Stat 479 Final Report

Introduction

Yelp is a company that provides recommendations, reviews and ratings about local restaurants and services. Thousands of customers upload their comments about restaurants each day. And people trust this service because this is the voice from people and its reliability. "Yelp averages more than 178 million unique visitors every month, across its mobile, desktop, and app versions" (Bassig 2019) shows people are making their decision based on comments and rating from yelp. Yelp can greatly influence people's opinion or attitude toward restaurants and services.

For the past few years, Yelp has hosted the Yelp Dataset Challenge annually, which encourages people to use Yelp's huge data resources to further the restaurant-review field. We decided to join this challenge with focusing on restaurants in the Dane County area in WI. We want to study how comments length, sentiments, and other factors can reflect the rating. Words can also be determined to show people's real feelings and attitude.

Raw data used in this project were provided by Professor Pixu Shi from Duke University and was first used in Stat 333, 2020 Spring. This raw data contains 92,236 reviews of 1,361 Madison businesses. Our goal is to figure out what words can reflect people's positive or negative attitude and predict the rating of restaurants given those comments based on words and some environmental factors.

Data Preprocessing and Visualization

In the original dataset, one sample contains the sample id, rating stars, actual comments, date, number of characters (nchar), number of words (nword), restaurant's categories and sentiment score. We first randomly select 5000 from the original dataset and then deleted all

samples with missing values. Although using all the data in the dataset can improve the performance of our model, we still choose to use a subset of the original dataset due to limited computing power we can get access to. We also changed the format of some features so it's easier to operate. The values of actual comments, restaurant's categories were converted into strings.

	Month
Winter	12-2
Spring	3-5
Summer	6-8
Autumn	9-11

	Time
Night	22:00 - 4:00
Morning	3:00 - 5:00
Noon	6:00 - 8:00
Evening	9:00 - 11:00

Table 2. Method to get two new factors from date and time.

The feature of date is composed of the actual date and time the comment is made. To make use of this feature, we converted it into season and time of the day, two new features. Table 2 shows how we divided the date into different seasons and time of the day. The newly generated feature of season and day time is then treated as dummy variables. Since the feature of category in the raw data is a string containing all the relevant categories of a restaurant, we also splitted it and created individual dummy variables for each category.

Then, we did some exploratory data analysis on our data. Figure 1 shows the distribution of comments based on rating stars, nchar, nword and the sentiment score . As we can see, people are more willing to give comments with high rating stars, and the majority of comments are less than 2000 characters and less than 200 words. Both the distributions of nchar and nword is highly right skewed. To make sure we can satisfy the assumption of the model we want to fit later, we did a log transformation to nchar and nword. Figure 2 are the distributions of these two features after transformation. They now both have a distribution similar to the normal distribution and their scales become smaller. The distribution of the sentiment score is slightly

left skewed. We think it is fine to have such distribution, so we did not apply a transformation to the sentiment score.

By splitting the filtered and modified data, we have a training set and a validation set. The validation set contains about 20% of the randomly chosen samples in the original dataset. The training set contains about 80% of the randomly chosen samples in the original dataset. Since both the validation set and the training set are composed of samples randomly selected from the original dataset, we expect the distributions of different features in these two datasets to be the same as the distributions we have in the original dataset.

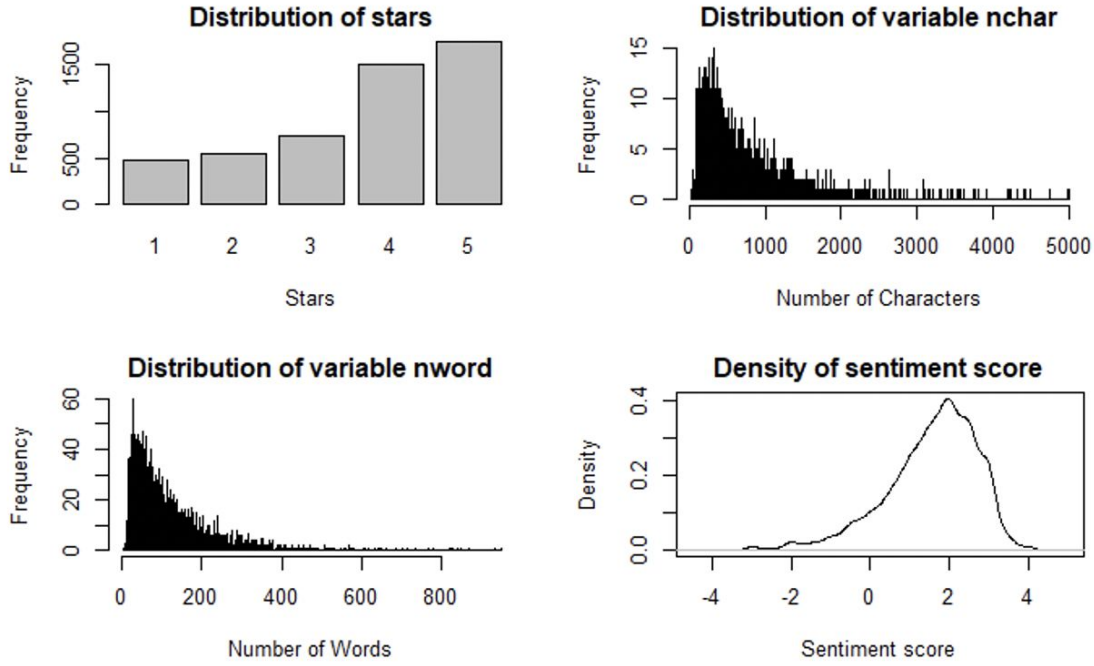

Figure 1: The distribution of number of comments based on different factors.

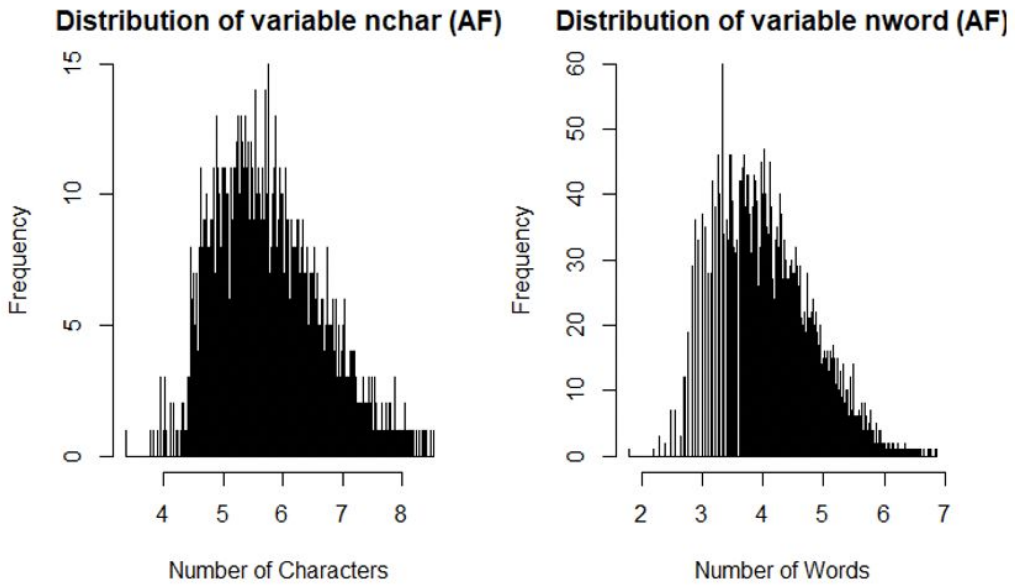

Figure 2: The distribution of comments based on factors. The data are transformed by log(1+variables data) for better visualization.

Data Processing

Because we want to analyze the comments given by the customers, we set up additional features as the number of occurrences of different words. Since there are only 4000 samples in the training set, we want to have a relatively small number of features. Thus, we choose to only count the occurrence of unigram words in the training set.

Before counting the occurrences of the words, we processed these comments. We first converted all the comments into lower cases, then removed the punctuations, white spaces, numbers and stopwords of these text comments. In addition, we performed document stemming to ensure that the same words will not appear in the data sets as multiple versions in our final bags of worlds.

In terms of building up predictors using unigram words, we used a local dictionary that is composed of unigram words extracted from the original customer comments and an external dictionary. The external dictionary can be found in *Opinion Mining, Sentiment Analysis, Opinion*

Extraction [1]. The dictionary contains 4167 words with positive and negative sentiment meanings. We choose to use an external dictionary so that more words with strong sentiment meanings can be used as features. We applied the same transformations we did to the customer comments to these unigram words in the external dictionary in order to maintain the consistency of the word predictors. For the local dictionary, we constructed a document-term matrix for the customer comments with all the entries collected from the training set using tf-idf as the weighting method. The inverse document frequency (idf) decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. The inverse document frequency is combined with the frequency of a word to calculate the word's tf-idf, the frequency of a term adjusted for how rarely it is used. In this way, we can measure the importance of the word. We then removed all words with a sparsity over 0.999 from the document-term matrix to form our final local dictionary. There are 3270 words in our local dictionary.

Given the external dictionary and local diction, we built another document-term matrix for the training set and the validation set. In total, there are 1002 words in these two document-term matrixes. What we did next is to merge these two document-term matrices into the training set and the validation set respectively. In such a way, we set up predictors based on the occurrence of different words used in the customer comments in the training set.

Proposed Method and Model Implementation

After data preparation, there are 1244 predictors in our training set and validation set in total. Due to the large size of our predictors, we decided to try two different regression methods, linear regression and lasso regression.

[1] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

Linear regression is a linear approach to model the relationship between a response and one or more explanatory independent variables. In our case, we are trying to establish a relation between the star ratings and the other 1244 variables representing customer comments and other features of restaurants. The fitting of the model is based on a loss function called least squares method. The least squares method estimates the coefficients of each variable and an intercept to minimize aggregated error for all samples. We used lm() to fit linear regression in this project.

Lasso regression can handle a large number of variables. Lasso regression is a kind of penalized regression. In cases when there are very few of the variables in a model can have influence on the response, Lasso regression can give us a smaller MSE than linear regression. It reduces our model size by a soft-threshold determined by the value of lambda. Since there is a bias-variance tradeoff, we want to find the lambda that minimizes the MSE. In this project, we adopted a 10-fold cross-validation to find the lambda that minimized the MSE of the model. Then we used this lambda we found to penalize our model.

In order to have a better accuracy, if the predicted rating is greater than 5, we change it to 5, and if the predicted rating is smaller than 1, we change it to 1.

Model Result & Evaluation

Linear Regression

We use multiple linear regression with 1244 predictors. The R-square is 0.7374 and Adjust R-square is 0.6236. This means 0.7374 of the variability in the response variable, rating stars, can be explained by the predictors. The Adjust R-square is 0.1 less than R-square but it still shows out predictors fit this model. Based on this linear regression model, we calculate the MSE of the training data between the real rating and the predicted rating which is 0.4303173. And the MSE of validation data is 0.830368.

```
Residual standard error: 0.8073 on 2790 degrees of freedom
Multiple R-squared:  0.7374,    Adjusted R-squared:  0.6236
F-statistic: 6.479 on 1209 and 2790 DF,  p-value: < 2.2e-16
```

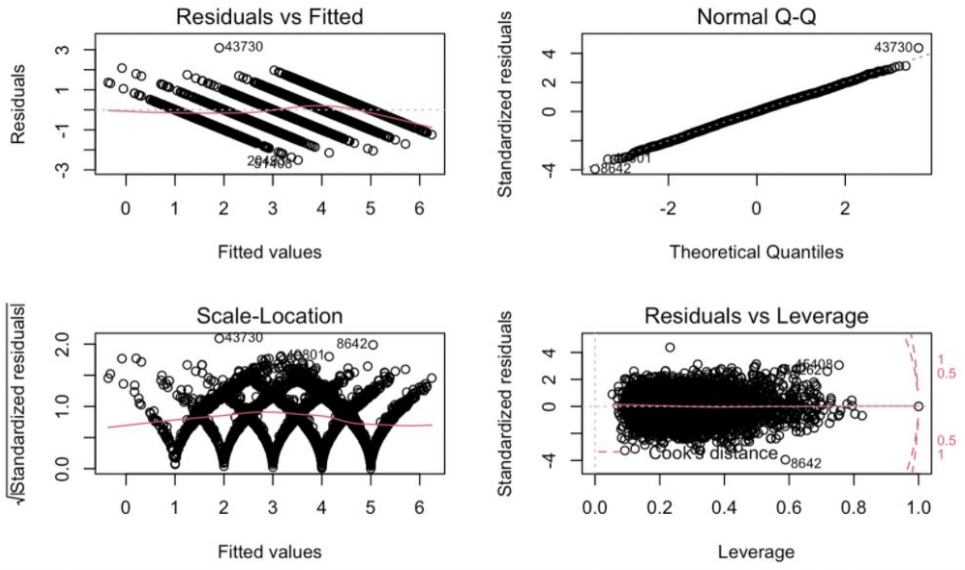

Figure 4. Diagnostic plots for Linear regression model

Figure 4 shows some characteristics of this model fitting. The residuals vs fitted plot shows most of the data points are centered around zero which only a few outliers. Five lines exist because the response variable, rating stars, has five levels. This plot concludes the linearity assumption is not violated. The scale-location plot gives a straight fitting line which means the homoscedasticity assumption is not violated. QQ plot shows the vast majority of the points are on the straight line which indicate the normality assumption is also not violated. It is true that there exist some outliers like 43730 and 8642. But these outliers will not have a great impact on the prediction model compared with a large number of sample sizes.

It is true that the MSE of validation data is relatively big. This is potentially because of small sample size and the small predictor variable size. Thinking about comments, people with different backgrounds have different word-choosing. It is hard to include all predictors that are significant with only about 1000 predictors and 4000 samples might not be enough for a good model fitting.

Fold	RMSE	R-square
1	1.07	0.413
2	1.11	0.401
3	1.07	0.459
4	1.05	0.423
5	1.05	0.458

Table 3. Linear regression model has 5 folds. This listed RMSE and R-square for each fold

To check the stability of the model, we utilized k-fold cross validation after we fitted our linear regression model. When k=5, for each fold, we have stable RMSE and R-square values, which means that our conclusion does not highly depend on the random dataset we picked and the model is relatively stable.

Worth noting, we received a "rank deficiency" error when fitting this linear regression model. We reckon that this may be caused by the facts that the matrix we used for model fitting is sparse and the number of features is large. Under the combined influence of these two facts, the rank of the matrix for model fitting may be low. In this case, there may be multiple solutions to this linear equation system. The performance of the model for predicting the response variable may not be greatly affected. But the coefficients to the features may not be unique. We need to use the coefficients of the features with caution.

Lasso Regression

By using lasso regression with 10 folds, we found the minimum MSE is 0.818, which is reached when lambda = 0.0161. In the left part of the plot in figure 3, the MSE decreases when log(λ) increases, while in the right part of the plot, MSE increases when log (λ) increases. Thus, there is a lambda that can minimize the MSE.

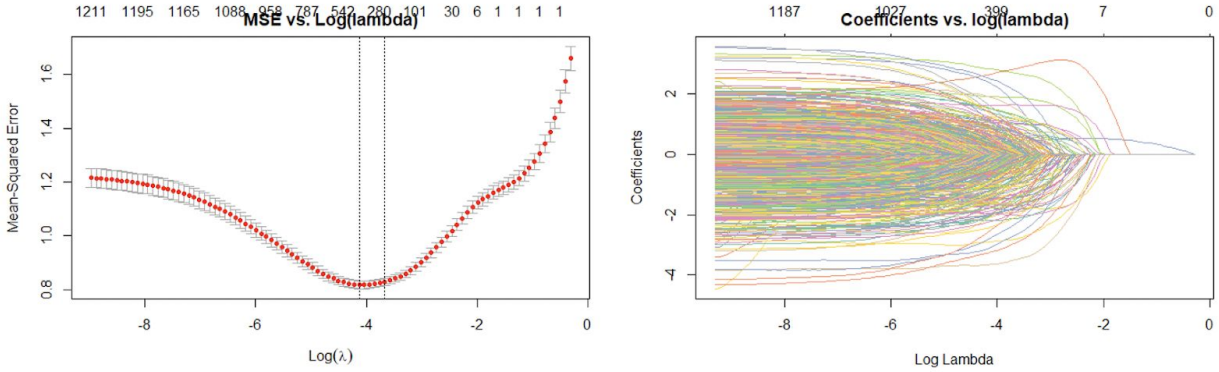

Figure 3. The Lasso regression model with sample size. = 4000. Left: Mean Square Error vs. Log(lambda). Right: Coefficient vs. Log(lambda)

After regularization, some of the coefficients became zero and were eliminated from the model. There were 428 significant predictors after we applied the Lasso regression. Based on this lasso regression model, we calculated the MSE of the training data between the real rating and the predicted rating which is 0.62. And the MSE of validation data is 1.31.

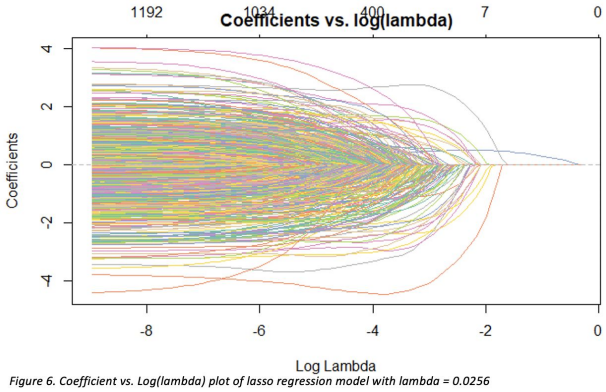

Figure 6. Coefficient vs. Log(lambda) plot of lasso regression model with lambda = 0.0256

To further assure the model stability, we tried to manually increase the lambda and fit the regression model with new predictors. The new lambda = 0.0256 which is the largest value such that error is within 1 standard error of the minimum with MSE = 0.83. Using this regression, MSE of the training data is 0.711. The value of MSE matches our expectations. It gets larger because we didn't use the lambda with minimum MSE in lasso regression which increases error. Although we change our value of lambda, the MSE we obtained is still very close to the MSE we get with the best lambda. The number of predictors selected in this case is 244. It is smaller than the number of predictors selected when the best lambda value is applied. This shrinkage of predictors can be explained by the soft-thresholding formula below. When

lambda is increased, more coefficients would be less than the threshold and be set to 0. Both the changes of MSE and number of selected variables indicate that the model is relatively stable.

$$\hat{\beta}_L = S_{\frac{\lambda}{n}}(\frac{\tilde{y}}{\sqrt{n}})$$

The MSE of validation data is 1.25 which is very close but slightly smaller than MSE of validation data with lambda = 0.0161. This value might not be trustful and might indicate that lasso is not a good model fit.

To validate the lasso regression model, we also tried lasso regression with 3500 sample size rather than 4000. The minimum MSE is 0.828 with lambda = 0.0216. By comparing Figure

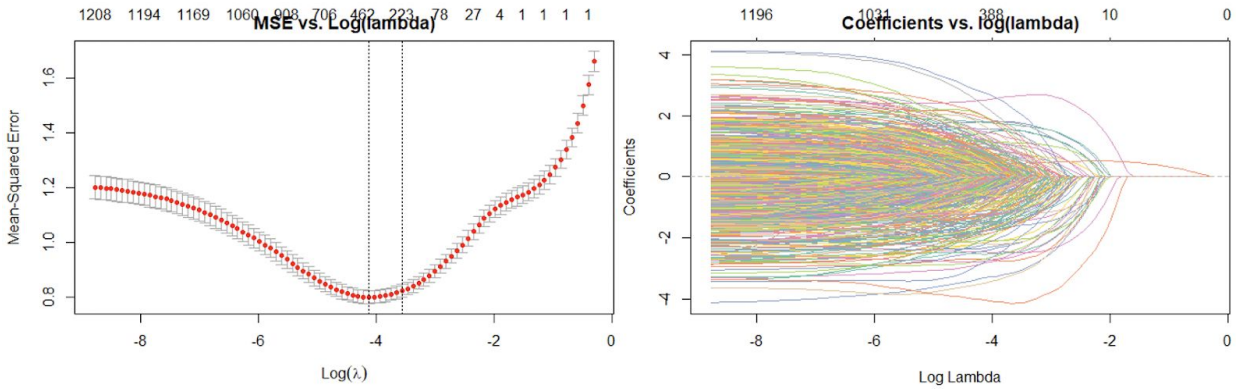

Figure 5. The Lasso regression model with sample size. = 3500. Left: Mean Square Error vs. Log(lambda). Right: Coefficient vs. Log(lambda)

3 and 5, we can easily see that lambda and MSE are very close. And it is reasonable that MSE from a smaller sample size fitting is larger because larger sample size could increase the precision and accuracy of results and decrease the margin of error. There are 329 predictors left after lasso regression since the lambda is larger compared to the lambda we used when sample size is 4000. MSE for training data is 0.0663 and for validation data is 1.25. The changes in the MSE of the model and the selected variable size matches our expectation. Other metrics also did

not change much. Thus, it would be reasonable to say that this model is relatively stable and does not depend on the training dataset much.

Result Comparison

	lasso_pred_y				
actual_y	1	2	3	4	5
1	6	33	48	13	1
2	1	11	50	35	4
3	0	7	82	64	7
4	1	15	121	154	17
5	1	11	108	181	29
	linear_pred_y				
actual_y	1	2	3	4	5
1	41	32	21	7	0
2	17	39	33	11	1
3	4	35	69	37	15
4	1	13	58	134	102
5	0	1	42	109	178

Table 1. Lasse regression and linear regression predicted result vs. real result from validation data

Because both training and validation data MSE from linear regression are smaller than lasso regression, which shows linear regression is the better model fitting.

By using linear regression and lasso regression model, we try to predict the validation dataset and compare it with real rating stars. The correct rate of linear regression model is 0.461 while the for lasso regression model is 0.282. Lasso regression has a relatively worse prediction accuracy with false rate 0.718.

sentiment	delici	die	rude	told	tast	manag	favorit	great	satisfi
1.02e-37	9.50e-09	1.05e-07	1.08e-07	2.42e-06	3.17e-06	4.88e-06	1.34e-05	1.84e-05	3.48e-05

Table 4: Top 10 significant predictors of linear regression model with lowest p-value.

Table 4 gives the top 10 significant predictor variables from linear regression model. It is clear that these words can express a customer's attitude and give better prediction towards star rating.

Conclusion & Discussion

Linear regression model gives the best performance with 1237 predictors. It also gives the lowest Root Mean Square Error value of 0.958. Lasso regression model with 399 final predictors gived a RMSE value of 1.15.

Based on MSE and cross-validation, linear regression model gives the best performance. It is reasonable that lasso regression is not able to give the best performance under such a limited

number of predictors. It is highly possible that there are not many significant predictors. But it still can give direction about which predictors are more significant. Also lasso regression might not be suitable under this circumstance because word-choice is very personal and it is hard to have a very significant predictor (word) even if their star rating is the same.

This model can help us determine which words are more or less significant for determining people's attitude. All significant predictors and meaningful words in some ways indicating people's opinions toward local restaurants and companies. This study can benefit those restaurant and company owners better their services from learning customer' comments. Our model can also help linguisticians understand more underlying meanings of vocabularies and expressions.

Some future direction we could do to better this model is including interaction terms if enough data examples are provided or utilizing other models like graphical models. We can also use algorithms with higher prediction power such as random forest, gradient boost, and neural networks.

Contribution

Andy Xie found the dataset and conducted data preparation and visualization. He was also responsible for cleaning the data, building the Linear regression model, and method validation. Bob Dai was responsible for building the Lasso regression model and using cross-validation to tune the parameter. For the writing of the final report, Jihua Liu was responsible for the introduction, model result & evaluation, result comparison, and conclusion & discussion. Andy was responsible for data cleaning and visualization, and model implementation. Bob was responsible for proposed method and model implementation.

Appendix

````
---
title: "Stat 479 Project"
author: "Anze X, Bob D, Jihua L"
date: "2020/12/14"
output:
  word_document: default
  pdf_document: default
---

```{r setup}
library(qdap)
library(dplyr)
library(tm)
library(ggplot2)
library(ggthemes)
library(wordcloud)
library(plotrix)
library(dendextend)
library(RWeka)
library(quanteda)
library(stringr)
library(glmnet)
library(lubridate)
library(tidyverse)
library(caret)
library(pls)
library(qdapTools)
library(nnet)
library(glmnet)
library(RColorBrewer)
library(gplots)
library(DAAG)
library(MLmetrics)
library(magrittr)

# Set the memory limit
memory.limit(100000)

```

```{r functions for seperate season and daytime into categories}
factorizeSeason <- function(date){
  season <- vector(mode = "character", length = length(date))
  time <- ymd_hm(date)
````
````

```
  for (i in 1:length(time)){
    if ((month(time[i]) <= 2) | (month(time[i]) >= 12)){
      season[i] <- "Winter" # 12-2
    }
    else if ((month(time[i]) <= 5) & (month(time[i]) >= 3)){
      season[i] <- "Spring" # 3-5
    }
    else if ((month(time[i]) <= 8) & (month(time[i]) >= 6)){
      season[i] <- "Summer" # 6-8
    }
    else{
      season[i] <- "Autumn" # 9-11
    }
  }
  return(season)
}

factorizeDaytime <- function(date){
  dayTime <- vector(mode = "character", length = length(date))
  time <- ymd_hm(date)

  for (i in 1:length(time)){
    hour <- hour(time[i])

    if ((hour >= 22) | (hour <= 4 )){
      dayTime[i] <- "Night"
    }
    else if ((hour >= 4) & (hour <= 10)){
      dayTime[i] <- "Morning"
    }
    else if ((hour >= 10) & (hour <= 16)){
      dayTime[i] <- "Noon"
    }
    else{
      dayTime[i] <- "Evening"
    }
  }
  return(dayTime)
}
```

```{r function for check NA}
checkNA <- function(df){
  if (sum(is.na(df$stars)) + sum(is.na(df$text)) + sum(is.na(df$date))
```

```
    + sum(is.na(df$city)) + sum(is.na(df$categories)) + sum(is.na(df$nchar)) +
sum(is.na(df$nword)) + sum(is.na(df$Id))
    + sum(is.na(df$sentiment)) != 0){
   ?stop
  }
}
```

```{r function for corpus processing}
# basic transformations for texts
# input: a dataframe with $text
# output: transformed workable format by tm package
textTransformation <- function(totalText){
  ## Make a vector source and a corpus
  corpus_review <- Corpus(VectorSource(totalText$text))

  ## Remove punctuation
  ## create "addSpace" function that finds a specified pattern and substitute it with a space
  addSpace <- content_transformer(function(x, pattern){
    return(gsub(pattern, " ", x))
  })

  corpus_review <- tm_map(corpus_review, addSpace, "-")
  corpus_review <- tm_map(corpus_review, removePunctuation)

  ## Remove extra whitespaces
  corpus_review <- tm_map(corpus_review, stripWhitespace)
  ## Convert to lower case
  corpus_review <- tm_map(corpus_review, content_transformer(tolower))
  ## Remove numbers
  corpus_review <- tm_map(corpus_review, removeNumbers)
  ## Remove stopwords
  corpus_review <- tm_map(corpus_review, removeWords, stopwords("english"))
  ## Stemming document
  corpus_review <- tm_map(corpus_review, stemDocument)

  return(corpus_review)
}


## merge the DocumentTermMatrix into yelp (original dataset)
decomposeMatrix <- function(sparseRemoved, df){
  new_X <- as.matrix(sparseRemoved)

  new_df <- cbind(df, new_X)
```

```
  ## remove useless cols
  textIndex <- which(colnames(df) == "text")
  new_df <- new_df[, -c(textIndex)]

  return(new_df)

  ## decompose the above matrix for yelp and ylep_out
  # new_X_yelp <- new_X[1:55342, ]
  # new_X_yelp_out <- new_X[55343:92236, ]
  # corpus_yelp <- cbind(yelp, new_X_yelp)
  # corpus_yelp_out <- cbind(yelp_out, new_X_yelp_out)


  # if (c == 1){
  #   ## Remove the column of categories, text, id, city, date from yelp
  #   textIndex <- which(colnames(corpus_yelp) == "text")
  #   categoriesIndex <- which(colnames(corpus_yelp) == "categories")
  #   idIndex <- which(colnames(corpus_yelp) == "Id")
  #   cityIndex <- which(colnames(corpus_yelp) == "city")
  #   dateIndex <- which(colnames(corpus_yelp) == "date")
  #   corpus_yelp_2 <- corpus_yelp[, -c(textIndex, categoriesIndex, idIndex, cityIndex,
dateIndex)]
  #
  #   return(corpus_yelp_2)
  # }
  # else{
  #   ## Remove the column of categories, text, id, city, date from yelp_out
  #   textIndex <- which(colnames(corpus_yelp_out) == "text")
  #   categoriesIndex <- which(colnames(corpus_yelp_out) == "categories")
  #   idIndex <- which(colnames(corpus_yelp_out) == "Id")
  #   cityIndex <- which(colnames(corpus_yelp_out) == "city")
  #   dateIndex <- which(colnames(corpus_yelp_out) == "date")
  #   corpus_yelp_out_2 <- corpus_yelp_out[, -c(textIndex, categoriesIndex, idIndex, cityIndex,
dateIndex)]
  #
  #   return(corpus_yelp_out_2)
  # }
}

```

```{r Data reading & preprocessing}
## read in data
rawData <- read.csv("raw_data/yelp_data.csv")
yelp <- subset(rawData, select = c(1:12))
yelp <- subset(yelp, select = -c(5,6,7))
```

```
## randomly choose 5000
checkNA(yelp)
random_rowNum <- sample(1:55342, 5000, replace = F)
yelp <- yelp[c(random_rowNum), ]
yelp["Id"] <- c(1:5000)


## convert text into actual strings
yelp$text <- as.character(yelp$text)
yelp$categories <- as.character(yelp$categories)


## generate season and daytime categorical variable from variable date
season <- factorizeSeason(yelp$date)
yelp <- cbind(yelp, season)
yelp <- cbind(yelp, mtabulate(strsplit(as.character(yelp$season), ', ')))
yelp <- yelp[-which(colnames(yelp) == "season")]
yelp$Spring <- as.factor(yelp$Spring)
yelp$Summer <- as.factor(yelp$Summer)
yelp$Autumn <- as.factor(yelp$Autumn)
yelp$Winter <- as.factor(yelp$Winter)

dayTime <- factorizeDaytime(yelp$date)
yelp <- cbind(yelp, dayTime)
yelp <- cbind(yelp, mtabulate(strsplit(as.character(yelp$dayTime), ', ')))
yelp <- yelp[-which(colnames(yelp) == "dayTime")]
yelp$Evening <- as.factor(yelp$Evening)
yelp$Night <- as.factor(yelp$Night)
yelp$Morning <- as.factor(yelp$Morning)
yelp$Noon <- as.factor(yelp$Noon)

## factorize categories
yelp <- cbind(yelp, mtabulate(strsplit(as.character(yelp$categories), ', ')))
cols <- c(20:50)
yelp[,cols] <- lapply(yelp[,cols], as.factor)


## save preprocessed data and remove unnecessary data to save memory
rm(list = setdiff(ls(), c("yelp", "rawData")))
save(list = ls(), file = "./intermidiate_data/preprocessed_data.Rdata")
```

```{r DATA VISUALIZATION & DATA TRANSFORMATION}
# DATA VISUALIZATION
par(mfrow = c(2,2))
```

```
barplot(table(yelp$stars),main="Distribution of stars",xlab="Stars",ylab="Frequency")
hist(yelp$nchar, breaks=10000,main="Distribution of variable nchar",xlab="Number of
Characters",ylab="Frequency")
hist(yelp$nword, breaks=10000,main="Distribution of variable nword",xlab="Number of
Words",ylab="Frequency")
plot(density(yelp$sentiment), main = "Density of sentiment score", xlab = "Sentiment score",
ylab = "Density")


# DATA TRANSFORMATION
yelp$nchar <- log(1+yelp$nchar)
yelp$nword <- log(1+yelp$nword)

## check distribution of transformed data
par(mfrow = c(1,2))
hist(yelp$nchar, breaks=10000,main="Distribution of variable nchar (AF)",xlab="Number of
Characters",ylab="Frequency")
hist(yelp$nword, breaks=10000,main="Distribution of variable nword (AF)",xlab="Number of
Words",ylab="Frequency") # AF - after transformation

## remove useless features
categoriesIndex <- which(colnames(yelp) == "categories")
idIndex <- which(colnames(yelp) == "Id")
dateIndex <- which(colnames(yelp) == "date")
yelp <- yelp[, -c(categoriesIndex, idIndex, dateIndex)]

## remove city to prevent level problem
yelp <- yelp[, -which(colnames(yelp) == "city")]

## seperate into train&test + validation (20%)
rand <- sample(1:5000, 5000* 0.2, replace = F)
yelp_validation <- yelp[rand,]
yelp <- yelp[-c(rand),]

## save data
rm(list = setdiff(ls(), c("yelp", "yelp_validation")))
save(list = ls(), file = "./intermidiate_data/transformed_data.Rdata")
```

```{r LOAD STOPWORDS & DICTIONARY}
## load a list of stopWords and do additional removal of stopwords
stopWords <- scan("./raw_data/stopwords.txt", sep = ",", character())
stopWords <- trimws(stopWords, which = "both")

## load dictionary
positiveWords <- scan("./raw_data/positive-words.txt", sep = "",character(), skip = 31)
```

```
negativeWords <- scan("./raw_data/negative-words.txt", sep = "",character(), skip = 31)
dictionaryWords <- data.frame(text = c(positiveWords, negativeWords))

## Garbage Collection
rm(list = setdiff(ls(), c("yelp", "yelp_validation", "stopWords", "positiveWords",
"negativeWords", "dictionaryWords")))
save(list = ls(), file = "./intermidiate_data/transformed_data_with_dict.Rdata")
```

```` ```{r CORPUS PROCESSING} ````

```
# load("./intermidiate_data/transformed_data_with_dict.Rdata")

## transform original text
text <- data.frame("text" = yelp$text)
corpus_review <- textTransformation(text)

## transform external text
dictionary_review <- data.frame(text = paste(dictionaryWords$text, collapse = " "))
dictionary_review <- textTransformation(dictionary_review)
dictionaryBag <- DocumentTermMatrix(dictionary_review)$dimnames$Terms

## GENERATE A DOCUMENT TERM MATRIX FROM CORPUS_REVIEW OF YELP
TEXT
## Sparsity: # of 0 entries / # of total entries
## Non-/sparse entries: # of non-zero / # of 0

## Interpretation:  stemmed words,  documents, % of the words are sparse, longest word:

## Build DocumentTermMatrix for corpus_review with original words
dtm_noDict <- DocumentTermMatrix(corpus_review, control = list(weighting = weightTfIdf))
dtm_noDict_SparseRemoved <- removeSparseTerms(dtm_noDict, 0.999) #  3270 terms

## Build DocumentTermMatrix for corpus_review with dictionary dictionaryBag
dtm_onlyDict <- DocumentTermMatrix(corpus_review, control = list(weighting = weightTfIdf,
dictionary = dictionaryBag))
dtm_onlyDict_SparseRemoved <- removeSparseTerms(dtm_onlyDict, 0.9999) # 1980 terms

## Build DocumentTermMatrix for corpus_review with both original words and external
dictionary
aggrBag <- c(dictionaryBag, dtm_noDict_SparseRemoved$dimnames$Terms) # 4167 words
aggrDtm <- DocumentTermMatrix(corpus_review, control = list(weighting = weightTfIdf,
dictionary = aggrBag))
aggrDtm_SparseRemoved <- removeSparseTerms(aggrDtm, 0.992) # 0.9997 - 3843 terms
```
```

```
## Build DocumentTermMatrix for validation set using same configuration as above
## dtm_noDict
## dtm_noDict_SparseRemoved
## aggrDtm_SparseRemoved
validation_text <- data.frame("text" = yelp_validation$text)
validation_review <- textTransformation(validation_text)
aggrDtm_SparseRemoved_validation <- DocumentTermMatrix(validation_review, control =
list(weighting = weightTfIdf, dictionary = aggrDtm_SparseRemoved$dimnames$Terms))



# 4 OPTIONS: CHOOSE ONE DICT TO RUN
option <- 4

if (option == 1){
  # OPTION 1: Only dictionary
  onlyDict_yelp <- decomposeMatrix(dtm_onlyDict, yelp, 1)
  yelp_validation <- decomposeMatrix(dtm_onlyDict, yelp_validation, 2)
  save(list = c("onlyDict_yelp", "yelp_validation"), file = "./intermidiate_data/onlyDict.Rdata")
}else if (option == 2){
  # OPTION 2: Only dictionary with sparse remove
  onlyDict_SparseRemoved_yelp <- decomposeMatrix(dtm_onlyDict_SparseRemoved, yelp, 1)
  yelp_validation <- decomposeMatrix(dtm_onlyDict_SparseRemoved, yelp_validation, 2)
  save(list = c("onlyDict_SparseRemoved_yelp", "yelp_validation"), file =
"./intermidiate_data/onlyDict_SparseRemoved.Rdata")
}else if (option == 3){
  # OPTION 3: Only yelp$text with sparse remove
  noDict_SparseRemoved_yelp <- decomposeMatrix(dtm_noDict_SparseRemoved, yelp, 1)
  yelp_validation <- decomposeMatrix(dtm_noDict_SparseRemoved, yelp_validation, 2)
  save(list = c("noDict_SparseRemoved_yelp", "yelp_validation"), file =
"./intermidiate_data/noDict_SparseRemoved.Rdata")
}else if (option == 4){
  # OPTION 4: Raw dictionary + yelp$text with sparse remove
  aggr_SparseRemoved_yelp <- decomposeMatrix(aggrDtm_SparseRemoved, yelp)
  yelp_validation <- decomposeMatrix(aggrDtm_SparseRemoved_validation, yelp_validation)
  save(list = c("aggr_SparseRemoved_yelp", "yelp_validation"), file =
"./intermidiate_data/aggr_SparseRemoved.Rdata")
}

```

```{r model_1: Linear Regression}
load("./intermidiate_data/aggr_SparseRemoved.Rdata")
mlrModel <- lm(stars ~ ., data = aggr_SparseRemoved_yelp)
```

```
stars_prediction <- predict(mlrModel, newdata = yelp_validation[,
-which(colnames(yelp_validation) == "stars")])
stars_prediction <- pmax(stars_prediction, 1)
stars_prediction <- pmin(stars_prediction, 5)

## for plotting
cv.mlrModel <- cv.lm(aggr_SparseRemoved_yelp, mlrModel, m = 5, seed = 479, plotit =
c("Observed","Residual"), printit = F)

set.seed(479)
train.cl <- trainControl(method = "cv", number = 5 )
cv.mlrModel <- train(stars~., data = aggr_SparseRemoved_yelp, method = "lm", trControl =
train.cl)
cv.mlrModel$resample

mse_validation = MSE(stars_prediction, as.double(yelp_validation$stars)) # mse by comparing
predicted and true in validation set
rmse_validation = RMSE(stars_prediction, as.double(yelp_validation$stars))
Adjusted_R_square = 0.624
# "mse_cv" = c(0.99, 1.3, 1.29, 1.24, 1.27) # extracted from output directly
# "SS_cv" = c(796, 1039, 1033, 991, 1017) # extracted from output directly
# "mse_cv_mean_and_sd" = c(mean(c(0.99, 1.3, 1.29, 1.24, 1.27)), sd(c(0.99, 1.3, 1.29, 1.24,
1.27)))

# check most significant vars

mlrModel$rank
significant_ones <- summary(mlrModel)$coef[summary(mlrModel)$coef[,4] <= 0.05, 4]
length(significant_ones)
significant_ones[order(significant_ones)[1:10]]

```

```{r }
# Xmat <- data.matrix(aggr_SparseRemoved_yelp[,-which(colnames(yelp_validation) ==
"stars")])
# Ymat <- aggr_SparseRemoved_yelp$stars
#
# # Fit lasso model
# # model_lasso <- glmnet(Xmat, Ymat, standardize = F, alpha=1, lambda =
model_lasso_cv$lambda.min)
#
# set.seed(1)
# model_lasso_cv <- cv.glmnet(Xmat, Ymat, nfold=10, alpha=1)
# plot(model_lasso_cv, main = "MSE vs. Log(lambda)")
#
```

```
# # prediction on train set
# pred_train_lasso <- predict(model_lasso_cv,
#                 newx = Xmat,
#                 s = model_lasso_cv$lambda.min)
#
# pred_train_lasso <- pmax(pred_train_lasso, 1)
# pred_train_lasso <- pmin(pred_train_lasso, 5)
# MSE(Ymat, pred_train_lasso)
#
#
# # prediction on validation set
# pred_validation_lasso <- predict(model_lasso_cv,
#                 newx = data.matrix(yelp_validation[,-which(colnames(yelp_validation) ==
"stars")]),
#                 s = model_lasso_cv$lambda.min)
#
# pred_validation_lasso <- pmax(pred_validation_lasso, 1)
# pred_validation_lasso <- pmin(pred_validation_lasso, 5)
# MSE(yelp_validation$stars, pred_validation_lasso)
# RMSE(yelp_validation$stars, pred_validation_lasso)
#
# sig_ones <- summary(pred_train_lasso)$coef
```

```{r model_2: Lasso Regression}
set.seed(479)

x <- model.matrix(stars~., data = aggr_SparseRemoved_yelp)
y <- as.numeric(aggr_SparseRemoved_yelp$stars)

new_x <- model.matrix(~., data = yelp_validation[, -which(colnames(yelp_validation) ==
"stars")])

yelp.lasso <- glmnet(x, y, alpha = 1)
cv.lasso <- cv.glmnet(x, y, alpha = 1, nfold = 5)

review_logreg <- glmnet(x, y, alpha = 1, lambda = cv.lasso$lambda.min)

logregcoef = as.matrix(coef(review_logreg))
odds_ratio = as.matrix(exp(coef(review_logreg)))

mypalette <- brewer.pal(8, "Set2")
plot(yelp.lasso, xvar="lambda", lebel=TRUE,lwd=0.01, col=mypalette, main = "Coefficients vs.
log(lambda)")
abline(h=0, lwd=0.01, lty=2, col="grey")
```

```
plot(cv.lasso, main = "MSE vs. Log(lambda)")

m <- predict(review_logreg, s = cv.lasso$lambda.min, newx = new_x)
m <- pmax(m, 1)
m <- pmin(m, 5)

m_train <- predict(review_logreg, s = cv.lasso$lambda.min, newx = x)
m_train <- pmax(m_train, 1)
m_train <- pmin(m_train, 5)

MSE(m_train, aggr_SparseRemoved_yelp$stars)
MSE(m, yelp_validation$stars)
RMSE(m, yelp_validation$stars)
sum(review_logreg$beta != 0)
cv.lasso$cvm[which(cv.lasso$lambda == cv.lasso$lambda.min)]
cv.lasso$lambda.min
```

```{r lasso with less sample}
set.seed(479)

rand <- sample(1:5000, 500)

x <- model.matrix(stars~., data = aggr_SparseRemoved_yelp[-c(rand), ])
y <- as.numeric(aggr_SparseRemoved_yelp[-c(rand), ]$stars)

new_x <- model.matrix(~., data = yelp_validation[, -which(colnames(yelp_validation) ==
"stars")])

yelp.lasso <- glmnet(x, y, alpha = 1)
cv.lasso <- cv.glmnet(x, y, alpha = 1, nfold = 5)

review_logreg <- glmnet(x, y, alpha = 1, lambda = cv.lasso$lambda.min)

logregcoef = as.matrix(coef(review_logreg))
odds_ratio = as.matrix(exp(coef(review_logreg)))

mypalette <- brewer.pal(8, "Set2")
plot(yelp.lasso, xvar="lambda", lebel=TRUE,lwd=0.01, col=mypalette, main = "Coefficients vs.
log(lambda)")
abline(h=0, lwd=0.01, lty=2, col="grey")

plot(cv.lasso, main = "MSE vs. Log(lambda)")
```

```
m <- predict(review_logreg, s = cv.lasso$lambda.min, newx = new_x)
m <- pmax(m, 1)
m <- pmin(m, 5)

m_train <- predict(review_logreg, s = cv.lasso$lambda.min, newx = x)
m_train <- pmax(m_train, 1)
m_train <- pmin(m_train, 5)

MSE(m_train, aggr_SparseRemoved_yelp[-c(rand), ]$stars)
MSE(m, yelp_validation$stars)
RMSE(m, yelp_validation$stars)
sum(review_logreg$beta != 0)
cv.lasso$cvm[which(cv.lasso$lambda == cv.lasso$lambda.min)]
cv.lasso$lambda.min
```
```{r tuned lasso}
set.seed(479)

x <- model.matrix(stars~., data = aggr_SparseRemoved_yelp)
y <- as.numeric(aggr_SparseRemoved_yelp$stars)

new_x <- model.matrix(~., data = yelp_validation[, -which(colnames(yelp_validation) ==
"stars")])

yelp.lasso <- glmnet(x, y, alpha = 1)
cv.lasso <- cv.glmnet(x, y, alpha = 1, nfold = 5)

review_logreg <- glmnet(x, y, alpha = 1, lambda = cv.lasso$lambda.1se)

logregcoef = as.matrix(coef(review_logreg))
odds_ratio = as.matrix(exp(coef(review_logreg)))

mypalette <- brewer.pal(8, "Set2")
plot(yelp.lasso, xvar="lambda", lebel=TRUE,lwd=0.01, col=mypalette, main = "Coefficients vs.
log(lambda)")
abline(h=0, lwd=0.01, lty=2, col="grey")

plot(cv.lasso, main = "MSE vs. Log(lambda)")

m <- predict(review_logreg, s = cv.lasso$lambda.1se, newx = new_x)
m <- pmax(m, 1)
m <- pmin(m, 5)

m_train <- predict(review_logreg, s = cv.lasso$lambda.1se, newx = x)
m_train <- pmax(m_train, 1)
m_train <- pmin(m_train, 5)
```

```
MSE(m_train, aggr_SparseRemoved_yelp$stars)
MSE(m, yelp_validation$stars)
RMSE(m, yelp_validation$stars)
sum(review_logreg$beta != 0)
cv.lasso$cvm[which(cv.lasso$lambda == cv.lasso$lambda.1se)]
cv.lasso$lambda.1se
```

```
```{r function for converting regression to classification}
classification <- function(pred_mat){
  pred_mat[(pred_mat < 1.5)] <- 1
  pred_mat[(pred_mat >= 1.5) & (pred_mat < 2.5)] <- 2
  pred_mat[(pred_mat >= 2.5) & (pred_mat < 3.5)] <- 3
  pred_mat[(pred_mat >= 3.5) & (pred_mat < 4.5)] <- 4
  pred_mat[(pred_mat >= 4.5)] <- 5
  return(pred_mat)
}


lasso_pred_y = classification(m)
actual_y = yelp_validation$stars

table(actual_y, lasso_pred_y)

linear_pred_y = classification(stars_prediction)
table(actual_y, linear_pred_y)

```
```