R Markdown File:

```
---
title: "Writeup"
author: "u2101274"
date: "`r Sys.Date()`"
output:
  pdf_document: default
  html_document: default
---


```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Github Link: https://github.com/Harris-Go/EC349_Project

We're part of an academic community at Warwick.

Whether studying, teaching, or researching, we’re all taking part in an expert conversation which must meet standards of academic integrity. When we all meet these standards, we can take pride in our own academic achievements, as individuals and as an academic community.

Academic integrity means committing to honesty in academic work, giving credit where we've used others' ideas and being proud of our own achievements.

In submitting my work I confirm that:

1. I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.

2. I declare that the work is all my own, except where I have stated otherwise.

3. No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.

4. Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.

5. I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.

6. Where a proof-reader, paid or unpaid was used, I confirm that the proofreader was made aware of and has complied with the University's proofreading policy.

7. I consent that my work may be submitted to Turnitin or other analytical technology. I understand the use of this service (or similar), along with other methods of maintaining the integrity of the academic process, will help the University uphold academic standards and assessment fairness.

Analysis of the method (1249 words)

The Methodology - Problem Definition

To approach this project, I used the General Data Science Methodology by John Rallins due to its wide applicability and detailed process that would allow me to step through the project carefully. Although it is not always the best suited for teams, the methodology was effective for an individual project.

The first step is understanding the problem which was predicting how many stars a certain user would give a business on yelp. The data provided was a selection from the Yelp database, comprising predominantly of user reviews, user account characteristics and information on the businesses. This data could be used to train a model that given equivalent data on a user and a business, could predict how many stars that user was likely to give the business.

Data Understanding and Organisation

With a vast selection of data available, it was important to understand how useful certain parts would be in answering the given question. Primarily, an understanding of the user and the business characteristics as well as their history with stars and ratings on yelp would provide a solid foundation to construct the model from. Initially, I removed all the open text fields from the data as although they could prove useful in predicting based on specific words in the text, this model is not using sentiment analysis. Additionally, most of the business attribute fields were full of missing data, with as little as 0.03% of some categories having any data in. The nature of these attributes of being very specific, often referring to specific sectors (ResaurantPriceRange2) and being difficult to impute, they were removed from the data to avoid reducing the analysis to only restaurants with no missing values.

The data sets were then combined, in total, the data consisted of 33 independent variables across 1,398,056 observations. The data was then split into a training and test data set to have 10000 observations in the test set which would provide a sufficient test of the model without reducing too much information needed to train it.

Large amounts of missing values were present with the user's data (about 80% missing) but the nature (and potential importance) of this data made it more manageable to impute. As most of the data was count data, and was already skewed towards zero, the missing data that suggested a lack of account meant the user could also be attributed a zero value as any non-zero value would have been a false attribution to the user and would have shifted the distribution more. The most difficult data to impute was the user's stars as they were values in the range of 1 to 5 based on the user's average review. In this case the distribution was analysed, and the median value was assigned to all the missing values which, although not ideal, was done to ensure the average was not misrepresented.

Modelling

Despite stars being a categorical variable, due to the nature of the categories having an order and that logically a person who voted a one was more likely to vote a two than any other number, they could be modelled using a linear regression model. The only two categorical variables in the cleaned data set were state and city, so they would need to be removed in order to run a regression model, but otherwise variables like business stars and user stars were already showing linear relation to the stars variable in the graphs.

Linear Models

After creating a base linear model using all the data, the coefficients show that there are 17 coefficients with statistically significant correlation and one that has perfect collinearity with another independent variable. The linear model has a mean squared error of 1.502983. Due to the stars data being discrete and positive, the model was run again with the Poisson distribution, but the MSE was much higher, at 7.83759, so shows no improvement on the original model, and when run with fewer variables (using only the significant ones), the model is also worse with an error of 7.837722. The results from the linear model show that not all the predictors are working to help predict and therefore the model can be improved.

LASSO and Ridge

By utilising a Least Absolute Shrinkage and Selection Operator (LASSO) model, it's penalisation parameter will reduce variables that are not helpful in predicting stars to zero, and shrink the others, especially useful with 31 variables. Using the LASSO model with a lambda value of 1 (arbitrarily picked at the minute), the MSE of the model is 2.18729 so is currently performing worse than the linear model. However, by utilising cross validation to split the training data set into 3 folds, it allows the model to be trained on two of the folds and then validated on the other fold; the optimal lambda value can be determined which will then be plugged back into the model. The value determined by cross validation was 0.005216 and using that in the LASSO model led to a MSE of 1.506196, which is slightly larger than the standard linear model's MSE. It achieved 29.4% correct values out of the total predictions. The original linear model accuracy is still marginally better, at 29.7%. Therefore, it appears keeping all the predictors involved may well improve the model, so the model was run with a ridge shrinkage estimator that smoothly shrinks all the predictors towards 0. The error of the ridge model was 1.509303, and the accuracy was 28.8% which were worse than the previous models.

Ordinal and Multinomial Model

Another final option with the linear model was to implement an ordinal model that treats stars as an ordered categorical variable. This was implemented and used to predict stars, and performed much better than the linear model, predicting 51.2% of values correctly despite an error of 2.8946. This suggests that the straight linear model is not an accurate representation of the data. Interestingly, the ordinal model predicts only 5s and 1s, apart from 218 4s, which suggests that due to the shape of the data outlined earlier, it is better for the model to bank on either a 5 or a 1 than to worry about the values in the middle. The multinomial LASSO model was an even better prediction with an accuracy of 53.56%, the best of all the models, demonstrating the benefit of fitting the data correctly and using a shrinkage estimator to reduce the unnecessary predictors.

Decision Tree

To test whether the model created matches against an alternative option, a decision tree model was built and compared against the ordinal and multinomial models. The decision tree only had marginally better accuracy with 51.81% and equally predicted mostly 1-star and 5-star reviews with a small number of 4-star reviews. Therefore, the models are acting similarly despite using two very different approaches.

Evaluation

The ordinal and multinomial models performed a lot better than the linear model because they fit the data much more accurately than their linear counterparts. It appears that there may not be much ability to increase accuracy because the descriptiveness of the data doesn't allow any model to predict well using the 2 and 3 stars. If the model was constructed initially with decision trees it would be possible to increase the predictive power of the tree slightly utilising bootstrap aggregating and random forests to reduce the variance in the predictors and the correlation between samples or boosting to improve the predictors by building trees on the residuals.

##Table of Results

Model	Code Name	Features	MSE	Accuracy	Misclassification Error Rate
Linear	linear1	All the data	1.5029	29.76%	
Linear	linear2	Poisson Distribution	7.8375		
Linear	linear3	Reduced Variables	7.8377		
LASSO	lasso1	Lambda = 1	2.1872		
LASSO	lasso2	Cross Validated	1.5061	29.41%	
Ridge	ridge1	Cross Validated	1.5093	28.89%	
Ordinal	ordinal	Uses only business stars and user stars	2.9252	51.24%	
LASSO	lasso_multi	Multinomial	-16.38	53.56%	
Tree	tree1	Decision Tree		51.81%	0.4819

Methodology (127 words)

I chose and implemented the General Data Science methodology created by John Rallins, which was easy to implement due to the detailed methodology which was clear to follow. Flexibility of the methodology allowed me to tidy the data first, and then attempt to model, but when the model needed a change in the parameters, I went back and adapted the data more. Additionally, I found the loop between modelling and evaluation was extremely apt as I often found in evaluating my model that I needed to rethink my approach. For example, the use of a linear model initially seemed to be well founded but ended up being a wrong approach, so rethinking my modelling allowed me to return and attempt ordinal models that were much more successful.

Challenges (193 words)

The biggest challenge I faced was trying to deal with the computational restrictions imposed when dealing with large data sets. Initially I attempted to use the full-size data sets, but as I found out, R stores all data in the RAM, so a bottleneck in RAM made it impossible to stream in the data. Therefore, I used the smaller data sets to save RAM. This worked for the linear models, but when I came to apply the bagging, boosting and random trees models, the number of trees they had to build to be effective ensured they spent a huge amount of time running (Some of them took over 3 hours). I attempted to streamline the models by reducing the amount of bags, boosts or forests involved, and I looked at correlations between independent variables to work out if any could be removed as they suffered from multicollinearity. In the end, I decided to focus on the linear models as the computational limitations were lower, although the final multinomial LASSO model required a sample of observations to finish finding the cross validated results but could run the training and prediction on the full data.

Modelling:

```
# Import the relavant packages
library(jsonlite)     #Importing JSON Files
library(glmnet)       #Building Linear Models
library(ggplot2)      #Plotting Graphs
library(tidyverse)    #Manipulating the Data
library(corrplot)     #Plotting Correlation Matrices
library(sandwich)     #Robust Standard Errors
library(lmtest)       #Testing the Linear Model
library(caret)        #Splitting Data into Train and Test
require(MASS)         #Used for Ordinal Prediction
require(tree)         #Used for Decision Trees
require(rpart)        #Used for Decision Trees
require(rpart.plot)   #Plot Decision Trees



# Clear the R Workspace and free up unused RAM
cat("\014")
rm(list=ls())
gc()

# Set Working Directory
setwd("D:/Documents/EC349_Project")

################################

# Load in json files and save them as R datasets
business_data <- stream_in(file("yelp_academic_dataset_business.json"))
saveRDS(business_data, file = "business_data.rds")
```

```
checkin_data <- stream_in(file("yelp_academic_dataset_checkin.json"))
saveRDS(checkin_data, file = "checkin_data.rds")

tip_data <- stream_in(file("yelp_academic_dataset_tip.json"))
saveRDS(tip_data, file = "tip_data.rds")

#Load in datasets to avoid streaming them in every time
tip_data <- readRDS("tip_data.rds")

checkin_data <- readRDS("checkin_data.rds")

business_data <- readRDS("business_data.rds")

load("yelp_review_small.Rda")

load("yelp_user_small.Rda")

################################################

##Explore the data
#Business Data:
names(business_data)
dim(business_data)
head(business_data)
#Check the percentage of missing fields
(colSums(!is.na(business_data))/nrow(business_data))*100
#Removes the open text fields and fields with large amounts of missing data
business_data[c('name','address','postal_code','attributes','hours','categories')] <- list(NULL)
#Renames the fields to avoid duplication with other data sets
names(business_data) <-
c('business_id','city','state','latitude','longitude','business_stars','business_review_count','is_open')
```

```
(colSums(!is.na(business_data))/nrow(business_data))*100



#Checkin Data:
names(checkin_data)
head(checkin_data)
#Splits the checkin dates into a string and sums to find the number of checkins for each business
checkin_data <- checkin_data %>% mutate(checkin_number = 1 + str_count(date, ","))
#Removes the surplus date field
checkin_data[c('date')] <- list(NULL)

#Review Data:
names(review_data_small)
dim(review_data_small)
summary(review_data_small)
#Removes the open text field
review_data_small[c('text')] <- list(NULL)
#Seperates the date field into year, month and day, making them all numeric
review_data_small <- review_data_small %>% separate(date,c('year','month','day'))
review_data_small$year <- as.numeric(review_data_small$year)
review_data_small$month <- as.numeric(review_data_small$month)
review_data_small$day <- as.numeric(review_data_small$day)
#Checks for percentage of missing data
(colSums(!is.na(review_data_small))/nrow(review_data_small))*100

#Tip Data
names(tip_data)
dim(tip_data)
head(tip_data)
tip_data %>% summary()
#Remove text column
```

```
tip_data[c('text','date')] <- list(NULL)

#Adds an instrument that when combined with the other data sets should capture whether there was a tip

tip_data <- tip_data %>% mutate(tip = 1)

#Renames to avoid duplicated IDs

names(tip_data) <- c('user_id','business_id','tip_compliments','tip')


#User Data

names(user_data_small)

dim(user_data_small)

summary(user_data_small)

#Removes text fields

user_data_small[c('name','friends','elite')] <- list(NULL)

#Separates the yelping since field into a year only

user_data_small <- user_data_small %>% separate('yelping_since',c('yelping_since_year'))

user_data_small$yelping_since_year <- as.numeric(user_data_small$yelping_since_year)

#Renames all the fields to avoid duplication

names(user_data_small) <-
c('user_id','user_review_count','yelping_since','user_useful','user_funny','user_cool','user_fans','user_stars','compliment_hot','compliment_more','compliment_profile','compliment_cute','compliment_list','compliment_note','compliment_plain','compliment_cool','compliment_funny','compliment_writer','compliment_photos')

#Checks the percentage of missing data

(colSums(!is.na(user_data_small))/nrow(user_data_small))*100


#####################################


#Join the data sets together and removing them once combined

review_data_small <- left_join(review_data_small,user_data_small,'user_id')

rm(user_data_small)

review_data_small <- left_join(review_data_small,business_data,'business_id')

rm(business_data)

review_data_small <- left_join(review_data_small,checkin_data,'business_id')
```

```
rm(checkin_data)

#Due to the nature of tip data having multiple potential matches, the joining process added observations

#To the data set (Around 23,000), so it has been left out to avoid spoiling the data

#review_data_small <- left_join(review_data_small,tip_data,by = c('business_id' = 'business_id','user_id'='user_id'))

rm(tip_data)

#Checks to see the percentages of missing data

(colSums(!is.na(review_data_small))/nrow(review_data_small))*100


#Remove the IDs to make modelling easier

review_data_small[c('review_id','user_id','business_id')] <- list(NULL)


#Save

save(review_data_small,file = "review_data.Rda")

#load(file = "review_data.Rda")

##############################


#Split the data into train and test

set.seed(1)

#Partition the data

parts = createDataPartition(review_data_small$stars, p = 1-(10002/nrow(review_data_small)), list = F)

#Select the training and test data using the random selection above

train = review_data_small[parts, ]

test = review_data_small[-parts, ]

#Remove the unnecessary variables to save RAM

rm(parts)

rm(review_data_small)


###################################
```

```
(colSums(!is.na(train))/nrow(train))*100

#Sort out the missing values:

train %>% summary()

#User Review Count

ggplot(train, aes(user_review_count)) + geom_histogram()

train <- train %>% mutate(user_review_count = ifelse(is.na(user_review_count), 0,
user_review_count))

ggplot(train, aes(user_review_count)) + geom_histogram()

#Yelping Since

ggplot(train, aes(yelping_since)) + geom_histogram()

ggplot(train, aes(year)) + geom_histogram()

#Despite the difference in distributions, the only available information we have for the user

#Is when they wrote the review,and it would be wrong to attribute them being on yelp any

#Earlier than when they wrote their review.

train <- train %>% mutate(yelping_since = ifelse(is.na(yelping_since), year, yelping_since))

ggplot(train, aes(yelping_since)) + geom_histogram()

#User Values

#As most of the user values have similar distributions they have all been treated the same,

#As most of the given data is 0, and since they can't have recieved any praise or comments

#on their account if they didn't have one, they have been treated as if they had an account

#with 0 reactions on any of their posts.

ggplot(train, aes(user_useful)) + geom_histogram()

train <- train %>% mutate(user_useful = ifelse(is.na(user_useful), 0, user_useful),

                          user_funny = ifelse(is.na(user_funny), 0, user_funny),

                          user_cool = ifelse(is.na(user_cool), 0, user_cool),

                          user_fans = ifelse(is.na(user_fans), 0, user_fans),

                          compliment_hot = ifelse(is.na(compliment_hot), 0, compliment_hot),

                          compliment_more = ifelse(is.na(compliment_more), 0, compliment_more),

                          compliment_profile = ifelse(is.na(compliment_profile), 0, compliment_profile),

                          compliment_cute = ifelse(is.na(compliment_cute), 0, compliment_cute),
```

```
                compliment_list = ifelse(is.na(compliment_list), 0, compliment_list),
                compliment_note = ifelse(is.na(compliment_note), 0, compliment_note),
                compliment_plain = ifelse(is.na(compliment_plain), 0, compliment_plain),
                compliment_cool = ifelse(is.na(compliment_cool), 0, compliment_cool),
                compliment_funny = ifelse(is.na(compliment_funny), 0, compliment_funny),
                compliment_writer = ifelse(is.na(compliment_writer), 0, compliment_writer),
                compliment_photos = ifelse(is.na(compliment_photos), 0, compliment_photos)
)
#User stars
#User stars are again going to be attributed to the review that they have given, an approach that
#will only work in the training data set. The test data will need to use a median or mean value
#from the training data.
ggplot(train, aes(user_stars)) + geom_histogram()
train <- train %>% mutate(user_stars = ifelse(is.na(user_stars), median(user_stars, na.rm=TRUE),
user_stars))
ggplot(train, aes(user_stars)) + geom_histogram()

#Checkin Data
ggplot(train, aes(checkin_number)) + geom_histogram()
train <- train %>% mutate(checkin_number = ifelse(is.na(checkin_number),0,checkin_number))

##################################################

#Do the same for the test data:
test <- test %>% mutate(user_review_count = ifelse(is.na(user_review_count), 0,
user_review_count))
test <- test %>% mutate(yelping_since = ifelse(is.na(yelping_since), year, yelping_since))
test <- test %>% mutate(user_useful = ifelse(is.na(user_useful), 0, user_useful),
                user_funny = ifelse(is.na(user_funny), 0, user_funny),
                user_cool = ifelse(is.na(user_cool), 0, user_cool),
                user_fans = ifelse(is.na(user_fans), 0, user_fans),
                compliment_hot = ifelse(is.na(compliment_hot), 0, compliment_hot),
```

```
                compliment_more = ifelse(is.na(compliment_more), 0, compliment_more),
                compliment_profile = ifelse(is.na(compliment_profile), 0, compliment_profile),
                compliment_cute = ifelse(is.na(compliment_cute), 0, compliment_cute),
                compliment_list = ifelse(is.na(compliment_list), 0, compliment_list),
                compliment_note = ifelse(is.na(compliment_note), 0, compliment_note),
                compliment_plain = ifelse(is.na(compliment_plain), 0, compliment_plain),
                compliment_cool = ifelse(is.na(compliment_cool), 0, compliment_cool),
                compliment_funny = ifelse(is.na(compliment_funny), 0, compliment_funny),
                compliment_writer = ifelse(is.na(compliment_writer), 0, compliment_writer),
                compliment_photos = ifelse(is.na(compliment_photos), 0, compliment_photos)
)
test <- test %>% mutate(user_stars = ifelse(is.na(user_stars), median(user_stars, na.rm=TRUE),
user_stars))
test <- test %>% mutate(checkin_number = ifelse(is.na(checkin_number),0,checkin_number))

(colSums(!is.na(test))/nrow(test))*100

save.image()

#Unneeded from when I was trying to reduce the RAM consumption when modelling
#save(test, file = "test.Rda")
#load(file = "test.Rda")
#rm(test)
#gc()

########################################

#Explore the data - Look at some key variables and their interaction with stars
ggplot(train, aes(x=stars)) + geom_bar()
ggplot(train, aes(x=user_review_count, y=stars)) + geom_point() + geom_smooth(method="lm")
ggplot(train, aes(x=business_review_count, y=stars)) + geom_point() + geom_smooth(method="lm")
```

```
ggplot(train, aes(x=user_stars, y=stars)) + geom_point() + geom_smooth(method="lm")
ggplot(train, aes(x=business_stars, y=stars)) + geom_point() + geom_smooth(method="lm")
ggplot(train, aes(x=business_stars, y=stars)) + geom_bin_2d() + geom_smooth(method = "lm")
ggplot(train, aes(x=year, y=stars)) + geom_point() + geom_smooth(method = "lm")

#Looks at the correlation matrix between variables
cor(train)
corrplot(cor(train[,-c(26,27)], use = "na.or.complete"))

#Tests some of the key variables and their correlation with stars
cor.test(test$user_stars,test$stars)
cor.test(test$business_stars,test$stars)

########################################

#Remove city and state as variables from the data:
train[c('city','state')] <- list(NULL)
test[c('city','state')] <- list(NULL)

#Initial Linear Model
linear1 <- glm(stars ~ ., data = train, family = gaussian)
summary(linear1)
linear1_prediction <- predict(linear1,newdata=test)
linear1_error <- mean((test$stars - linear1_prediction)^2)

#Linear Model with Poisson Distribution
linear2 <- glm(stars ~ ., data = train, family = poisson)
summary(linear2)
linear2_prediction <- predict(linear2,newdata=test)
linear2_error <- mean((test$stars - linear2_prediction)^2)
summary(linear2_prediction)
```

```
#Linear Model with fewer variables

linear3 <- glm(stars ~ useful + funny + cool + year + month + yelping_since + user_useful +
user_funny + user_cool + user_fans + user_stars + compliment_list + compliment_writer +
business_stars + is_open +checkin_number, data = train, family = poisson)

summary(linear3)

linear3_prediction <- predict(linear3,newdata=test)

linear3_error <- mean((test$stars - linear3_prediction)^2)


#LASSO Model

lasso1 <- glmnet(as.matrix(train[,-1]), as.matrix(train[,1]), lambda = 1, alpha = 1)

coef(lasso1)

lasso1_prediction <- predict(lasso1, s = 1, newx = as.matrix(test[,-1]))

lasso1_error <- mean((test$stars - lasso1_prediction)^2)


#LASSO with cross validation

#Find the possible values of Lambda

cross_validation <- cv.glmnet(as.matrix(train[,-1]), as.matrix(train[,1]), alpha = 1, nfolds = 3)

plot(cross_validation)

#Find the one that minimises MSE

lambda_lasso <- cross_validation$lambda.min

#Run the lasso model with that optimal value of Lambda

lasso2 <- glmnet(as.matrix(train[,-1]), as.matrix(train[,1]), lambda = lambda_lasso, alpha = 1)

coef(lasso2)

#Predict using the new model

lasso2_prediction <- predict(lasso2, s = lambda_lasso, newx = as.matrix(test[,-1]))

#Find the MSE

lasso2_error <- mean((test$stars - lasso2_prediction)^2)

summary(lasso2_prediction)

#Clean the output into integers

lasso2_prediction <- round(lasso2_prediction, 0)

lasso2_prediction[lasso2_prediction<1] <- 1
```

```
lasso2_prediction[lasso2_prediction>5] <- 5
summary(lasso2_prediction)
#Test the new accuracy and MSE
lasso2_error <- mean((test$stars - lasso2_prediction)^2)
lasso2_accuracy <- sum(lasso2_prediction == test$stars)/nrow(test)


#Clean the output of the linear model as well
summary(linear1_prediction)
linear1_prediction <- round(linear1_prediction, 0)
linear1_prediction[linear1_prediction<1] <- 1
linear1_prediction[linear1_prediction>5] <- 5
summary(linear1_prediction)
#Test the accuracy and error of the new model
linear1_error <- mean((test$stars - linear1_prediction)^2)
linear1_accuracy <- sum(linear1_prediction == test$stars)/nrow(test)

#Implement a cross-validated Ridge Model
cross_validation1 <- cv.glmnet(as.matrix(train[,-1]), as.matrix(train[,1]), alpha = 0, nfolds = 3)
plot(cross_validation1)
ridge_lambda <- cross_validation1$lambda.min
ridge1 <- glmnet(train[,-1], train[,1], alpha = 0, lambda = ridge_lambda, thresh = 1e-12)

ridge_prediction <- predict(ridge1, s = ridge_lambda, newx = as.matrix(test[,-1]))
ridge_error <- mean((ridge_prediction - test[,1]) ^ 2)

#Clean the output and check accuracy of ridge model
summary(ridge_prediction)
ridge_prediction <- round(ridge_prediction, 0)
ridge_prediction[ridge_prediction<1] <- 1
ridge_prediction[ridge_prediction>5] <- 5
```

```
summary(ridge_prediction)

ridge_error <- mean((test$stars - ridge_prediction)^2)

ridge_accuracy <- sum(ridge_prediction == test$stars)/nrow(test)


#Implementing an ordinal model

#Convert stars into a categorical variable

train$stars <- as.factor(train$stars)


#Run the ordinal model with business_stars and user_stars

ordinal <- polr(stars ~ year + month + yelping_since + user_stars + business_stars + is_open, data =
train, Hess=TRUE)

summary(ordinal)

ordinal_prediction <- predict(ordinal, newdata = test)

ordinal_error <- mean((test$stars - as.numeric(ordinal_prediction))^2)

ordinal_accuracy <- sum(as.numeric(ordinal_prediction) == test$stars)/nrow(test)

summary(ordinal_prediction)


set.seed(1)

parts1 = createDataPartition(train$stars, p = 0.1, list = F)

train1 = train[parts1, ]

rm(parts1)


#LASSO with multinomial

#Find the possible values of Lambda

cross_validation_multi <- cv.glmnet(as.matrix(train1[,-1]), as.matrix(train1[,1]), alpha = 1, nfolds =
3,family="multinomial",type.multinomial="grouped")

plot(cross_validation_multi)

#Find the one that minimises MSE

lambda_lasso_multi <- cross_validation_multi$lambda.min

#Run the lasso model with that optimal value of Lambda

lasso_multi <- glmnet(as.matrix(train[,-1]), as.matrix(train[,1]), lambda = lambda_lasso_multi, alpha
= 1,family="multinomial")
```

```
coef(lasso_multi)

#Predict using the new model

lasso_prediction_multi <- predict(lasso_multi, s = lambda_lasso_multi, newx = as.matrix(test[,-1]),type="class")

#Find the Errors and Accuracy

summary(as.numeric(lasso_prediction_multi))

lasso_multi_error <- mean((test$stars - as.numeric(lasso_prediction_multi)^2))

lasso2_accuracy <- sum(as.numeric(lasso_prediction_multi) == test$stars)/nrow(test)



#Remove the prediction matrices to save RAM

rm(linear1_prediction)

rm(linear2_prediction)

rm(linear3_prediction)

rm(lasso1_prediction)

rm(lasso2_prediction)

rm(ridge_prediction)

rm(ordinal_prediction)

##########################################

#Decision Tree with 'rpart'

#Train the tree

tree1 <- rpart(stars ~ ., data = train)

#Plot the tree

rpart.plot(tree1)

#Predict using the tree

tree1_prediction <- predict(tree1, newdata = test, type="class")

summary(tree1)

summary(tree1_prediction)

#Determine the Error and Accuracy
```

```
tree1_error <- sum(as.numeric(tree1_prediction) != test$stars)/nrow(test)
tree1_accuracy <- sum(as.numeric(tree1_prediction) == test$stars)/nrow(test)



##########################
#Attempts to improve the decision tree were hampered by severe
#problems in handling the computational load so despite best efforts
#this has been removed from the analysis



#Remove Excess variables to reduce the computational load
#train1$stars <- as.numeric(train1$stars)
#corrplot(cor(train1))
#train1 <- train1[,c(1,4,5,8,11,14,24,26,27,28,29,30)]
#corrplot(cor(train1))
#train1$stars <- as.factor(train1$stars)

#Decision Tree with Boosting
#boost <- boosting(stars~., data=train1, boos=TRUE, mfinal=4)
#summary(boost)
#boost_prediction <- predict(boost, newdata = test)
#boost_prediction

#Random Forests
#set.seed(1312)
#model_RF<-randomForest(stars~.,data=train, ntree=25)
#pred_RF_test = predict(model_RF, test)
#mean(model_RF[["err.rate"]])



#Decision Tree with Bagging
```

```
#bag <- bagging(stars~., data=train1, nbagg = 5, coob = TRUE, control = rpart.control(minsplit = 1000, cp = 10))

#bag #MSE 1.3074
```