Modern Data Mining - HW 4

Jose Cervantez Bethany Hsaio Rob Kuan

Due: 11:59Pm, April 7th, 2024

Contents

Overview

In this homework, we will explore the transition from linear models to more flexible, tree-based methods and ensemble techniques in predictive modeling. Unlike linear models, a model-free approach, such as binary decision trees, offers a more intuitive understanding by illustrating direct relationships between predictors and responses. Although simple, binary decision trees are highly interpretable and can unveil valuable insights.

However, to harness greater predictive power, we can extend beyond a single decision tree. By aggregating multiple models, particularly those that are uncorrelated, we significantly enhance our predictive accuracy.

A prime example of this concept is the RandomForest algorithm. Here, we create a multitude of decision trees through bootstrap sampling – a method where each tree is built from a random subset of data and variables. The aggregation of these diverse trees results in a robust final prediction model.

Ensemble methods extend this idea further by combining various models to improve predictive performance. This could involve averaging or taking a weighted average of numerous distinct models. Often, this approach surpasses the predictive capability of any individual model at hand, providing a powerful tool for tackling complex data mining challenges.

Boosting, particularly Gradient Boosting Machines, stands out as another potent predictive method. Unlike traditional ensemble techniques that build models independently, boosting focuses on sequentially improving the prediction by specifically targeting the errors of previous models. Each new model incrementally reduces the errors, leading to a highly accurate combined prediction.

All the methods mentioned above can handle diverse types of data and predict outcomes ranging from continuous to categorical responses, including multi-level categories.

In Homework 4, we will delve into these advanced techniques, moving beyond the limitations of linear models and exploring the expansive potential of trees, ensembles, and boosting in modern data mining. This journey will provide you with a solid foundation in leveraging sophisticated algorithms to uncover deeper insights and achieve superior predictive performance.

Objectives

- Understand trees
  - single tree/displaying/pruning a tree
  - RandomForest
  - Ensemble idea
  - Boosting
- R functions/Packages
  - `tree`, `RandomForest`, `ranger`
  - Boosting functions
- Json data format
- text mining
  - bag of words

Data needed:

- `IQ.Full.csv`
- `yelp_review_20k.json`

Problem 0: Study lectures

Please study all three modules. Understand the main elements in each module and be able to run and compile the lectures

- textmining
- trees
- boosting

Problem 1: IQ and successes

Background: Measurement of Intelligence

Case Study: how intelligence relates to one's future successes?

Data needed: `IQ.Full.csv`

ASVAB (Armed Services Vocational Aptitude Battery) tests have been used as a screening test for those who want to join the army or other jobs.

Our data set IQ.csv is a subset of individuals from the 1979 National Longitudinal Study of Youth (NLSY79) survey who were re-interviewed in 2006. Information about family, personal demographic such as gender, race and education level, plus a set of ASVAB (Armed Services Vocational Aptitude Battery) test scores are available. It is STILL used as a screening test for those who want to join the army! ASVAB scores were 1981 and income was 2005.

Our goals:

- Is IQ related to one's successes measured by Income?
- Is there evidence to show that Females are under-paid?
- What are the best possible prediction models to predict future income?

The ASVAB has the following components:

- Science, Arith (Arithmetic reasoning), Word (Word knowledge), Parag (Paragraph comprehension), Numer (Numerical operation), Coding (Coding speed), Auto (Automative and Shop information), Math (Math knowledge), Mechanic (Mechanic Comprehension) and Elec (Electronic information).
- AFQT (Armed Forces Qualifying Test) is a combination of Word, Parag, Math and Arith.
- Note: Service Branch requirement: Army 31, Navy 35, Marines 31, Air Force 36, and Coast Guard 45,(out of 100 which is the max!)

The detailed variable definitions:

Personal Demographic Variables:

- Race: 1 = Hispanic, 2 = Black, 3 = Not Hispanic or Black
- Gender: a factor with levels "female" and "male"
- Educ: years of education completed by 2006

Household Environment:

- Imagazine: a variable taking on the value 1 if anyone in the respondent's household regularly read magazines in 1979, otherwise 0
- Inewspaper: a variable taking on the value 1 if anyone in the respondent's household regularly read newspapers in 1979, otherwise 0
- Ilibrary: a variable taking on the value 1 if anyone in the respondent's household had a library card in 1979, otherwise 0
- MotherEd: mother's years of education
- FatherEd: father's years of education

Variables Related to ASVAB test Scores in 1981 (Proxy of IQ's)

- AFQT: percentile score on the AFQT intelligence test in 1981

- Coding: score on the Coding Speed test in 1981
- Auto: score on the Automotive and Shop test in 1981
- Mechanic: score on the Mechanic test in 1981
- Elec: score on the Electronics Information test in 1981
- Science: score on the General Science test in 1981
- Math: score on the Math test in 1981
- Arith: score on the Arithmetic Reasoning test in 1981
- Word: score on the Word Knowledge Test in 1981
- Parag: score on the Paragraph Comprehension test in 1981
- Numer: score on the Numerical Operations test in 1981

Variable Related to Life Success in 2006

- Income2005: total annual income from wages and salary in 2005. We will use a natural log transformation over the income.

Note: All the Esteem scores shouldn't be used as predictors to predict income

1. EDA: Some cleaning work is needed to organize the data.

- The first variable is the label for each person. Take that out.
- Set categorical variables as factors.
- Make log transformation for Income and take the original Income out
- Take the last person out of the dataset and label it as **Michelle**.
- When needed, split data to three portions: training, testing and validation (70%/20%/10%)
  - training data: get a fit
  - testing data: find the best tuning parameters/best models
  - validation data: only used in your final model to report the accuracy.

```
data = read.csv('data/IQ.full.csv')
data$Gender = as.factor(data$Gender)
data$logIncome = log(data$Income2005)
data = select(data, -c(Income2005))
Michelle = data[nrow(data),]
```

2. Factors affect Income

We start with linear models to answer the questions below.

i. To summarize ASVAB test scores, create PC1 and PC2 of 10 scores of ASVAB tests and label them as ASVAB_PC1 and ASVAB_PC2. Give a quick interpretation of each ASVAB_PC1 and ASVAB_PC2 in terms of the original 10 tests.

PC1 is roughly equivalent to the sum of all 10 scores (we can negate all of the signs because all of the loadings are negative).

```
asvab_cols = c("Coding", "Auto", "Mechanic", "Elec", "Science", "Math", "Arith", "Word", "Parag", "Numer
data_asvab = data %>% select(asvab_cols)
pca = prcomp(data %>% select(asvab_cols), scale. = T, center=TRUE)
pca$rotation
```

```
##                PC1         PC2         PC3         PC4          PC5
## Coding   -0.2339521 -0.51455964  0.51524933  0.24500645  0.575849028
## Auto     -0.2722745  0.47349800  0.42195184  0.15797866 -0.183791738
## Mechanic -0.3160465  0.33448607  0.25224685 -0.19954794  0.229388609
## Elec     -0.3238265  0.33533678  0.08604426  0.13533434 -0.057152114
## Science  -0.3518223  0.14188501 -0.22203592  0.16073380  0.030878280
## Math     -0.3360033 -0.14260221 -0.22316966 -0.57750436  0.138069594
## Arith    -0.3543323 -0.04872007 -0.09187558 -0.47645927  0.057410768
## Word     -0.3495605 -0.06148398 -0.34292321  0.37521282  0.006897479
## Parag    -0.3261262 -0.18973335 -0.39113650  0.35216707 -0.043273536
## Numer    -0.2749630 -0.45174873  0.32758639 -0.07558727 -0.743975903
##                 PC6          PC7         PC8          PC9         PC10
## Coding   -0.11404768 -0.021378265 -0.08490888  0.044110745  0.02763096
## Auto      0.10023865  0.272163351 -0.59345162  0.025870347 -0.16706730
## Mechanic  0.48426654  0.100841118  0.61279729 -0.079485484 -0.07513453
## Elec     -0.37424432 -0.774050667  0.11072065  0.044934182  0.01445331
## Science  -0.42877809  0.482680997  0.23079572  0.460204538  0.31323227
## Math     -0.15866489 -0.002352397 -0.16314355  0.219252157 -0.60375476
## Arith     0.06314755 -0.050327779 -0.29800327 -0.324336209  0.65787841
## Word     -0.13085686  0.173463139  0.06553515 -0.704170391 -0.26121799
## Parag     0.60995010 -0.221742040 -0.16697976  0.357891519  0.03979597
## Numer    -0.01403521  0.036991300  0.22794859  0.007208544 -0.01463333
```

ii. Is there any evidence showing ASVAB test scores in terms of ASVAB_PC1 and ASVAB_PC2, might affect the Income? Show your work here. You may control a few other variables, including gender.

There is evidence. The estimate for PC1 is significant at alpha = 0.01 as shown in the table, and the coefficient of -0.016184 indicates that as PC1 increases by 1, the logIncome falls by -0.016184. The estimate for PC2 is not significant. The estimate for Gender is also significant, indicating that being male leads to a 0.621681 increase in log income.

```
pc1_asvab = pca$rotation[, 1]
pc2_asvab = pca$rotation[, 2]
data$PC1_asvab = as.vector(as.matrix(data[,asvab_cols]) %*% pc1_asvab)
data$PC2_asvab = as.vector(as.matrix(data[,asvab_cols]) %*% pc2_asvab)

fit1 = lm(logIncome ~ PC1_asvab + PC2_asvab + Gender, data=data)
summary(fit1)
```

```
##
## Call:
## lm(formula = logIncome ~ PC1_asvab + PC2_asvab + Gender, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7262 -0.3497  0.1283  0.5180  2.6171
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.986107   0.072829 123.386   <2e-16 ***
## PC1_asvab   -0.016184   0.001447 -11.186   <2e-16 ***
## PC2_asvab   -0.004151   0.002139  -1.941   0.0524 .
## Gendermale   0.621681   0.042332  14.686   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8771 on 2580 degrees of freedom
## Multiple R-squared:  0.1898, Adjusted R-squared:  0.1889
## F-statistic: 201.5 on 3 and 2580 DF,  p-value: < 2.2e-16
```

iii. Is there any evidence to show that there is gender bias against either male or female in terms of income in the above model?

Yes there is evidence because the estimate for gender is significant and the coefficient is not 0, indicating that being male is correlated with higher incomes and that there is a gender bias against women.

We next build a few models for the purpose of prediction using all the information available. From now on you may use the three data sets setting (training/testing/validation) when it is appropriate.

3. Trees

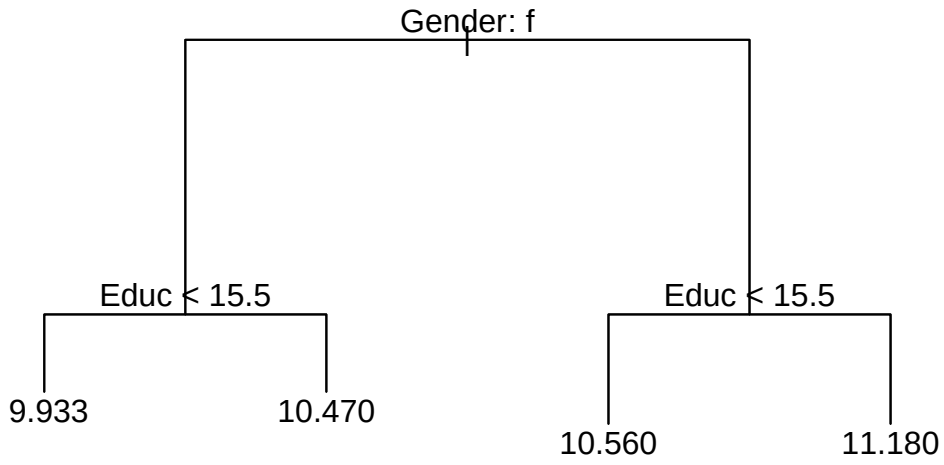
```r
# split data into train/val/test for trees
# Split data into training, testing, and validation sets
set.seed(123)
train_index <- sample(1:nrow(data), 0.7*nrow(data))
test_index <- sample(setdiff(1:nrow(data), train_index), 0.2*nrow(data))
valid_index <- setdiff(1:nrow(data), union(train_index, test_index))

data.train <- data[train_index,]
data.test <- data[test_index,]
data.valid <- data[valid_index,]
```

i. fit1: tree(Income ~ Educ + Gender, data.train) with default set up

a) Display the tree
b) How many end nodes? Briefly explain how the estimation is obtained in each end nodes and deescribe the prediction equation
c) Does it show interaction effect of Gender and Educ on Income?
d) Predict Michelle's income

Part a:

```r
# part i
fit1 = tree(logIncome ~ Educ + Gender, data.train)
plot(fit1)
text(fit1, pretty=TRUE)
```



Part b: There are 4 end nodes. Each end node/leaf represents the prediction we obtain when we follow the path from the root to the leaf (and since there is only one unique path between any 2 nodes in a tree, there is only 1 way to reach each leaf). We describe the end nodes from left to right. The leftmost prediction means that for a female with fewer than 15.5 years of education, we would predict her log income to be 9.933. Next, for a female with at least 15.5 years of education, we would predict her log income to be 10.470. Next, for a male with fewer than 15.5 years of education, we would predict his log income to be 10.560. Finally, for a male with at least 15.5 years of education, we would predict his log income to be 11.180.

```
data.table(fit1$frame)
```

```
##       var     n       dev      yval splits.cutleft splits.cutright
##    <fctr> <num>     <num>     <num>         <char>          <char>
## 1: Gender  1808 1759.4397 10.429101             :a              :b
## 2:   Educ   889  847.0578 10.095814          <15.5           >15.5
## 3: <leaf>   621  530.5423  9.932855
## 4: <leaf>   268  261.8120 10.473417
## 5:   Educ   919  718.1046 10.751508          <15.5           >15.5
## 6: <leaf>   631  407.2630 10.555475
## 7: <leaf>   288  233.4646 11.181012
```

Part c: Yes, we do see an interaction effect. Our predictions of log income depending on the combination of Gender and Education that we have.

Part d: We predict Michelle's log income to be 9.932855. This translates to $20,596.06.

```
print("Log income prediction")
```

```
## [1] "Log income prediction"
```

```
predict(fit1, Michelle)
```

```
##     2584
## 9.932855
```

```
print("Income prediction")
```

```
## [1] "Income prediction"
```

```
exp(predict(fit1, Michelle))
```

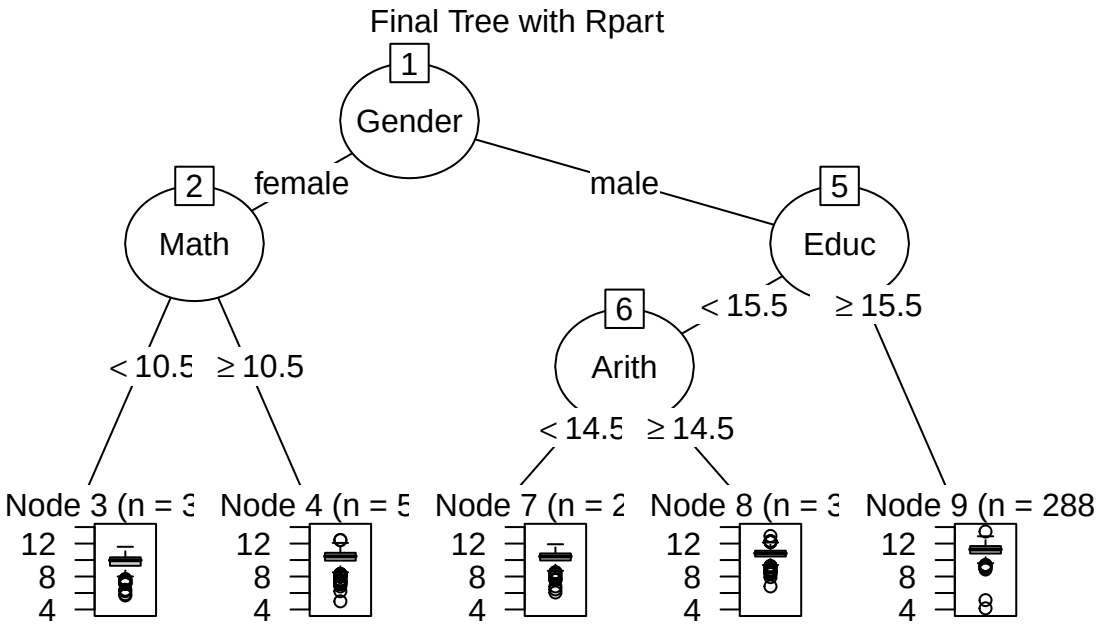
```
##     2584
## 20596.06
```

ii. fit2: fit2 <- rpart(Income2005 ~., data.train, minsplit=20, cp=.009)

a) Display the tree using plot(as.party(fit2), main="Final Tree with Rpart")
b) A brief summary of the fit2
c) Compare testing errors between fit1 and fit2. Is the training error from fit2 always less than that from fit1? Is the testing error from fit2 always smaller than that from fit1?
d) You may prune the fit2 to get a tree with small testing error.

Part a:

```
fit2 = rpart(logIncome ~., data.train, minsplit=20, cp=.009)
plot(as.party(fit2), main="Final Tree with Rpart")
```



Part b: This tree has 9 terminal nodes. Gender is the most important variable, while FatherEd is the least important variable. In this tree, we first split on Gender. For females, we then split by scores on Math, while for males, we then split by years of Education. For males with less than 15.5 years of education, we then split on Arithmetic scores.

```
summary(fit2)
```

```
## Call:
## rpart(formula = logIncome ~ ., data = data.train, minsplit = 20,
##     cp = 0.009)
##   n= 1808
##
##           CP nsplit rel error    xerror       xstd
## 1 0.11041994      0 1.0000000 1.0010788 0.05620134
## 2 0.04397820      1 0.8895801 0.8908234 0.05391106
## 3 0.03386862      2 0.8456019 0.8692166 0.05553848
## 4 0.02034414      3 0.8117332 0.8418125 0.05522198
## 5 0.00900000      4 0.7913891 0.8049285 0.05403338
##
## Variable importance
##    Gender      Math      Auto      AFQT  Mechanic     Arith      Educ PC1_asvab
##        18        10        10         8         8         8         7         7
## PC2_asvab      Elec    Coding   Science      Word     Parag
##         7         6         4         3         2         2
##
## Node number 1: 1808 observations,    complexity param=0.1104199
##   mean=10.4291, MSE=0.9731414
##   left son=2 (889 obs) right son=3 (919 obs)
##   Primary splits:
##       Gender splits as  LR, improve=0.11041990, (0 missing)
##       Educ   < 15.5      to the left,  improve=0.07702635, (0 missing)
##       Auto   < 14.5      to the left,  improve=0.07557397, (0 missing)
##       Arith  < 15.5      to the left,  improve=0.07530312, (0 missing)
##       Math   < 10.5      to the left,  improve=0.07240290, (0 missing)
##   Surrogate splits:
##       Auto      < 14.5      to the left,  agree=0.770, adj=0.533, (0 split)
##       PC2_asvab < -29.34425 to the left,  agree=0.695, adj=0.380, (0 split)
##       Mechanic  < 16.5      to the left,  agree=0.686, adj=0.361, (0 split)
##       Elec      < 13.5      to the left,  agree=0.685, adj=0.359, (0 split)
##       Coding    < 47.5      to the right, agree=0.613, adj=0.214, (0 split)
##
## Node number 2: 889 observations,    complexity param=0.03386862
##   mean=10.09581, MSE=0.952821
##   left son=4 (320 obs) right son=5 (569 obs)
##   Primary splits:
##       Math      < 10.5      to the left,  improve=0.07034915, (0 missing)
##       AFQT      < 21.91     to the left,  improve=0.06622025, (0 missing)
##       PC1_asvab < -56.15321 to the right, improve=0.06538350, (0 missing)
##       Educ      < 15.5      to the left,  improve=0.06458066, (0 missing)
##       Word      < 17.5      to the left,  improve=0.05660296, (0 missing)
##   Surrogate splits:
##       AFQT      < 42.3865   to the left,  agree=0.861, adj=0.612, (0 split)
##       PC1_asvab < -57.29809 to the right, agree=0.841, adj=0.559, (0 split)
##       Arith     < 14.5      to the left,  agree=0.816, adj=0.487, (0 split)
```

```
##       Word      < 21.5      to the left,  agree=0.773, adj=0.369, (0 split)
##       Parag     < 11.5      to the left,  agree=0.757, adj=0.325, (0 split)
##
## Node number 3: 919 observations,    complexity param=0.0439782
##   mean=10.75151, MSE=0.7813979
##   left son=6 (631 obs) right son=7 (288 obs)
##   Primary splits:
##       Educ      < 15.5      to the left,  improve=0.10775170, (0 missing)
##       Arith     < 13.5      to the left,  improve=0.10108110, (0 missing)
##       AFQT      < 45.73     to the left,  improve=0.09229685, (0 missing)
##       Math      < 10.5      to the left,  improve=0.08281826, (0 missing)
##       PC1_asvab < -51.80592 to the right, improve=0.08224283, (0 missing)
##   Surrogate splits:
##       Math      < 20.5      to the left,  agree=0.802, adj=0.368, (0 split)
##       AFQT      < 79.5415   to the left,  agree=0.800, adj=0.361, (0 split)
##       PC1_asvab < -77.91082 to the right, agree=0.767, adj=0.257, (0 split)
##       Arith     < 26.5      to the left,  agree=0.763, adj=0.243, (0 split)
##       Science   < 21.5      to the left,  agree=0.753, adj=0.212, (0 split)
##
## Node number 4: 320 observations
##   mean=9.750578, MSE=0.9124729
##
## Node number 5: 569 observations
##   mean=10.28997, MSE=0.8707851
##
## Node number 6: 631 observations,    complexity param=0.02034414
##   mean=10.55548, MSE=0.6454247
##   left son=12 (248 obs) right son=13 (383 obs)
##   Primary splits:
##       Arith     < 14.5      to the left,  improve=0.08788987, (0 missing)
##       Imagazine < 0.5       to the left,  improve=0.06530764, (0 missing)
##       PC1_asvab < -50.90182 to the right, improve=0.05947948, (0 missing)
##       AFQT      < 26.8245   to the left,  improve=0.05805357, (0 missing)
##       Numer     < 22.5      to the left,  improve=0.05776184, (0 missing)
##   Surrogate splits:
##       AFQT      < 32.821    to the left,  agree=0.876, adj=0.685, (0 split)
##       PC1_asvab < -57.22508 to the right, agree=0.870, adj=0.669, (0 split)
##       Math      < 8.5       to the left,  agree=0.824, adj=0.552, (0 split)
##       Science   < 14.5      to the left,  agree=0.797, adj=0.484, (0 split)
##       Mechanic  < 12.5      to the left,  agree=0.797, adj=0.484, (0 split)
##
## Node number 7: 288 observations
##   mean=11.18101, MSE=0.8106411
##
## Node number 12: 248 observations
##   mean=10.25949, MSE=0.7114552
##
## Node number 13: 383 observations
##   mean=10.74713, MSE=0.509211
```

Part c:

fit1 has a test MSE of 0.7600874, while fit2 has a test MSE of 0.7573498. For this specific seed, the training and test errors are lower for fit2 than fit1. However, depending on the seed and how the data is split, it is possible that fit1 may produce a tree with lower training and/or test errors.

```r
print("Fit 1 test error: ")
```

```
## [1] "Fit 1 test error: "
```

```r
test.error.fit1 <- mean((predict(fit1, data.test) - data.test$logIncome)^2)
test.error.fit1
```

```
## [1] 0.7600874
```

```r
print("Fit 2 test error: ")
```

```
## [1] "Fit 2 test error: "
```

```r
test.error.fit2 <- mean((predict(fit2, data.test) - data.test$logIncome)^2)
test.error.fit2
```

```
## [1] 0.7573498
```

```r
print("Fit 1 train error: ")
```

```
## [1] "Fit 1 train error: "
```

```r
train.error.fit1 <- mean((predict(fit1, data.train) - data.train$logIncome)^2)
train.error.fit1
```

```
## [1] 0.7926338
```

```r
print("Fit 2 train error: ")
```

```
## [1] "Fit 2 train error: "
```

```r
train.error.fit2 <- mean((predict(fit2, data.train) - data.train$logIncome)^2)
train.error.fit2
```

```
## [1] 0.7701335
```

Part d: By pruning fit2, we obtain a lower testing MSE of 0.7573498.

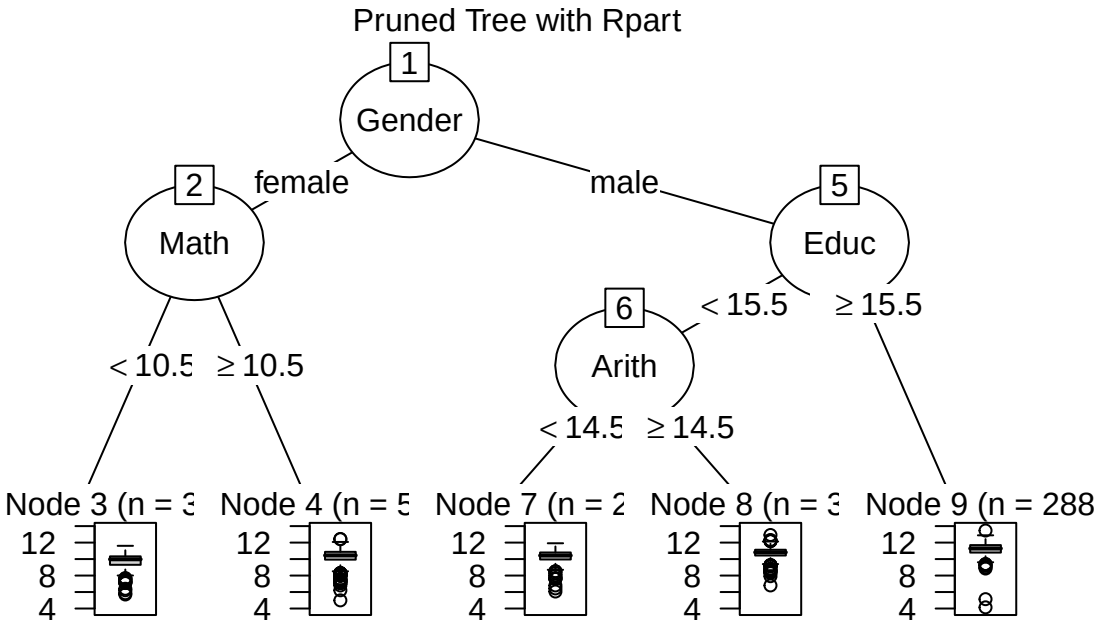
```r
fit2.p = prune(fit2, cp=0.01)
summary(fit2.p)
```

```
## Call:
## rpart(formula = logIncome ~ ., data = data.train, minsplit = 20,
##     cp = 0.009)
##   n= 1808
##
##           CP nsplit rel error    xerror       xstd
```

```
## 1 0.11041994      0 1.0000000 1.0010788 0.05620134
## 2 0.04397820      1 0.8895801 0.8908234 0.05391106
## 3 0.03386862      2 0.8456019 0.8692166 0.05553848
## 4 0.02034414      3 0.8117332 0.8418125 0.05522198
## 5 0.00900000      4 0.7913891 0.8049285 0.05403338
##
## Variable importance
##    Gender      Math      Auto      AFQT  Mechanic     Arith      Educ PC1_asvab
##        18        10        10         8         8         8         7         7
## PC2_asvab      Elec    Coding   Science      Word     Parag
##         7         6         4         3         2         2
##
## Node number 1: 1808 observations,    complexity param=0.1104199
##   mean=10.4291, MSE=0.9731414
##   left son=2 (889 obs) right son=3 (919 obs)
##   Primary splits:
##       Gender splits as  LR, improve=0.11041990, (0 missing)
##       Educ   < 15.5      to the left,  improve=0.07702635, (0 missing)
##       Auto   < 14.5      to the left,  improve=0.07557397, (0 missing)
##       Arith  < 15.5      to the left,  improve=0.07530312, (0 missing)
##       Math   < 10.5      to the left,  improve=0.07240290, (0 missing)
##   Surrogate splits:
##       Auto      < 14.5      to the left,  agree=0.770, adj=0.533, (0 split)
##       PC2_asvab < -29.34425 to the left,  agree=0.695, adj=0.380, (0 split)
##       Mechanic  < 16.5      to the left,  agree=0.686, adj=0.361, (0 split)
##       Elec      < 13.5      to the left,  agree=0.685, adj=0.359, (0 split)
##       Coding    < 47.5      to the right, agree=0.613, adj=0.214, (0 split)
##
## Node number 2: 889 observations,    complexity param=0.03386862
##   mean=10.09581, MSE=0.952821
##   left son=4 (320 obs) right son=5 (569 obs)
##   Primary splits:
##       Math      < 10.5      to the left,  improve=0.07034915, (0 missing)
##       AFQT      < 21.91     to the left,  improve=0.06622025, (0 missing)
##       PC1_asvab < -56.15321 to the right, improve=0.06538350, (0 missing)
##       Educ      < 15.5      to the left,  improve=0.06458066, (0 missing)
##       Word      < 17.5      to the left,  improve=0.05660296, (0 missing)
##   Surrogate splits:
##       AFQT      < 42.3865   to the left,  agree=0.861, adj=0.612, (0 split)
##       PC1_asvab < -57.29809 to the right, agree=0.841, adj=0.559, (0 split)
##       Arith     < 14.5      to the left,  agree=0.816, adj=0.487, (0 split)
##       Word      < 21.5      to the left,  agree=0.773, adj=0.369, (0 split)
##       Parag     < 11.5      to the left,  agree=0.757, adj=0.325, (0 split)
##
## Node number 3: 919 observations,    complexity param=0.0439782
##   mean=10.75151, MSE=0.7813979
##   left son=6 (631 obs) right son=7 (288 obs)
##   Primary splits:
##       Educ      < 15.5      to the left,  improve=0.10775170, (0 missing)
##       Arith     < 13.5      to the left,  improve=0.10108110, (0 missing)
##       AFQT      < 45.73     to the left,  improve=0.09229685, (0 missing)
##       Math      < 10.5      to the left,  improve=0.08281826, (0 missing)
##       PC1_asvab < -51.80592 to the right, improve=0.08224283, (0 missing)
##   Surrogate splits:
```

```
##       Math      < 20.5      to the left,  agree=0.802, adj=0.368, (0 split)
##       AFQT      < 79.5415   to the left,  agree=0.800, adj=0.361, (0 split)
##       PC1_asvab < -77.91082 to the right, agree=0.767, adj=0.257, (0 split)
##       Arith     < 26.5      to the left,  agree=0.763, adj=0.243, (0 split)
##       Science   < 21.5      to the left,  agree=0.753, adj=0.212, (0 split)
##
## Node number 4: 320 observations
##   mean=9.750578, MSE=0.9124729
##
## Node number 5: 569 observations
##   mean=10.28997, MSE=0.8707851
##
## Node number 6: 631 observations,    complexity param=0.02034414
##   mean=10.55548, MSE=0.6454247
##   left son=12 (248 obs) right son=13 (383 obs)
##   Primary splits:
##       Arith     < 14.5      to the left,  improve=0.08788987, (0 missing)
##       Imagazine < 0.5       to the left,  improve=0.06530764, (0 missing)
##       PC1_asvab < -50.90182 to the right, improve=0.05947948, (0 missing)
##       AFQT      < 26.8245   to the left,  improve=0.05805357, (0 missing)
##       Numer     < 22.5      to the left,  improve=0.05776184, (0 missing)
##   Surrogate splits:
##       AFQT      < 32.821    to the left,  agree=0.876, adj=0.685, (0 split)
##       PC1_asvab < -57.22508 to the right, agree=0.870, adj=0.669, (0 split)
##       Math      < 8.5       to the left,  agree=0.824, adj=0.552, (0 split)
##       Science   < 14.5      to the left,  agree=0.797, adj=0.484, (0 split)
##       Mechanic  < 12.5      to the left,  agree=0.797, adj=0.484, (0 split)
##
## Node number 7: 288 observations
##   mean=11.18101, MSE=0.8106411
##
## Node number 12: 248 observations
##   mean=10.25949, MSE=0.7114552
##
## Node number 13: 383 observations
##   mean=10.74713, MSE=0.509211
```

```
plot(as.party(fit2.p), main="Pruned Tree with Rpart")
```




```
print("Fit 2 pruned test error: ")
```

```
## [1] "Fit 2 pruned test error: "
```

```
test.error.fit2p <- mean((predict(fit2.p, data.test) - data.test$logIncome)^2)
print(test.error.fit2p)
```

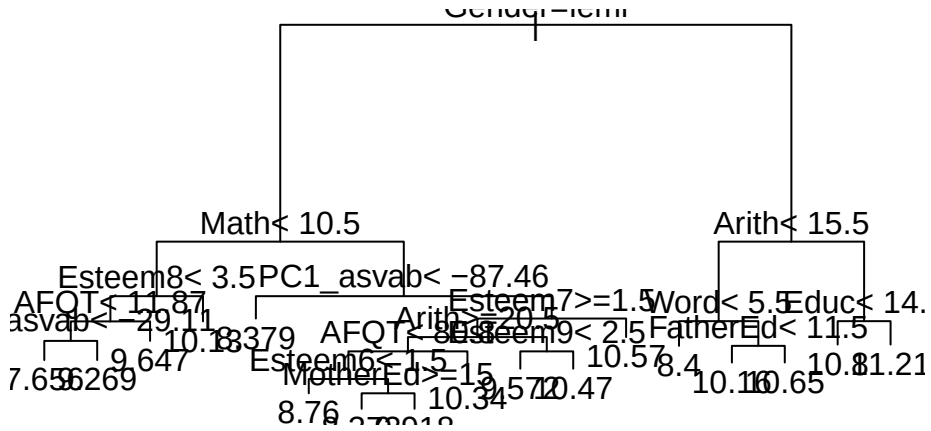
```
## [1] 0.7573498
```

iii. fit3: bag two trees

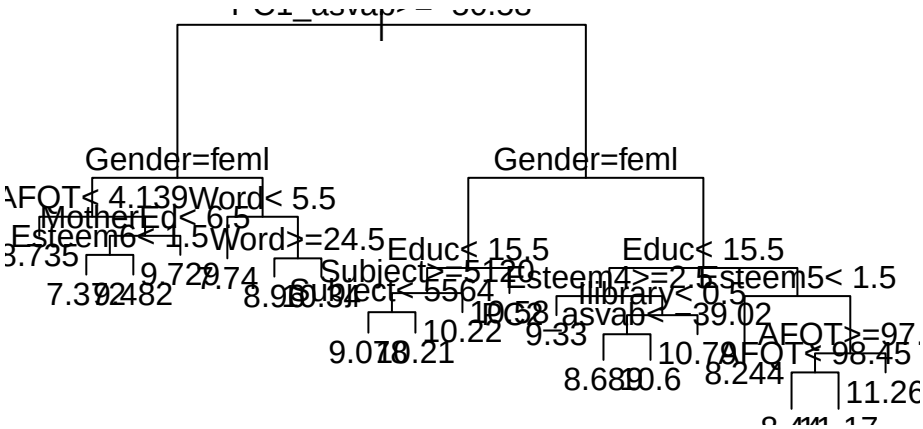
```
a) Take 2 bootstrap training samples and build two trees using the
rpart(Income2005 ~., data.train.b, minsplit=20, cp=.009). Display both trees.
b) Explain how to get fitted values for Michelle by bagging the two trees obtained above. Do not use th
c) What is the testing error for the bagged tree. Is it guaranteed that the testing error by bagging th
```

Part a:

```
n = nrow(data.train)

index1 <- sample(n, n, replace = TRUE)
boot.data1 <- data.train[index1, ]
boot.1.tree <- rpart(logIncome ~., boot.data1, minsplit=20, cp=.009)

plot(boot.1.tree)
title(main = "First bootstrap tree")
text(boot.1.tree, pretty=TRUE)
```




```
index2 <- sample(n, n, replace = TRUE)
boot.data2 <- data.train[index2, ]
boot.2.tree <- rpart(logIncome ~., boot.data2, minsplit=20, cp=.009)

plot(boot.2.tree)
title(main = "Second bootstrap tree")
text(boot.2.tree, pretty=TRUE)
```



Part b: To get fitted values for Michelle, we would take the average of the predictions from boot.1.tree and boot.2.tree.

Part c: The testing MSE of the bagged tree is 0.7862542. It is not guaranteed that the bagged tree's testing MSE is always lower than that of a single tree. Since the bagged tree outputs the average of the trees it contains, it's possible that if one of the single trees is optimal that the averaging will move the output results away from the optimal results.

```
bag.predict <- (predict(boot.1.tree, data.test) + predict(boot.2.tree, data.test))/2 #bagging
mean((bag.predict - data.test$logIncome)^2)
```

```
## [1] 0.7977995
```

iv. fit4: Build a best possible RandomForest

a) Show the process how you tune mtry and number of trees. Give a very high level explanation how fit4 is built.
b) Compare the oob errors form fit4 to the testing errors using your testing data. Are you convinced that oob errors estimate testing error reasonably well.
c) What is the predicted value for Michelle?

```
## part iv

mtry_vals <- seq(2, ncol(data.train)-1, by=2)
oob_errors <- c()
for (m in mtry_vals) {
  fit <- randomForest(logIncome ~ ., data=data.train, mtry=m, ntree=500)
  oob_errors <- c(oob_errors, fit$mse[500])
}
```

```r
mtry_opt <- mtry_vals[which.min(oob_errors)]

fit4 <- randomForest(logIncome ~ ., data=data.train, mtry=mtry_opt, ntree=500, importance=TRUE)
oob_error <- sqrt(fit4$mse[500])
test_error <- sqrt(mean((predict(fit4, data.test) - data.test$logIncome)^2))
# Add PC1_asvab and PC2_asvab variables to Michelle data
Michelle$PC1_asvab = as.vector(as.matrix(Michelle[,asvab_cols]) %*% pc1_asvab)
Michelle$PC2_asvab = as.vector(as.matrix(Michelle[,asvab_cols]) %*% pc2_asvab)

print(paste("OOB Error:", round(oob_error,3)))
```

```
## [1] "OOB Error: 0.871"
```

```r
print(paste("Test Error:", round(test_error,3)))
```

```
## [1] "Test Error: 0.835"
```

```r
print(paste("Michelle's Predicted logIncome:", round(predict(fit4, Michelle),3)))
```

```
## [1] "Michelle's Predicted logIncome: 9.622"
```

- RandomForest combines multiple decision trees to improve prediction accuracy and reduce overfitting. The algorithm constructs a large number of decision trees (500 in this case) on bootstrap samples of the training data. At each split in a tree, a random subset of predictors (determined by mtry) is considered for the best split. This process introduces randomness and diversity among the trees, which helps in reducing the correlation between them and improving the overall performance of the ensemble.

v. Now you have built so many predicted models (fit1 through fit4 in this section). What about build a fit5 which bags fit1 through fit4. Does fit5 have the smallest testing error?

```r
# Create a list of the models to be bagged
models <- list(fit1, fit2, fit4)

# Define a custom bagging function
bag_predictions <- function(models, newdata) {
  preds <- lapply(models, function(model) predict(model, newdata))
  rowMeans(do.call(cbind, preds))
}

# Make predictions on the test set using the bagged models
bagged_preds <- bag_predictions(models, data.test)

# Calculate the test error for the bagged model
fit5_test_error <- sqrt(mean((bagged_preds - data.test$logIncome)^2))
print(paste("Bagged Model Test Error:", round(fit5_test_error,3)))
```

```
## [1] "Bagged Model Test Error: 0.845"
```

```r
# Calculate test errors for fit1, fit2, fit3, and fit4
fit1_test_error <- sqrt(mean((predict(fit1, data.test) - data.test$logIncome)^2))
fit2_test_error <- sqrt(mean((predict(fit2, data.test) - data.test$logIncome)^2))
#fit3_test_error <- sqrt(mean((predict(fit3, data.test) - data.test$logIncome)^2))
fit4_test_error <- sqrt(mean((predict(fit4, data.test) - data.test$logIncome)^2))

# Compare test errors
print(paste("fit1 Test Error:", round(fit1_test_error,3)))
```

```
## [1] "fit1 Test Error: 0.872"
```

```r
print(paste("fit2 Test Error:", round(fit2_test_error,3)))
```

```
## [1] "fit2 Test Error: 0.87"
```

```r
#print(paste("fit3 Test Error:", round(fit3_test_error,3)))
print(paste("fit4 Test Error:", round(fit4_test_error,3)))
```

```
## [1] "fit4 Test Error: 0.835"
```

```r
print(paste("fit5 (Bagged) Test Error:", round(fit5_test_error,3)))
```

```
## [1] "fit5 (Bagged) Test Error: 0.845"
```

iv. Now use `XGBoost` to build the fit6 predictive equation. Evaluate its testing error. Also briefly explain how it works.

```r
# vi. XGBoost

train_data <- as.matrix(sapply(data.train[,-ncol(data.train)], as.numeric))
test_data <- as.matrix(sapply(data.test[,-ncol(data.test)], as.numeric))

dtrain <- xgb.DMatrix(train_data, label=data.train$logIncome)
dtest <- xgb.DMatrix(test_data, label=data.test$logIncome)

watchlist <- list(train=dtrain, test=dtest)
fit6 <- xgb.train(data=dtrain, max.depth=3, eta=0.1, nrounds=100, watchlist=watchlist,
                  verbose=0, objective = "reg:squarederror")
fit6_test_error <- sqrt(mean((predict(fit6, dtest) - data.test$logIncome)^2))
print(paste("XGBoost Test Error:", round(fit6_test_error,3)))
```

```
## [1] "XGBoost Test Error: 0.032"
```

vii. Summarize the results and nail down one best possible final model you will recommend to predict income. Explain briefly why this is the best choice. Finally for the first time evaluate the prediction error using the validating data set.

```r
# vii. Final model selection
# Based on the test errors, the RandomForest model fit4 has the lowest error and is selected as the fin
# Evaluation on validation set
valid_error <- sqrt(mean((predict(fit4, data.valid) - data.valid$logIncome)^2))
print(paste("Final Model (RandomForest) Validation Error:", round(valid_error,3)))
```

```
## [1] "Final Model (RandomForest) Validation Error: 0.863"
```

- The best possible final model to predict income is the RandomForest model fit4. This model has the lowest test error among all the models built.

viii. Use your final model to predict Michelle's income.

```r
# viii. Predict Michelle's income using fit4
print(paste("Michelle's Predicted logIncome:", round(predict(fit4, Michelle),3)))
```

```
## [1] "Michelle's Predicted logIncome: 9.622"
```

Problem 2: Yelp challenge 2019

Note: This problem is rather involved. It covers essentially all the main materials we have done so far in this semester. It could be thought as a guideline for your final project if you want when appropriate.

Yelp has made their data available to public and launched Yelp challenge. More information. It is unlikely we will win the $5,000 prize posted but we get to use their data for free. We have done a detailed analysis in our lecture. This exercise is designed for you to get hands on the whole process.

For this case study, we downloaded the data and took a 20k subset from **review.json**. *json* is another format for data. It is flexible and commonly-used for websites. Each item/subject/sample is contained in a brace *{}*. Data is stored as **key-value** pairs inside the brace. *Key* is the counterpart of column name in *csv* and *value* is the content/data. Both *key* and *value* are quoted. Each pair is separated by a comma. The following is an example of one item/subject/sample.

```
{
  "key1": "value1",
  "key2": "value2"
}
```

Data needed: yelp_review_20k.json available in Canvas.

yelp_review_20k.json contains full review text data including the user_id that wrote the review and the business_id the review is written for. Here's an example of one review.

```
{
    // string, 22 character unique review id
    "review_id": "zdSx_SD6obEhz9VrW9uAWA",

    // string, 22 character unique user id, maps to the user in user.json
    "user_id": "Ha3iJu77CxlrFm-vQRs_8g",

    // string, 22 character business id, maps to business in business.json
```

```
    "business_id": "tnhfDv5Il8EaGSXZGiuQGg",

    // integer, star rating
    "stars": 4,

    // string, date formatted YYYY-MM-DD
    "date": "2016-03-09",

    // string, the review itself
    "text": "Great place to hang out after work: the prices are decent, and the ambience is fun. It's a

    // integer, number of useful votes received
    "useful": 0,

    // integer, number of funny votes received
    "funny": 0,

    // integer, number of cool votes received
    "cool": 0
}
```

Goal of the study

The goals are

1) Try to identify important words associated with positive ratings and negative ratings. Collectively we have a sentiment analysis.

2) To predict ratings using different methods.

1. JSON data and preprocessing data

i. Load *json* data

The *json* data provided is formatted as newline delimited JSON (ndjson). It is relatively new and useful for streaming.

```
{
  "key1": "value1",
  "key2": "value2"
}
{
  "key1": "value1",
  "key2": "value2"
}
```

The traditional JSON format is as follows.

```
[{
  "key1": "value1",
  "key2": "value2"
```

```
},
{
  "key1": "value1",
  "key2": "value2"
}]
```

We use `stream_in()` in the `jsonlite` package to load the JSON data (of ndjson format) as `data.frame`. (For the traditional JSON file, use `fromJSON()` function.)

```
pacman::p_load(jsonlite)
yelp_data <- jsonlite::stream_in(file("data/yelp_review_20k.json"), verbose = F)
str(yelp_data)
```

```
## 'data.frame':    19999 obs. of  9 variables:
##  $ review_id  : chr  "Q1sbwvVQXV2734tPgoKj4Q" "GJXCdrto3ASJOqKeVWPi6Q" "2TzJjDVDEuAW6MR5Vuc1ug" "yi0
##  $ user_id    : chr  "hG7b0MtEbXx5QzbzE6C_VA" "yXQM5uF2jS6es16SJzNHfg" "n6-Gk65cPZL6Uz8qRm3NYw" "dac
##  $ business_id: chr  "ujmEBvifdJM6h6RLv4wQIg" "NZnhc2sEQy3RmzKTZnqtwQ" "WTqjgwHlXbSFevF32_DJVw" "ikC
##  $ stars      : num  1 5 5 5 1 4 3 1 2 3 ...
##  $ useful     : int  6 0 3 0 7 0 5 3 1 1 ...
##  $ funny      : int  1 0 0 0 0 0 4 1 0 0 ...
##  $ cool       : int  0 0 0 0 0 0 5 1 0 1 ...
##  $ text       : chr  "Total bill for this horrible service? Over $8Gs. These crooks actually had the
##  $ date       : chr  "2013-05-07 04:34:36" "2017-01-14 21:30:33" "2016-11-09 20:09:03" "2018-01-09 2
```

```
# different JSON format
# tmp_json <- toJSON(yelp_data[1:10,])
# fromJSON(tmp_json)
```

Write a brief summary about the data:

a) Which time period were the reviews collected in this data?

```
range(as.Date(yelp_data$date))
```

```
## [1] "2004-10-19" "2018-10-04"
```

- The reviews were collected between 2004-10-12 and 2018-11-14.

b) Are ratings (with 5 levels) related to month of the year or days of the week? Only address this through EDA please.

```
yelp_data$month <- format(as.Date(yelp_data$date), "%m")
yelp_data$day_of_week <- format(as.Date(yelp_data$date), "%A")
# Ratings by month
table(yelp_data$month, yelp_data$stars)
```

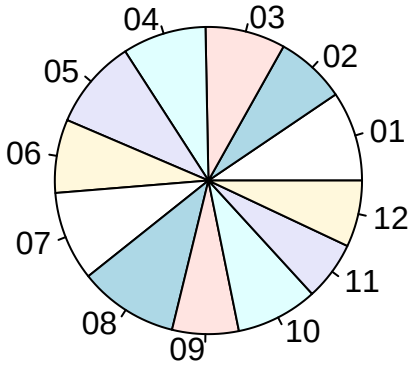
```
##
##        1   2   3   4   5
##   01 259 128 246 459 802
##   02 196 130 177 322 655
```

```
##   03 246 125 197 366 753
##   04 242 133 223 372 792
##   05 280 149 213 420 828
##   06 245 114 141 291 755
##   07 293 148 206 384 851
##   08 317 183 218 473 905
##   09 214 108 135 269 674
##   10 241 113 178 418 771
##   11 155 100 169 262 555
##   12 244  99 142 320 595
```

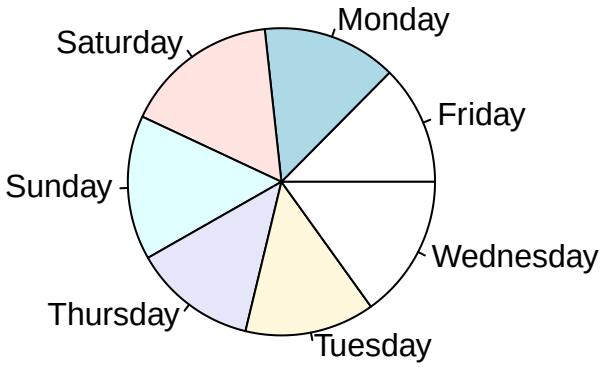
```r
# Ratings by weekday
table(yelp_data$day_of_week, yelp_data$stars)
```

```
##
##                1    2    3    4    5
##   Friday     381  182  250  530 1173
##   Monday     397  226  333  648 1227
##   Saturday   508  259  317  677 1498
##   Sunday     449  241  383  672 1297
##   Thursday   376  180  297  564 1182
##   Tuesday    429  206  303  575 1220
##   Wednesday  392  236  362  690 1339
```

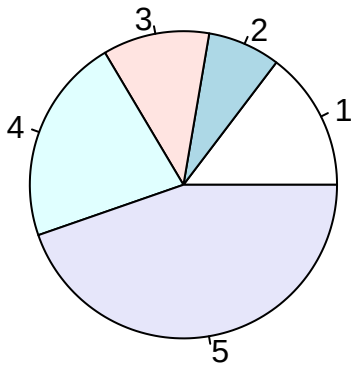
```r
pie(table(yelp_data$month))
```



```
pie(table(yelp_data$day_of_week))
```




```
pie(table(yelp_data$stars))
```



- From the tables and pie charts, there doesn't seem to be a discernible pattern or correlation between ratings and months or weeks.

ii. Document term matrix (dtm)

Extract document term matrix for texts to keep words appearing at least .5% of the time among all 20000 documents. Go through the similar process of cleansing as we did in the lecture.

```
pacman::p_load(tm)
mycorpus1 <- VCorpus(VectorSource(yelp_data$text))
mycorpus_clean <- tm_map(mycorpus1, content_transformer(tolower))
mycorpus_clean <- tm_map(mycorpus_clean, removeWords, stopwords("english"))
mycorpus_clean <- tm_map(mycorpus_clean, removePunctuation)
mycorpus_clean <- tm_map(mycorpus_clean, removeNumbers)
mycorpus_clean <- tm_map(mycorpus_clean, stemDocument, lazy = TRUE)
mycorpus_clean <- tm_map(mycorpus1, removeWords, stopwords("english"))
dtm <- DocumentTermMatrix(mycorpus_clean)
dtm <- removeSparseTerms(dtm, 0.995)
```

a) Briefly explain what does this matrix record? What is the cell number at row 100 and column 405? What does it represent?

- The document term matrix records the frequency of each term (word) in each document (review). For example, the cell number at row 100 and column 405 represents the frequency of the 405th term in the 100th document.

b) What is the sparsity of the dtm obtained here? What does that mean?

```
sparsity <- sum(dtm == 0) / prod(dim(dtm))
print(paste("Sparsity of the dtm:", round(sparsity, 3)))
```

```
## [1] "Sparsity of the dtm: 0.979"
```

- The sparsity of the dtm is 0.986, which means that 98.6% of the cells in the matrix are zero. This indicates that most documents contain only a small subset of the total terms.

iii. Set the stars as a two category response variable called rating to be "1" = 5,4 and "0"= 1,2,3. Combine the variable rating with the dtm as a data frame called data2.

```
yelp_data$rating <- ifelse(yelp_data$stars >= 4, "1", "0")
data2 <- cbind(as.data.frame(as.matrix(dtm)), rating = yelp_data$rating)
```

Analysis

Get a training data with 13000 reviews and the 5000 reserved as the testing data. Keep the rest (2000) as our validation data set.

2. LASSO

i. Use the training data to get Lasso fit. Choose lambda.1se. Keep the result here.

ii. Feed the output from Lasso above, get a logistic regression.

a) Pull out all the positive coefficients and the corresponding words. Rank the coefficients in a decreasing order. Report the leading 2 words and the coefficients. Describe briefly the interpretation for those two coefficients.

b) Make a word cloud with the top 100 positive words according to their coefficients. Interpret the cloud briefly.

c) Repeat i) and ii) for the bag of negative words.

d) Summarize the findings.

iii. Using majority votes find the testing errors i) From Lasso fit in 3) ii) From logistic regression in 4) iii) Which one is smaller?

3. Random Forest

i. Briefly summarize the method of Random Forest

ii. Now train the data using the training data set by RF. Get the testing error of majority vote. Also explain how you tune the tuning parameters (`mtry` and `ntree`).

4. Boosting

Now use `XGBoost` to build the fourth predictive equation. Evaluate its testing error.

5. Ensemble model

i. Take average of some of the models built above (also try all of them) and this gives us the fifth model. Report it's testing error. (Do you have more models to be bagged, try it.)

6. Final model

Which classifier(s) seem to produce the least testing error? Are you surprised? Report the final model and accompany the validation error. Once again this is THE only time you use the validation data set. For the purpose of prediction, comment on how would you predict a rating if you are given a review (not a tm output) using our final model?