Exercise Feature Importance

Exercise 1: PDP and ICE in case of interactions

a) Derivation of PD function for $S = \{1\}$ (with $C = \{2\}$) given

$$\hat{f}(\mathbf{x}) = \hat{f}(x_1, x_2) = \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1 x_2 + \hat{\beta}_0$$

$$\begin{aligned}
f_{1,PD}(x_1) = \mathbb{E}_{x_2}\left(\hat{f}(x_1, x_2)\right) &= \int_{-\infty}^{\infty} \left(\hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1 x_2 + \hat{\beta}_0\right) d\mathbb{P}(x_2) \\
&= \hat{\beta}_1 x_1 + \int_{-\infty}^{\infty} \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1 x_2 \, d\mathbb{P}(x_2) + \hat{\beta}_0 \\
&= \hat{\beta}_1 x_1 + \int_{-\infty}^{\infty} (\hat{\beta}_2 + \hat{\beta}_3 x_1) x_2 \, d\mathbb{P}(x_2) + \hat{\beta}_0 \\
&= \hat{\beta}_1 x_1 + (\hat{\beta}_2 + \hat{\beta}_3 x_1) \cdot \int_{-\infty}^{\infty} x_2 \, d\mathbb{P}(x_2) + \hat{\beta}_0 \\
&= \hat{\beta}_1 x_1 + (\hat{\beta}_2 + \hat{\beta}_3 x_1) \cdot \mathbb{E}_{x_2}(x_2) + \hat{\beta}_0
\end{aligned}$$

b) PD function for $\hat{\beta}_0 = 0$, $\hat{\beta}_1 = -8$, $\hat{\beta}_2 = 0.2$, $\hat{\beta}_3 = 16$, $X_1 \sim Unif(-1, 1)$ and $X_2 \sim B(1, 0.5)$.

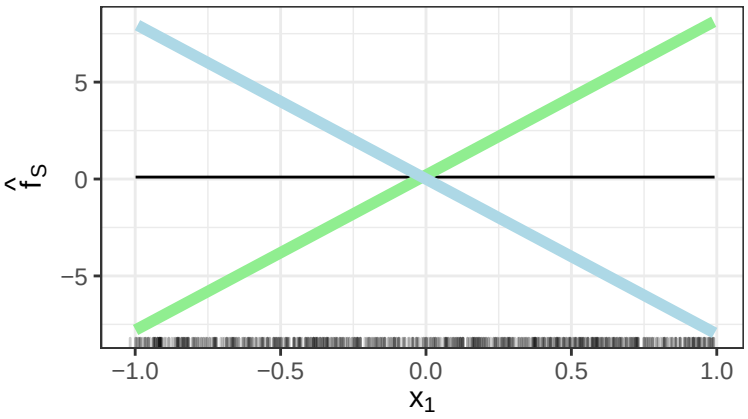
$$\begin{aligned}
f_{1,PD}(x_1) = \hat{\beta}_1 x_1 + (\hat{\beta}_2 + \hat{\beta}_3 x_1) \cdot \mathbb{E}_{x_2}(x_2) + \hat{\beta}_0 &= -8x_1 + (0.2 + 16x_1) \cdot \mathbb{E}_{x_2}(x_2) + 0 \\
&= -8x_1 + (0.2 + 16x_1) \cdot 0.5 \\
&= -8x_1 + 0.1 + 8x_1 \\
&= 0.1 + 0x_1
\end{aligned}$$

c) ICE functions for group $X_2 = 1$ and for group $X_2 = 0$:

$$f_1(x_1) = \begin{cases} -8x_1 + (0.2 + 16x_1) \cdot 1 = 8x_1 + 0.2 & x_2 = 1 \\ -8x_1 + (0.2 + 16x_1) \cdot 0 = -8x_1 & x_2 = 0 \end{cases}$$

The light green dots correspond to group $X_2 = 1$, the light blue dots to group $X_2 = 0$.

d) The example illustrates that by averaging of ICE curves for a PD plot, we might obfuscate heterogeneous effects and interactions. Although ICE curves showed a strong effect of X_1 on Y, the effect was not apparent in PDPs. Therefore, it is highly recommended to plot PD plots and ICE curves together.



Exercise 2: ALE

2a)

```
get_bounds <- function(X, s, n_intervals = 100) {
  ### Luckily for us the seq()-function already implements
  ### exactly what we want if we take two neighboring
  ### equidistant points as the boundaries of one interval.
  ### Now we need n_intervals + 1 points since we always
  ### need two points for one interval resulting
  ### in one point more than the number of intervals we want.
  seq(min(X[ , s]), max(X[ , s]), length.out = n_intervals + 1)
}
```

2b)

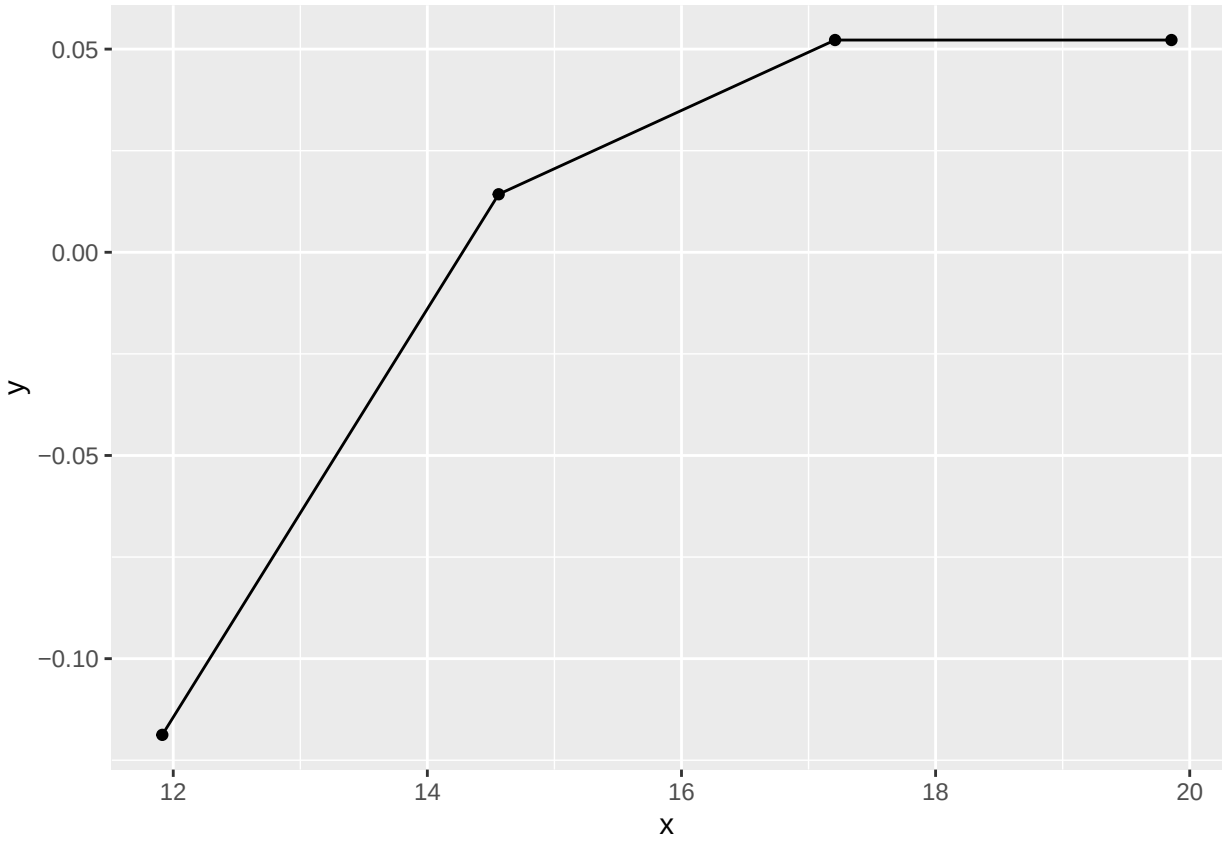
```
calculate_ale <- function(model, X, s, n_intervals = 100, centered = FALSE) {
  ### Get the feature column specified by s from the input data.
  x_s <- X[ , s]

  ### Save all possible lower and upper bounds in
  ### lowerbounds and upperbounds, respectively.
  boundary_points <- get_bounds(X, s, n_intervals)
  lowerbounds <- boundary_points[1:n_intervals]
  upperbounds <- boundary_points[2:(n_intervals + 1)]

  ### Now we want to iterate over all intervals. For this we
  ### will use mapply() in the following but for readability we define the
  ### function to apply before. This function returns the
  ### uncentered ALE value for one interval given the left and the right
  ### boundary of the interval.
  uncentered_ale_calculation <- function(left_boundary, right_boundary) {
    ### The first interval is of the form [left_boundary, right_boundary]
    ### (so that we do not ignore the points with the minimal feature value).
    if (left_boundary == lowerbounds[1]) {
      idxs <- which(x_s >= left_boundary & x_s <= right_boundary)
    }
    ### All other intervals are of the form (left_boundary, right_boundary]
    ### (so that no point falls into two intervals).
    else {
      idxs <- which(x_s > left_boundary & x_s <= right_boundary)
    }

    ### If there is no point in the interval return 0
    ### (as stated in the task).
    if (length(idxs) == 0) {
      return(0)
    }

    ### Else we compute the ALE value. The first step is to
    ### replace the feature values of all the points that fall into the interval
    ### with the left and the right boundary of the interval.
    X_min <- X[idxs, ]
    X_max <- X[idxs, ]
    X_min <- replace(X_min, s, left_boundary)
    X_max <- replace(X_max, s, right_boundary)
```

```
    ### Then we let the model predict these "new" data points.
    y_min <- predict(model, X_min)
    y_max <- predict(model, X_max)

    ### We return the arithmetic mean of the prediction differences.
    mean(y_max - y_min)
  }

  ### Now we can calculate the ALE values all at once with the mapply()-function.
  ### We accumulate them via the cumsum()-function.
  ale_values <- cumsum(mapply(uncentered_ale_calculation, lowerbounds, upperbounds))

  ### If the centered argument is set to TRUE the user wants us to center the ALE values.
  if (centered)
    ale_values <- ale_values - mean(ale_values)

  ### Return the boundary points as well as the centered or uncentered ALE values.
  list(bounds = boundary_points, ale = ale_values)
}
```

2c)

```
prepare_ale <- function(model, X, s, n_intervals = 100, centered = TRUE) {
  ### Get the ALE values via our function from task b).
  ale <- calculate_ale(model, X, s, n_intervals, centered)

  ### Get the bounds and y from the list.
  boundary_points <- array(unlist(ale[1]))
  y <- array(unlist(ale[2]))

  ### Reconstruct the lowerbounds and upperbounds vectors.
  lowerbounds <- boundary_points[1:n_intervals]
  upperbounds <- boundary_points[2:(n_intervals + 1)]

  ### Get the center of each interval.
  x <- rowMeans(cbind(lowerbounds, upperbounds))

  ### Now for ggplot2 the expected format is a data.frame.
  data.frame(x = x, y = y)
}

### A function to create the ALE plot from the previous generated objects.
ale_plot <- function() {
  library(randomForest)
  library(ggplot2)

  ### Set up your working directory using swd() and get the dataset file.
  df <- read.csv(file = 'rsrc/datasets/wheat_seeds.csv')

  ### Split the dataset to 70% train data and 30% test data.
  set.seed(100)
  train <- sample(nrow(df), 0.7 * nrow(df), replace = FALSE)
  trainData <- df[train, ]
  testData <- df[-train, ]

  ### Normalize the target to be between 0 and 1.
```

```r
  min_max_norm <- function(x) {
    (x - min(x)) / (max(x) - min(x))
  }

  trainData$Type <- min_max_norm(trainData$Type)

  ### Build a Random Forest model.
  model <- randomForest(Type ~ ., data = trainData, mtry = 4, importance = TRUE)

  ### Use the first feature from the dataset and set up
  ### 4 intervals to test the get_bounds function
  bounds <- get_bounds(df, 1, 4)

  ### Test the calculate_ale function, with centered = FALSE
  uncentered_ale <- calculate_ale(model, df, 1, 4, FALSE)

  ### Test the prepare_ale function, with centered = TRUE
  prepared_ale <- prepare_ale(model, df, 1, 4, TRUE)

  ggplot(data = prepared_ale, mapping = aes(x = x, y = y)) +
    geom_line() +
    geom_point()
}

ale_plot()
```



Exercise 3: Categorical ALE

a) Overall, all customers, regardless of their personal status and gender, have on average a high probability of being a low (good) risk for the bank. The average marginal prediction for divorced or separated male customers reveals a slightly higher risk for this group.

b) The ALE is faster to compute and unbiased. Unbiasedness means that it does not suffer from the extrapolation problem which is especially apparent in PDPs when features are correlated.

c)

```
order_levels <- function(data, feature.name) {
  feature <- data[ , feature.name]
  others <- setdiff(colnames(data), feature.name)
  feature.lev <- unique(feature)

  ### Iterate over other features
  dists <- lapply(others, function(k) {
    feature.k <- data[ , k]
    if (inherits(feature.k, "factor") | inherits(feature.k, "character")) {
      dists <- get_diff_cat(feature.k, feature)
    } else {
      dists <- get_diff_numeric(feature.k, feature)
    }
    dists
  })
  dists.cumulated.long <- as.data.frame(Reduce(function(d1, d2) {
    d1$dist <- d1$dist + d2$dist
    d1
  }, dists))

  ### Create a matrix of distances
  dists.cumulated <- reshape2::dcast(dists.cumulated.long,
                                     class1 ~ class2, value.var = "dist")[ , -1]
  rownames(dists.cumulated) <- colnames(dists.cumulated)

  ### conduct multi-dimensional scaling (here: principal coordinates analysis)
  ### based on dissimilarity matrix it assigns
  ### to each item a location in a low dimensional space
  ### the closer the location, the more similar the items are.
  scaled <- cmdscale(dists.cumulated, k = 1)
  feature.lev[order(scaled)]
}

get_diff_numeric <- function(feature.k, feature.j) {
  ### set up data.frame which we will fill later
  dists <- expand.grid(unique(feature.j), unique(feature.j))
  colnames(dists) <- c("class1", "class2")

  ### get decentiles
  quants <- quantile(feature.k, probs = seq(0, 1, length.out = 10),
                     na.rm = TRUE, names = FALSE)

  ### derive empirical distribution function for each category
  ecdfs <- data.frame(lapply(unique(feature.j), function(lev) {
    x.ecdf <- ecdf(feature.k[feature.j == lev])(quants)
    return(x.ecdf)
  }))
  colnames(ecdfs) <- unique(feature.j)
```

```r
  ### get pairwise absolute distances of empirical distribution function
  ### between the different categories
  ecdf.dists.all <- abs(ecdfs[ , dists$class1] - ecdfs[ , dists$class2])

  ### get maximum distance over decentiles for each pair of categories
  dists$dist <- apply(ecdf.dists.all, 2, max)
  dists
}

get_diff_cat <- function(feature.k, feature.j) {
  ### set up data.frame which we will fill later
  dists <- expand.grid(unique(feature.j), unique(feature.j))
  colnames(dists) <- c("class1", "class2")

  ### get relative frequency table
  x.count <- as.numeric(table(feature.j))
  A <- table(feature.j, feature.k) / x.count

  ### compute pairwise absolute distances
  dists$dist <- rowSums(
    abs(A[as.character(dists[ , "class1"]), ] - A[as.character(dists[ , "class2"]), ]))
  dists
}

if (FALSE) {
  credit <- read.csv("datasets/credit.csv")

  ### This should already work WITHOUT get_diff_cat()
  credit.sub <- credit[ , c("age", "personal_status_sex")]
  order_levels(credit.sub, "personal_status_sex")
  get_diff_numeric(feature.k = credit.sub[ ,"age"],
                   feature.j = credit.sub[ ,"personal_status_sex"])
  ### to see what the methods does step by step use
  ### debug(order_levels) or debug(get_diff_numeric)

  ### This should work AFTER you have implemented get_diff_cat()
  order_levels(credit, "personal_status_sex")
  get_diff_cat(feature.k = credit[ ,"employment_duration"],
               feature.j = credit[ ,"personal_status_sex"])
}
```

Bonus: ALE and PDP are global interpretation tools which base their insights on averages (of predictions or prediction differences) over whole test sets. Indeed vulnerable groups are typically not the majority of a population but have a low proportion, and biases might be overlooked. Therefore, local explanation tools should be consulted, in addition to these methods in order to identify pointwise biases or discriminatory behavior.