Statistical Learning

Group 19 - Le superchicche

Cammarota Sara, Iadisernia Giulia, Petrucci Ilaria

May 15th, 2023

Part I

1.1 "Be linear in transformed feature space"

bla bla bla

1.2 Truncated power basis functions

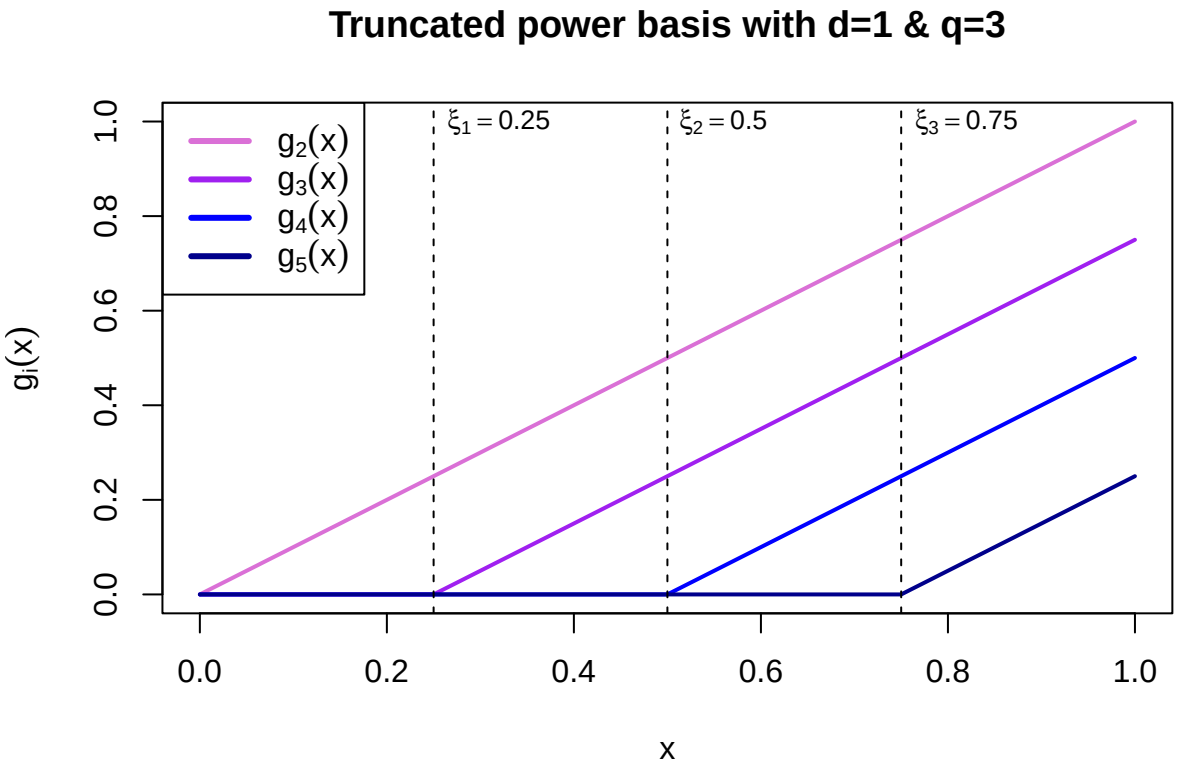
Truncated power basis functions $G_{d,q}$ are defined as $\{g_1(x) = 1, g_2(x) = x, ..., g_{d+1}(x) = x^d\}$ and $\{g_{(d+1)+j}(x) = (x - \xi_j)_+^d\}_{j=1}^q$ where$(x)_+ = max\{0, x\}$

We choose *degree* $d \in \{1, 3\}$ and the number of equi-spaced *knots* $q \in \{3, 10\}$ where $\{\xi_j\}_{j=1}^q \in (0, 1)$.

1.2.1 Our implementation vs...

Let's suppose $d = 1$ and $q = 3$

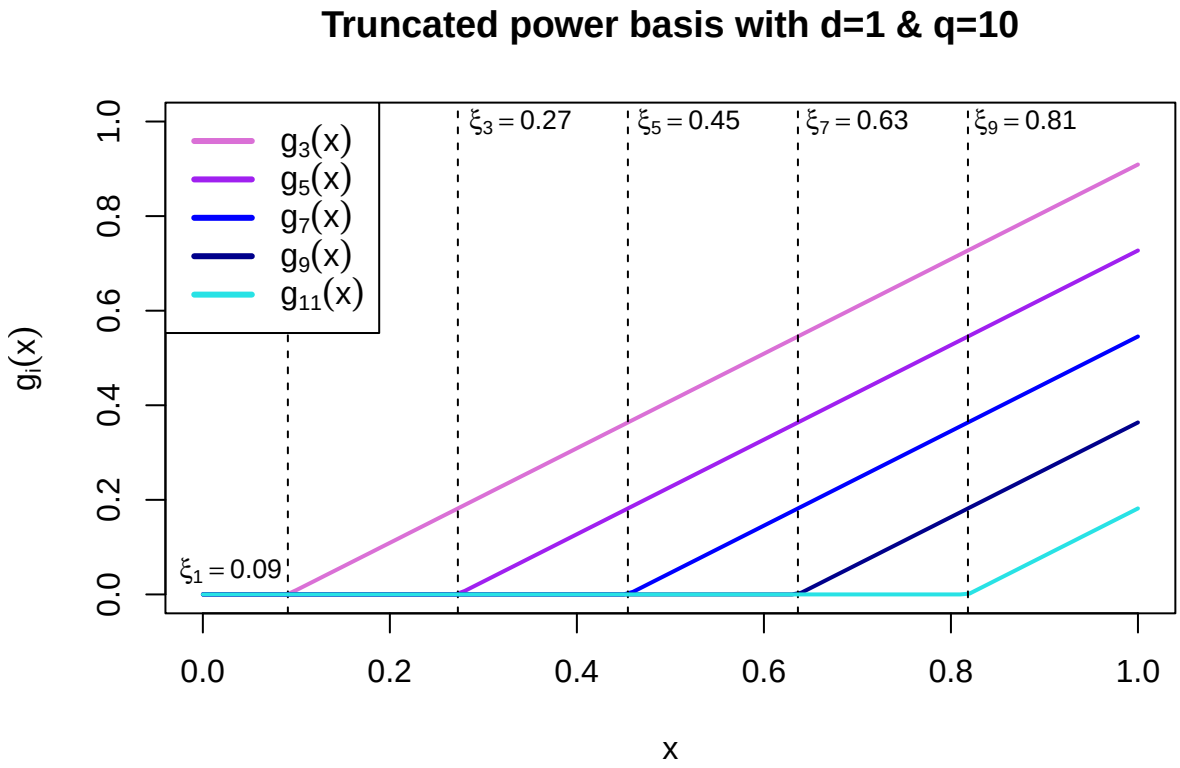
$\{g_i(x)\}_{i=1}^2 = \{g_1(x) = 1, g_2(x) = x\}$ and $\{g_{2+j}(x)\}_{j=1}^3 = \{g_3(x) = (x - \xi_1)_+, g_4(x) = (x - \xi_2)_+, g_5(x) = (x - \xi_3)_+\}$ where the knots $\{\xi_j\}_{j=1}^3$ are respectively equal to $0.25, 0.50, 0.75$.



When $d = 1$ and $q = 10$

$$\{g_i(x)\}_{i=1}^{2} = \{g_1(x) = 1, g_2(x) = x\} \quad \text{and} \quad \{g_{2+j}(x)\}_{j=1}^{10}$$

where the knots $\{\xi_j\}_{j=1}^{3}$ are respectively equal to $0.09, 0.18, 0.27, 0.36, 0.45, 0.54, 0.63, 0.72, 0.81, 0.91$. In the figure below we plot some of these basis functions.



Now, let's have a look at what changes when we increase the degree to $d = 3$. For simplicity, let's choose $q = 3$.

rifare plots con x, x**2,** **x**3 + i tre knots

1.2.2 . . . ChatGPT's reply

1.3 Our very first Kaggle competition

Import modules

```
library(caret)
library(glmnet)
library(elasticnet)
library(doParallel)
library(dplyr)
```

The `wmap` dataset

We start by importing the training and testing datasets. The training dataset consists of 675 angular power spectrum observations (rows) and 3 columns:

- `id` which uniquely identifies each row (necessary to correctly ascertain the "goodness of fit" of our predictions)
- `x` the predictor variable
- `y` the target variable

The `test` data frame contains another 224 angular power spectrum observations and solely two columns (`id` and `x`).

```
train <- read.csv('train.csv')
test <- read.csv('test.csv')
```

1.3.1 Equi-spaced knots

In this section we aim at fining the "best" (d, q)-combination, firstly via plain old vanilla k-fold cross validation, and secondly by implementing Bates' nested cross validation algorithm.

Vanilla Cross-Validation: K-fold CV

First, we must define some essential **functions** which will make our life much easier when testing our models as the values of q, d and other hyper-parameters change.

`remap` maps the feature vector x into a higher-dimensional feature space X, a matrix containing $(d + q)$ columns (excluding $g_1(x) = 1$ which is automatically accounted for when using `glmnet`) and `length(x)` rows (675 in case of the training set). This function takes in input d, q, a vector containing the position of the `knots = seq(1/(q+1), 1 - 1/(q+1), 1/(q+1))`, and obviously `x`. The j^{th} column of the feature matrix `X` stores the truncated power basis function $g_j(x)$ evaluated at `x` with $2 \leq j \leq d + q + 1$.

```
remap <- function(d, q, knots, x){
  n = length(x)
  X <- matrix(0, n, d+q)
  for(i in 1:d){
    X[,i] <- (x)**i
  }
  for(j in 1:q){
    idx = d+j
    X[,idx] <- pmax(0, x - rep(knots[j], n))**d
  }
  return(X)
}
```

`bestmodel` applies 5-fold CV to find the "best" combination of d and q, namely the one that yields the lowest `rmse` score averaged over the 5 validation folds. The function takes in input `qmax`, `dmax`, `train$x`, `train$y`, `pos` indicating the way we want to position the knots (for now let's suppose the knots are equi-spaced) and `w`, a vector of weights. For each (d, q) couple we

- `set.seed(123)` inside the function.
- `remap` the feature vector `x` into the feature matrix `X`.
- create the data frame `train_data` containing the $d + q$ feature columns together with the target variable y values.
- choose a tuning grid of *lambda* values. Specifically, we give `lambda=seq(1e-3, 1, length=100)`, while *alpha* always equals zero (ridge regression). We experimented with other forms of penalization such as lasso and elastic net which, however, did not drastically improve our model performance while heavily slowing down the search of the best (d, q)-combination (running `bestmodel` was slower when not using ridge regression).
- exploit the `train` function of the `caret` package to perform 5-fold CV which spits out the `model` (and hence hyperparameters) yielding the lowest `rmse` score for that choice of `d` and `q`.
- save `lambda`, `rmse` and features matrix `X` into lists.
- return the (d, q)-combination which minimizes the validation error.

```
bestmodel <- function(qmax=1, dmax=1, train_x, train_y, pos = 'equi', w){
  set.seed(123)
  D <- c(1, 3)
  Q <- 1:qmax
  combinations <- data.frame(expand.grid(D, Q))
  rmse <- c(rep(NA, nrow(combinations)))
  alpha_values <- c(rep(NA, nrow(combinations)))
  lambda_values <- c(rep(NA, nrow(combinations)))
  knots_pos <- list()
  features <- list()
  if(pos == 'equi'){
    knots_pos = lapply(Q, function(q)seq(1/(q+1), 1 - 1/(q+1), 1/(q+1)))
  }
  if (pos == 'quantile') {
    knots_pos = lapply(Q, function(q)
      unname(quantile(train_x,
                      probs = seq(
                        1 / (q + 1), 1 - 1 / (q + 1), 1 / (q + 1)))))
  }
  for(row in 1:nrow(combinations)){
    d = combinations[row, 1]
    q = combinations[row, 2]
    if(pos == 'cluster'){
      dist <- dist(train_y)
      hc <- hclust(dist)
      hc_labels <- cutree(hc, k = q)
```

```
      knots <-
        sort(unlist(lapply(1:q, function(i)
          mean(train_x[hc_labels == i]))))
      knots_pos[[q]] <- knots
    }
    else{
      knots <- knots_pos[[q]]
    }
    X <- remap(d, q, knots, train_x)
    train_data <- data.frame(cbind(X, "y" = train_y))
    train.control <- trainControl(method = "cv",
                                  number = 5, allowParallel = TRUE)
    my_grid <- expand.grid(alpha = 0, lambda = seq(1e-3, 1, length = 100))
    model <- train(form = y ~. , data = train_data, method = "glmnet",
                   trControl = train.control, metric = "RMSE",
                   tuneGrid = my_grid, weights = w)
    alpha_values[row] <- model$bestTune$alpha
    lambda_values[row] <- model$bestTune$lambda
    rmse[row] <- model$results$RMSE[as.integer(rownames(model$bestTune))]
    features <- append(features, list(X))
  }
  idx_min <- combinations[which.min(rmse),]
  return(list(idx_min, features, rmse, knots_pos, alpha_values, lambda_values))
}
```

`prediction` returns the prediction values `y_pred` necessary to evaluate the performance of the model. Specifically, it

- unpacks the output list of the `bestmodel` function,
- trains a `glmnet` model having as `alpha` and `lambda` parameters the one chosen by the `bestmodel` function
- remaps the test set feature `x` into a feature matrix `Xtest`
- and returns the predictions `y_pred`

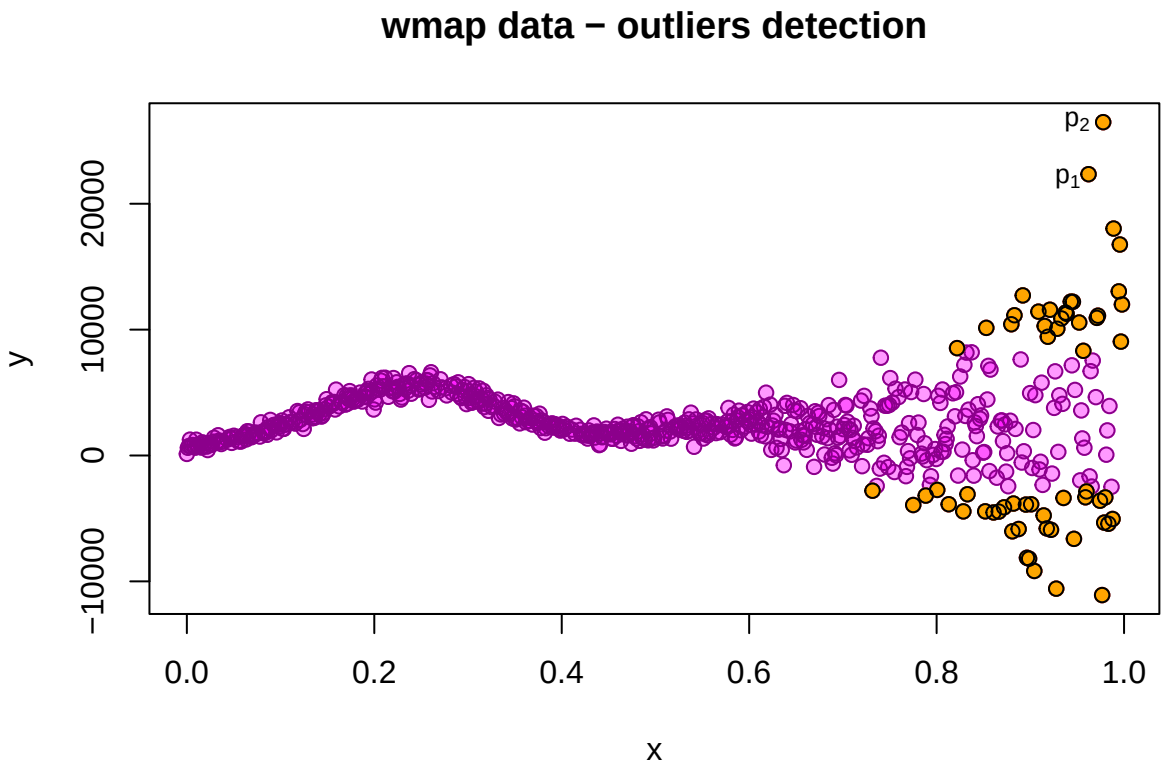
```
prediction <- function(res, train, test){
    # best model hyperparameters
    idx_min = res[[1]]
    d = idx_min$Var1
    q = idx_min$Var2
    X = res[[2]][[as.integer(rownames(idx_min))]]
    rmse = res[[3]][as.integer(rownames(idx_min))]
    knots = res[[4]][[q]]
    alpha = res[[5]][as.integer(rownames(idx_min))]
    lambda = res[[6]][as.integer(rownames(idx_min))]

    # train the model + predict values on test set data
```

```
    train_data <- data.frame(cbind(X, "y" = train$y))
    colnames(train_data) <- c(paste0('X', 2:ncol(train_data)-1), 'y')
    mod <- glmnet(X, train$y, alpha = alpha, lambda = lambda,
                  family = "gaussian")
    Xtest = remap(d, q, knots, test$x)
    y_pred <- predict(mod, newx = Xtest)

    return(y_pred)
}
```

How to deal with outliers

Typically, outliers are considered to be values that are below `Q1 - 1.5*IQR` or above `Q3 + 1.5*IQR`. We define a new data frame `no_outliers` which excludes the 59 outlier observations, leaving us with 616 rows. The orange points in the plot below are the outliers.

```
Q1 <- quantile(train$y, .25)
Q3 <- quantile(train$y, .75)
IQR <- unname(Q3 - Q1)

no_outliers <- subset(train, train$y > (Q1 - 1.5*IQR) & train$y < (Q3 + 1.5*IQR))
out <- anti_join(train, no_outliers, by = c('id'='id', 'x'='x', 'y'='y'))
```



Instead of just getting rid of all outliers, we can just remove the two farthermost points p_1 and p_2 in the plot and weight less those outliers when training the model. The `weights` vector can be passed directly to the `weights` parameter of `caret`'s `train` function. We set all weights to 1 except for outliers observations whose weights equal 0.5.

```
train <- subset(train, train$y < 20000) # get rid of two farthermost obs
x_train = train$x
y_train = train$y

weights <- c(rep(NA, nrow(train)))
for (row in 1:nrow(train)) {
  weights[row] <- ifelse(train[row, 1] %in% out[,1], 0.5, 1)
}
```

Run the `bestmodel` function

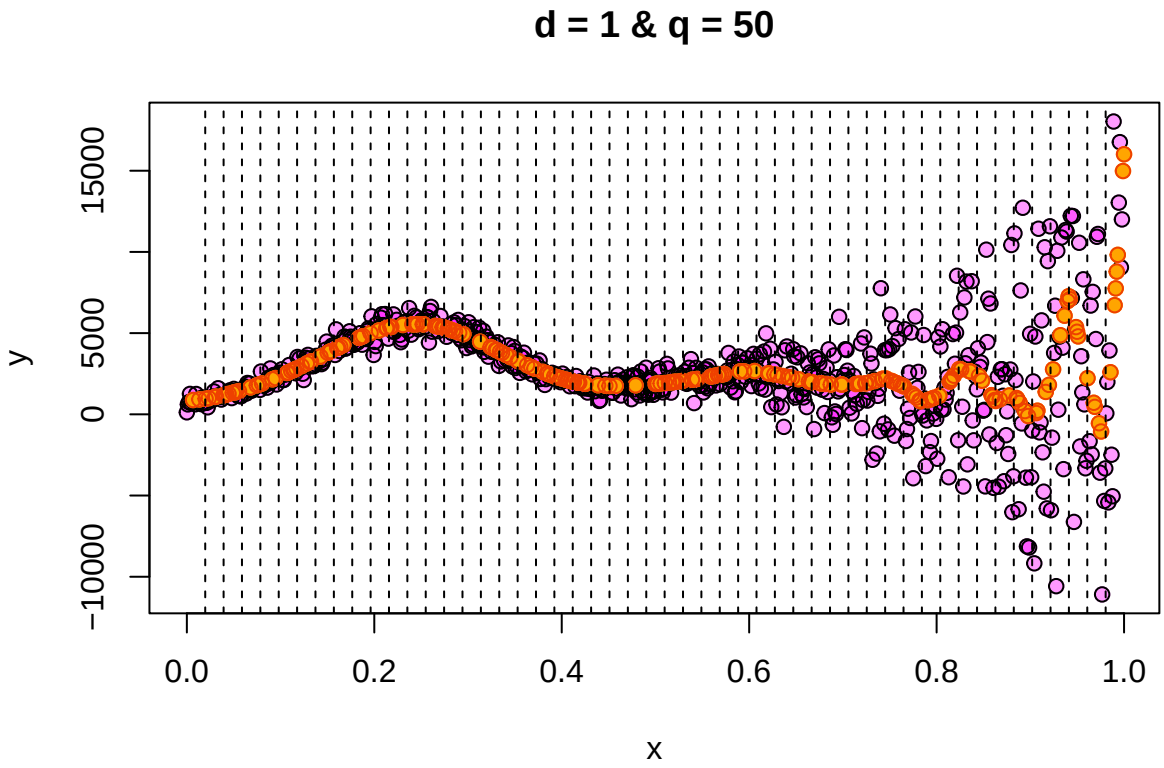
Let's set `qmax = 50` and `dmax = 3`.

```
res <- bestmodel(qmax=50, dmax=3, x_train, y_train, pos='equi', w=weights)

idx_min = res[[1]]
d_best = idx_min$Var1
q_best = idx_min$Var2
X = res[[2]][[as.integer(rownames(idx_min))]]
mse_list = res[[3]]**2
mse_min = mse_list[as.integer(rownames(idx_min))]
knots = res[[4]][[q_best]]
alphas = res[[5]]
lambdas = res[[6]]
alpha_min = res[[5]][as.integer(rownames(idx_min))]
lambda_min = res[[6]][as.integer(rownames(idx_min))]
idx_min
```

```
##    Var1 Var2
## 99    1   50
```

The best model according to vanilla cv is the one having `d=1` and `q=50`. That's an awful lot of knots. The cross validation`rmse` equals approximately `2689`, the training set error equals `2537` compared to the public leaderboard test set score of ... RIEMPIRE (submission 12)

Let's have a look at the plot of the fit.



Bates' et al. Nested CV

!!!explain what problem nested cv wants to solve

`inner_crossval` performs k-fold cross validation. Precisely, we

- repeatedly split the training set derived from the outer loop into a new training set (K-1 remaining folds) `in_train` (K-2 folds) and a validation set `val` (1 fold).
- we then train the model on the `in_train` data and fit the model to `val` data.
- finally, we append to `e_in` the vector of residual values `e_temp`.

```
inner_crossval <- function(d=1, q=1, K=10, knots,
                           out_folds, data, kout, pos='equi'){
    e_in <- c()
    kval <- 1:K
    for(k in kval[kval!=kout]){
        idx <- which(out_folds==k,arr.ind=TRUE)
        val <- data[idx, ]
        temp <- data[-idx, ] # do not consider validation data
        idx_out <- which(out_folds==kout,arr.ind=TRUE)
        in_train <- temp[-idx_out, ] # do not consider holdout data
        if(pos=='cluster'){
            dist <- dist(in_train$y)
```

```
            hc <- hclust(dist)
            hc_labels <- cutree(hc, k = q)
            knots_train <-
              sort(unlist(lapply(1:q, function(i)
                mean(in_train$x[hc_labels == i]))))
        }
        else{
            knots_train <- knots
        }
        X <- remap(d, q, knots_train, in_train$x)
        model <- glmnet(X, in_train$y, alpha=0)
        Xval <- remap(d, q, knots, val$x)
        y_pred <- predict(model, newx=Xval)
        e_temp <- (y_pred - val$y)**2
        e_in <- append(e_in, e_temp)
    }
    return(list(e_in))
}
```

`outer_crossval` performs the outer loop of the nested cross validation algorithm.

- we repeatedly split the training set `train` into a smaller (K-1 fold) `training` set and a holdout fold (1 fold).
- we apply the `inner_crossval` function to the remaining (K-1) folds and obtain a vector of losses `e_in` which we append to another vector `es`.
- we train a `glmnet` model on the `training` data and evaluate it on the `holdout` data obtaining another vector of losses `e_out`.
- we append `(mean(e_in) - mean(e_out))**2` to the `a_list` vector.
- we append `var(e_out)/nrow(holdout)` to the `b_list` vector.
- finally, we return `es`, `a_list` and `b_list`.

```
outer_crossval <- function(d=1, q=1, K=10, knots,
                           out_folds, data=data, pos='equi'){
    es <- c()
    a_list <- c()
    b_list <- c()
    for(kout in 1:K){
        idx <- which(out_folds==kout,arr.ind=TRUE)
        holdout <- data[idx, ]  # specify holdout
        training <- data[-idx, ] # specify training
        res_in <- inner_crossval(d, q, K, knots, out_folds, data=data, kout=kout)
        e_in <- res_in[[1]]
        if(pos=='cluster'){
            dist <- dist(training$y)
            hc <- hclust(dist)
            hc_labels <- cutree(hc, k = q)
```

```
            knots_train <-
              sort(unlist(lapply(1:q, function(i)
                mean(training$x[hc_labels == i]))))
        }
        else{
            knots_train <- knots
        }
        Xtrain <- remap(d, q, knots_train, training$x)
        Xout <- remap(d, q, knots, holdout$x)
        model <- glmnet(Xtrain, training$y, alpha=0)
        y_pred <- predict(model, newx=Xout)
        e_out <- (y_pred - holdout$y)**2
        a_list <- append(a_list, (mean(e_in) - mean(e_out))**2)
        b_list <- append(b_list, var(e_out)/nrow(holdout))
        es <- append(es, e_in)
    }
    return(list(es, a_list, b_list))
}
```

`nested_cv` is an implementation of Bates' et al. nested cross validation algorithm.

- we perform the `outer_crossval` function `R` times, each time randomly sampling from the `train` dataset which is split into `K=10` folds.
- We define the `MSE` as `mean(a_list) - mean(b_list)`.

```
nested_cv <- function(d=1, q=1, K=10, knots, train, R=5, pos='equi'){
    set.seed(123)
    a_list <- c()
    b_list <- c()
    es <- c()
    pb <- txtProgressBar(min = 0, max = R, style = 3)
    for(r in 1:R){
        data <- train[sample(nrow(train)),]
        outer_folds <- cut(seq(1,nrow(data)), breaks=K, labels=FALSE)
        res <- outer_crossval(d, q, K, knots, out_folds=outer_folds, data=data)
        es <- append(es, res[[1]])
        a_list <- append(a_list, res[[2]])
        b_list <- append(b_list, res[[3]])
        setTxtProgressBar(pb, r)
    }
    MSE <- mean(a_list) - mean(b_list)
    Err <- mean(es)
    return(list(MSE, Err))
}
```

`genmodel`

```
genmodel <- function(qmax=1, mse_list, upper_bound){
    D <- c(1, 3)
    Q <- 1:qmax
    combinations <- data.frame(expand.grid(D, Q))
    d_gen = d_best
    q_gen = q_best
    for(model in 1:length(mse_list)){
        d = combinations[model, 1]
        q = combinations[model, 2]
        if((mse_list[model] <= upper_bound) & (d+q <= d_gen+q_gen)){
            d_gen = d
            q_gen = q
        }
    }
    return(list(d_gen, q_gen))
}
```

1.3.2 Hierarchical clustered knots

Vanilla Cross-Validation: K-fold CV

the functions have already been listed before just explain how we choose the knots and show model performance.

Bates' et al. Nested CV

Conclusion

Part II