Lab 2 - Machine Learning for Social Science

To be handed in no later than September 3rd. The submission should include code, relevant output, as well as answers to questions. We recommend the use of RMarkdown to create the report.

Part 1: Social Network Ad Purchase (cont.)

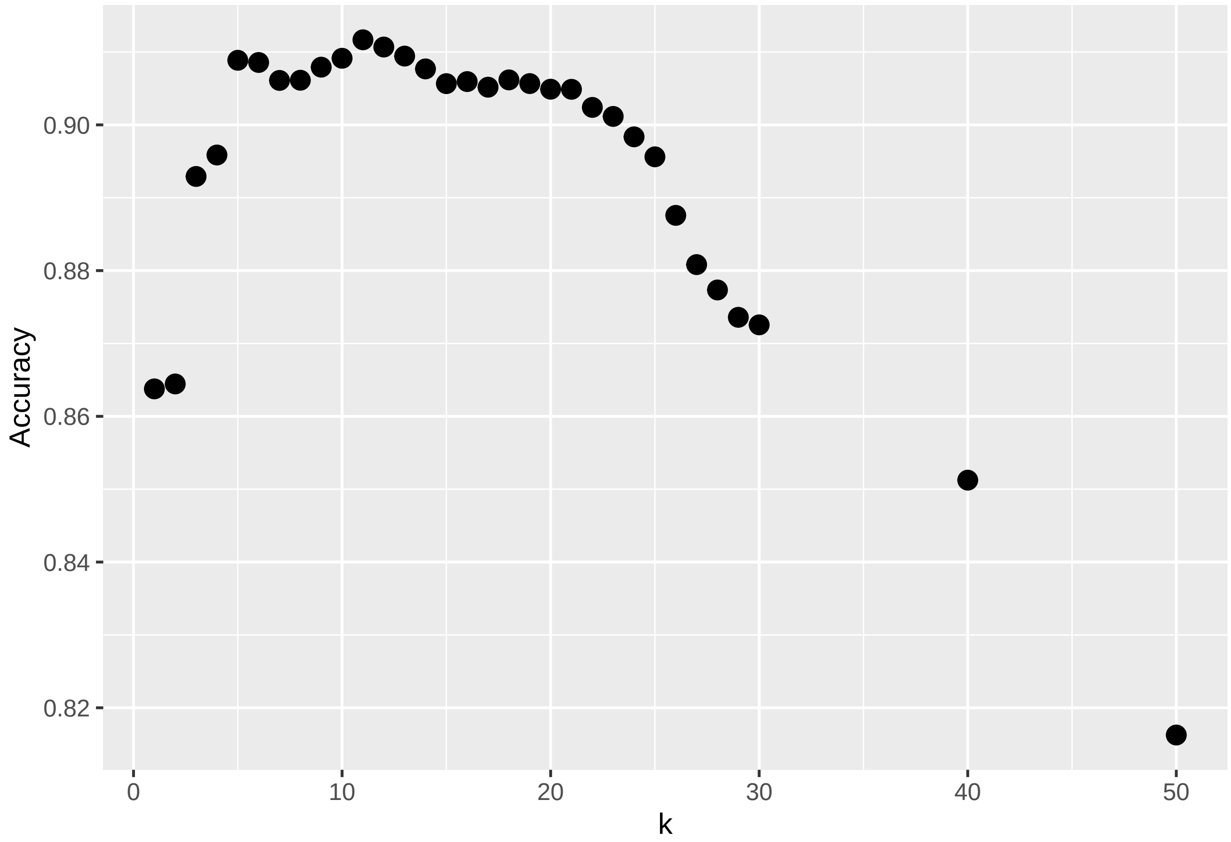
In the first part of this lab, we will again consider the *ad purchase* data that we used in lab 1 (containing information about individuals' purchasing behavior under exposure to online ads). As in lab 1, we will build classification models to predict ad purchase (0/1) based on a set of covariates about 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.

```
dt <- fread(input = '/Users/marar08/Documents/Teaching/MLSS_HT2025/Labs/W2/data/Kaggle_Social_Network_Ad
dt[,user_id := NULL]
dt[,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?

```
# Either of the below are OK
# The first repeats the CV 10 times
# And thus provide more stable results
tc <- trainControl(method="repeatedcv",
                   repeats = 10,
                   number = 10)
# tc <- trainControl(method = 'cv',
#                    number = 10)
set.seed(1)
knn_model <- train(Purchased ~ .,
                   data = dt,
                   method = "knn",
                   preProcess = c("center","scale"),
                   tuneGrid = expand.grid(k = c(1:30,40,50)),
                   trControl = tc)
ggplot(knn_model$results, aes(x=k,y=Accuracy)) +
  geom_point(size=3)
```



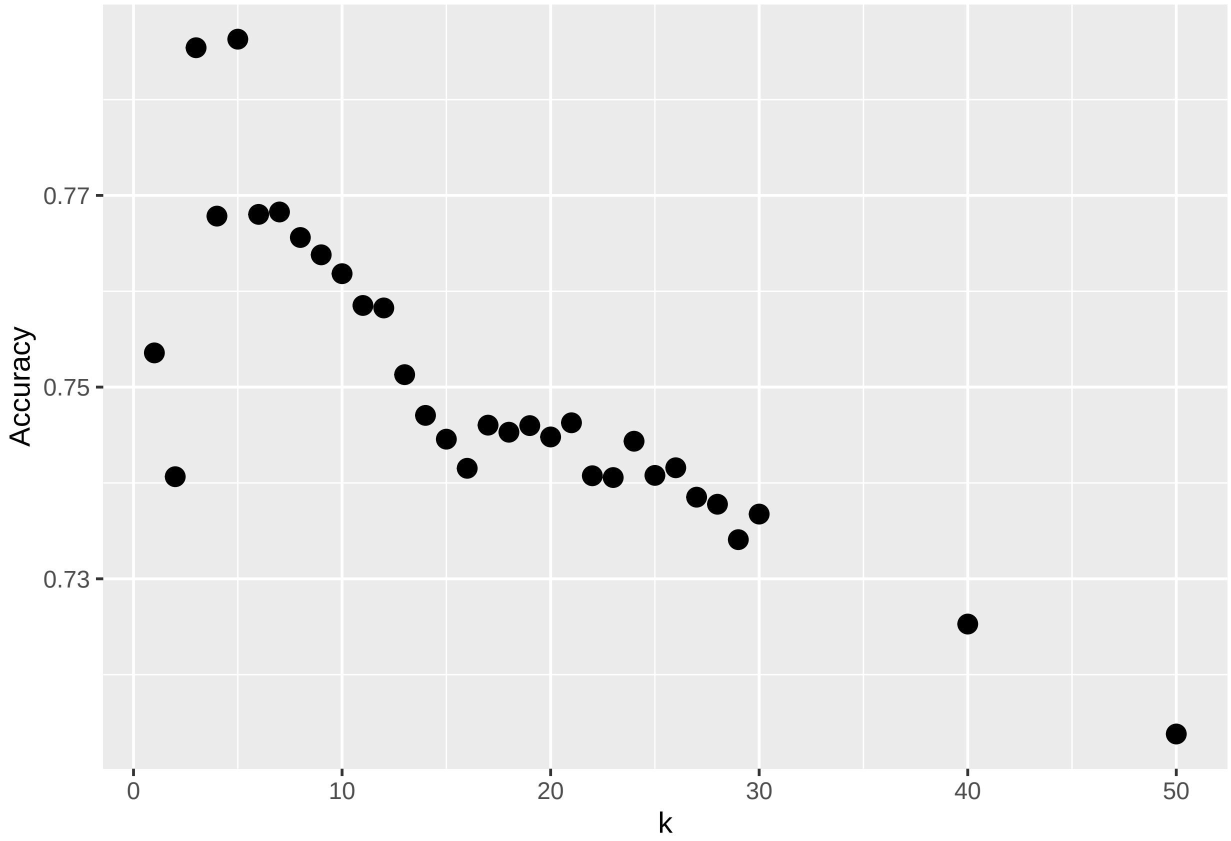
The highest test accuracy is obtained in the region $k \in [5, 20]$. Making k smaller than this, the model becomes too flexible and overfits the data. Conversely, as we increase k, predictions are pooled over increasingly distant observations, the flexibility is decreased, and we underfit the data. Relating the $k \in [5, 20]$ *NN* models to the best *GAMs* of the previous lab, we see a similar performance (accuracy $\sim 90\%$). While they show similar prediction performance, the *GAM* offer a high degree of interpretability as well—which, for (most) social scientific purposes is important. Therefore, I would generally go with the *GAM*.

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 $X_1, X_2, \ldots, X_{20}$. 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.

```
# Import and process
dt2 <- fread('/Users/marar08/Documents/Teaching/MLSS_HT2025/Labs/W2/data/Kaggle_Social_Network_Ads_Augme
dt2[,Purchased := as.factor(Purchased)]
dt2[,user_id := NULL]
```

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
# Fit kNN on augmented
set.seed(1)
knn_model <- train(Purchased ~ .,
                   data = dt2,
                   method = "knn",
                   preProcess = c("center","scale"),
                   tuneGrid = expand.grid(k = c(1:30,40,50)),
                   trControl = tc)
ggplot(knn_model$results, aes(x=k,y=Accuracy)) +
  geom_point(size=3)
```



The general pattern of the plot remains roughly the same; but now with an overall lower test accuracy. This difference can be explained by *kNN* giving "equal weight" to each dimension; it identifies the closest neighbour by considering all dimensions. That together with the fact that a majority of the added variables were noisy and not useful for predicting the outcome at hand.

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?

Yes. As we discussed in the lecture, decision trees classify data through a recursive procedure split procedure that—at every step—seeks the variable & split that maximizes the reduction in error (maximizes homogeneity in leaves). This importantly implies that only a *subset of variables* is used to partition the data space; the subset which is found to be useful. In such a procedure, we would expect the noisy/independent variables to be unlikely to be selected as a variable to be split upon (unless the tree is grown very large).

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).

   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?

   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, \ldots, X_{20}$) show up in the tree?

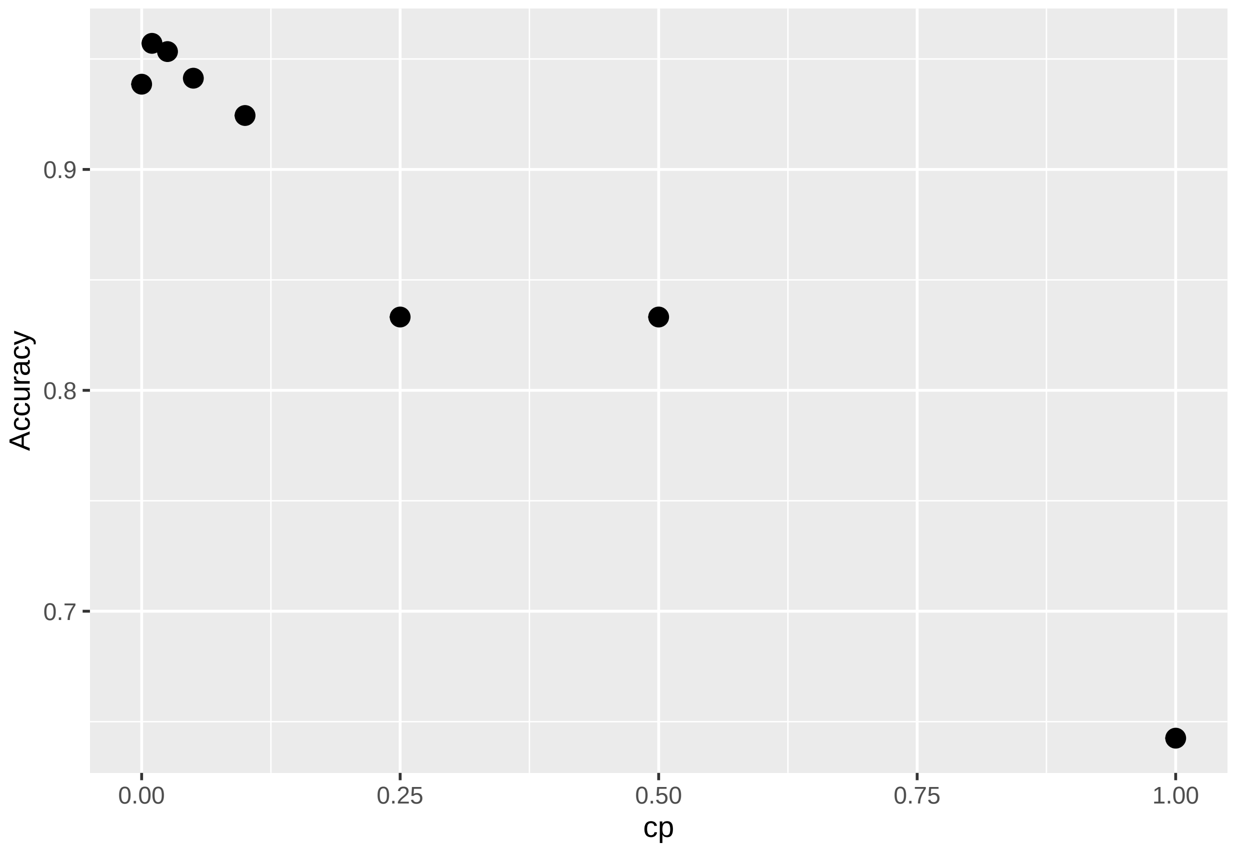
   d. Use `caret`'s `varImp()` function to calculate *variable importance* scores. Interpret.

Regaring a—when we use a very *large minbucket*, this inhibits the model to make any (or just few) splits, which in turn is likely to make the decision tree underfit the data. Conversely, a very *small "minbucket"* allows the tree to grow very large, such that—without pruning—it is likely to overfit the data.

Regaring b—we observe a similar type of pattern as we did for *kNN*. The highest accuracy in the cp $\in [0.01, 0.025]$ region. Increasing *cp* beyond this increases the bias more than it reduces the variane (underfitting). Conversely, decreasing *cp* we retain the original full tree, which seems to slightly overfit the data. The performance is substantially improved compared to *kNN*. This can be attributed to the fact that whereas *kNN* use all variables, *decision trees* only use a subset; those that improve predictions.

```r
set.seed(1)
tree_model <- train(Purchased ~ .,
                    data = dt2,
                    method = "rpart",
                    tuneGrid = expand.grid(cp=c(0,0.01,0.025,0.05,0.10,0.25,0.5,1)),
                    control = rpart.control(minbucket = 2, minsplit=4),
                    trControl = tc)

ggplot(tree_model$results, aes(x=cp,y=Accuracy)) +
  geom_point(size=3)
```

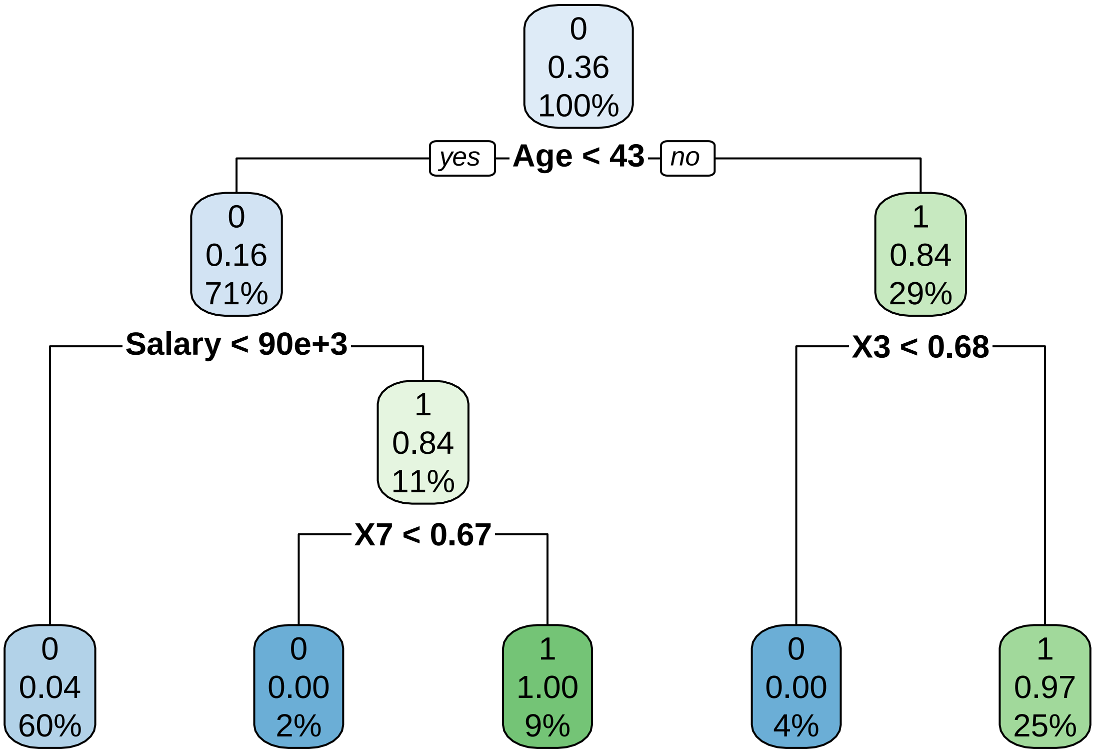


```
tree_model$bestTune
```

```
##     cp
## 2 0.01
```

Regarding c—the retrieved tree has four *internal* nodes and five *terminal* nodes. The tree first shows—via its first split—that, for individuals with an age ≥ 43, a large majority (84%) purchase the item, while below age 43, a large majority (84%) do not. Second, zooming in (going further down) the right-branch of the tree, we see that among those age ≥ 43 **and** have an $X_3 \geq 0.68$, 97% purchase the item, while 100% of those below this X_3 value do not. A similar type of interpretation can be made for the left-hand side of the tree. Overall, the plot shows that the decision tree is able to identify a subset of digital trace variables that *are* relevant (X_3,X_7), while ignoring the rest of them.

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



Regarding d—we find that *Salary* is the variable which—either as split variable or as surrogate split variable—reduced the error the most. Among the original variables, *Salary* and *Age* are found to be by far the most important (*Gender* has a score of 0). Additionally, we see the two digital trace variables that appeared in the tree (X_3,X_7) to also be ranked high. After the first four variables, there is a major drop in reduction of error.

```
varimp <- varImp(tree_model$finalModel)
varimp <- data.table(var = rownames(varimp),varimp)
varimp <- varimp[,Overall := Overall / max(varimp$Overall)]
varimp <- varimp[order(Overall,decreasing = T),]
print(varimp)
```

```
##            var    Overall
##         <char>      <num>
##  1:     Salary 1.00000000
##  2:        Age 0.80627311
##  3:         X3 0.68847206
##  4:         X7 0.40668840
##  5:        X18 0.05818060
##  6:         X2 0.02844318
##  7:         X5 0.02844318
##  8:        X15 0.02821830
##  9:        X19 0.02760122
## 10:        X13 0.02494862
## 11:        X17 0.01808483
## 12:        X10 0.01616313
## 13: GenderMale 0.00000000
## 14:         X1 0.00000000
## 15:         X4 0.00000000
## 16:         X6 0.00000000
## 17:         X8 0.00000000
## 18:         X9 0.00000000
## 19:        X11 0.00000000
## 20:        X12 0.00000000
## 21:        X14 0.00000000
## 22:        X16 0.00000000
## 23:        X20 0.00000000
##            var    Overall
```

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.)

```
dt3 <- fread(input = '/Users/marar08/Documents/Teaching/MLSS_HT2025/Labs/W2/data/trumpbernie2.csv')
dt3[,trump_tweet := factor(trump_tweet,levels = c(0,1))]
```

2. 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:

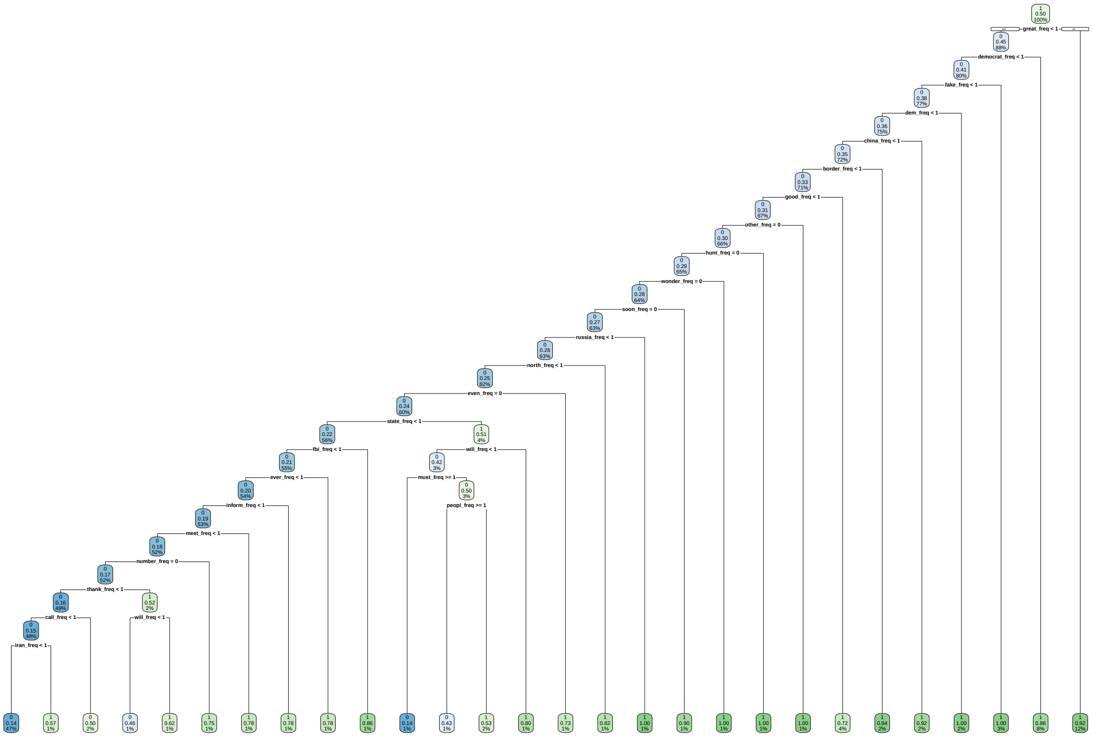
    a. Plot the tree which achieved the highest test accuracy (hint: again, you can extract the best model using `$finalModel`, and plot it using `rpart.plot`). Is it a big or a small tree?

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

```
control <- trainControl(method='cv',number=5)
set.seed(1)
tree2 <- caret::train(trump_tweet ~ .,
                      data = dt3,
                      trControl = control,
                      method = 'rpart',
                      tuneGrid=expand.grid(cp=c(0,0.001,0.01,0.05,0.1)))
# Results
tree2$results
```

```
##      cp  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.000 0.7716567 0.5435611 0.01677852 0.03352256
## 2 0.001 0.7716567 0.5435796 0.01677852 0.03351498
## 3 0.010 0.7686617 0.5380751 0.01657314 0.03277739
## 4 0.050 0.6879453 0.3782090 0.03081055 0.06130127
## 5 0.100 0.6051940 0.2142841 0.01633678 0.03262702
```

```
# (a)
rpart.plot(tree2$finalModel)
```



```
# (b)
varimp <- varImp(tree2$finalModel)
varimp <- data.table(var = rownames(varimp),varimp)
varimp <- varimp[,Overall := Overall / max(varimp$Overall)]
varimp <- varimp[order(Overall,decreasing = T),]
cat(paste0('Number of 0-scores:', nrow(varimp[Overall==0])))
```

```
## Number of 0-scores:735
```

The best accuracy is obtained when there virtually is no penalty applied (cp → 0). This provides an accuracy of ≈ 77%. On the one hand, this is much better than the considerably overfitted *standard logistic regression*. To a first approximation, this can be attributed to the fact that the decision tree only used a small subset out of all variables (55/780). On the other, it does not do as well as the *ridge regression*. This is likely explained by the flip side of the point made a sentence ago; that the great majority (735/780) of the variables were left unused by the decision tree. Yet, the tree is rather large, and is therefore still likely to have high variance. In other words, decision trees—unlike ridge—do not have an effective way of utilizing many variables without getting too complex and overfit. Ridge does this naturally by assuming a linear and additive relationship.

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:

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

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?

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

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.

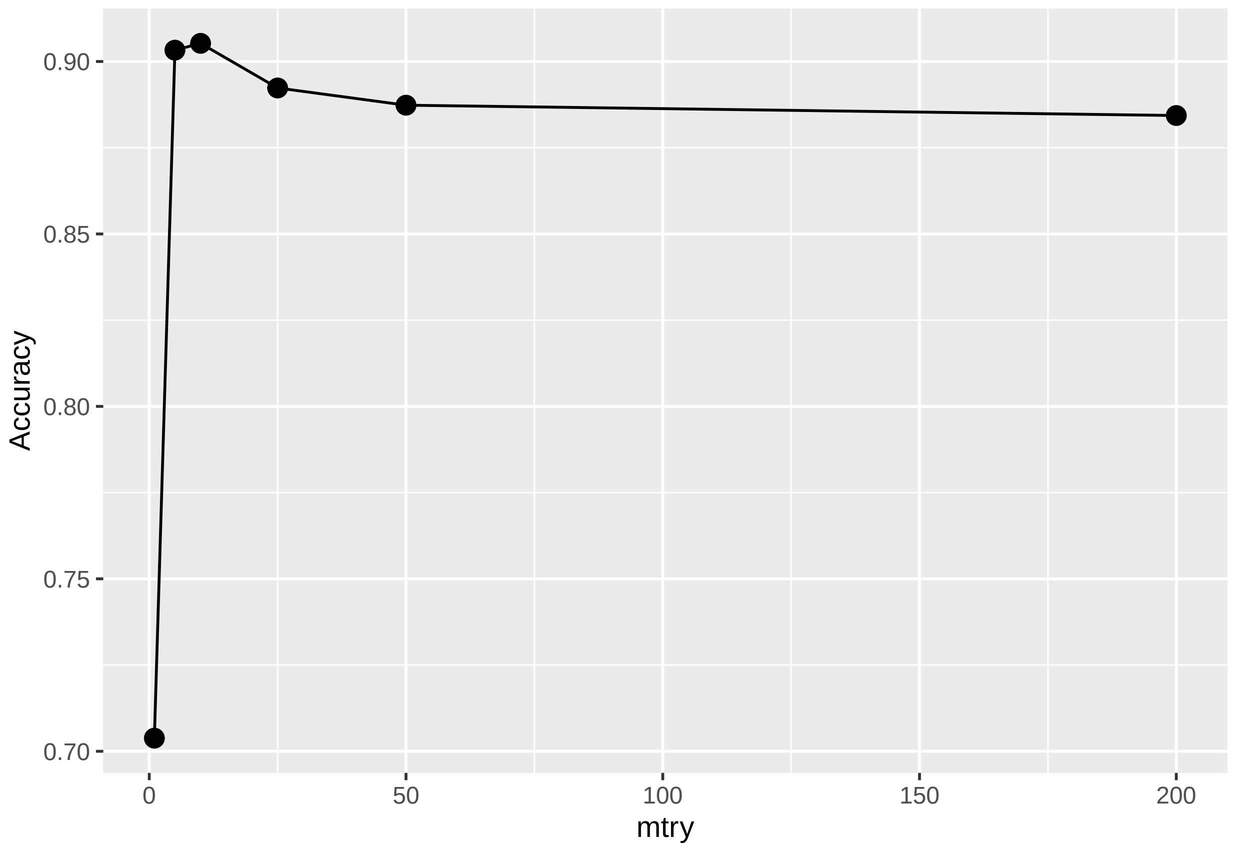
Regarding a—decorrelating the trees does indeed help. A moderate *mtry* ($[5, 10]$) provides the best prediction performance. If we set *mtry* "too high", the model starts to converge to a *bagging approach* (in which trees tend to be relatively more correlated with each other, and gains from pooling predictions between trees are reduced). If we set *mtry* "too low", on the other hand, then the model becomes too restricted, and underfits the data.

```r
# (a)
control <- trainControl(method='cv', number=5)
set.seed(1)
my_rf <- caret::train(trump_tweet ~.,
                      data=dt3,
                      method = "rf",
                      tuneGrid = expand.grid(mtry=c(1,5,10,25,50,200)),
                      ntree=200,
                      trControl = control)
```

```r
# Print results
my_rf$results
```

```
##   mtry  Accuracy     Kappa AccuracySD    KappaSD
## 1    1 0.7038010 0.4102300 0.04919054 0.09683562
## 2    5 0.9032886 0.8066884 0.01036920 0.02073839
## 3   10 0.9052637 0.8106205 0.02449124 0.04893474
## 4   25 0.8923234 0.7847444 0.01089310 0.02177879
## 5   50 0.8873284 0.7747983 0.02076551 0.04150362
## 6  200 0.8843383 0.7687882 0.02016346 0.04033800
```

```r
# (a)
ggplot(my_rf$results, aes(x=mtry,y=Accuracy)) + geom_point(size=3) + geom_line()
```



```
# (b)
varimp <- varImp(my_rf$finalModel)
varimp <- data.table(var = rownames(varimp),varimp)
varimp <- varimp[,Overall := Overall / max(varimp$Overall)]
varimp <- varimp[order(Overall,decreasing = T),]
cat(paste0('Number of 0-scores:', nrow(varimp[Overall==0])))
```

```
## Number of 0-scores:0
```

Regarding b—the random forest does indeed provide a substantial improvement compared to the standard decision tree. As already noted above; decision trees have to be grown very large in order for them to utilize many variables. This usually leads to overfitting, and thus the algorithm only uses a smaller size (at the cost not being able to model some of the patterns in the data). Comparing the variable importance scores, the (single) decision tree did not use a majority of the variables. With random forests, by contrast, all have > 0 values. Which happens as a result of random forests growing very differnt trees, with different variables included, and then pooling their predictions.

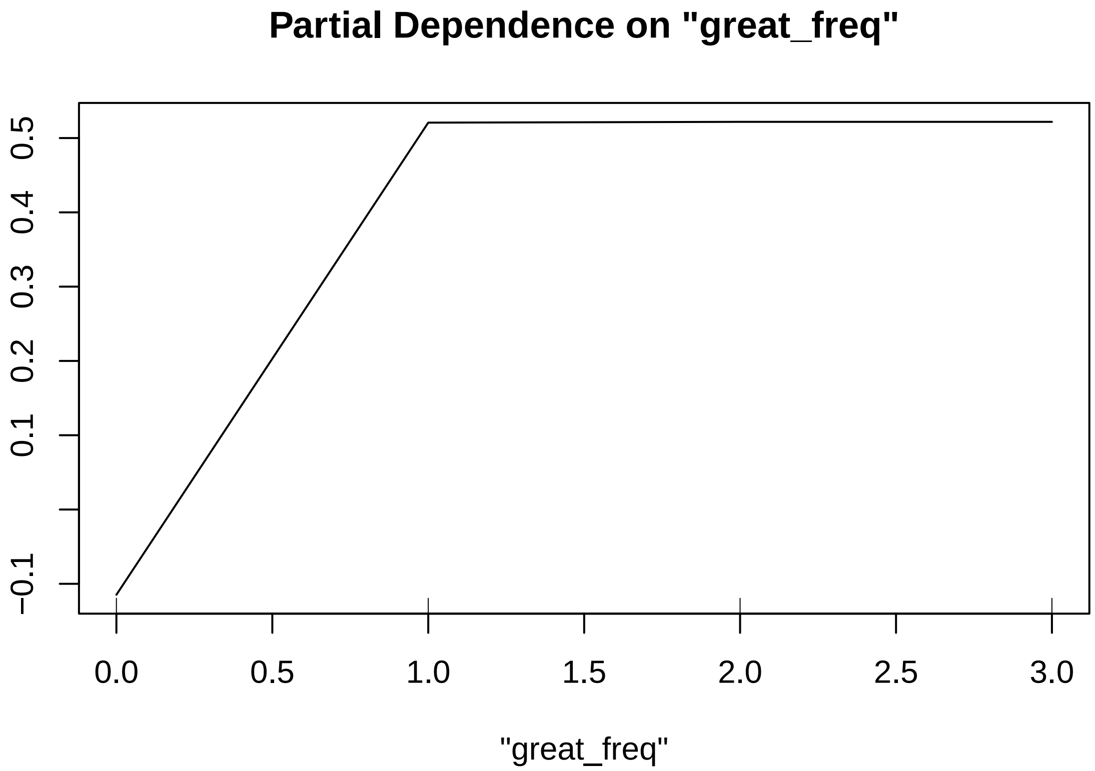
Regarding c—here I notice that I added the wrong abbreviation. The intended comparison was between *random forest* (RF) and *ridge regression* (RR). Sorry about that. So: what about the difference we observe between RF and RR? Both utilize pretty much all variables. What differs between them is that RR assumes linearity and additivity, while RF does not. So, when RF outperforms ridge, this is suggestive of the relationship not being fully additive and linear; but containing some more complex relationships.

```
# Print 20 most important variables for random forest
print(varimp[1:20,])
```

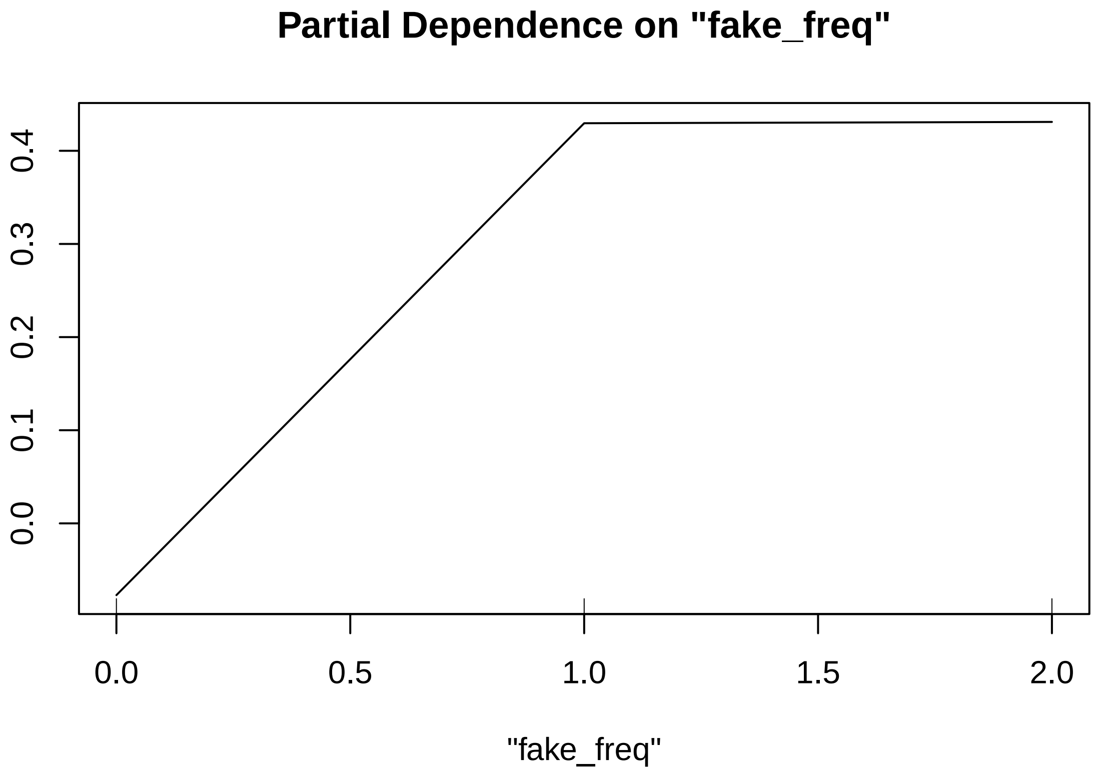
```
##                 var   Overall
##              <char>     <num>
##  1:      great_freq 1.0000000
##  2:   democrat_freq 0.3846509
##  3:       must_freq 0.3692877
##  4:       fake_freq 0.3385410
##  5:       news_freq 0.3021878
##  6:     health_freq 0.2954552
##  7:     border_freq 0.2725088
##  8:       will_freq 0.2576549
##  9:    million_freq 0.2526332
## 10:       live_freq 0.2472230
## 11:   movement_freq 0.2456402
## 12:        dem_freq 0.2401753
## 13:     worker_freq 0.2073440
## 14:      stand_freq 0.1997543
## 15: billionair_freq 0.1918473
## 16:     togeth_freq 0.1873690
## 17:       just_freq 0.1806720
## 18:       good_freq 0.1722828
## 19:     corpor_freq 0.1611651
## 20:       join_freq 0.1609100
##                 var   Overall
```

Regarding d—yes, the top 20 does indeed contain words that I had expected be words that distinguish between Bernie Sanders and Donald Trump. For example "fake", "news", and "border" on the side of Trump, and "health", "worker", and "corporate" on the side of Bernie. Note that the importance scores does not reveal the directionality, however. To confirm, we use “‘partialPlot()“‘. And they do indeed confirm the abovestated expectations.

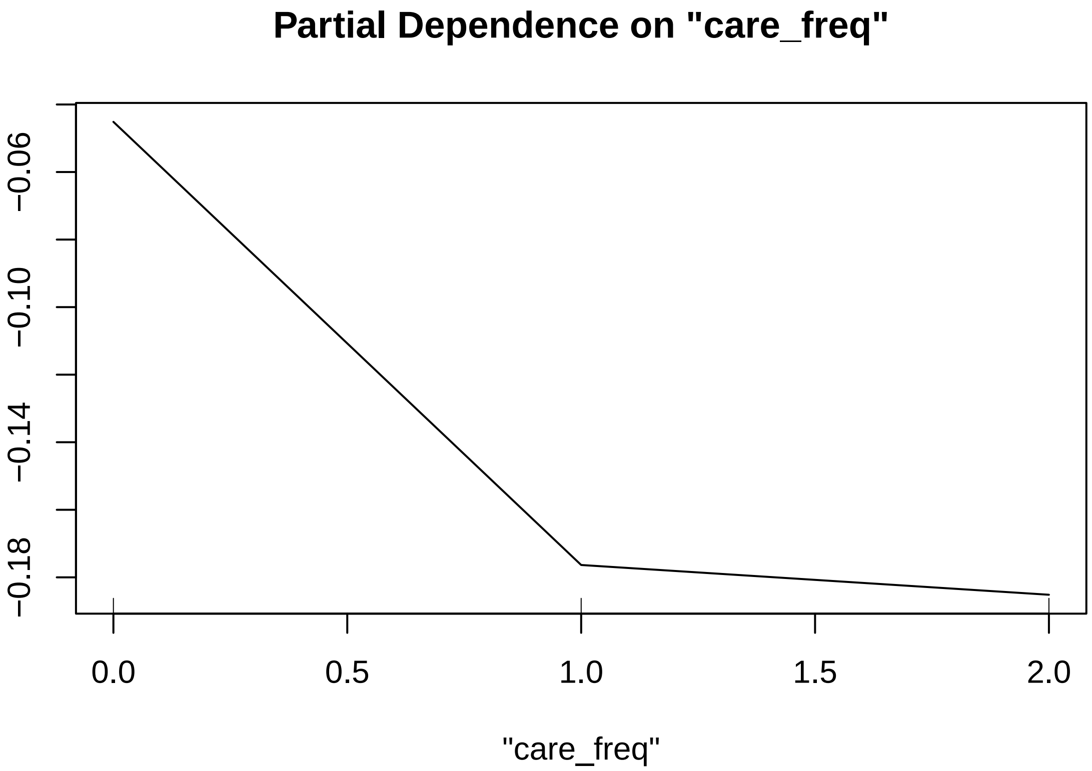
```
# Confirm directions
# =======
# Note 1: by setting which.class="1",
# positive dependence implies increased
# probability of Trump, and vice versa.
# =======
# Note 2: we do n care too much about
# the difference between 1 and above
# on the x-axis. What is important and
# revealing is the 0 --> 1 difference.
# That is, being used in a document or not.
# =======
# - Expected Trump words
randomForest::partialPlot(x = my_rf$finalModel,
                          pred.data=dt3,
                          x.var = "great_freq",
                          which.class="1")
```



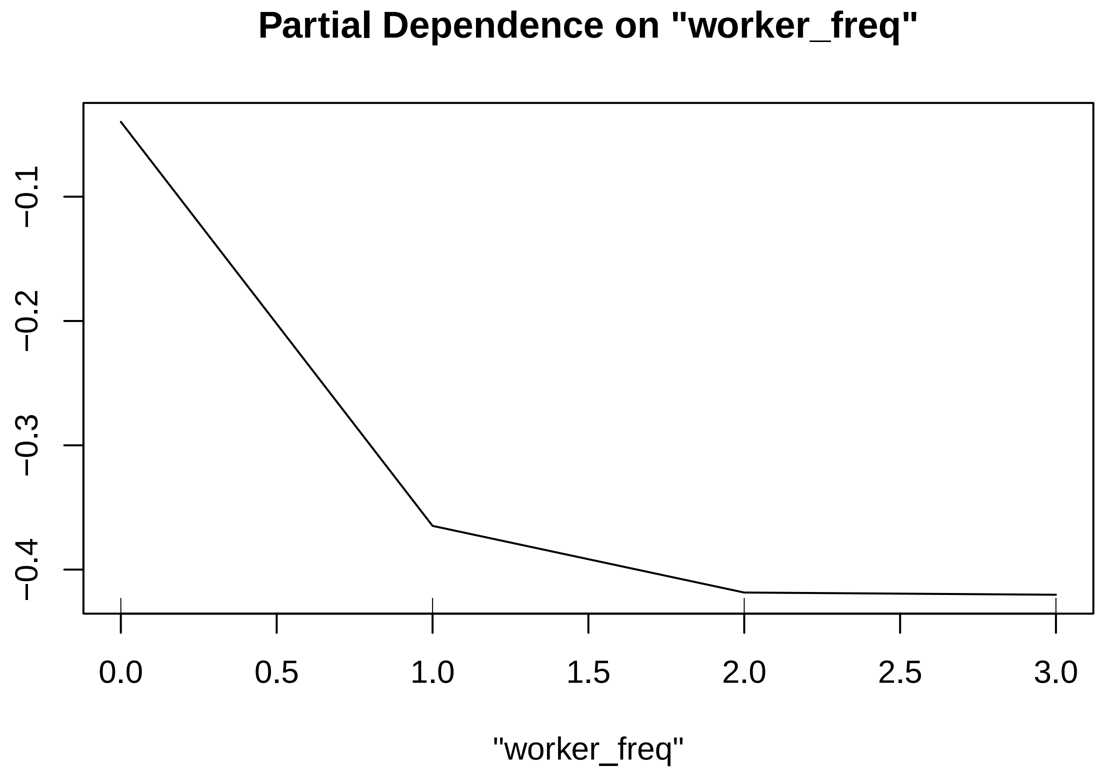
```
randomForest::partialPlot(x = my_rf$finalModel,
                          pred.data=dt3,
                          x.var = "fake_freq",
                          which.class="1")
```



```
# - Expected Bernie words
randomForest::partialPlot(x = my_rf$finalModel,
                          pred.data=dt3,
                          x.var = "care_freq",
                          which.class="1")
```



```r
randomForest::partialPlot(x = my_rf$finalModel,
                          pred.data=dt3,
                          x.var = "worker_freq",
                          which.class="1") # Trump = 1
```


Partial Dependence on "worker_freq"
−0.1
−0.2
−0.3
−0.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
"worker_freq"