EXAM V23 BAN404

Sander Eide Dahling

2023-05-15

```
# Loading relevant libraries into memory

library(tidyverse)      # Used for pipe-operator and dplyr functions
library(stargazer)      # Used for summarizing data
library(glmnet)         # Used for LASSO regression
library(tree)           # Used when creating regression trees
library(randomForest)   # Used for creating randomForest model
library(leaps)          # Used for the regsubsets() function
library(boot)           # Used for bootstrapping
```

Task 1

In this exam, the data set contained in the file `Churn.csv` will be worked on and analyzed. This data set contains information related to the the customer throughput of an internet provider. In Task 1, the average size of the customers bill will be analyzed, whilst in task 2 the customer churns will be the main focus.

In order to make sure that any leftover memory won't cause any interference in this analysis, I will remove all items from the working memory with the `rm()` function. Then, the data is loaded into memory using the `read.csv()` function in base R. Note: Make sure that the .csv file is stored in the working directory!

```
# Removing leftover items from working memory
rm(list = ls())

# Loading the required data into memory
Churn <- read.csv("Churn.csv")
```

a)

Before any analysis is conducted, it is important to get an understanding of the data, and make decisions about how the variables should be programmed and which variables should be included in our analysis. Therefore, I will first take a look at the `head()` of the data, before any further re-programming and removal is done.

```
head(Churn)
```

```
##   id is_tv_subscriber is_movie_package_subscriber subscription_age bill_avg
## 1 15                1                           0            11.95       25
## 2 18                0                           0             8.22        0
## 3 23                1                           0             8.91       16
## 4 27                0                           0             6.87       21
## 5 34                0                           0             6.39        0
## 6 56                1                           1            11.94       32
```

```
##   remaining_contract service_failure_count download_avg upload_avg
## 1               0.14                     0          8.4        2.3
## 2               0.00                     0          0.0        0.0
## 3               0.00                     0         13.7        0.9
## 4               0.00                     1          0.0        0.0
## 5               0.00                     0          0.0        0.0
## 6               1.38                     0         69.4        4.0
##   download_over_limit churn
## 1                   0     0
## 2                   0     1
## 3                   0     1
## 4                   0     1
## 5                   0     1
## 6                   0     0
```

The data set contains information about different customer relationships of the provider. The only variable I will remove at this stage is the `id` variable as it merely denotes the individual customers in the data set. Variables which might not be available at the time of the prediction such as `churn` in the case of task 1, will also be removed.

As a footnote, any changes done to the data set will be stored in a new variable, `xdata` as to make sure that any changes can be reverted.

```
# Removing the id column
xdata <- Churn[, !names(Churn) %in% c("id")]
```

From this point, the re-programming of the variables as factors can be done. I choose to go by a method which focuses on the amount of unique observations in each column, which is a good indicator of the nature of a given variable. Variables with a high number of unique observations are not likely to be a factor variable by nature, and vice versa. First I will present an overview of the unique observations of each variable.

```
# Looking at possible reprogramming variables to factors
unique_count <- function(df){apply(df, 2, function(x) length(unique(x)))}
ucnt <- unique_count(xdata)

# Displaying the unique counts of the features
ucnt
```

```
##            is_tv_subscriber is_movie_package_subscriber
##                           2                           2
##            subscription_age                    bill_avg
##                        1109                         179
##         remaining_contract       service_failure_count
##                         247                          19
##               download_avg                  upload_avg
##                        2856                         802
##         download_over_limit                       churn
##                           8                           2
```

From these results there are some variables which looks very much like factor variables. These include `is_tv_subscriber`, `is_movie_package_subscriber` and `churn`. The variable `download_over_limit` can be interpreted as a factor as well, as it only has 8 unique observations, yet given its definition as a count of times the data limit is reached, I would expect it to behave like a integer variable rather than a factor variable.

I will therefore re-programm variables which has two or less unique obervations in them.

```r
#The variables which has less than 2 unique observations could be re-coded as factors.
for(i in 1:length(ucnt)) if(ucnt[i]<=2) xdata[,i]=as.factor(xdata[,i])
```

Now the `churn` variable can be removed from the `xdata` data set, as it most likely is not known if one wants to predict the billing of a current customer.

```r
# Removing the churn column
xdata <- xdata[, !names(xdata) %in% c("churn")]
```

When the initial wrangling and importing is done, I can start the analysis. The first step is to utilize a 50/50 split cross validation technique, which is used to validate a models ability to predict new and unseen observations.

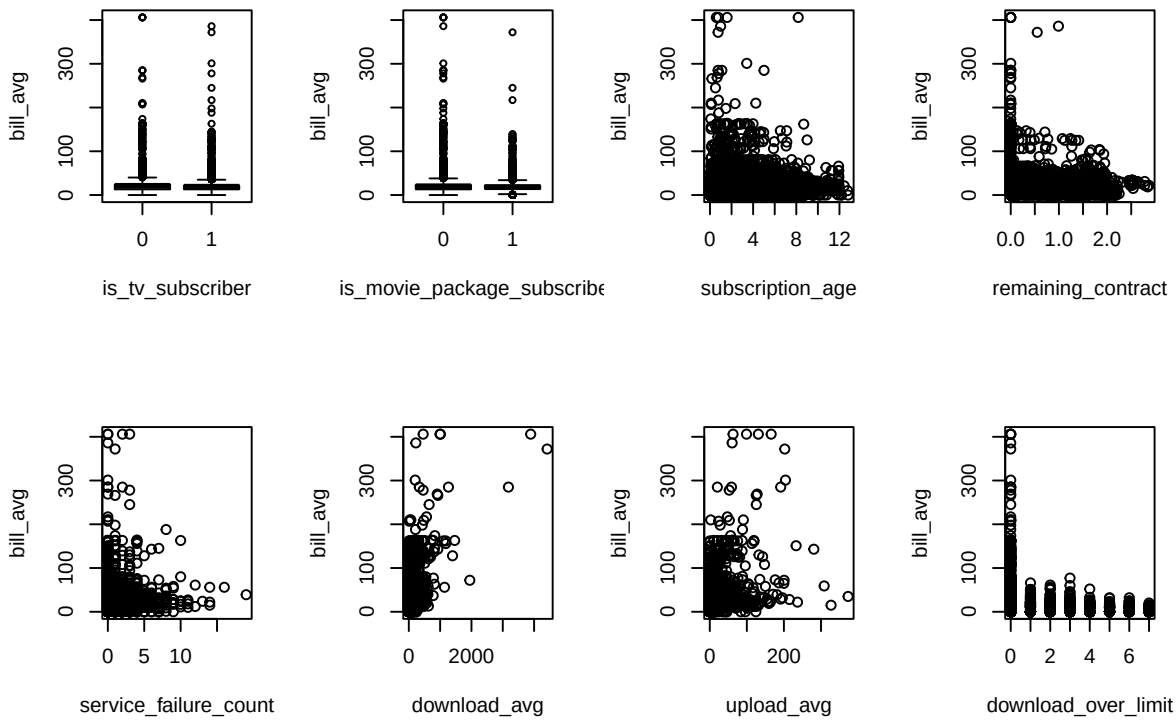
```r
# Setting the required seed
set.seed(65923764)

# Dividing the data into a training and testing set
n <- nrow(xdata)
ind <- sample(1:n, floor(n/2))

train <- xdata[ind,]
test <- xdata[-ind,]
```

b)

With the initial groundwork done, I can start getting into the analysis of the task. Before any models are created, it is important to get a grasp of the relationships inherent in the data. Descriptive statistics is important as a first step in an analysis of a data set because it allows for the ease of data visualization. It allows for data to be presented in a meaningful and understandable way, which, in turn, allows for a simplified interpretation of the data set in question1. Descriptive statistics can also help create an overview of the entire data set by summarizing it and generate an actionable set of information from the large data set having multiple variables.

First of all, I will create some plots to visualize the relationship between the variables and the `bill_avg` variable which we will use as the basis of the model. I plot all possible variables on the x-axis against the `bill_avg` on the y-axis.

```r
# Making plots for all variables compared to response variable
{
  par(mfrow=c(2,4))
  plot(bill_avg ~ ., data = train)
}
```



From these plots is is hard to find many strong indicators of the `bill_avg`, although the `upload_avg` and `download_over_limit` variables seems to have some relationship between them and the average bill size. There seems to be several clusters in the plots, and I suspect that this data is stretched out a bit, as both the scatter and box plots shows many outliers.

```
# Measures of central tendencies
stargazer::stargazer(train, type = "text", align=T, digits=4, no.space=T)
```

```
##
## ===============================================================
## Statistic                N      Mean    St. Dev.    Min      Max
## ---------------------------------------------------------------
## subscription_age      35,946 2.4440    2.0151   -0.0200  12.8000
## bill_avg              35,946 19.0366 13.4209       0        406
## remaining_contract    35,946 0.4969    0.6652   0.0000    2.8700
## service_failure_count 35,946 0.2783    0.8205       0        19
## download_avg          35,946 43.7858 67.3821   0.0000  4,415.2000
## upload_avg            35,946 4.1530    9.3189   0.0000   373.1000
## download_over_limit   35,946 0.2105    1.0011       0        7
## ---------------------------------------------------------------
```

This theory of outliers are supported by this summary of the data set, as the St. dev and Min/Max deviate quite a lot from the mean, especially taking into account the number of observations which is quite large at 35946.

Lastly, most internet provider plan comes in packages with three criteria. These are (in the US):

- Data Limit
- Upload Speed
- Download Speed

I assume that these come in packages, and therefore there is a possibility that the upload and download speeds are highly correlated. This could lead to issues with multicolinearity later down the line. I will therefore check if my assumption is true.

```
cor(train$download_avg, train$upload_avg)
```

```
## [1] 0.5776961
```

As the correlation between these two variables are shown to be over 50%, I choose to remove the upload average from the model in order to avoid co-linearity issues.

```
# Remove the variable from all defined data sets
xdata <- xdata[, !names(xdata) %in% c("upload_avg")]

train <- train[, !names(train) %in% c("upload_avg")]
test <- test[, !names(test) %in% c("upload_avg")]
```

Given that I did not find many strong indicators of `bill_avg` in the data as of yet, I do will keep the variables as is for the time being.

b)

Now we move on to the model fitting part of the task. First of all, a standard OLS model will be fitted to the data. This model will be stored in the variable `ols.fit`. Before the model is presented, I will discuss how the variables are chosen and how the predictions and the fit of the model will be evaluated.

I will fit the model to all variables which is left in the data set, as I see no reason to remove any of them as of yet, as I did not deem them worthy of removal in the previous two tasks. On the other hand, there was some variables which in the plots seemed like they has a weak relationship with the `bill_avg`. I will therefore give them the benefit of the doubt, and include them in this initial model.

When it comes to evaluation methods, I will base my interpretation of the model on two criteria, namely R^2 and the *testMSE* of the model, and the *p-value* when evaluating the performance of the explanatory variables and the intercept. R squared measures how many percent of the change in the response variable can be allocated the predictive power of the explanatory variables. The *testMSE* measures the average squared error between the actual values and the predicted values on the test data set, with a model trained on the train data set.

testMSE is defined as

$$testMSE = \frac{1}{m} \sum_{i=n+1}^{n+m} (y_i - \hat{f}(x_i))$$

```
# We fit a OLS model to the data
ols.fit <- lm(bill_avg~. , data = train)
summary(ols.fit)
```

```
##
## Call:
## lm(formula = bill_avg ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -129.49   -5.33   -0.10    4.26  351.76
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 19.4098580  0.1571820 123.487  < 2e-16 ***
```

```
## is_tv_subscriber1            -4.0302157  0.1658607 -24.299  < 2e-16 ***
## is_movie_package_subscriber1 -0.5575293  0.1415529  -3.939 8.21e-05 ***
## subscription_age              0.2759533  0.0299602   9.211  < 2e-16 ***
## remaining_contract           -2.7768363  0.0985008 -28.191  < 2e-16 ***
## service_failure_count         0.9192653  0.0728100  12.626  < 2e-16 ***
## download_avg                  0.0945820  0.0009129 103.606  < 2e-16 ***
## download_over_limit          -2.8319316  0.0603347 -46.937  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.28 on 35938 degrees of freedom
## Multiple R-squared:  0.2934, Adjusted R-squared:  0.2933
## F-statistic:  2132 on 7 and 35938 DF,  p-value: < 2.2e-16
```

With all variables deemed significant by the model, as the *p-value* is small, I think it is appropriate to keep them as of yet. The R^2 is on the other hand not very high, as it seems the model can only account for around 29.34% of the variance in the response variable.

```
# The R-squared of the model
summary(ols.fit)$r.squared
```

```
## [1] 0.2934491
```

```
# The testMSE of the OLS model
ols.pred <- predict(ols.fit, newdata = test)

ols.mse <- mean((test$bill_avg - ols.pred)^2)
ols.mse
```

```
## [1] 126.9305
```

The testMSE of the model is 126.93, which is useful when comparing this OLS model to other models later in the task, though by it self it is hard to interpret.

d)

In this part, I will fit a LASSO regression to the data. This method is similar to OLS, as it minimizes the RSS between the observed variables and a linear line. It differs slightly though, as in addition to the minimizing of the RSS, it also adds a new parameter which penalizes larger slopes (which can be interpreted as "stronger" predictions). This leads the model to be a slightly worse fit to the training data, in exchange for hopefully providing a better fit to the test data. It is also very similar to Ridge regression, as the only change is that the extra parameter is squared in Ridge, whilst it is measured in absolute terms in LASSO. This allows for some coefficient to be exactly zero, as opposed to Ridge regression.

The formula for calculating LASSO is given below:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{k=1}^{p}|\beta_k|$$

Where the λ is a scale parameter which determines how much weight is given to the extra parameter. Choosing the optimal λ is also the goal of many optimization methods associated with this method. I will first find the optimal lambda for both the scaled and not scaled data sets. Of these I will choose one based on the pros and cons of scaling. By default `nfold = 10` is found by k-fold CV with 10 folds. 10 observations are left out, the model is fitted with a particular on the rest of the data and the testMSE is computed on the 10 observations. This is then repeated for the other folds and the average of the testMSE for the different folds are computed. The with the smallest average testMSE is chosen.

```r
# Creating a temporary data frame to convert the factor variables into dummy variables
dummy_lasso <- train %>%
  mutate(is_tv_subscriber = as.numeric(is_tv_subscriber),
         is_tv_subscriber = ifelse(is_tv_subscriber == 2, 1, 0),
         is_movie_package_subscriber = as.numeric(is_movie_package_subscriber),
         is_movie_package_subscriber = ifelse(is_movie_package_subscriber == 2, 1, 0)) %>%
  select(1:2)

# Creating a scaled data frame with the rest of the variables
scaled_lasso <- train %>%
  select(-(1:2)) %>%
  scale(center = TRUE, scale = TRUE)

# Combing the two temporary data sets into a new data fram
lasso_train_scale <- cbind(dummy_lasso, scaled_lasso)
lasso_train_noscale <- cbind(dummy_lasso, select(train, -(1:2)))

# Storing the results in matrixes, which is needed for the glmnet function
X <- as.matrix(lasso_train_noscale[, !names(lasso_train_noscale) %in% c("bill_avg")])
y <- as.matrix(lasso_train_noscale[, names(lasso_train_noscale) %in% c("bill_avg")])

# Storing the results in matrixes, which is needed for the glmnet function
Xs <- as.matrix(lasso_train_scale[, !names(lasso_train_scale) %in% c("bill_avg")])
ys <- as.matrix(lasso_train_scale[, names(lasso_train_scale) %in% c("bill_avg")])

# Removing the temporary data frame
rm(dummy_lasso)
rm(scaled_lasso)

# Finding the optimal lambda of the scaled function using the cv.lambda function
lassolambda_scaled <- cv.glmnet(Xs,ys,alpha=1, nfold=10)$lambda.min
lassolambda_scaled
```

```
## [1] 0.001441521
```

```r
# Finding the optimal lambda of the not scaled function using the cv.lambda function
lassolambda <- cv.glmnet(X,y,alpha=1, nfold=10)$lambda.min
lassolambda
```

```
## [1] 0.01934657
```

With both the scaled and not scaled versions of the data set used to find their optimal lambda in regards to a LASSO regression, I can lay out my motivation for using one over the other. The main benefit of scaling is that the Lambda parameter will affect all coefficients equally by making sure that all features are on the same scale. It also helps in reducing the impact of outliers in the model.

The cons of using scaling however is also worth noting, as it makes the coefficients of the model harder to interpret, which makes it more difficult to compare the model to other alternatives. As the main goal of this task is to find the best model to use when predicting the average billing amount of a given customer, it is important to reliably compare a given model to others. This is why I have chosen to go with the not scaled model.

With the output from the previous code section, I can compare the β coefficients from this LASSO model to the OLS model.

```r
# The OLS beta coefficients
coef.df1 <- as.data.frame(as.matrix(ols.fit$coefficients)[-1,])
```

```r
# The LASSO model
lasso.fit <- glmnet(X,y,lambda=lassolambda, alpha=1)

# The LASSO beta coefficients
coef.df2 <- as.data.frame(as.matrix(lasso.fit$beta))

cbind(coef.df1, coef.df2) %>%
  rename("OLS_Coeff" = 1,
         "LASSO_Coeff" = 2) %>%
  mutate("Diffrence" = OLS_Coeff - LASSO_Coeff)
```

```
##                                 OLS_Coeff LASSO_Coeff     Diffrence
## is_tv_subscriber1            -4.03021568 -3.98718055 -0.0430351288
## is_movie_package_subscriber1 -0.55752935 -0.52519734 -0.0323320027
## subscription_age              0.27595334  0.26522491  0.0107284272
## remaining_contract           -2.77683625 -2.75276443 -0.0240718225
## service_failure_count         0.91926534  0.89801633  0.0212490075
## download_avg                  0.09458195  0.09425999  0.0003219588
## download_over_limit          -2.83193158 -2.81032550 -0.0216060842
```

Laid out side by side, the differences in coefficients are not that large. On average, the predictors are closer to zero in the LASSO model, which is expected given the extra parameter in the regression minimization. Some parameters are reduced more than others, which indicated that the LASSO model deems these parameters as less important in predicting the slope of the model. All in all, the low λ indicates that the models are similar in performance.

e)

We can use *testMSE* in order to see if the LASSO model fares better or worse when predicting the `bill_avg` of the data set.

```r
# Creating matrixes for the test data set (The same operations as before)
dummy_lasso_test <- test %>%
  mutate(is_tv_subscriber = as.numeric(is_tv_subscriber),
         is_tv_subscriber = ifelse(is_tv_subscriber == 2, 1, 0),
         is_movie_package_subscriber = as.numeric(is_movie_package_subscriber),
         is_movie_package_subscriber = ifelse(is_movie_package_subscriber == 2, 1, 0)) %>%
  select(1:2)

lasso_test_noscale <- cbind(dummy_lasso_test, select(test, -(1:2)))

testX <- as.matrix(lasso_test_noscale[, !names(lasso_test_noscale) %in% c("bill_avg")])
testy <- as.matrix(lasso_test_noscale[, names(lasso_test_noscale) %in% c("bill_avg")])

# Using the new testX data set to calculate the testMSE of this model
lasso.predict <- predict(lasso.fit, newx = testX, s = lassolambda, type = "response")

# The testMSE of LASSO model
lasso.mse <- mean((testy - lasso.predict)^2)
lasso.mse
```

```
## [1] 126.9285
```

```r
# The testMSE of OLS model
ols.mse
```

```
## [1] 126.9305
```

Compared to the OLS testMSE, the values are extremely similar. This was to be expected given the similar coefficients. This makes me conclude that there are no evidence of the LASSO model being a better predictor of `bill_avg` than standard OLS, even though the testMSE is marginally smaller.

f)

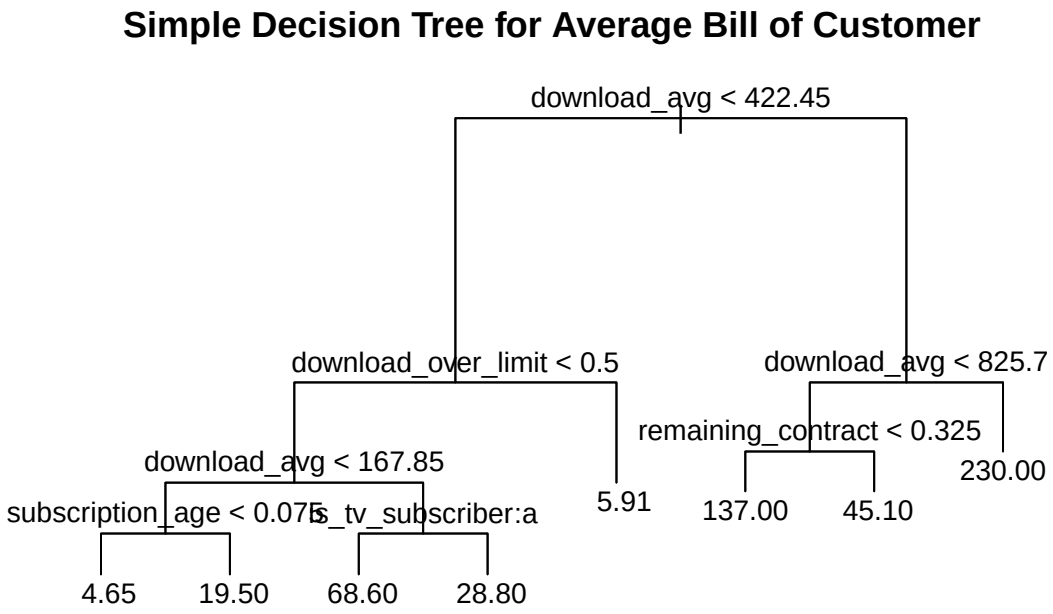
A regression tree is a type of supervised learning algorithm that is mostly used for prediction problems. It is made up of branch and leaf nodes. Branch nodes perform a logical check on the new data, which based on the output of TRUE/FALSE is sent further down the tree. If the logical check returns TRUE, the observation goes down the left side of the branch. When the observation reaches a leaf-node, the value of the leaf assigns a value to the new observation.

In order to create a decision tree, we need to divide the predictor space spanned by $X_1, ..., X_p$ into J non-overlapping regions, $R_1, ..., R_J$. In simpler terms, the region which engulfs the observations are carved up into prediction regions which minimizes RSS. The optimal regions are defined by which carving minimizes the RSS formula given below. (Lecture 11, 2023)

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}R_j)^2$$

With the theory laid out, the code for creating a regression tree is relatively simple. I load in the `tree` library, and run a regression in the same format as the standard `lm()` function. The tree can be plotted in order to get a grasp of the chosen variables. I choose to use all available variables as explanatory variables for this tree.

```
# Fitting the regression tree
simple.tree <- tree(bill_avg~., data=train)

# Plotting the regression tree
{
plot(simple.tree)
text(simple.tree, digits=3, cex = 0.9)
title("Simple Decision Tree for Average Bill of Customer")
}
```



According to the plot which has been created following the regression tree being fitted, the most important predictors are the ones as the highest branch of the tree. The `download_avg` and `download_over_limit` seems to be the main variables used when determining which value the model assigns a new observation. It seems that the main cause of the outliers in the `bill_avg` is the download amount, which places the bills at very high levels, if the value is over 422.45. On the other hand, if the average download is low, the bill is generally small.

In a realistic scenario, this makes sense as so called *power users* which downloads a lot of content, usually prefers higher speeds and higher limits to their broadband, which in turn makes it more expensive. At the lower stages, the various details surround the age of the subscription and if a TV package is included determines the small variations at the lower end of the billing rates according to the regression tree.

g)

In order to compare this decision tree with the LASSO and OLS models, I will calculate the testMSE of the regression tree in order to see if it is better at predicting new observations than the previous models.

```
# Predictions from the regression tree
simple.tree.pred <- predict(simple.tree, newdata = test)

# testMSE of the simple regression tree
simple.tree.mse <- mean((test$bill_avg - simple.tree.pred)^2)
simple.tree.mse
```

```
## [1] 135.5329
```

```
# testMSE of OLS and Lasso
ols.mse
```

```
## [1] 126.9305
```

```
lasso.mse
```

```
## [1] 126.9285
```

Compared to the previous model, this regression tree has a higher testMSE to the data. Although this means that on average the predictions of the regression tree is worse, there are reasons behind this. As the regression tree has set boundaries between values which often has a high difference between them, the variance of a regression tree is often very high. There are ways to attend to this, as we will see in the next task.

h)

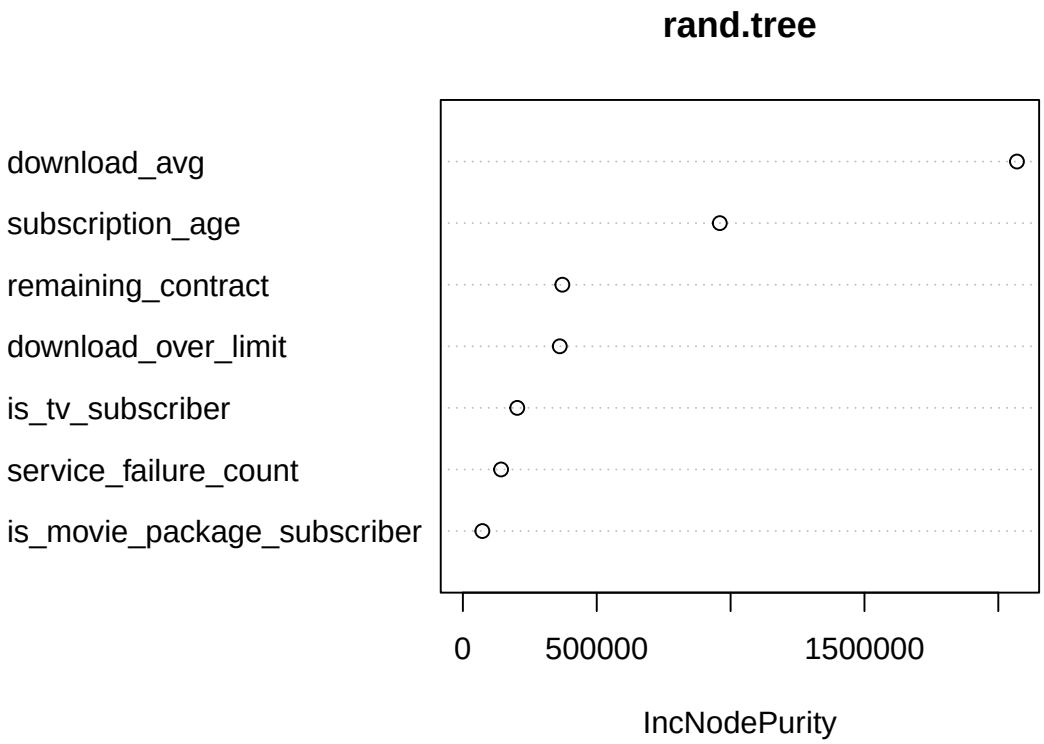
Random Forest is a form of bagging of a given regression tree. Bagging is short for bootstrap aggregation, and the main goal of this methodology is to reduce the variance of regression trees by creating many models and aggregating their results. Both of the models are described as ensemble learning methods. Whilst normal bagging uses full re-sampling with replacement for each iteration, random forest makes use of randomness in its feature selection, which is more advanced. Another difference is that many iterations of bagging can be done simultaneously, whilst random forest is iterative, which means it cant make use of more cores in a computer that easily. This must be remembered when choosing between the methods.

In this case however, it is specified that a random forest model must be applied to the data set at hand. In this section the `randomForest` package and function will be used.

Mtry: `mtry` is the number of variables randomly sampled at each split. In a bagged regression tree, the `mtry` is equalto the number of explanatory variables, as all features are selected for each iteration. In this model however, the `mtry` is set lower to account for the random feature selection in this scenario.

ntree: The `ntree` parameter is set low in comparison to other randomForest models, as the computation power needed to fit 100+ trees is very high, as this training set contains over 35 000 observations.

```
# Fitting the randomForest model to the data
rand.tree <- randomForest(bill_avg~., data=train, mtry=3, ntree=10)
                                    # ntree is set at 10 to improve computation
                                    # times. The program crashes at ntree = 1000
# Variable Importance
{
  varImpPlot(rand.tree)
}
```



VarImpPlot: In a random forest regression tree, the IncNodePurity represents the improvement in the mean squared error (MSE) of the response variable when a split is made on a predictor variable. The IncNodePurity is calculated as the difference between the MSE of the parent node and the weighted average of the MSEs of the child nodes. The higher the IncNodePurity, the better the split is at separating observations into different groups based on their response variable values. In this case, the `download_avg` and `subscription_age` are the variables with the highest purity, which tells us that these are the most important predictors in the random forest model.

i)

With the model in h) fitted, we can use it to predict the `bill_avg` in the test set, and compare its testMSE with the other models fitted earlier in the task. We use the same method as before when calculating the testMSE.

```
# Predicting the bill_avg of the test data set
rand.pred <- predict(rand.tree, newdata = test)

# Calculating the testMSE of the randomForest model
rand.mse <- mean((test$bill_avg - rand.pred)^2)
rand.mse
```

```
## [1] 114.4888
```

The testMSE of this regression model is very good compared to its earlier counterparts. Especially compared to the simple tree, which tells us that the variance in the regression tree impacted the predictive power of that model.

```
# OLS testMSE
ols.mse
```

```
## [1] 126.9305
```

```
# LASSO testMSE
lasso.mse
```

```
## [1] 126.9285
```

```
# Simple Regression Tree testMSE
simple.tree.mse
```

```
## [1] 135.5329
```

With all models fitted, it is possible to conclude that, given that testMSE is a good measure of accuracy for the predicitons, the best model when it comes to accuracy is the randomForest model fitted in h).

j)

In this last sub-task in Task 1, I will make use of my previous analysis in order to determine which features is most prominent in the high and low end of billig amounts. Although this could be a very long section in which one could categorize the billig levels and determine feature importance for the various categories, I have chosen to interpret the task in a way which limits further analysis, which in turn leads me to reference my earlier answers when discussing the importance of various features for the categories, instead of making new analyses.

In the OLS and LASSO model, the `is_tv_subscriber`, `remaining_contract` and `download_over_limit` were the features which dragged down the expected billing of a given customer. The increase in these variables led to quite a big jump down in expected billig, which might suprise some. My interpretation of this fact is the structure of the deals given by the company. Based on these observations, I assume that casual consumers has a TV package included in their deal, whilst the *power users* mentioned previously might not need this, as they might be small businesses or people who use their internet a lot. If the company offers discounts for binding contracts, it makes sense that customers with more time remaining on their contract has gotten "better" deals and have lower monthly billing costs. Customers who have downloaded more than their limit might have lower billig because they have an initial lower download cap, which is cheaper on average.

As the intercept is quite high, the remaining features doesn't have any striking effect which lifts the price way higher up, yet the `service_failure_count` and `download_avg` are predicted to raise the average billing of customers, which makes sense given the analysis of which features are important predictors for customers billed at the low end.

In our regression tree models, it looks like the main feature which determines wheter a customer will get a bill in the low or high range is `download_avg`. This is reflected in the randomForest model as well, as it has a very high IncNodePurity. The `subscription_age` is also an important predictor in the randomForest model, whilst it is not the most indicative of billing in the simple regression tree. On the low end, the simple regression tree seems to value `is_tv_subscriber`, `download_over_limit` and `subscription_age`.

In summary, based on the analysis of the previous tasks, the variables can be placed into groups in which they are most able to predict the outcome of the average billig of a customer.

At the high end, the `download_avg` and `remaining_contract` is the most important predictors of higly billed customers, whilst on the low end, the variables `is_tv_subscriber`, `remaining_contract`, `subscription_age` and `download_over_limit` are most indicative of the customers with lower billings.

Task 2

In this task, we will change perspective and focus on predicting the `churn` variable in the data set. This is a dummy variable which indicated whether or not a customer has stopped using the service. This

is useful for the company, as it makes it easier to target customers who are predicted to churn with campaigns or offers which might make them stay on as a customer.

In order to make sure that the analysis from the previous tasks does not interfere with the analysis from this task (as they are quite separate) I will remove all values in the working memory of the script, and load the data in again.

```
# Removing leftover items from working memory
rm(list = ls())

# Loading the required data into memory
Churn <- read.csv("Churn.csv")
```

In order to be able to use this data in the same vein as before, I will perform the same reprogramming and wrangling as I did i Task 1, as the data is the same. The reasoning behind the choices made in this wrangling and re-programming can be found in Task 1 a) and b).

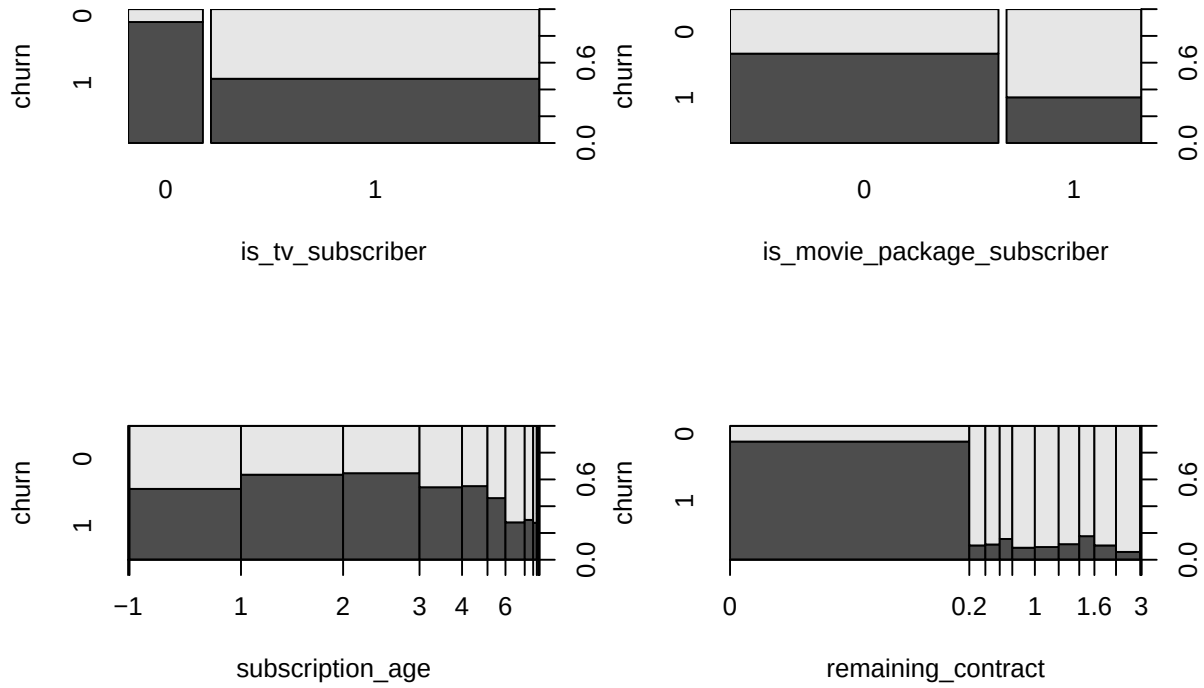
```
# Removing the id column
xdata <- Churn[, !names(Churn) %in% c("id", "upload_avg")]

# Looking at possible reprogramming variables to factors
unique_count <- function(df){apply(df, 2, function(x) length(unique(x)))}
ucnt <- unique_count(xdata)

#The variables which has less than 2 unique observations could be re-coded as factors.
for(i in 1:length(ucnt)) if(ucnt[i]<=2) xdata[,i]=as.factor(xdata[,i])
```

Although it is not specified, I have chosen to make use of the 50/50 cross validation technique in order to create a testing and training set in this task as well. I will use the same seed as before.

```
# Setting the required seed
set.seed(65923764)

# Dividing the data into a training and testing set
n <- nrow(xdata)
ind <- sample(1:n, floor(n/2))

train <- xdata[ind,]
test <- xdata[-ind,]
```

With all the changes completed, and the new data set stored in `xdata`, I can start by using some descriptive statistics on the data set which might be able to help us when determining the churn feature of an observation.

```
# Making plots for all variables compared to response variable
{
  par(mfrow=c(2, 2))
  plot(churn ~ is_tv_subscriber, data = train)
  plot(churn ~ is_movie_package_subscriber, data = train)
  plot(churn ~ subscription_age, data = train)
  plot(churn ~ remaining_contract, data = train)
}
```



These plots show some promising patterns. The customers who are TV and Movie subscribers seem to be much lower churn rate than the customers who do not have those subscriptions. The people who churn based on the age of the subscription follows something which looks like a beta probability curve, with the probability increasing at first, but later it decreases drastically. The customers who has a contract are very liekly not to churn, as they most likely will suffer penalties based on breaching the contract, whilst the customers with no contract has a very high churn rate.

```
# Comparing promosing determinators of churn
by(train$download_over_limit, train$churn, mean)
```

```
## train$churn: 0
## [1] 0.03150599
## ------------------------------------------------------------
## train$churn: 1
## [1] 0.3519625
```

```
by(train$download_avg, train$churn, mean)
```

```
## train$churn: 0
## [1] 65.10445
## ------------------------------------------------------------
## train$churn: 1
## [1] 26.93347
```

In this comparison, we can see that customers who churn tend to have lower data usage and often go over their data capacity for a given month. This could be useful when determining which customers are likely to churn. In summary, I have found that `is_tv_subscriber`, `is_movie_package_subscriber`, `subscription_age`, `remaining_contract`, `download_over_limit` and `download_avg` seems like good predictors of churn, and will be used going forward in the analysis.

b)

In this part I will use bootstrapping in order to compute the 95% confidence interval for p. As the churn variable is stored as a factor variable in the `xdata` data set, I have to transform the observations into numeric dummy variables if they are ot be used in the following analysis.

```
# Transforming churn into a numeric dummy variable
churn.num <- as.numeric(xdata$churn[1:50])-1
```

Using the `boot`-package, it is now possible to estimate `p` based on the values presented in the first 50 observations of the data set.

```
# Setting the required seed
set.seed(65923764)

# Defining the function used in the boot function, which needs a
# data parameter and a index parameter
mean.boot.func <- function(data, i) mean(data[i])

# When the function is defined, the boot() function in the boot package can be used to
# to estimate the 95% confidence interval of the actual mean for p. I use 1000 reps.
obj.boot<- boot(churn.num, mean.boot.func, R = 1000)

# In order to extract the quantiles of this bootstrapped data set, I use the boot.ci function
ci.boot <- quantile(obj.boot$t, c(0.025, 0.975))
ci.boot
```

```
##  2.5% 97.5%
##  0.68  0.90
```

This shows us that the 95% quantile of the bootstrap sample is (0.68, 0.90).

Comparing this bootstrap confidence interval to the standard approximation method can be done by calculating the p based on a mathematical formula. This method is defined as

$$\hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

where $\hat{p}$ is the sample fraction (i.e. the mean of the sample) and n is the number of observations, which in this case is equal to 50.

```
# We define N as the length of the churn.num num variable and p_hat as the mean of the
# churn.num sample
N <- length(churn.num)
p_hat <- mean(churn.num)

# We implement the the function laid out mathematically above
std_ci <- c(p_hat - 1.96 * sqrt(p_hat * (1 - p_hat) / N), p_hat + 1.96 * sqrt(p_hat * (1 - p_hat) /
round(std_ci, 4)
```

```
## [1] 0.6891 0.9109
```

For reference, a similar figure to this and the output of the quantile function is shown in the output of the `boot.ci`, although I found it useful to lay out my though process with the full formula. In normal use cases, the output of that function would be sufficient for most cases.

c)

In this task, I will fit a logistic regression model to the data, in order to predict the categorical variable `churn`. I will make use of the features presented in a). In order to be able to evaluate the models predictive power, I will make use of the training and testing data set created in task a).

```
# Defining the formula used in this logistic regression and tree model later
form <- churn~is_tv_subscriber+
  is_movie_package_subscriber+
  subscription_age+
  remaining_contract+
  download_over_limit+
  download_avg

# Fitting the model to the training data with family = binomial as we
# are using the logistic regression to classify a factor variable
log.fit <- glm(form, data = train, family = "binomial")
summary(log.fit)
```

```
##
## Call:
## glm(formula = form, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4196  -0.4067   0.1923   0.5385   4.3265
##
## Coefficients:
##                                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    4.2630724  0.0683463  62.375   <2e-16 ***
## is_tv_subscriber1             -1.7750974  0.0635679 -27.924   <2e-16 ***
## is_movie_package_subscriber1 -0.0728537  0.0360082  -2.023    0.043 *
## subscription_age              -0.2538756  0.0083859 -30.274   <2e-16 ***
## remaining_contract            -3.1889069  0.0363860 -87.641   <2e-16 ***
## download_over_limit            0.4835534  0.0401605  12.041   <2e-16 ***
## download_avg                  -0.0109605  0.0003526 -31.081   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 49338  on 35945  degrees of freedom
## Residual deviance: 24579  on 35939  degrees of freedom
## AIC: 24593
##
## Number of Fisher Scoring iterations: 6
```

The coefficients for factor variables in logistic regression models represent the difference in the log odds of the outcome variable between the reference category and the category represented by the coefficient1. In this case, since there are only two possible states for each variable, the reference category is likely to be 0. Therefore, we can interpret the coefficients as follows:

- The coefficient for `is_tv_subscriber` is -1.7750974. This means that holding all other variables constant, the odds of the churn happening are expected to be 16.95% lower when `is_tv_subscriber` changes from 0 to 1.
- The coefficient for `is_movie_package_subscriber` is -0.0728537. This means that holding all other variables constant, the log odds of the churn happening are expected to be 92.97% lower when `is_movie_package_subscriber` changes from 0 to 1.

When it comes to the signs of each coefficient, there were some noteworthy aspects located in the summary. The sign of each coefficient indicates whether the relationship between the predictor variable and the response variable is positive or negative. A positive coefficient indicates that as the predictor variable increases, so does the probability of the response variable being 1 (or "churn"). A negative coefficient indicates that as the predictor variable increases, the probability of the response variable being 1 decreases.

All the coefficients are negative except for `download_over_limit`. This tells me that for every unit added to each of those variables, the probability of churn decreases. This fits with the analysis done in task a), which leads me to conclude that the signs were as I expected initially based on my findings in a).

d)

In this task i will evaluate the prediction power of the fitted logistic regression model. As this is a classification model, it is not possible to get a meaningful testMSE from the predictions of the model. The best way to determine the accuracy of a classification model (within the scopes of this course) is a **confusion matrix**. A confusion matrix compares the number of correctly classified observations against the number of wrongly classified observations in a matrix, which enables us to see if the model has any bias towards having false positives over false negatives etc.

With a confusion matrix it is also possible to measure the accuracy by dividing the number of correctly classified observations against the total number of classified results, which indicates how often the model estimates the class of a new observation correctly. I set the threshold `th` as 0.5, where predictions over 0.5 are classified as churn, and predictions below 0.5 are classified as not churn.

```
# Threshold
th <- 0.5

# Predicting the class of the observations in the test data set
log.prob <- predict(log.fit, newdata = test, type = "response")
log.pred <- ifelse(log.prob > th, 1, 0)

# Creating a confusion matrix
log.cm <- table(test$churn, log.pred)
log.cm
```

```
##    log.pred
##         0     1
##   0 12976  2997
##   1  1636 18338
```

In addition to this, it is possible to create a similar matrix which uses percentages of the total instead of absolute numbers, which makes it easier to compare against other confusion matrixes.

```
# Confusion Matrix as fraction
log.cm.perc <- round(prop.table(table(test$churn, log.pred), margin = 1), 3)
log.cm.perc
```

```
##    log.pred
##         0     1
##   0 0.812 0.188
##   1 0.082 0.918
```

The accuracy of the model can be calculated as specified above, by dividing the diagonal on the sum of the table.

```
# Accuracy of the model
log.acc <- sum(diag(log.cm))/sum(log.cm)
log.acc
```

```
## [1] 0.8711158
```

The accuracy of the model is around 87% which means that it is able to classify 87% of new observations correctly.

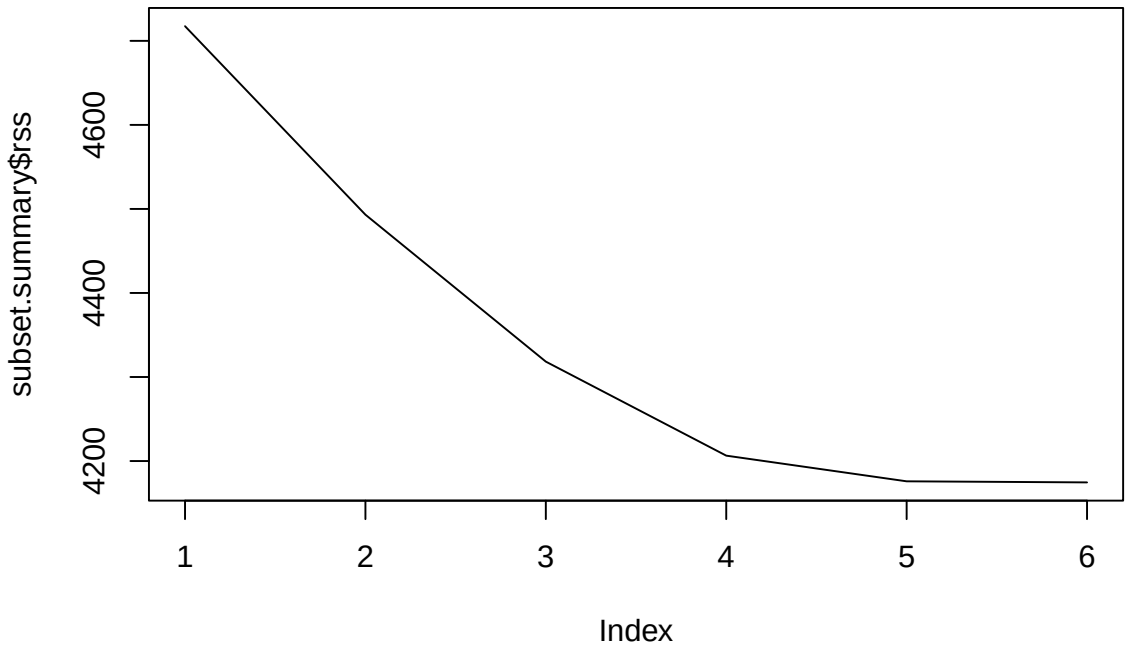
We can use this framework in order to see of the predictions are improved if some of the variables are removed from the regression. I will present a new regression model with fewer features, and use the confusion matrix and accuracy measures in order to determine if the new model has better predictive power than the first model.

In order to decide which features to remove, I will make use of the `regsubsets()` function in the `leaps`-package. The output of this function shows which variables contributes the least to lowering the RSS of the model.

```
# Using the regsubsets() function in the leaps package
log.subset <- regsubsets(form, data = train, nvmax = 8)

# Extracting the output of the summary which is useful to us
subset.summary <- summary(log.subset)
subset.summary$outmat
```

```
##          is_tv_subscriber1 is_movie_package_subscriber1 subscription_age
## 1  ( 1 ) " "               " "                          " "
## 2  ( 1 ) "*"               " "                          " "
## 3  ( 1 ) "*"               " "                          " "
## 4  ( 1 ) "*"               " "                          "*"
## 5  ( 1 ) "*"               " "                          "*"
## 6  ( 1 ) "*"               "*"                          "*"
##          remaining_contract download_over_limit download_avg
## 1  ( 1 ) "*"                " "                  " "
## 2  ( 1 ) "*"                " "                  " "
## 3  ( 1 ) "*"                " "                  "*"
## 4  ( 1 ) "*"                " "                  "*"
## 5  ( 1 ) "*"                "*"                  "*"
## 6  ( 1 ) "*"                "*"                  "*"
```

```
# Plotting the RSS against number of features in the subsets
plot(subset.summary$rss, type = "l")
```



From here I can see that after 4 features, the decrease in RSS is not very large. Given that models with fewer features are less suceptible to overfitting, I will create a new logistic regression model which only includes 4 features.

```
# Defining the formula used in this logistic regression and tree model later
form2 <- churn~is_tv_subscriber+
  subscription_age+
  remaining_contract+
  download_avg

# Fitting the model to the training data with family = binomial as we
# are using the logistic regression to classify a factor variable
log.fit2 <- glm(form2, data = train, family = "binomial")
summary(log.fit2)
```

```
##
## Call:
## glm(formula = form2, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9473  -0.4080   0.1974   0.5345   4.5037
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)         4.404801   0.068323   64.47   <2e-16 ***
## is_tv_subscriber1  -1.858038   0.062490  -29.73   <2e-16 ***
## subscription_age   -0.253549   0.008288  -30.59   <2e-16 ***
## remaining_contract -3.217199   0.035906  -89.60   <2e-16 ***
## download_avg       -0.011745   0.000353  -33.27   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 49338  on 35945  degrees of freedom
## Residual deviance: 24847  on 35941  degrees of freedom
## AIC: 24857
##
## Number of Fisher Scoring iterations: 6
```

Although the AIC is a bit higher in the new model, it will be interesting to see how this model measures up against the previous model fitted in c).

```
# Threshold
th2 <- 0.5

# Predicting the class of the observations in the test data set
log.prob2 <- predict(log.fit2, newdata = test, type = "response")
log.pred2 <- ifelse(log.prob2 > th2, 1, 0)

# Creating a confusion matrix
log.cm2 <- table(test$churn, log.pred2)
log.cm2
```

```
##    log.pred2
##         0     1
##   0 12908  3065
##   1  1650 18324
```

```
# Confusion Matrix as fraction
log.cm.perc2 <- round(prop.table(table(test$churn, log.pred2), margin = 1), 3)
log.cm.perc2
```

```
##    log.pred2
##         0     1
##   0 0.808 0.192
##   1 0.083 0.917
```

```
# Accuracy of the model
log.acc2 <- sum(diag(log.cm2))/sum(log.cm2)
log.acc2
```

```
## [1] 0.8688347
```

```
# The accuracy compared to the previous model
log.acc
```

```
## [1] 0.8711158
```

Even though the accuracy is a bit lower in this new model compared to the old one, it is a very small amount, given that we have removed two explanatory variables. With this information it is not possible to conclude that the predictions are not better given this data set, although in the long run the new model will most likely prove more useful and be less prone to overfitting given new data.

e)

As the theory behind random forest models are very similar to the ones we created in Task 1h), I won't go into detail about the usecases and diffrences between randomForest and bagging. I will, however, highlight a important difference between how a classification tree and regression tree is created. As classification trees cannot make use of the RSS function presented earlier to create the binary splits, we need to minimize some other measure of the performance of the model in order to create the important subsets of the data set.

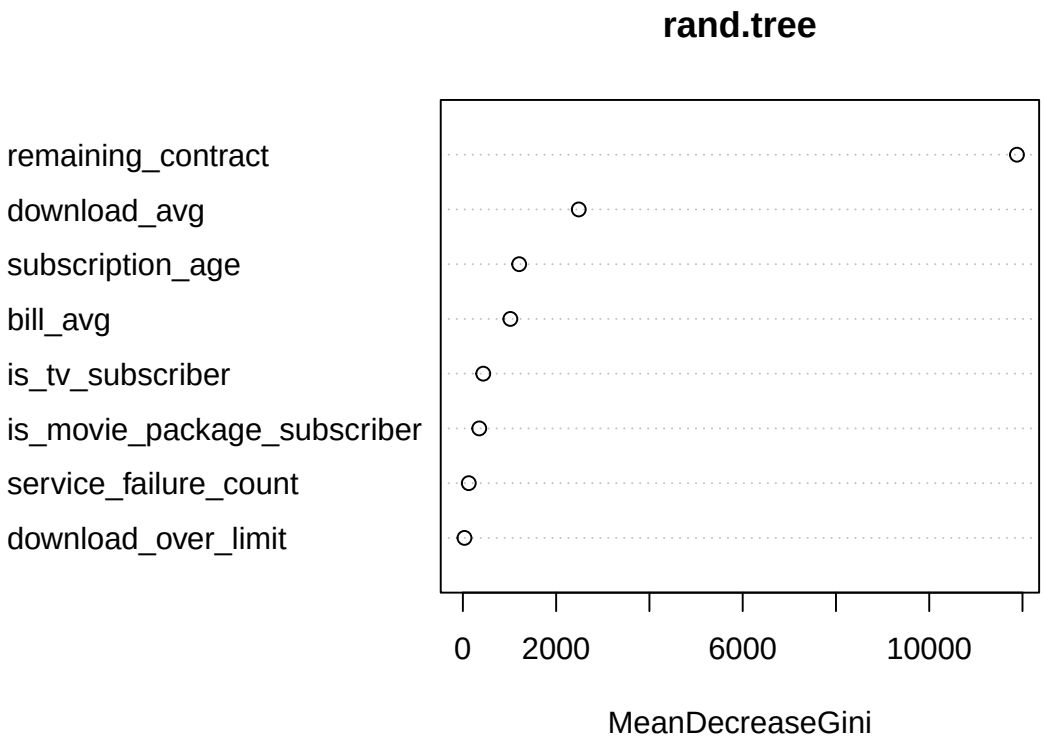
In order to divide the predictor space, we need to let $\hat{p}_{mk}$ be the proportion of observations in R_m from class k. Then one can find the rectangles $R_1, ..., R_J$ (Lecture 11, 2023). These measures are as follows:

- Classification Error
$$E = 1 - \max_k(\hat{p}_{mk})$$
- Gini Index
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$
- Cross Entropy
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

Out of these, the Classification Error is not that useful for creating decision trees, which leaves us with the **Gini Index** and the **Entropy** of the splitting.

With the main differences between classification and regression trees defined, we can start by fitting a random forest model to the data set. The syntax behind this is very similar to the previous tasks. I will make use of all the available features in the data set, as there might be some features which was deemed not important in the logistic regression, but is useful in the fitting of the random forest. I will make use of 4 predictors for each random tree, which gives us `mtry = 4`.

```
# Fitting the randomForest model to the data
rand.tree <- randomForest(churn~., data=train, mtry=4, ntree=10)
                                    # ntree is set at 10 to improve computation
                                    # times. The program crashes at ntree = 1000
# Variable Importance
{
  varImpPlot(rand.tree)
}
```



Using almost the same code from Task 1, one can see that the `IncNodePurity` has changed to `MeanDecreaseGini`. This is because the function which is minimized has changed to the Gini Index, which is what we predicted! When it comes to the importance of each variable we can see that the `remaining_contract` is by far the feature with the most impact on the Gini Index, followed by `download_avg`, `subscription_age` and `bill_avg`.

Although there are similarities between this model and the logistic regression, some of the features in the random forest are less important than in the logistic regression, which is an observation which points to there being non-linear features in some of the variables.

Moving on to evaluating the predictive performance of the model, we once again use the confusion matrix and accuracy measure as tools to evaluate the predictive power of the model. As the code is similar to the one presented in the previous task, I will type it out and jump straight to comparing the models.

```
# Predicting the class of the observations in the test data set
tree.pred <- predict(rand.tree, newdata = test)

# Creating a confusion matrix
tree.cm <- table(test$churn, tree.pred)
tree.cm
```

```
##    tree.pred
##         0     1
##   0 14982   991
##   1  1251 18723
```

```
# Confusion Matrix as fraction
tree.cm.perc <- round(prop.table(tree.cm, margin = 1), 3)
tree.cm.perc
```

```
##    tree.pred
##         0     1
```

```
##   0 0.938 0.062
##   1 0.063 0.937
```

```
# Accuracy of the model
tree.acc <- sum(diag(tree.cm))/sum(tree.cm)
tree.acc
```

```
## [1] 0.9376304
```

```
# The accuracy compared to the previous model
log.acc
```

```
## [1] 0.8711158
```

```
log.acc2
```

```
## [1] 0.8688347
```

Looking at the accuracy of the model, it has 6.65% higher accuracy than the best logistic classification model. This means that for new, unseen observations the model is able to predict at a 93.67% accuracy whether or not the customer will churn or not. This is a very high number given that the number of people who have churned/not churned are split at around 50% each.

f)

In this last task, I will make use of the analysis from tasks a) - e), and summaries based on the inference gathered from the models which features are the most typical of customers who churn. The structure of this task will be similar to that of Task 1 j).

From the descriptive statistics, it was very clear that some customer properties were indicative of a higher churn rate. For example, customers without TV or Movie subscriptions were much more likely to churn. This behavior is reasonable, as additional services may keep customers from moving to competitors, even though the base product is similar. Another factor which became evident is the fact that the longer a customer has stayed, the less likely they were to churn, which also makes sense given that the customers who have changed to this provider recently, are more inclined to change provider given a better offer. The variable showing the remaining contract time is interesting, albeit obvious, as a customer with a contract is likely to suffer economic penalties for changing providers, which leads to a low churn rate.

From the logistic regression we saw that four of the features accounted for much of the explanatory power in the model. I will use these as the basis of my analysis from this part. In addition to the features mentioned already, such as `is_tv_subscriber`, `subscription_age` and `remaining_contract` we saw that the '`download_avg` was an important measure of churn. Therefore, the output of the regression model which told us that for each unit of extra data downloaded, the likelihood of a customer churning was decreased by 98.91%. One reason as to why this percentage is as high as it is, is because of the fact that many customers don't use their broadband, so any use at all signifies a great chance of the churn not taking place. With more time, a useful analysis might be to separate out customers which has had no internet usage and see if this is a good predictor of churn, but I digress.

When it comes to the random forest model, is is not as easy as the logistic regression (or a simple classification tree for that matter) to determine which variables was most indicative of a customer which churned. We can see from the variable importance as to which variables were the most important when determining churn, which was the remaining time on their contract, the average download amount, age of subscription and the average monthly bill. Based on the descriptive statistics, it is possible to make an educated guess as to which values were most indicative of a customer churning, which is a customer with a long remaining time on their contract, and a customer who uses their deal to actually download data from the internet.

In summary, based on my interpretation of the analysis conducted in this paper, I believe that some variables has stood out in several models, and can give us a picture of the traits which a churned customers possesses. For example, a customer without any TV or Movie packages, who don't use their internet very much and has no time left on their contract is very likely to churn compared to the other factors. Based on my understanding of the internet service provider business, this seems like a logical conclusion to draw based on the data and the interpretations of the analysis.