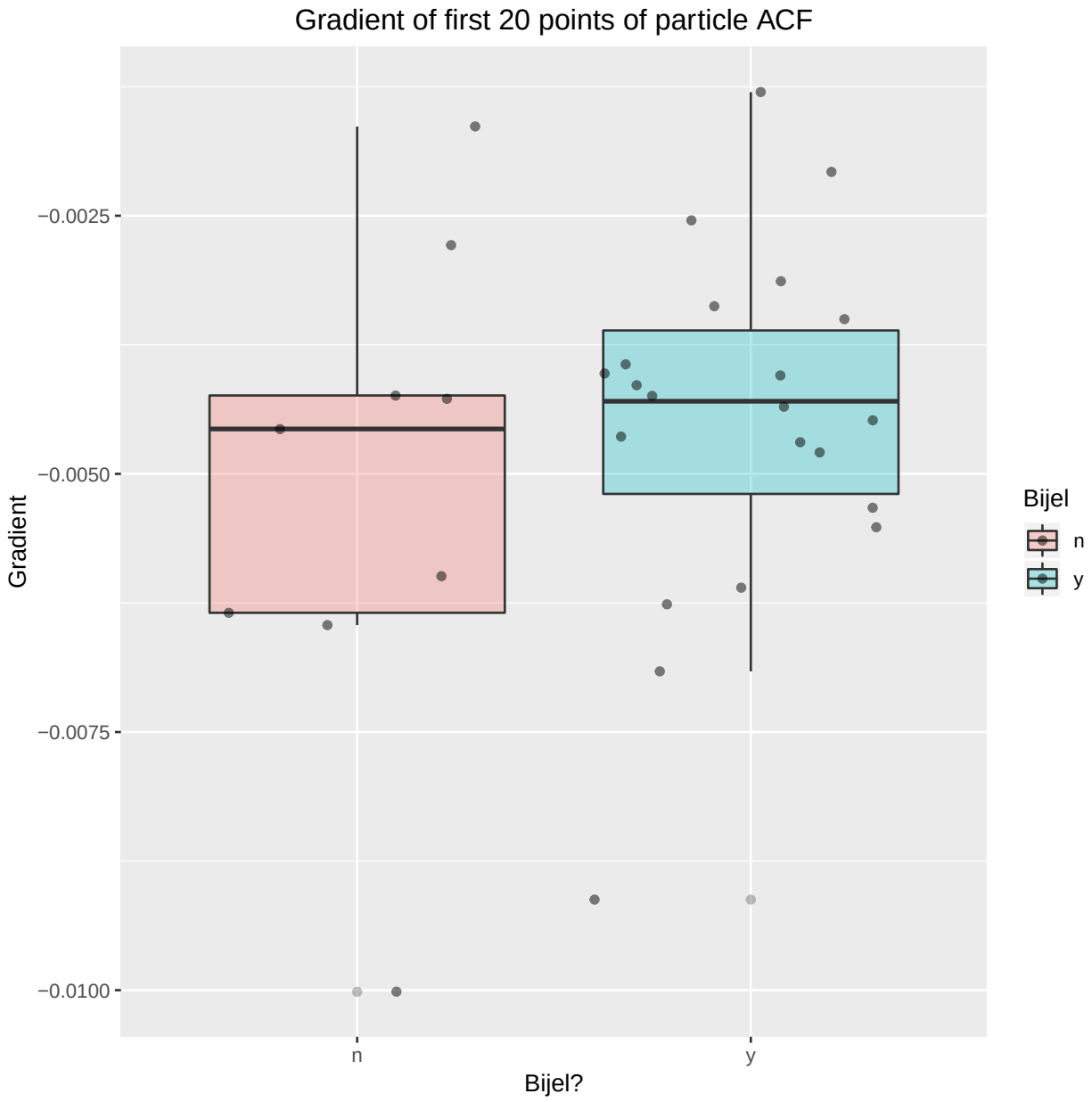
When performing this analysis on a set of data, we start with the autocorrelation functions of the two channels in each image, and labels of whether these images are bijels or not.

```
##      Sample.Number Bijel            ACF_1              ACF_2
## 19              19     n list(ACF_liquid) list(ACF_particle)
## 20i            20i     y list(ACF_liquid) list(ACF_particle)
## 20ii          20ii     y list(ACF_liquid) list(ACF_particle)
## 21              21     n list(ACF_liquid) list(ACF_particle)
## 22i            22i     n list(ACF_liquid) list(ACF_particle)
## 22ii          22ii     n list(ACF_liquid) list(ACF_particle)
```

We then need to turn these functions into a set of single-valued variables that describe features that may separate bijels from non-bijels, such as:

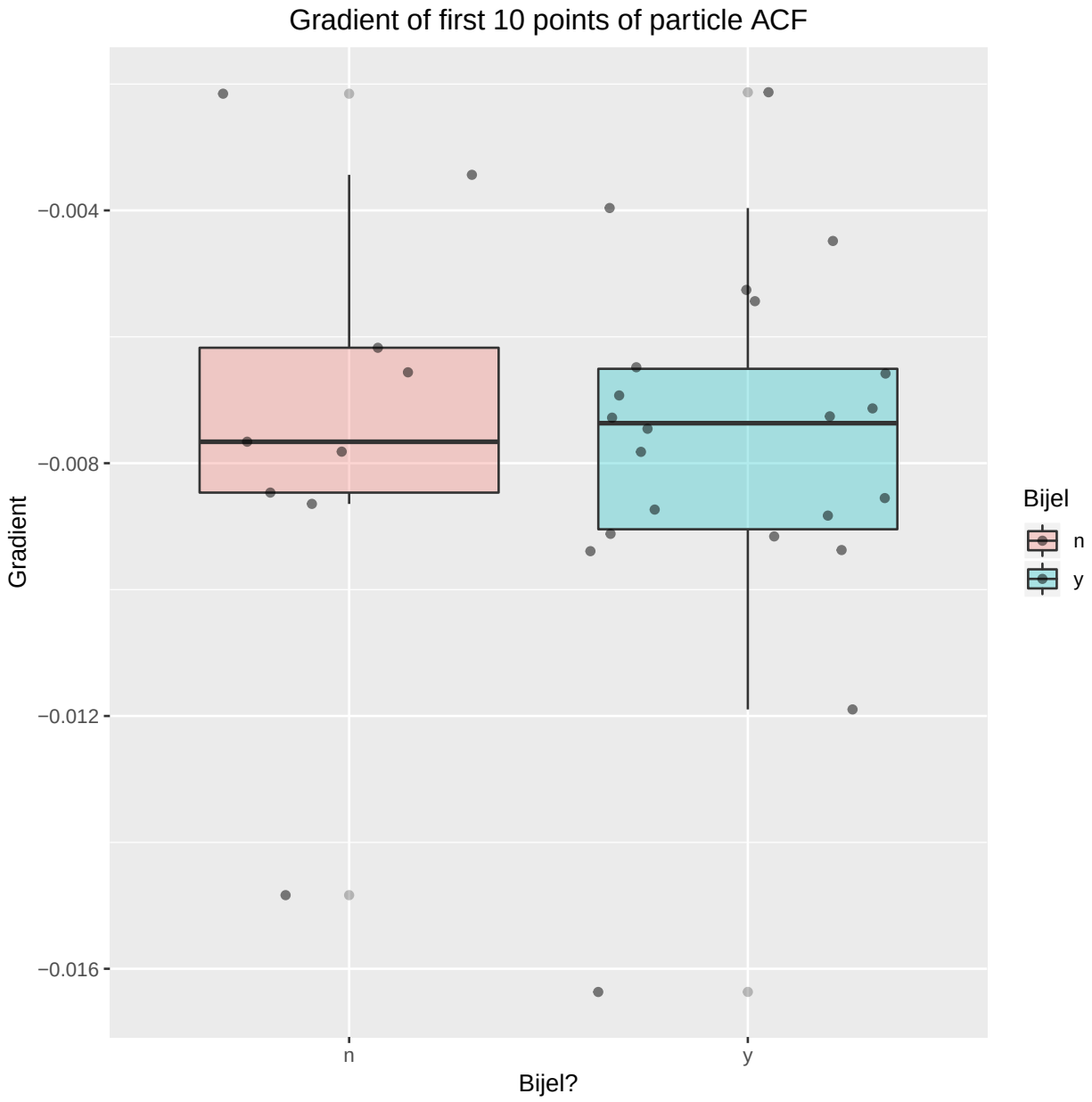
- The gradient of the particle channel autocorrelation function

```
r <- c(1:256)
num_points <- length(exp_Data$Sample.Number)
y <- exp_Data$Autocorrelation.Particle[1:20]
lineFits <- lapply(1:num_points,
                   function(n) lm(unlist(y[n,]) ~ r[1:20]))
lineCoeffs <- lapply(lineFits,
                     function(m) m$coefficients)
lineGradients <- lapply (1:num_points,
                         function(p) unname(lineCoeffs[[p]][2]))
exp_Data$Particle.Gradients.20 <- unlist(lineGradients)


library(ggplot2)
ggplot(exp_Data,
       aes(x=as.factor(Bijel), y=Particle.Gradients.20,
           fill=Bijel)) +
       geom_boxplot(alpha=0.3) +
       geom_jitter(alpha=0.5) +
       xlab("Bijel?") + ylab("Gradient") +
       ggtitle("Gradient of first 20 points of particle ACF") +
       theme(plot.title = element_text(hjust = 0.5))
```



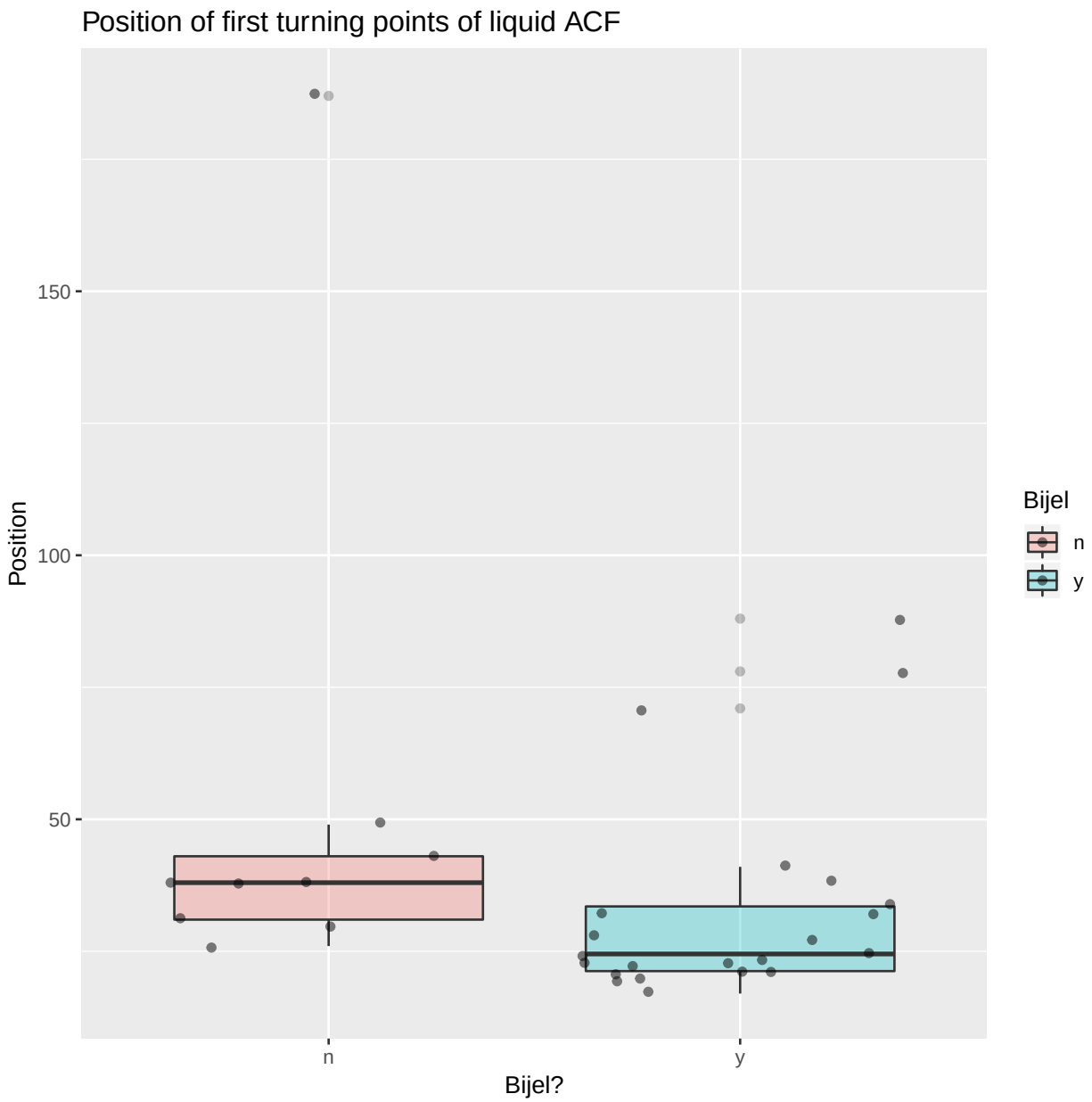

```
y2 <- exp_Data$Autocorrelation.Particle[1:10]
lineFits2 <- lapply(1:num_points,
                    function(n) lm(unlist(y2[n,]) ~ r[1:10]))
lineCoeffs2 <- lapply(lineFits2,
                      function(m) m$coefficients)
lineGradients2 <- lapply (1:num_points,
                          function(p) unname(lineCoeffs2[[p]][2]))
exp_Data$Particle.Gradients.10 <- unlist(lineGradients2)

ggplot(exp_Data,
       aes(x=as.factor(Bijel), y=Particle.Gradients.10,
```

```
        fill=Bijel)) +
    geom_boxplot(alpha=0.3) +
    geom_jitter(alpha=0.5) +
    xlab("Bijel?") + ylab("Gradient") +
    ggtitle("Gradient of first 10 points of particle ACF") +
    theme(plot.title = element_text(hjust = 0.5))
```



- The position of the first turning point in the liquid channel autocorrelation function

```
library(pastecs)
liquidTurns <- lapply(1:num_points,
                      function(y) turnpoints(unlist(
                        exp_Data$Autocorrelation.Liquid[y,])))
firstTurn <- lapply(1:num_points,
                    function(y) liquidTurns[[y]]$tppos[1])
exp_Data$Liquid.First.Turn <- unlist(firstTurn)

ggplot(exp_Data,
       aes(x=as.factor(Bijel), y=Liquid.First.Turn,
           fill=Bijel)) +
       geom_boxplot(alpha=0.3) +
       geom_jitter(alpha=0.5) +
       xlab("Bijel?") + ylab("Position") +
       ggtitle("Position of first turning points of liquid ACF")
```



We have generated box-and-jitter plots to see how the distribution of these variables is dependent on whether or not the sample is a bijel.

Once we have these variables, we can apply a machine learning model to it in one of two ways: training the model on the data and testing via cross-validation, and testing a previously trained model on the new data.

6.2 Training a model and testing with cross-validation

Here we have chosen to train a k-nearest neighbours model with the three variables shown above, based on our experience with a previous set of data. This can be done very simply using the CARET package in R, which contains a number of widely-used machine learning algorithms as well as cross-validation capabilities.

```r
library(caret)
```

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

```r
# set up the data
attach(exp_Data)
dat=data.frame(
  Particle.Gradients.20,
  Particle.Gradients.10,
  Liquid.First.Turn,
  Bijel)

# set a random number seed for reproducability
set.seed(1234)

# define the cross-validation parameters:
# here we use 10-fold cross-validation and repeat it 3 times
trCtrl <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3)

knnFit <- train(Bijel~., # Bijel = output, other variables = input
                data=dat, # define the data
                method="knn", # choose the algorithm
                trControl=trCtrl, # cross-validation as above
                tuneLength=10)
print(knnFit)
```

```
## k-Nearest Neighbors
##
## 31 samples
##  3 predictor
##  2 classes: 'n', 'y'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 28, 28, 28, 28, 28, 27, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.6611111   0.13928571
##    7  0.5611111  -0.01034483
##    9  0.6638889   0.19310345
##   11  0.7055556   0.22758621
##   13  0.7388889   0.25000000
##   15  0.7000000   0.00000000
##   17  0.7166667   0.00000000
##   19  0.7166667   0.00000000
##   21  0.7166667   0.00000000
##   23  0.7166667   0.00000000
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 13.
```

6.3 Testing a trained model on new data

If we have already trained a model, we can use it to classify new data. This is how the algorithm can be used as a tool to classify unlabelled data.

```
# predict whether the data points are bijels or not
# to do this, we have to omit the bijel label from the data
bijel_pred = predict(trainedKNN, newdata = dat[,-4])
bijel_true = dat[,4]
print(data.frame(bijel_pred, bijel_true=bijel_true))
```

```
##    bijel_pred bijel_true
## 1           y          n
## 2           y          y
## 3           y          y
## 4           y          n
## 5           y          n
## 6           y          n
## 7           y          y
## 8           y          y
## 9           y          n
## 10          y          n
## 11          y          y
## 12          y          y
## 13          y          n
## 14          n          n
## 15          n          n
## 16          n          y
## 17          y          y
## 18          y          y
## 19          y          y
## 20          y          y
## 21          y          y
## 22          y          y
## 23          y          y
## 24          y          y
## 25          n          y
## 26          n          y
## 27          y          y
## 28          n          y
## 29          y          y
## 30          n          y
## 31          y          y
```

We can now calculate the error rate, and also look at the results to see how many false positives, false negatives, etc. we have.

```
success_count=length(bijel_pred[bijel_pred==bijel_true])
success_rate=success_count/length(bijel_pred)
paste0("Success rate: ",round(100*success_rate),"%")

## [1] "Success rate: 61%"

library(gmodels)
CrossTable(x=bijel_pred, y=bijel_true, prop.chisq=FALSE)

##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  31
##
##
##              | bijel_true
##   bijel_pred |         n |         y | Row Total |
## -------------|-----------|-----------|-----------|
##            n |         2 |         5 |         7 |
##              |     0.286 |     0.714 |     0.226 |
##              |     0.222 |     0.227 |           |
##              |     0.065 |     0.161 |           |
## -------------|-----------|-----------|-----------|
##            y |         7 |        17 |        24 |
##              |     0.292 |     0.708 |     0.774 |
##              |     0.778 |     0.773 |           |
##              |     0.226 |     0.548 |           |
## -------------|-----------|-----------|-----------|
## Column Total |         9 |        22 |        31 |
##              |     0.290 |     0.710 |           |
## -------------|-----------|-----------|-----------|
##
##
```

In this case we can see that the error rate obtained from applying the old model to new data (39%) is much higher than the one obtained by directly training the same type of model on the data in question (13%). This is because the two datasets are from slightly different physical systems so although the same variables are useful, bijels are indicate at different values of these variables.