Lab 1 – SIRCSS – Machine Learning for Social Science (Solutions)

Part 1: Bernie Sanders and Donald Trump tweets.

For the first part of the lab, you will work with a dataset containing a sample of tweets from Donald Trump and Bernie Sanders. The objective is to explore how accurately we can predict who the author of a given tweet is based on its content, and to identify which words are the most discriminative.

The tweets have been preprocessed & cleaned for you, and are stored in a so-called document-term matrix format with rows indicating tweets, and columns indicating the frequency of words in different tweets.

1. Begin by importing the file "`trumpbernie_2016.csv`" (hint: for example by using `data.table`'s `fread()` function or the standard `read.csv()`). Report how many *rows* and *columns* there are in this dataset (hint: you may for example use `dim()`). Would you characterize this data set as being *high-dimensional* or *low-dimensional*? Based on this, do you expect that a standard logistic regression will work well for the purpose of prediction?

```r
# Load R packages
library(data.table)
library(rpart.plot)
library(ggplot2)
library(caret)

# Import data
trumpbernie <- fread(input = '/Users/marar08/Downloads/sings_ml_lecture/to_upload/lab/trumpbernie_2016.

# Examine dimensionality of data
dim(trumpbernie)
```

```
## [1] 1003 1339
```

As we discussed in the lecture, data sets which a large number of variables relative to the number of rows are considered to be high-dimensional. In this case, as we have more variables than rows, this is certainly high-dimensional. With regards to the appropriateness of logistic regression in this setting: From the lecture, we know that standard linear models a) cannot estimate more than n parameters, and b) that when $p \geq n$, such models will perfectly fit the training data—and thus produce considerable overfitting. Hence, this information leads me to believe that a standard logistic regression model will not produce good predictions here.

2. Estimate a *standard logistic regression model* on the whole data set using the `glm` function (recall to specify `family="binomial"` in `glm`). The outcome variable is `trump_tweet`, and the rest of the columns (the word frequencies) are the input variables. Note: estimating the model may take a couple of minutes. When the estimation of the model is finished, do the following:

    a. Extract the coefficients from the estimated model using the `coef()` function and inspect the coefficients that are placed 1010–1050 in the output from `coef()` (hint: to inspect the coefficients placed 1010–1050 you may use standard brackets, e.g., `coef(mymodel)[1010:1050]`). Do you notice anything special?

b. Examine the *training* accuracy of the estimated model. What does this result suggest about the predictive capacity of the model? You may use the following code to compute the training accuracy:

```
# 2) Estimate standard logistic regression
myglm <- glm(data = trumpbernie,
             trump_tweet ~ .,
             family = 'binomial')
```

```
## Warning: glm.fit: algorithm did not converge
```

```
# a) Extract specified coefficients
print(coef(myglm)[1010:1050])
```

```
##        rig     right      rise      risk      road      role      room roosevelt
##         NA        NA        NA        NA        NA        NA        NA        NA
##       rsvp      rule       run     rural    russia   russian         s      sach
##         NA        NA        NA        NA        NA        NA        NA        NA
##   sacrific       sad      safe     safer    safeti      said    salari sanctuari
##         NA        NA        NA        NA        NA        NA        NA        NA
##     sander     saudi      save       say      scam     scare    schiff   scholar
##         NA        NA        NA        NA        NA        NA        NA        NA
##     school   schumer     scott      seat    second secretari     secur       see
##         NA        NA        NA        NA        NA        NA        NA        NA
##       seem
##         NA
```

All the coefficients numbered 1010–1050 are "'NA"'. This, as remarked upon in #1 is expected as standard linear models can only estimate n parameters. As the variables are ordered by alphabetical order, this becomes a very arbitrary type of variable selection. Not good.

```
# b) Examine accuracy on training data
comparison_df <- data.frame(train_predictions=myglm$fitted.values,
                            observed=myglm$y)
# Apply prediction threshold
comparison_df$train_predictions<-ifelse(comparison_df$train_predictions>=0.5,
                                        yes = 1,
                                        no = 0)
# Compute accuracy (scale: 0-1, 0=0%, 1=100%)
nrow(comparison_df[comparison_df$train_predictions==comparison_df$observed,]) /
  nrow(comparison_df)
```

```
## [1] 1
```

The accuracy on the training set is 100%. Without any other information, this is suspiciously high—and one might reasonably suspect that we are *overfitting*. Knowing that we have fitted a standard linear model to a data set with more columns than rows, we know that we have overfitted the data. In other words, this suggests that predictions from this model are not likely to be good on test data.

3. Use the `caret` package to implement a 3-fold cross-validation procedure that estimates the *test* accuracy of a *standard logistic regression* (hint 1: two functions are relevant here: `trainControl` and `train` |

hint 2: specify `method="glm"` and `family='binomial'` in `train` to fit a standard logistic regression). Note, before you run `train()`, make sure you have formatted the outcome variable `trump_tweet` as a `factor` variable (this is needed for the "caret"'package to recognize the problem as a classification problem). Report the accuracy. Does this result align with your expectations from #1 and #2? Do the results from #2 and #3 provide any indications of either over- or underfitting?

```r
# Load caret package
library(caret)

# Set resamplig settings
tc <- trainControl(method = 'cv', number = 3)

# Format outcome variable as factor
trumpbernie[,trump_tweet := as.factor(trump_tweet)]
```

```r
# Run cross-validation-glm-estimation procedure
system.time(tc_glm <- train(trump_tweet ~ .,
                            method='glm',
                            data = trumpbernie,
                            family='binomial',
                            trControl=tc))
```

```
##    user  system elapsed
##  35.701   0.191  35.903
```

```r
# Extract results
tc_glm$results
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.5154735 0.0308542 0.01734293 0.03483811
```

The test accuracy is ≈ 50%, which is a massive reduction compared to 100%. Indeed, we would do just as well predicting by flipping a coin. In other words, this confirms the expectations from 1–2: our model is considerably overfitted.

4. Now we shall move beyond the standard logistic regression, and more specifically, turn to ridge regression for our prediction task. This importantly entails deciding on a value for the parameter λ. Use `glmnet`'s function `cv.glmnet` to find the λ that minimizes the test error, and report the associated test accuracy. Is this a better or worse prediction model compared to the one in #2–3? Which of the two models do you believe have a higher variance? Why?

   - When specifying `cv.glmnet`'s arguments, note the following:
     i. Unlike `glm` and `train`, the `cv.glmnet` function does not have a `data` argument. Instead, you have to specify `x` and `y` separately (hint: you can for example use `$` to extract and delete columns).
     ii. `alpha` dictates the model type (0=ridge, 1=lasso).
     iii. Finally, set the number of folds to 5 (`nfolds=5`), `family`="binomial" (to indicate that `y` should be treated as a binary variable), and `type.measure="class"` (to retrieve a measure of *accuracy*).
   - OBS: before using ridge regression, we must standardize X. You can do so using R's `scale()` function, e.g., `X <- scale(X)`.

```r
# Load R package
library(glmnet)

# Extract y
y <- trumpbernie$trump_tweet

# Extract X and make into matrix
X <- trumpbernie[,-c('trump_tweet'),with=F]
X <- as.matrix(X)
X <- scale(X)
```

```r
# Perform CV to identify best lambda
myglmnet <- cv.glmnet(x = X,
                      y = y,
                      alpha = 0,
                      nfolds = 5,
                      type.measure="class",
                      family='binomial')
```

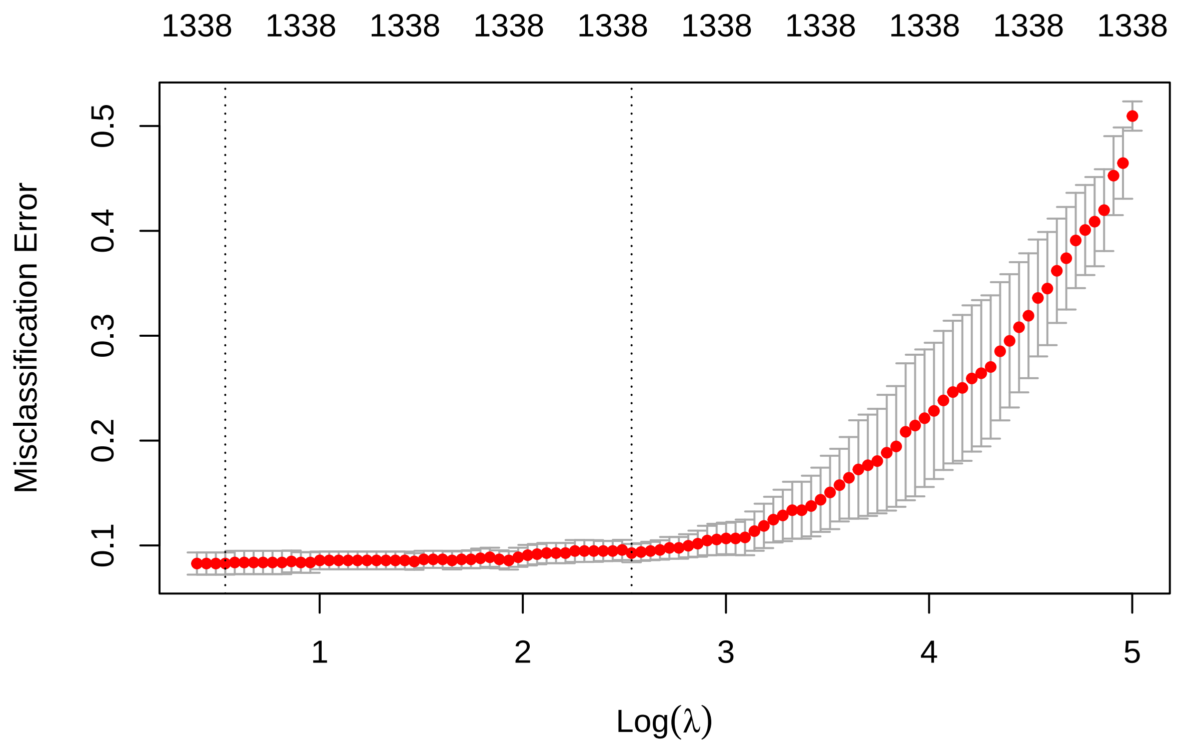
```r
# Extract classification error
print(myglmnet)
```

```
##
## Call:  cv.glmnet(x = X, y = y, type.measure = "class", nfolds = 5, alpha = 0,      family = "binomia
##
## Measure: Misclassification Error
##
##     Lambda Index Measure      SE Nonzero
## min  1.708    97 0.08275 0.01057    1338
## 1se 12.624    54 0.09272 0.00871    1338
```

This is indeed a better model than the one estimated in 1–2. This is because what we care about is the test error/accuracy: ridge ≈ 92% test accuracy compared to ≈ 50% for the standard logistic regression. The latter model is the one with the highest variance. The difference in accuracy on training data and test data for the standard logistic regression model is dramatic (100% vs. 50%).

5. Plot lambda against the classification error (hint: just `plot()` the object which you stored the output from `cv.glmnet`). Interpret the plot in terms of *bias* and *variance*. Note that the x-axis is plotted on a log scale; that explains why you *may* see negative values (negative logged values correspond to very small values on the original scale).

```r
# Plot classification error against lambda
plot(myglmnet, sign.lambda = 1)
```



The misclassification error is minimized when $\lambda \approx 1.6$ (note: $log(1.6) \sim 0.5$). This is the λ which balances the trade-off between bias and variance the best. If we increase the value of λ more (going to the right in this plot), the bias is increased more than the variance is reduced, and hence our overall test error is degraded. Conversely, if we reduce λ (going left in the plot), the variance increases more than the bias is reduced. Although note that (a) the range considered to the left is very small, and (b) the change in error is minimal.

6. Lastly, extract the coefficients associated with the lowest test error (hint: you can use `coef(myfit, s='lambda.min'` for this). Have a closer look at the coefficients with the largest *positive* and largest *negative* values. What do they reveal? Do the words you find on either side confine to your expectations?

```
# Extract coefficients associated with lowest test error
mycoefs <- coef(myglmnet, s = 'lambda.1se')
mycoefs_dt <- data.table(var=rownames(mycoefs),
                         coef=mycoefs[,1])
mycoefs_dt <- mycoefs_dt[order(coef,decreasing = T)]
# Print most positive coefs
print(head(mycoefs_dt,15))
```

```
##              var        coef
##           <char>       <num>
##  1: (Intercept) 0.014205337
##  2:       great 0.010697241
##  3:        news 0.007533270
##  4:        fake 0.007326290
##   5:    democrat 0.007201619
##   6:      border 0.006821191
##   7:         dem 0.006073030
##   8:        hunt 0.005459825
##   9:       witch 0.005439237
##  10:       china 0.005250204
##  11:       media 0.005227164
##  12:        good 0.005216147
##  13:        will 0.005042647
##  14:        mani 0.005031643
##  15:        just 0.004993576
```

```r
# Print most negative coefs
print(tail(mycoefs_dt,15))
```

```
##             var         coef
##          <char>        <num>
##   1:       wage -0.005189707
##   2:     climat -0.005228738
##   3:     defeat -0.005438814
##   4:     afford -0.005556927
##   5:      stand -0.005592485
##   6:     corpor -0.005877671
##   7:     togeth -0.005892902
##   8: billionair -0.005911233
##   9:       join -0.006034622
##  10:    million -0.006048422
##  11:     worker -0.006064288
##  12:       live -0.006548339
##  13:   movement -0.006719843
##  14:       must -0.006925683
##  15:     health -0.007138082
```

The results indeed align with my expectations. The words most predictive of a Trump tweet include "fake","news","witch","hunt". For Bernie, we instead find words like "afford","billionare","corporate","climate".

Part 2: Social Network Ad Purchase.

For the second part of the lab, you will work with a dataset that contains information about individuals' purchasing behavior under exposure to online ads. The data originates from an online shopping site, and contains sociodemographic variables `Age`, `Gender` and `Salary`, as well as 20 variables ($X_1, X_2, \ldots, X_{20}$) that reflect digital traces left by the individuals. Our goal is to examine how well we can predict purchases (`Purchase=0` or `Purchase=1`) on the basis of these variables, contrasting three methods: *logistic regression*, *kNN*, and *decision trees*. The people providing us with the data suggest that *some* of digital trace variables could be very relevant for predicting `Purchase`, but they also note that many of them likely are irrelevant. However, they do not know which is which.

1. Begin by importing the file "`Kaggle_Social_Network_Ads_Augmented.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.

```r
dt <- fread(input = '/Users/marar08/Downloads/sings_ml_lecture/to_upload/lab/Kaggle_Social_Network_Ads_
dt[,user_id := NULL]
dt[,Purchased := as.factor(Purchased)]
```

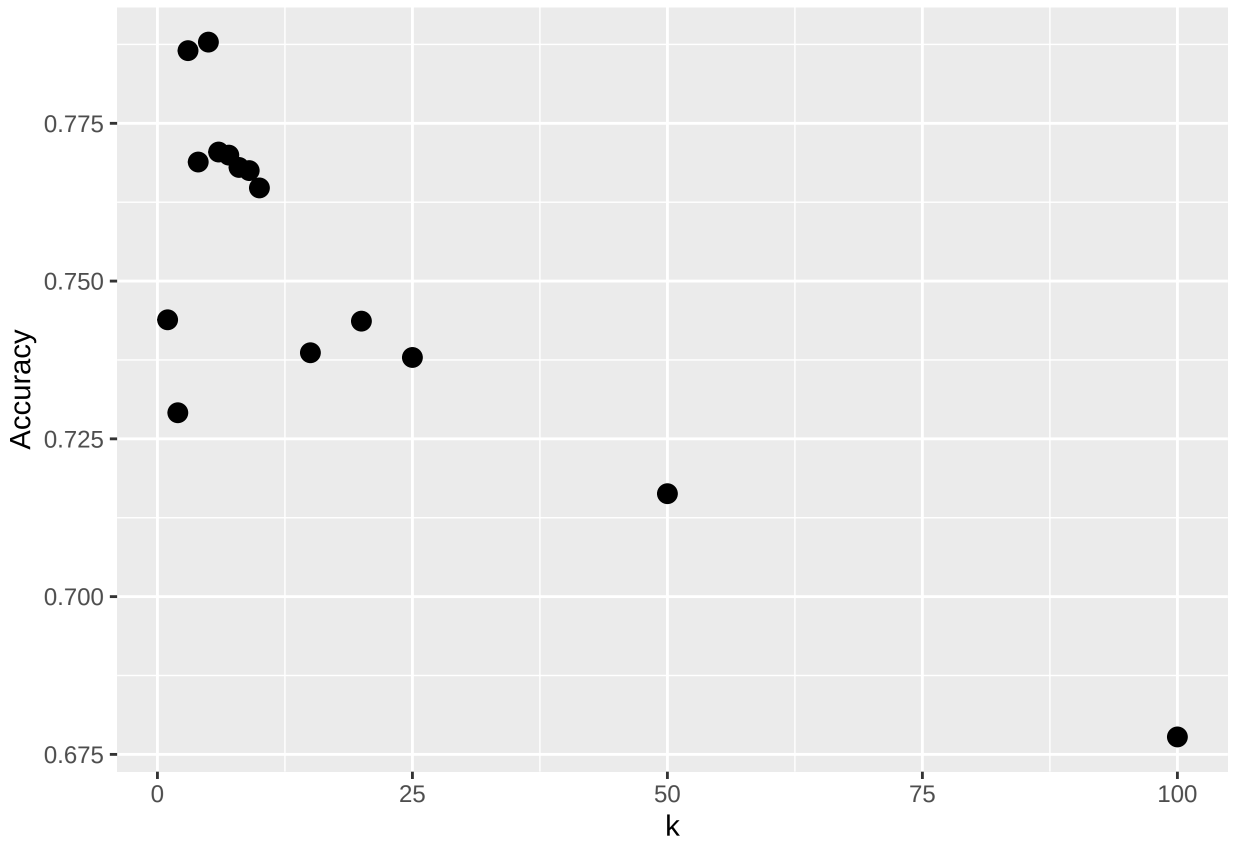
2. Use the `caret` package to implement a 10-fold cross-validation that assesses the test accuracy of a *standard logistic regression model*. Report its test accuracy.
   - Hint (i): two functions are relevant here: `trainControl` and `train`.
   - Hint (ii): to use a logistic regression model with *caret*'s `train` function, specify: `model="glm"` and `family="binimial"`.
   - Note: To ensure that the folds generated by `caret` are identical for different models, add a `set.seed(12345)` above each use of `train()`.

```r
set.seed(12345)
logit_model <- train(Purchased ~ .,
                     data = dt,
                     method = "glm",
                     family = 'binomial',
                     trControl = tc)
logit_model
```

```
## Generalized Linear Model
##
## 400 samples
##  23 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 266, 267, 267
## Resampling results:
##
##   Accuracy   Kappa
##   0.8100288  0.5765135
```

3. The second 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: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50, 100\}$. 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 what you deem to be the best k—compare to that of the *standard logistic regression*? Speculate on the reasons for the difference in predictability between the methods.

```r
# 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(12345)
knn_model <- train(Purchased ~ .,
                   data = dt,
                   method = "knn",
                   preProcess = c("center","scale"),
                   tuneGrid = expand.grid(k = c(1,2,3,4,5,6,7,8,9,10,15,20,25,50,100)),
                   trControl = tc)
ggplot(knn_model$results, aes(x=k,y=Accuracy)) +
  geom_point(size=3)
```



The highest test accuracy is obtained in the region $k \in [3, 6]$. 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. Compared to the *standard logistic regression* model, we see that kNN performs worse. 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 digital trace variables were noisy and not useful for predicting the outcome at hand.

4. Motivated by the results obtained in #3, 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*? If yes, why?

Yes. As we discussed in the lecture, decision trees classify data through a recursive 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 very unlikely to be selected as a variable to be split upon.

5. Estimating decision trees is what you will do now. Decision trees have one key parameter that a researcher must choose prior to estimation: the complexity parameter α (in `caret`: `cp`). Please do the following:

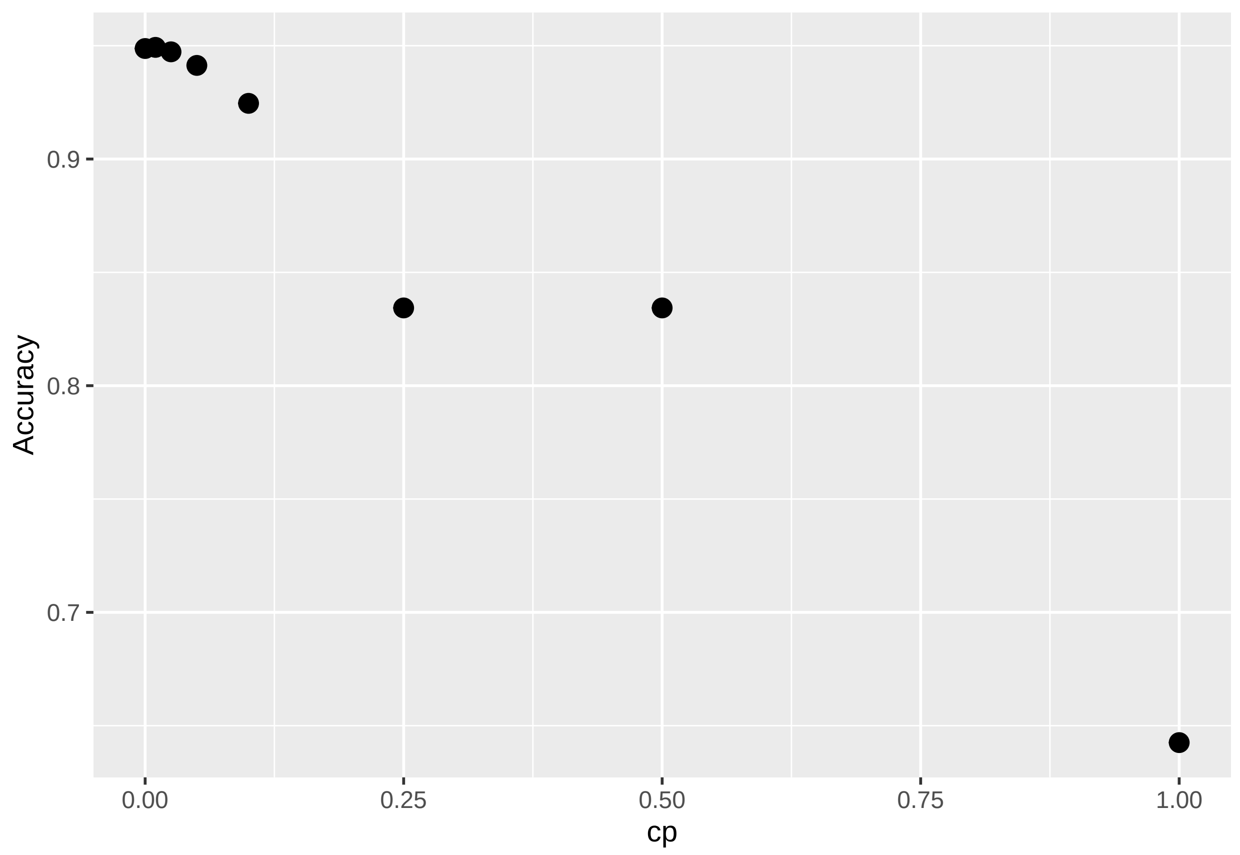
   a. Use the `caret` package (`method="rpart"`) to implement a 10-fold cross-validation procedure to assess how the test accuracy changes as a function of the complexity parameter (ISL: α; here: `cp`). Hint: use the `tuneGrid` argument to specify your grid of *cp* values. Consider the following values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

   b. 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`). Provide an interpretation. Does any of the digital trace variables $(X_1, \ldots, X_{20})$ show up in the tree?

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

Regaring a: we observe a similar type of pattern as we did for *kNN*. The highest accuracy in the cp $\in [0, 0.025]$ region. Increasing *cp* beyond this increases the bias more than it reduces the variane (underfitting). 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.

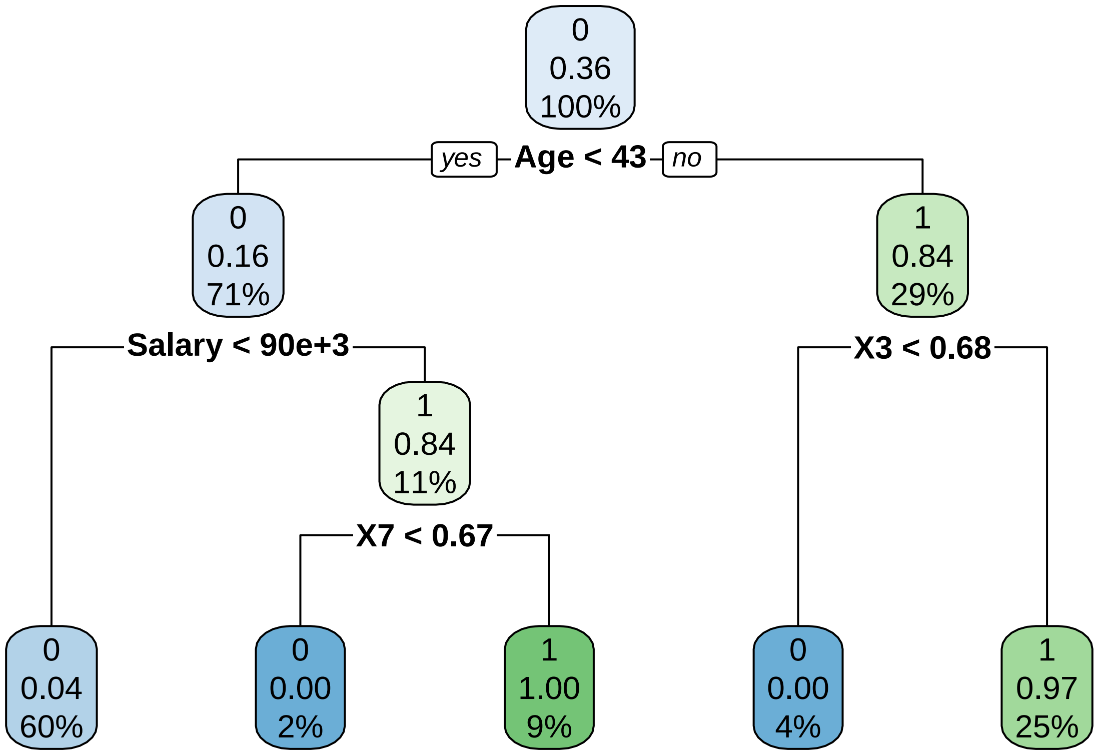
```r
set.seed(12345)
tree_model <- train(Purchased ~ .,
                    data = dt,
                    method = "rpart",
                    tuneGrid = expand.grid(cp=c(0,0.01,0.025,0.05,0.10,0.25,0.5,1)),
                    trControl = tc)

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



```
tree_model$bestTune
```

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

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



Regarding b: the tree first shows—via its first split—that, above/or age 43, a large majority (84%) of the individuals purchase the item, while below age 43, a large majority (71%) do not. Second, zooming in (going further down) the right-branch of the tree, we see that among those above/or 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 were able to identify digital trace variables that were relevant (X_3,X_7), while ignoring a majority of them.

```
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 0.67888256
##  4:         X7 0.40668840
##  5:        X18 0.05818060
##  6:        X13 0.04123056
##  7:         X5 0.01976680
##  8:        X17 0.01808483
##  9:         X8 0.01657369
## 10:         X2 0.01577321
## 11:        X12 0.01177979
## 12: GenderMale 0.00000000
## 13:         X1 0.00000000
## 14:         X4 0.00000000
## 15:         X6 0.00000000
## 16:         X9 0.00000000
## 17:        X10 0.00000000
## 18:        X11 0.00000000
## 19:        X14 0.00000000
## 20:        X15 0.00000000
## 21:        X16 0.00000000
## 22:        X19 0.00000000
## 23:        X20 0.00000000
##             var    Overall
```

Regarding c: we find that *Salary* was the variable which—either as split variable or as surrogate split variable—reduced the error the most. Among the original variables, *Salary* and *Age* were found to be by far the most important (*Gender* has a score of 0). Additionally, we the two digital trace variables that we saw in the plot (X_3,X_7). Beyond the first 4, there is a major drop.

Part 3: A fictive scenario.

For the third and final part of the lab, you will compare how well OLS and random forest perform on a *simulated* data set. Suppose in this fictive scenario that previous research has studied the phenomena in question using OLS and assumed linearity. You, having taken this course, hypothesize that there may be some interesting non-linearity present. This is what you will explore now.

1. Begin by importing the file "`fictive_scenario_dt.csv`". It contains an outcome variable Y and five input variables $X_1 \ldots X_5$. All variables are continuous.

```
dt <- fread(input = '/Users/marar08/Downloads/sings_ml_lecture/to_upload/lab/fictive_scenario_dt.csv')
```

2. To replicate the state of the art in this imagined literature, use the `caret` package to implement a 10-fold cross-validation that assesses the test error of a *standard linear regression model*. Report its test error (either *RMSE* or R^2).
   - Hint (i): two functions are relevant here: `trainControl` and `train`.
   - Hint (ii): to use a linear regression model with *caret*'s `train` function, specify: `model="lm"`.
   - Note: To ensure that the folds generated by `caret` are identical for different models, add a `set.seed(12345)` above each use of `train()`.

```
tc <- trainControl(method="repeatedcv",
                   repeats = 10,
                   number = 10)
set.seed(12345)
lm_model <- train(y ~ .,
                  data = dt,
                  method = "lm",
                  trControl = tc)
lm_model
```

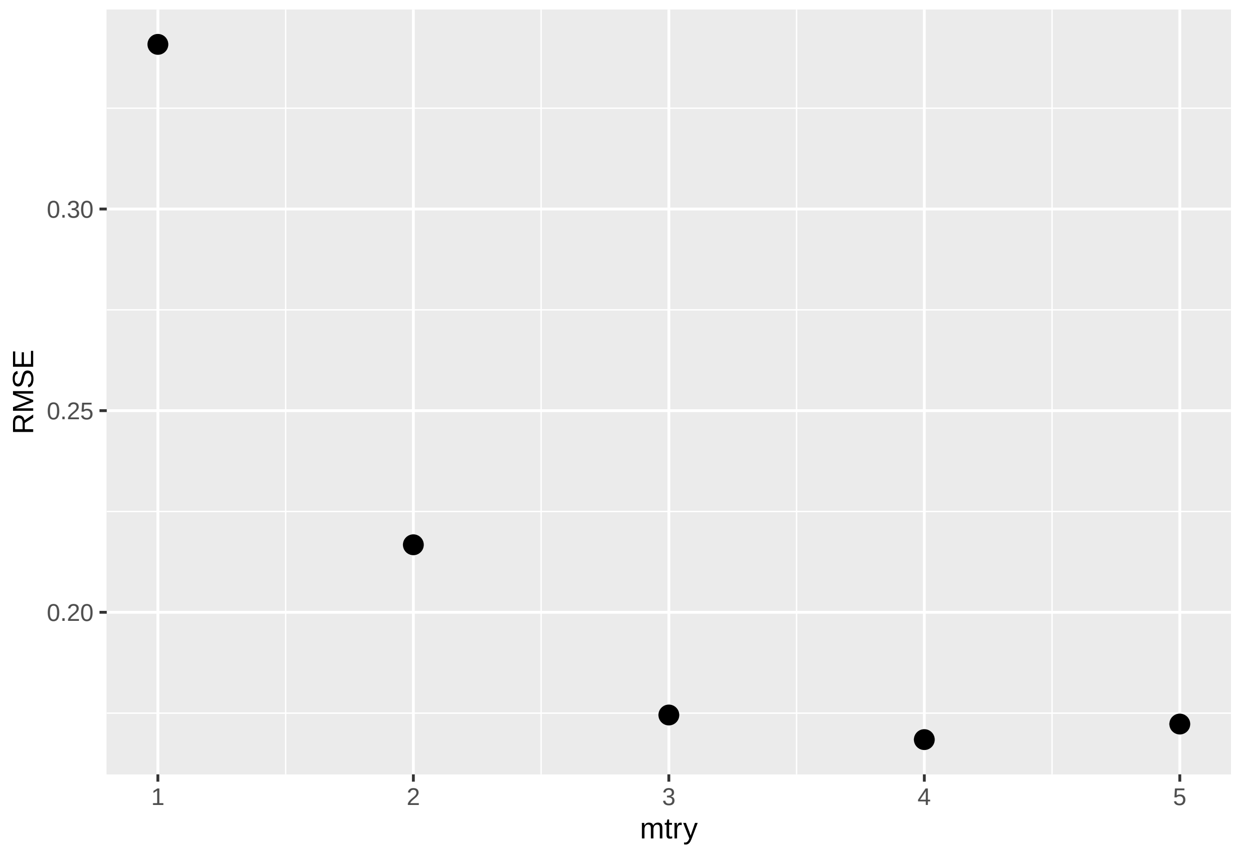
```
## Linear Regression
##
```

```
## 500 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 450, 450, 449, 451, 449, 451, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.6144584  0.2781393  0.5281909
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

3. To explore your hypothesis that there may be important non-linearities underlying this social phenomena, you shall apply *random forests*. Implement a 10-fold cross-validation procedure using the `caret` package (`method="rf"`). For random forests, a key parameter that the researcher must select is *mtry*, which 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*: $\{1, 2, 3, 4, 5\}$ (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 250 (hint: you may enter `ntree=250` as an argument in `train()` to do so). Once the estimation has finished, please do the following:

   a. Identify which mtry value results in the best predictive performance.
   b. Relate the performance of the best random forest model to the standard linear model. Do you find a meaningful improvement using random forest? What does this suggest about the data, and more specifically about the relationship between X and Y?
   c. Calculate the *variable importance scores* for both the random forest model and the standard linear regression model. Hint: you can use `caret`'s `varImp()` function to calculate variable importance. It takes as input the caret-model-object. Do you find any differences between the models, or are they in agreement on the most important variables?
   d. Create a *partial dependency plot* for the variable that attained the highest importance score for the random forest model. Create this plot both for the linear regression model and the random forest model, and compare. Hint: you can use `pdp`'s `partial()` function to create a partial dependency plot. For it, you need to specify the following arguments: `object` (set to your caret-train-object), `pred.var` (set to the name of the variable you want to plot for), `train` (set to the data you had as input for the caret-train-function), `grid.resolution` (set to 20). Interpret your findings.

```r
set.seed(12345)
rf_model <- train(y ~ .,
                  data = dt,
                  method = "rf",
                  ntree=250,
                  tuneGrid = expand.grid(mtry=c(1,2,3,4,5)),
                  trControl = tc)

ggplot(rf_model$results, aes(x=mtry,y=RMSE)) +
  geom_point(size=3)
```



```r
rf_model$bestTune
```

```
##   mtry
## 4    4
```

Regarding a: The lowest (root) mean squared error is achieved at $m = 4$ (highest R^2: $m = 3$).

Regarding b: The performance is substantially improved over the standard linear model. The random forest model explaining 95% of the variance while the standard linear model only explains circa 30%. What this suggests about the data is that there is a complex mapping between X and y — containing non-linearities and interactions.

```r
# Variable importance (rf)
varimp_rf <- varImp(rf_model$finalModel)
varimp_rf <- data.table(var = rownames(varimp_rf),varimp_rf)
varimp_rf <- varimp_rf[,Overall := Overall / max(varimp_rf$Overall)]
varimp_rf <- varimp_rf[order(Overall,decreasing = T),]
# Variable importance (lm)
varimp_lm <- varImp(lm_model$finalModel)
varimp_lm <- data.table(var = rownames(varimp_lm),varimp_lm)
varimp_lm <- varimp_lm[,Overall := Overall / max(varimp_lm$Overall)]
varimp_lm <- varimp_lm[order(Overall,decreasing = T),]
# Compare
print(varimp_rf)
```

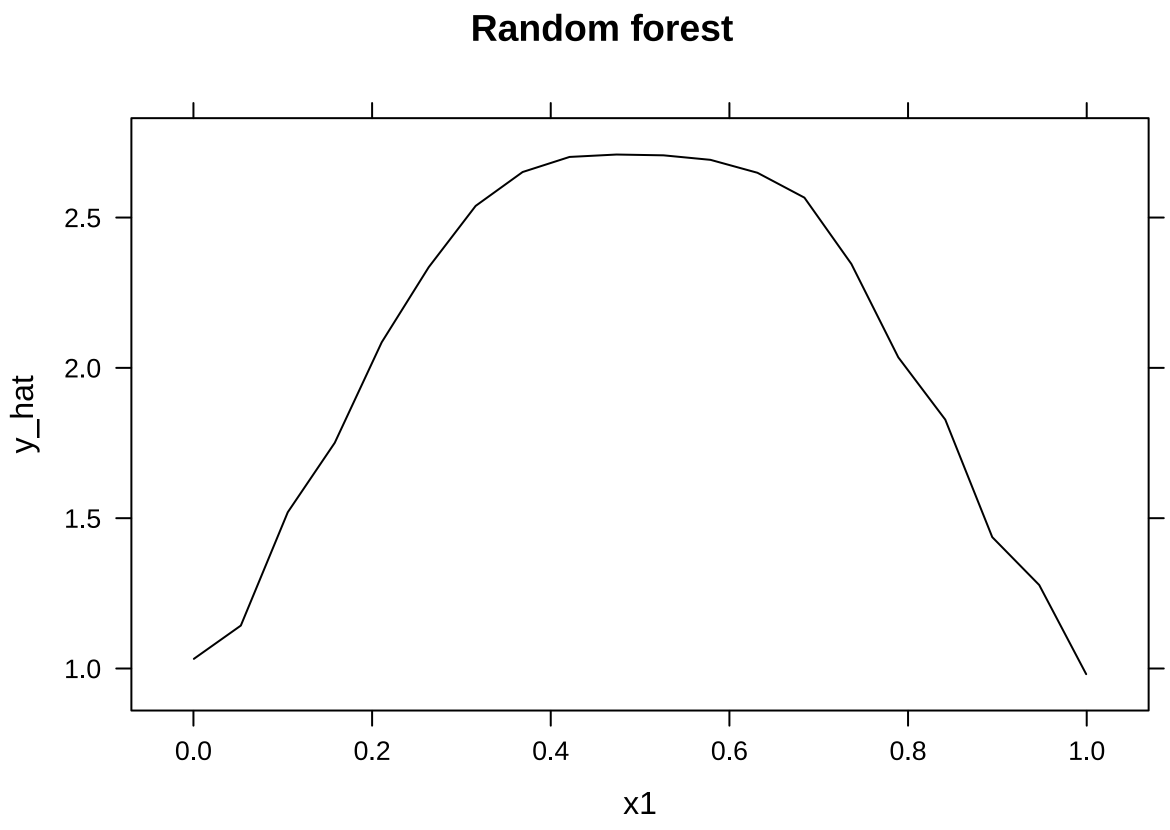
```
##        var    Overall
##     <char>      <num>
## 1:      x1 1.00000000
## 2:      x2 0.23189981
## 3:      x3 0.19970639
## 4:      x4 0.01552575
## 5:      x5 0.01489332
```

```r
print(varimp_lm)
```

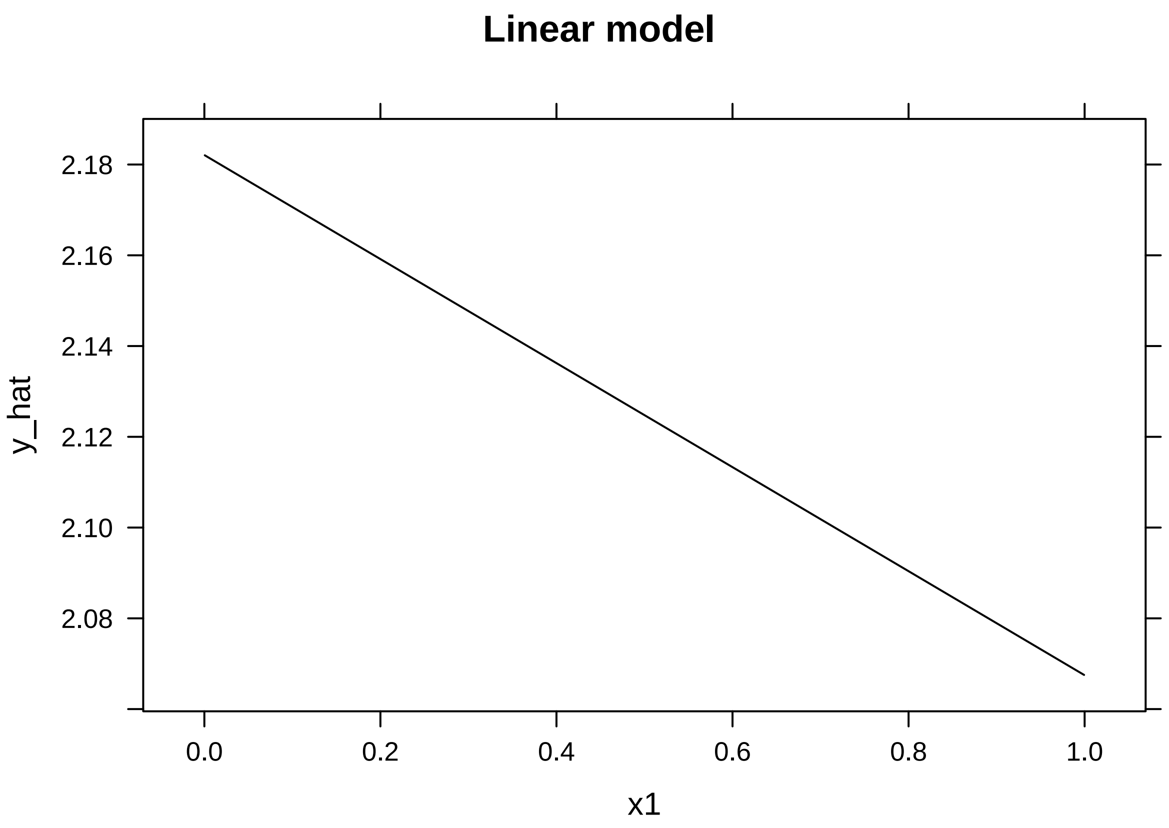
```
##        var    Overall
##     <char>      <num>
## 1:      x3 1.00000000
## 2:      x2 0.99555092
## 3:      x1 0.12001043
## 4:      x4 0.06625404
## 5:      x5 0.05552237
```

Regarding c: for the standard linear model, X_2 and X_3 are the most important variables. By contrast, X_1 dominate in the random forest model.

```r
# Plot Partial Dependence Plot for the selected variable
library(pdp)
pdp_plot_rf <- partial(rf_model$finalModel, pred.var = 'x1', train = dt, grid.resolution = 20)
pdp_plot_lm <- partial(lm_model$finalModel, pred.var = 'x1', train = dt, grid.resolution = 20)
plotPartial(pdp_plot_rf, main = 'Random forest', xlab = 'x1', ylab = "y_hat")
```



```r
plotPartial(pdp_plot_lm, main = 'Linear model', xlab = 'x1', ylab = "y_hat")
```



Regarding d: Holding other variables constant at their means, the random forest model predicts an inverse-u-shaped relationship between X_1 and the outcome; in other words, highly non-linear. By contrast, the linear model predicts a weakly negative relationship (notice y axis scale difference between the two plots). This helps explain why the random forest model performed so much better than the linear model. It also demonstrates an additional utility of more advanced model (like random forest): not only can they help improve predictions; by providing a better fit of the data; post-estimation techniques like partial dependency plots can help uncover novel relationships of scientific interest.