Lab 2

Thomas Haase

September 2, 2025

Table of contents

Part 1: Social Network Ad Purchase

In the first part of this lab, we will again consider the ad *purchase* data that we used in lab 1 (containing information about individuals' purchasing behavior under exposure to online ads). As in lab 1, we will build classification models to predict ad purchase (0/1) based on a set of covariates about individuals (including `Age`, `Gender` and `Salary`). In contrast to lab 1, we here focus on *non-parametric* methods.

1. Begin by importing the file "`Kaggle_Social_Network_Ads.csv`". Format the outcome variable `Purchased` as a factor variable (this is required for the subsequent analysis using the caret package). You may also delete the `user_id` column, as it will not be used.

```
data <- fread("Kaggle_Social_Network_Ads.csv")[, `:=`(
  user_id = NULL,
  Purchased = as.factor(Purchased)
)]
```

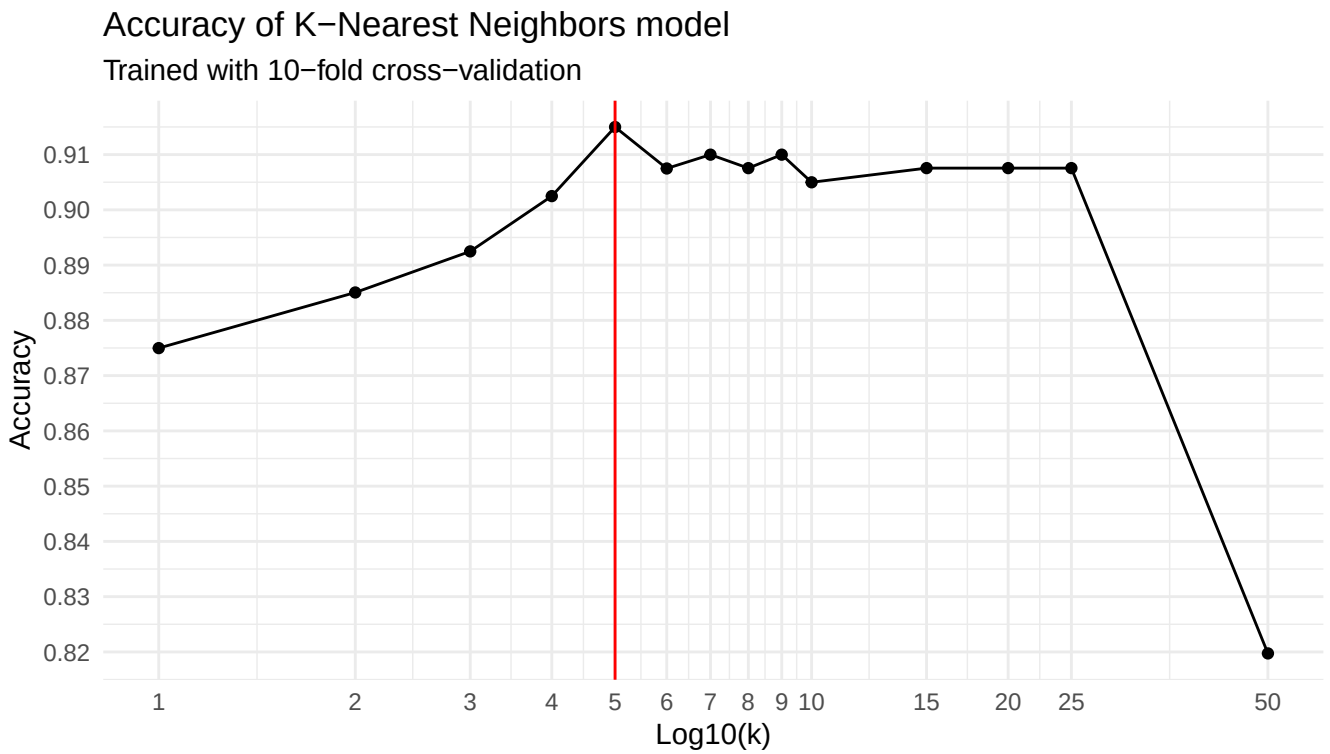
2. The first non-parametric method you will use to predict ad purchase is the *k-nearest neighbors* algorithm. As we talked about in the lecture, it has one key parameter, k, that the researcher must set. Use the `caret` package to implement a 10-fold cross-validation procedure that examines the test accuracy of *kNN* under different choices of k (hint 1: set `method="knn"` to estimate a *kNN* with caret hint 2: use the `tuneGrid` argument to specify your grid of *k* values). Consider the following values of *k*: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50}. Because *kNN* is sensitive to the scale of

variables, we must standardize our variables prior to estimation (hint: you can add the following argument to `train(): preProcess=c("center","scale")`, to do so). From the results, plot the test accuracy against k (hint: results can be extracted from the train object by using `$results`). Interpret this plot. How does the performance—for the best k—compare to that of *GAM* and *standard linear regression* (i.e., your results in lab 1)? Which do you prefer? Why?

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

model_knn <- caret::train(Purchased ~ .,
                          data = data,
                          preProcess = c("center","scale"),
                          tuneGrid = expand.grid(k = c(1:10,seq(15,25,5),50)),
                          method = "knn",
                          trControl = tc)


ggplot(model_knn$results, aes(x = k, y = Accuracy)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = model_knn$results$k[which.max(model_knn$results$Accuracy)],
             color = "red") +
  labs(title = "Accuracy of K-Nearest Neighbors model",
       subtitle = "Trained with 10-fold cross-validation",
       x = "Log10(k)") +
  scale_x_log10(breaks = c(1:10,seq(15,25,5),50)) +
  scale_y_continuous(breaks = seq(0.8,1,0.01)) +
  theme_minimal()
```



The performance of the best predicting KNN model is 91%. Small values of k probably underfit the data and too high values overfit the data. This is as good as the best predicting GAM model with 90,5%. Both the GAM and KNN model have a higher accuracy than the linear model. Just like the GAM the KNN is able to fit nonlinear data really well, which sets them both apart from the linear model.

I prefer the GAM and the KNN over the linear model because of its better fit, and the GAM over the KNN because it offers better interpretability than the KNN.

3. Suppose we now learn that those who collected the data have retrieved some additional information (in the form of digital traces) about these 400 individuals. This extra information consists of 20 variables $X1, X2, ..., X20$. The people providing us with the data suggest that *some* of these variables could be very relevant for predicting `Purchase`, but they also note that many of them likely are irrelevant. The problem is, they do not know which is which. This new data set is stored in the .csv file "`Kaggle_Social_Network_Ads_Augmented.csv`". Import it to R and process it the same way you did in #1.

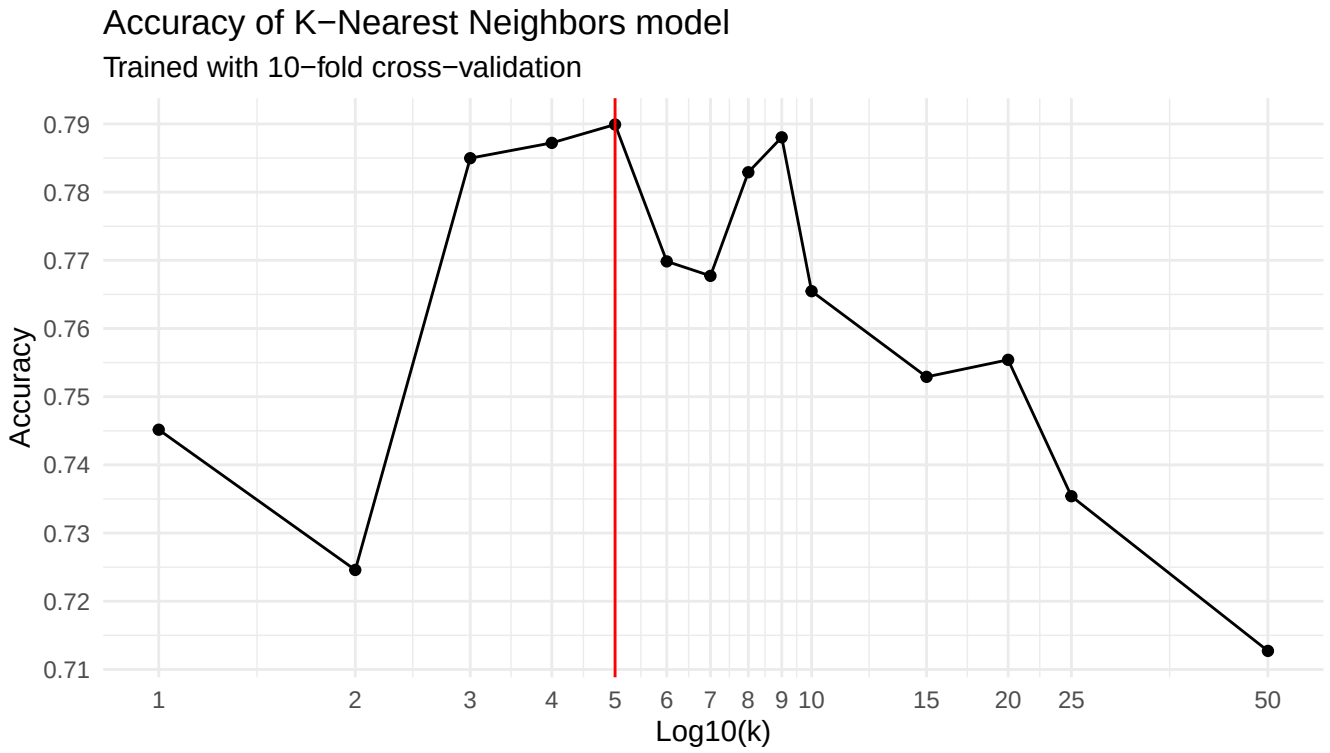
```
data <- fread("Kaggle_Social_Network_Ads_Augmented.csv")[, `:=`(
  user_id = NULL,
  Purchased = as.factor(Purchased)
)]
```

4. Repeat task #2 for this new data set (from step #3). How does the prediction accuracy change? How do you explain this difference (or lack thereof)?

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

model_knn <- caret::train(Purchased ~ .,
                          data = data,
                          preProcess = c("center","scale"),
                          tuneGrid = expand.grid(k = c(1:10,seq(15,25,5),50)),
                          method = "knn",
                          trControl = tc)


ggplot(model_knn$results, aes(x = k, y = Accuracy)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = model_knn$results$k[which.max(model_knn$results$Accuracy)],
             color = "red") +
  labs(title = "Accuracy of K-Nearest Neighbors model",
       subtitle = "Trained with 10-fold cross-validation",
       x = "Log10(k)") +
  scale_x_log10(breaks = c(1:10,seq(15,25,5),50)) +
  scale_y_continuous(breaks = seq(0,1,0.01)) +
  theme_minimal()
```



The prediction accuracy drops quite recognizable to 78%. Since KNN does not weigh the features based on their predictiveness it is sensitive to the inclusion of unimportant features.

5. Motivated by the results obtained in #4, we want to explore an alternative method. Based on the properties—which we discussed in the lecture—of *decision trees*, do you think it has the potential to perform better than *kNN* on the data set from #3? If yes, why?

A decision tree should perform better since it captures novel interactions of the high amount of variables in the dataset. Thereby it could pick up latent constructs hidden in the variables. On the other hand decision trees are not very robust and only a small change in the data could lead to a large change in the tree structure.

6. Estimating decision trees is what you will do now. In R, we can use the package `rpart` for this. Decision trees have two main parameters that a researcher must choose prior to estimation: (*i*) the minimum number of observation in each leaf node (in `rpart: minbucket`), and (*ii*) the complexity parameter α (in `rpart: cp`). In practice, the former is often set heuristically (using some rule-of-thumb) and the latter using cross-validation. Please do the following:

a. Answer why it generally is *not* a good idea to set `minbucket` to either a *very large* number or a very *small number* (relative to the size of your data).

As tree size (complexity) grows, training error always decreases, but test error first improves (↓ bias) and then worsens (↑ variance). A very large `minbucket` value will not allow the tree to reach its optimal size, the tree would be too biased. A very large `minbucket` value would lead to a tree with too high variance, basically overfitting the data.

b. Use the caret package (`method="rpart"`) to implement a 10-fold cross-validation procedure to assess how the test accuracy changes as a function of the complexity parameter (ISL: α; `rpart: cp`). Hint: use the `tuneGrid` argument to specify your grid of *cp* values. Consider the following values for *cp*: {0, 0.01, 0.025, 0.05, 0.10, 0.25, 0.5, 1.0}. (note: `minbucket` is here set to a default value=10). Plot the *test accuracy* against *cp* (hint: extract results as you did in #2 and #3). Interpret this plot. How does the accuracy for the best *cp* compare to *kNN*? What do you attribute this difference to?

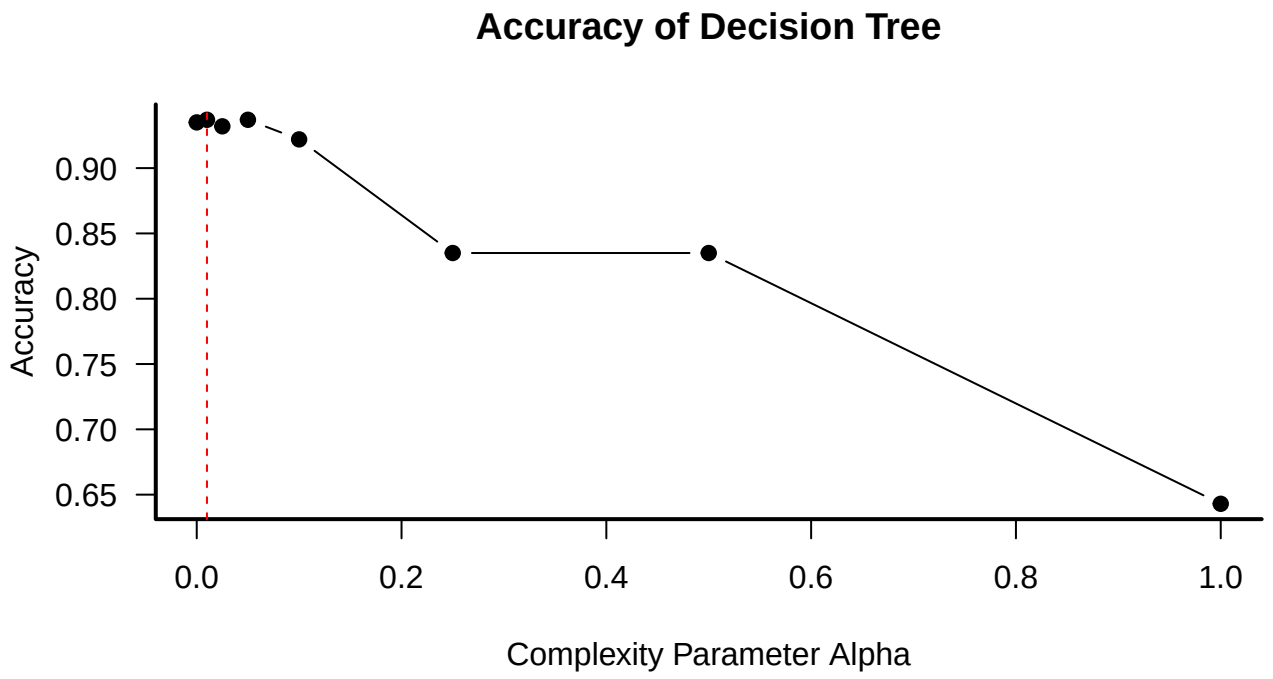
```
set.seed(12345)
model_tree <- caret::train(Purchased ~ .,
                           data = data,
                           tuneGrid = expand.grid(cp = c(0, 0.01, 0.025, 0.05,
                                                         0.10, 0.25, 0.5, 1)),
                           control = rpart.control(minbucket = 10),
                           #preProcess = "scale",
                           method = "rpart",
                           trControl = tc)

model_tree$results[,c("cp", "Accuracy", "AccuracySD")] |> round(3) |> kable()
```

cp	Accuracy	AccuracySD
0.000	0.935	0.032
0.010	0.937	0.034
0.025	0.932	0.034
0.050	0.937	0.036
0.100	0.922	0.043

cp	Accuracy	AccuracySD
0.250	0.835	0.066
0.500	0.835	0.066
1.000	0.643	0.009

```
model_tree$results[,c("cp", "Accuracy")] |>
  round(3) |>
  plot(type = "b", bty = "n", pch = 20, cex = 1.5, las = 1,
       xlab = "Complexity Parameter Alpha",
       main = "Accuracy of Decision Tree")
  box("plot", bty = "l", lwd = 2)
  abline(v = model_tree$results$cp[which.max(model_tree$results$Accuracy)], col = "red", lty = 2)
```



A higher complexity parameter α is overall related to a lower accuracy. Mathematically α is controlling the punishment of the amount of splits made/terminal leafs existing:

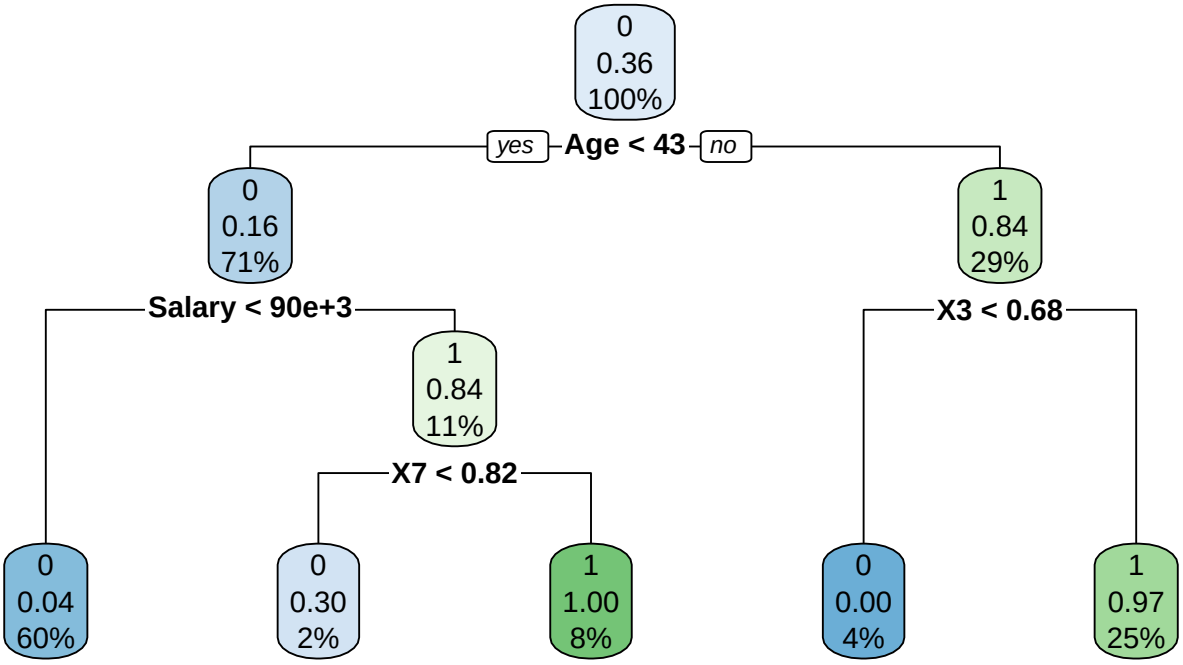
$$\underbrace{\sum_{j}^{J}\sum_{i\in R_j}(y_i-\bar{y}_{R_j})^2}_{\text{Summed error across leaves}} + \underbrace{\alpha * |T_0|}_{\text{Penalty}}$$

This punishment is called "tree pruning". The effectiveness of the punishment is displayed by the plot: the more the tree is punished the less it can accuratly learn the underlying relations of the data.

The accuracy of the best predicting tree is 0.94. It is much higher than the accuracy of the KNN Model. The KNN is sensitive to inclusion of unimportant features, but the tree is indirectly ranking variables for making the best predictive splits. Additionally this allows the tree to pick up inductivly on important interactions. The large amount of variables is fantastic for that.

c. Use the `rpart.plot` package to plot the decision tree you found to be the best in b (hint: you can extract the model with the highest accuracy from a `caret` model using `$finalModel`). First report how many *internal* and *terminal* nodes this tree has. Then, provide a interpretation. Does any of the "new" variables ($X_1, ..., X_20$) show up in the tree?

```
rpart.plot(model_tree$finalModel)
```



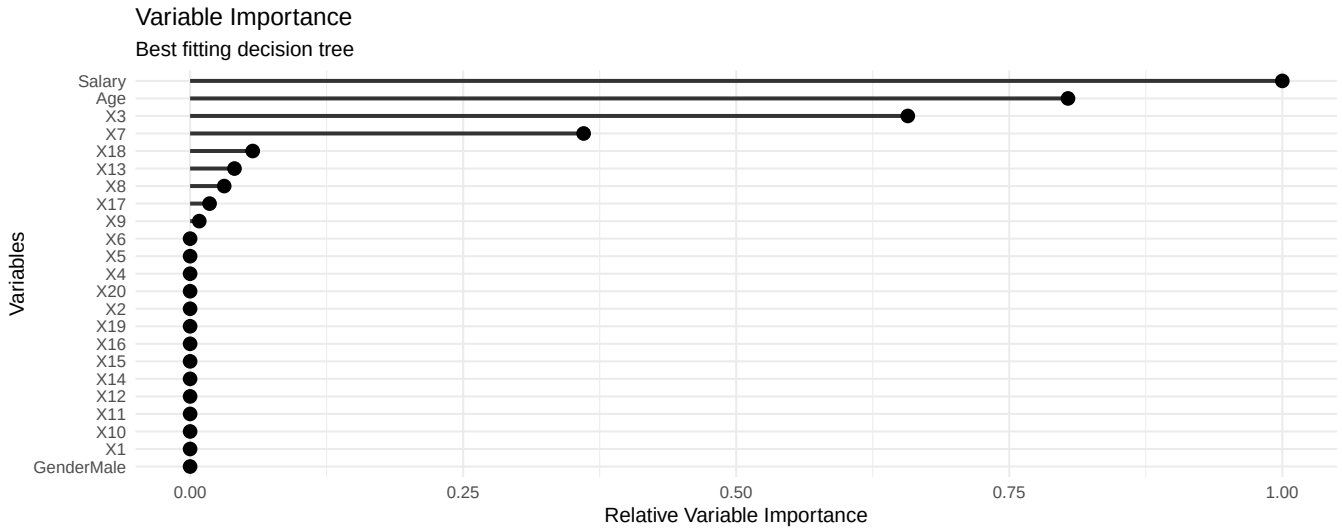
The tree has 4 internal leafs and 5 terminal leafs. Age is the strongest first split. Younger persons (< 43) are less likely to purchase (especially if salary is lower). Older persons ($ 43) tend to purchase, unless X3 is very small. Among the younger group the salary is the main decider: If the salary is greater or equal 90,000 the chance of purchasing increases. Within this subgroup, high values of X7 (0.82) almost guarantee class 1. New variables are definitly showing up at the bottom of the tree.

d. Use *caret'svarImp*() function to calculate *variable importance* scores. Interpret.

```
varimp <- data.table(var = rownames(caret::varImp(model_tree$finalModel)),
           imp = caret::varImp(model_tree$finalModel)$Overall)[order(imp,decreasing = T)]

varimp[,imp := imp / max(imp)] |>
ggplot(aes(y=reorder(var, imp),x=imp)) +
  geom_segment(aes(x = 0, xend = imp,
                   y = var, yend = var),
               color = "#333333",
               linewidth = 1) +
  geom_point(size=3) +
  labs(title = "Variable Importance",
       subtitle = "Best fitting decision tree",
       x = "Relative Variable Importance",
```

```
       y = "Variables") +
  theme_minimal()
```



The variable importance scores are calculated by tabulating the reduction in the loss function (e.g. mean squared error) attributed to each variable at each split and the sum is returned. The prediction if someone is a customer is mostly driven by 4 variables: Salary, age, X3 and X7. The most important variable for predicting if a person is a customer is their salary. After that Age and X3 play an important role aswell. lastly X7 aswell as X18, X17 and X13 play a role in the prediction too, most likely through interaction effects.

Part 2: Bernie Sanders and Donald Trump tweets (cont.)

In the second part of this lab, we will reconsider the *Bernie vs. Trump* twitter data set that we used in lab 1. As in lab 1, we shall build classification models to predict who authored a given tweet based on its linguistic content, and to identify which words are the most discriminative between Bernie and Trump.

1. Import the file "`trumpbernie2.csv`" to R (note: to reduce computational time, I have reduced the number of columns for this lab), and format the outcome variable trump_tweet as a factor variable (again, this is needed for the `caret` package to recognize it as a *discrete* variable.)

```
rm(list = ls())

data <- fread("trumpbernie2.csv")[, `:=`(
  trump_tweet = as.factor(trump_tweet)
)]
```

Use caret to implement a 5-fold cross-validation procedure that estimates the *test error* for a *decision tree* with different `cp` (ISL : α) values. Consider the following `cp` values: {0, 0.001, 0.01, 0.1, 0.25}. Report the accuracy of the best configuration. How does this compare to the performance of a standard logistic regression (LR) and a ridge regression (RR)? You do not need to estimate the LR and the RR yourself

— I provide their performance here: Accuracy$(LR) = 55\%$. Accuracy$(RR) = 86\%$. Why do think this is the case? Doing the following two things might help you answer this question:

```
set.seed(2025)
model_tree <- caret::train(trump_tweet ~ .,
                           data = data,
                           method = "rpart",
                           tuneGrid  = expand.grid(cp = c(0, 0.001, 0.01, 0.1, 0.25)),
                           control   = rpart.control(minbucket = 10),
                           trControl = trainControl(method = "cv", number = 5))

model_tree$results[,c("cp", "Accuracy", "AccuracySD")] |> round(3) |> kable()
```

cp	Accuracy	AccuracySD
0.000	0.772	0.011
0.001	0.773	0.010
0.010	0.753	0.021
0.100	0.637	0.030
0.250	0.503	0.001

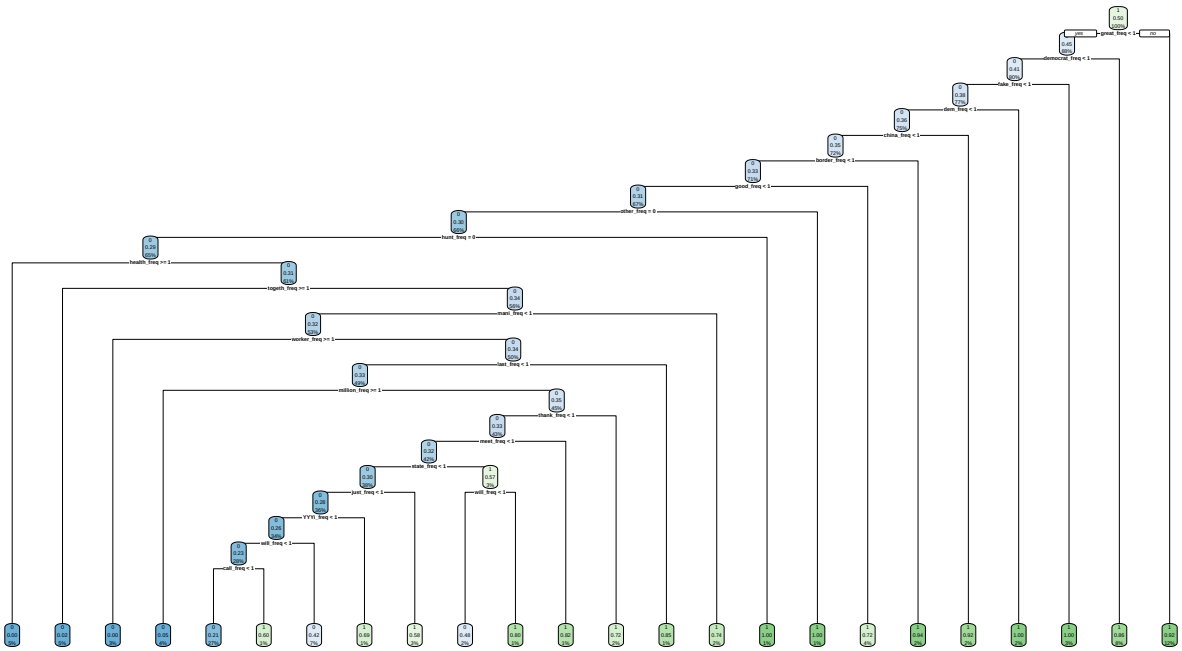
The best configuration reached an accuracy of 0.773. This is a higher accuracy than the standard logistic regression model reached, but less than the ridge regression.

The dataset is high dimensional. High dimensional data includes a few important signal variables and a high number of noise variables. Since decision trees use recursive binary splitting to grow, they can not make a distinction between variables that carry the signal and which are not affecting the data in a meaningful way. The result is an overfit decisiontree, that is highly variable does not perform well outside of the training dataset.

Ridgeregression uses regularization to tune the noise in the data down. This way it can focus the signal and reach a better accuracy

a. Plot the tree which achieved the highest test accuracy (hint: again, you can extract the best model using `$finalModel`, and plot it using `rpart.plot`). Is it a big or a small tree?

```
rpart.plot(model_tree$finalModel)
```

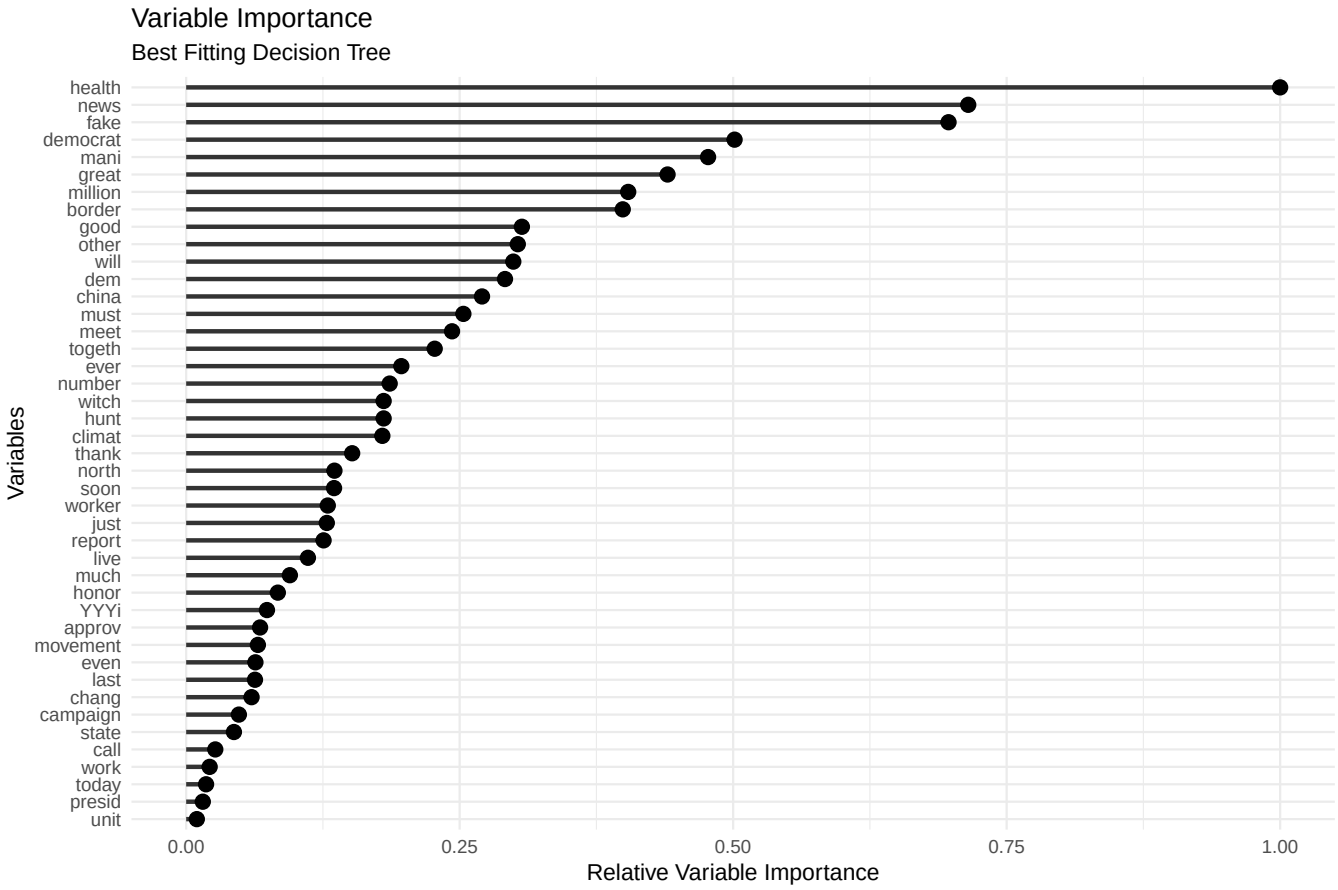


The tree is very big (good for environment).

b. Calculate variable importance. How many variables have non-zero values?

```
varimp <- data.table(var = rownames(caret::varImp(model_tree$finalModel)),
                     imp = caret::varImp(model_tree$finalModel)$Overall)[imp != 0][order(imp,decr

varimp[,imp := imp / max(imp)][, var := sub("_freq$", "", var)] |>
ggplot(aes(y=reorder(var, imp),x=imp)) +
  geom_segment(aes(x = 0, xend = imp,
                   y = var, yend = var),
               color = "#333333",
               linewidth = 1) +
  geom_point(size=3) +
  labs(title = "Variable Importance",
       subtitle = "Best Fitting Decision Tree",
       x = "Relative Variable Importance",
       y = "Variables") +
  theme_minimal()
```



43 Variables/Terms have non-zero values.

3. Given the results in #2, we want to explore other alternative methods. Using once again the `caret` package, you shall now implement a 5-fold cross-validation procedure that fits a random forest model(`method="rf"`) with different `mtry` values. `mtry` controls the number of variables that are considered at each split (corresponding to parameter `m` in ISL). Specifically, you shall consider the following values for `mtry`: {1, 5, 10, 25, 50, 200} (hint: again, you specify this grid using the argument tuneGrid). Another important parameter of random forest models is the number of trees to use. Here you shall use 200 (hint: you may enter ntree=200 as an argument in train() to do so). The estimation may take a couple of minutes to finish. Once it has finished, please do the following:

```
if(!file.exists("model_forest.rds")){
  set.seed(2025)
  model_forest <- caret::train(trump_tweet ~ .,
                               data = data,
                               method = "rf",
                               ntree = 200,
                               tuneGrid  = expand.grid(mtry = c(1, 5, 10, 25, 50, 200)),
                               trControl = trainControl(method = "cv", number = 5))
  saveRDS(model_forest, file = "model_forest.rds")
} else {
  model_forest <- readRDS("model_forest.rds")
```

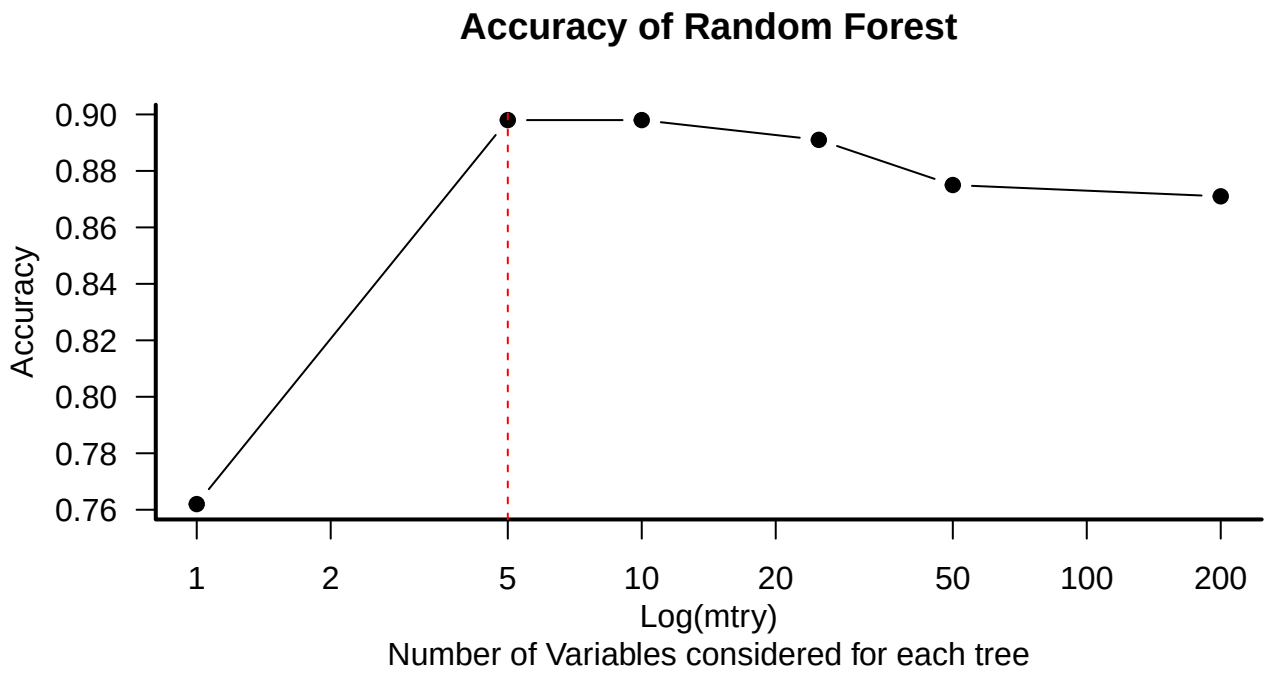
```
}


model_forest$results[,c("mtry", "Accuracy", "AccuracySD")] |> round(3) |> kable()
```

mtry	Accuracy	AccuracySD
1	0.762	0.064
5	0.898	0.033
10	0.898	0.022
25	0.891	0.027
50	0.875	0.028
200	0.871	0.025

a. Plot accuracy against mtry. Interpret this plot. Does decorrelating the trees seem to help performance?

```
model_forest$results[,c("mtry", "Accuracy")] |>
  round(3) |>
  plot(type = "b", bty = "n", pch = 20, cex = 1.5, las = 1,
       log = "x",
       xlab = "Log(mtry)\nNumber of Variables considered for each tree",
       main = "Accuracy of Random Forest")
  box("plot", bty = "l", lwd = 2)
  abline(v = model_forest$results$mtry[which.max(model_forest$results$Accuracy)], col = "red", lt
```

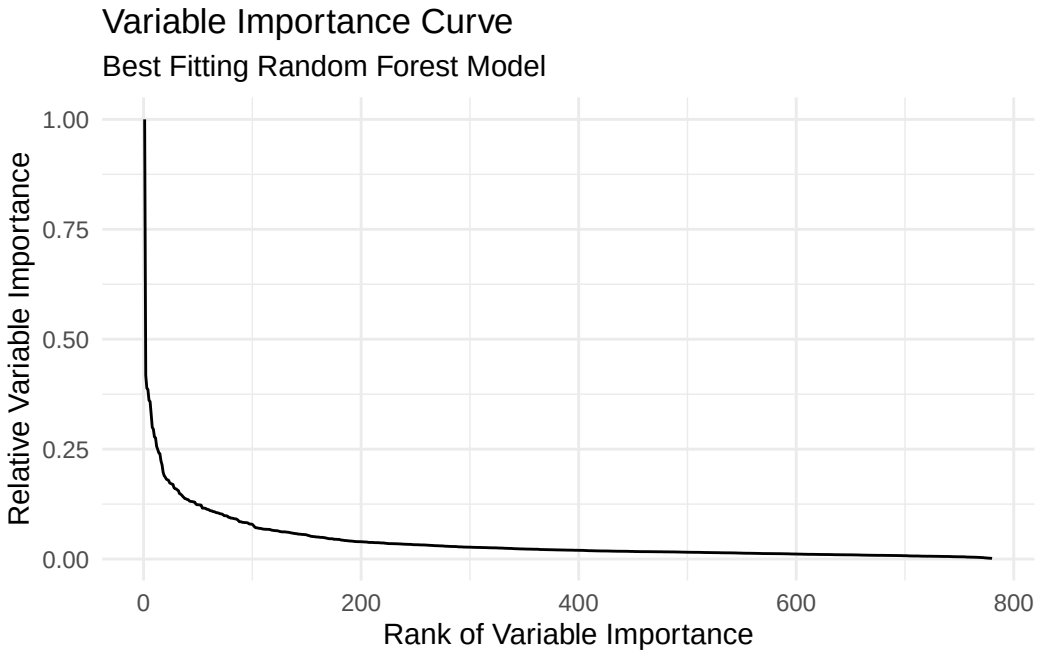


Random forest is an improvement of bagged trees. Bagged trees is a method where multiple trees are fit, each one to a bootstrapped version of the original dataset. In the end the mean of the trees is used as $\hat{y}$. Each tree is deep (unpruned) and the method tries to reduce variance of a single decisiontree by averaging multiple. Sadly the trees are still highly correlated, because they use the same variables and fall victim to the same noise of high dimensional data. Random forest improves this by limiting the amount of variables used to fit each tree. This decorrelates the trees by prohibiting the most important variables to dominate most trees. Imposing these restrictions weakens the individual trees, but strengthens the pooling. Limiting the number of considered variables for each split to 5 leads to the highest accuracy. Only fitting stumps still reaches an accuracy of 0.76. Stumps are not able to capture interactions like medium and taller trees can. Therefore increasing mtry to 5 yields in such a higher improvement of the accuracy. From there the more variables are considered, the more noise finds its way into the model leading it to overfit.

b. Relate the performance of the best random forest model to the best decision tree model. Do you find a meaningful improvement using random forest? Why do you think that is (or isn't)? Examining the variable importance scores may provide some clues; how many non-zero variables do you find for the random forest?

The best predicting random forest model had an accuracy of $\sim 90\%$, while the best predicting decision tree had an accuracy of $\sim 77\%$.

```
data.table(var = rownames(caret::varImp(model_forest$finalModel)),
           imp = caret::varImp(model_forest$finalModel)$Overall)[, imp := imp / max(imp)][order(-
  ggplot(aes(x = rank, y = imp)) +
  geom_line(linewidth = 0.5) +
  labs(title = "Variable Importance Curve",
       subtitle = "Best Fitting Random Forest Model",
       x = "Rank of Variable Importance",
       y = "Relative Variable Importance") +
  theme_minimal()
```



When investigating the variables closer no variable is exactly weighted with 0, but most of the variables are weighted with less than 10%. The random forest is averaging out the noise (decorrelating) by taking the mean of all trees. The noise variables still have a little bit of weight, but are less influential in mass.

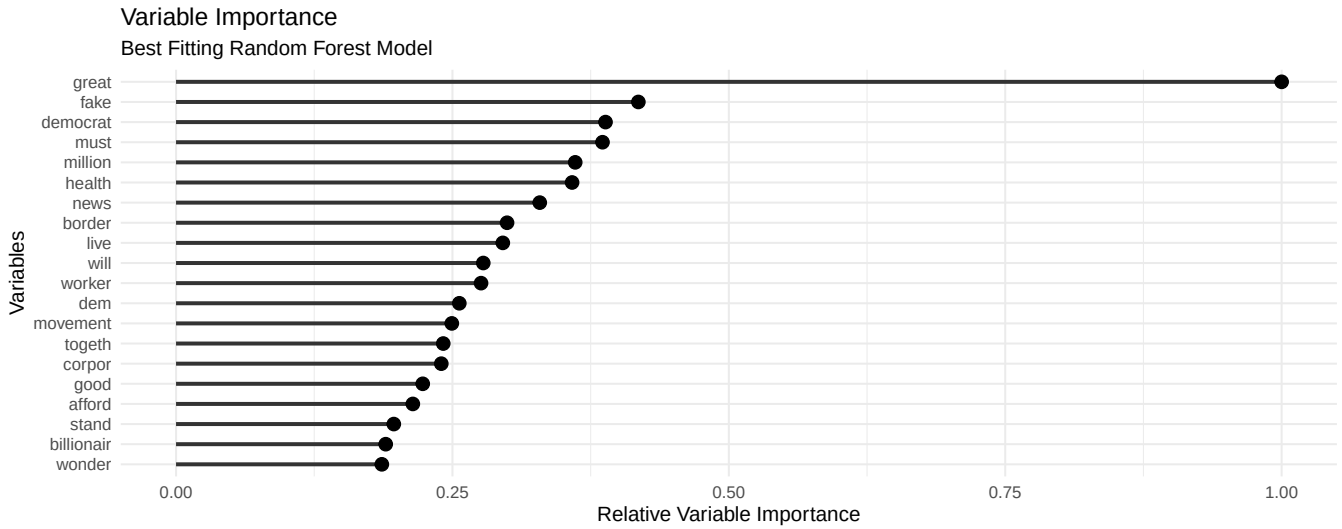
c. Relate the performance of the best random forest model and the ridge regression from #2. What do you think this difference indicates about the data, and more specifically the relation ship between X and y?

The best random forest model had an accuracy of $\sim 90\%$, while the best ridge regression had $\sim 86\%$ accuracy. Ridge regression is also able to focus the signal in high dimensional data, but it can not capture complex interactions of nonlinear data like the random forest model can. This seems to give the random forest model an edge when predicting.

d. Consider the variable importance scores in a bit more detail: does the variables that appear in the top 20 align with your expectations? Note that variable importance does not reveal the direction; e.g., whether a certain word is associated with Trump or with Bernie. For this, we can use the randomForest's function plotPartial() which produces a partial dependence plot for a variable of choice. Select three variables (words) which you find interesting, and create three separate partial dependency plot. In order to interpret the y-axis of the plot, use the argument which.class to specify which class ("0" or "1") should represent the positive class.

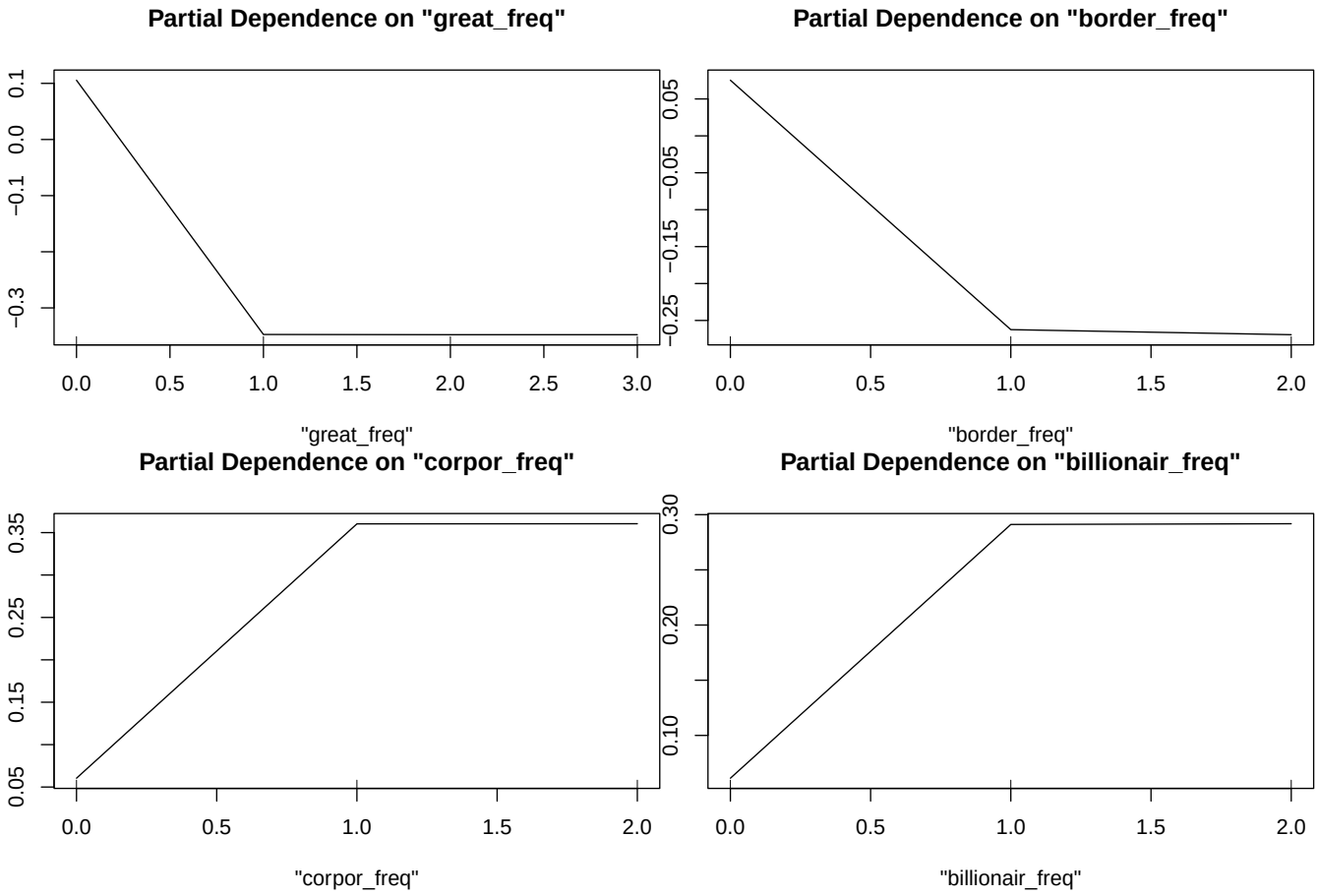
```
varimp <- data.table(var = rownames(caret::varImp(model_forest$finalModel)),
           imp = caret::varImp(model_forest$finalModel)$Overall)[order(imp,decreasing = T)]

varimp[,imp := imp / max(imp)][1:20][, var := sub("_freq$", "", var)]|>
ggplot(aes(y=reorder(var, imp),x=imp)) +
  geom_segment(aes(x = 0, xend = imp,
                   y = var, yend = var),
               color = "#333333",
               linewidth = 1) +
  geom_point(size=3) +
  labs(title = "Variable Importance",
       subtitle = "Best Fitting Random Forest Model",
       x = "Relative Variable Importance",
       y = "Variables") +
  theme_minimal()
```




```
randomForest::partialPlot(model_forest$finalModel, which.class = "0",
                          pred.data=data, x.var="great_freq")
randomForest::partialPlot(model_forest$finalModel, which.class = "0",
                          pred.data=data, x.var="border_freq")
randomForest::partialPlot(model_forest$finalModel, which.class = "0",
                          pred.data=data, x.var="corpor_freq")
randomForest::partialPlot(model_forest$finalModel, which.class = "0",
                          pred.data=data, x.var="billionair_freq")
```

The partial dependency plots display piecewise linear functions. When the word reaches appearence of `1.0` is the prediction platos at a binary level. For example once the word "great" appears once the predicted value reaches -0.3. The partial dependency plot shows that the predicted value are positive or negative. This way it is possible to see if an occuring word leads to an increase or decrease in predicted probability - one stands for bernie, one for trump. We can see that trump likes to use wordstems like "great" and "border", while bernie likes to use wordstems like "corpor" and "billionair".


Partial Dependence on "great_freq"
−0.3
−0.1
0.0
0.1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
"great_freq"
Partial Dependence on "border_freq"
−0.25
−0.15
−0.05
0.05
0.0
0.5
1.0
1.5
2.0
"border_freq"
Partial Dependence on "corpor_freq"
0.05
0.15
0.25
0.35
0.0
0.5
1.0
1.5
2.0
"corpor_freq"
Partial Dependence on "billionair_freq"
0.10
0.20
0.30
0.0
0.5
1.0
1.5
2.0
"billionair_freq"

Quiz Wrap Up

Provided here are three optional wrap-up questions that help recap some central ideas from the lecture, and which are taken from old exams.

1. For which of the following supervised learning methods does standardizing your input data (X) influence the predictions?

a. Decision tree

- no, optimal cut are about variance which stay the same in relation

b. Lasso

- yes, because its standard OLS which is trivial

c. kNN

- no, samme argument as decision tree

d. Standard OLS

- yes, trivial

2. Which among the following are important aspects distinguishing parametric and non-parametric supervised learning models?

a. Non-parametric methods make strong assumptions about the functional form of models; para- metric methods do not.

- Wrong

b. Parametric methods make strong assumptions about the functional form of models; non- parametric methods do not.

- True

c. Parametric methods are generally more appropriate for inference tasks compared to non- parametric methods.

- True

d. Non-parametric methods are generally more appropriate for inference tasks compared to para- metric methods.

- Wrong

3. In terms of the bias-variance decomposition, a decision tree with 20 leaves tends to have _____ than a decision tree with 3 leaves.

a. higher variance and lower bias.

- true

b. higher variance and higher bias.

- wrong

c. lower variance and lower bias.

- wrong

d. lower variance and higher bias.

- wrong

EXTRA STUFF - mlr3 boosting

```
# remotes::install_github('catboost/catboost', subdir = 'catboost/R-package')
# pak::pak("mlr-org/mlr3extralearners")
library(mlr3)
library(mlr3extralearners)
library(mlr3tuning)
library(mlr3hyperband)
library(catboost) # catboost performs well with categories more: https://aravindkolli.medium.com/

# NOTES ON HYPERPARAMETER OPTIMIZATION
# hyperband for quick and dirty, TunerMBO (bayesian model estimated to suggest new search paramet


tsk = as_task_classif(data[, (names(data)[sapply(data, is.integer)]) :=
                            lapply(.SD, as.numeric),
                          .SDcols = names(data)[sapply(data, is.integer)]],
                      target = "trump_tweet")

lrn = lrn("classif.catboost", logging_level = "Silent")

rsmp = rsmp("cv", folds = 5)


search_space = ps(
  iterations     = p_int(lower = 100, upper = 1000, tags = "budget"),
  learning_rate  = p_dbl(lower = 0.01, upper = 0.3),
  depth          = p_int(lower = 3, upper = 10)
)


terminator = trm("none")

instance = TuningInstanceSingleCrit$new(
  task           = tsk,
  learner        = lrn,
  resampling     = rsmp,
  measure        = msr("classif.acc"),
  search_space   = search_space,
  terminator     = terminator
)

tuner_hyperband$optimize(instance)

instance$result  # what is the best configuration
# 0.8893881 accuracy -> just slightly worse than the random forest
```

```
tuner_hyperband = tnr("hyperband", eta = 3) # eta specifies how strong the multi armed bandit red
```