Text Mining

Objectives

- Text may contain important, useful information about our response of interest.
  - Can we predict how much one likes a movie, a restaurant or a product based on his/her reviews?
- One simple but effective way of learning from a text is through bag of words to convert raw text data into a numeric matrix.
- Then we apply existing methods that use numerical matrices to either extract useful information or carry out predictions.
- We will extend the regularization technique (LASSO) to classification problems.

Objectives

- In this lecture through the `Yelp` case study, we will use the `tm` package to transform text into a word frequency matrix.
- We will build a classifier and conduct sentiment analysis.
- Finally we build a word cloud to exhibit words for good reviews and bad reviews respectively.

Table of Contents

Case Study: Yelp Reviews

- Founded in 2004, Yelp is a platform that holds reviews for services including restaurants, salons, movers, cleaners and so on.
- In this study, we use a subset of 100,000 restuarant reviews try to answer the following questions:
  - **How are reviews related to ratings?**
  - **How well can we predict star rankings based on the text of reviews?**
- Note: can do analysis in situations where only reviews are available but no quantitative evaluations are given.

Packages used

1. Text Mining Packages
   - `tm`: a popular text mining package
   - Note: Ingo Feinerer created text mining package `tm` in 2008 while he was a phd student at TU Vienna. Users can deploy `Hadoop` to handle large textual data.
   - `SnowballC`: For Stemming
2. Word Cloud Packages
   - `RColorBrewer`: a package for color palette
   - `wordcloud`: a package for creating wordcloud
3. LASSO and Random Forest packages
   - `glmnet`
   - `randomForest`
   - `ranger`
   - `stringr`: useful string package

For the remaining lecture

- Do EDA as usual.
- Digitize the reviews into a large dimension of word frequency vectors.
- Use `glm` and `LASSO` methods to build models of rating based on the reviews
- Report testing errors comparing different models.

Read data

Using package data.table to read and to manipulate data is much faster than using read.csv() especially when the dataset is large.Let's first take a small piece of it to work through. We use fread with nrows = 1000 to avoid loading the entire dataset.

```
# Note: We might need to shuffle the data in order to get a random sample.
data.all <- fread("data/yelp_subset.csv", stringsAsFactors = FALSE)
data <- fread("data/yelp_subset.csv", nrows = 1000, stringsAsFactors = FALSE)
names(data)
str(data)
n <- nrow(data)
```

```
##  [1] "user_id"      "review_id"    "text"         "votes.cool"
##  [5] "business_id"  "votes.funny"  "stars"        "date"
##  [9] "type"         "votes.useful"
## Classes 'data.table' and 'data.frame':   1000 obs. of  10 variables:
##  $ user_id     : chr  "RQU7dwZTdCLfy7DQU2TY1Q" "53QaFbmZojYK0vv3RQagcw" "OVwdQ7JFDiZ3JGICBYuIHw" "te_j2wG9cT
##  $ review_id   : chr  "bl8w2cxQEIFexrPQxVa_jw" "g01UnMSATfvlR83THcuYEw" "wyQi7ux65-dUKt9aWsnT3g" "JBTpvFkonR
##  $ text        : chr  "Super cute shop with great jewelry and gifts.  I also really love their baby stuff: b
##  $ votes.cool  : int  0 1 0 0 2 2 0 0 1 0 ...
##  $ business_id : chr  "LPmFKFCwEMauGfYF0lWGnw" "IMnTtFn3c5qZ7gW0gWqPzA" "CVpKlqrjYCyjxnlkBCUK5A" "0Da5sfXzUQ
##  $ votes.funny : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ stars       : int  5 4 5 5 4 4 3 4 2 5 ...
##  $ date        : IDate, format: "2011-11-15" "2010-04-05" ...
##  $ type        : chr  "review" "review" "review" "review" ...
##  $ votes.useful: int  0 1 0 0 1 2 0 0 1 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Response

- The `rating` available to us has five levels.
- Could treat as a continuous, ordinal, or categorical variable.
- Logistic regression or LASSO models could handle a 5-level categorical variable.
- For simplicity, we regroup them into a binary settings.
- We create a new response `rating` such that a review will be good or 1 if the original rating is at least 4 or 5. Otherwise we will code it as a bad or 0.

```
levels(as.factor(data$stars))
```

```
## [1] "1" "2" "3" "4" "5"
```

```
data$rating <- c(0)
data$rating[data$stars >= 4] <- 1
data$rating <- as.factor(data$rating)
#summary(data) #str(data)
```

Response

Proportion of good ratings:

```
prop.table(table(data$rating))
```

```
##
##     0     1
## 0.398 0.602
```

Notice that 60% of the reviews are good ones.

How to handle date

Does rating relate to month or day of the weeks?

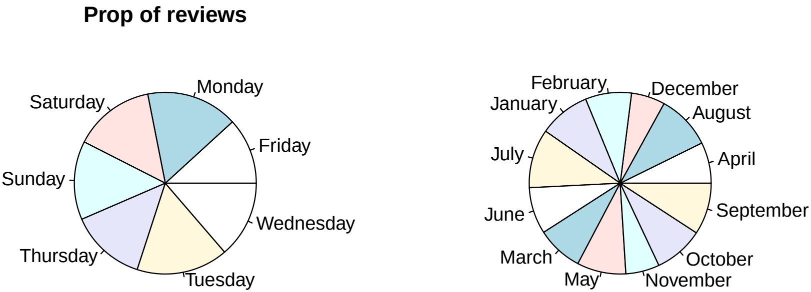
- Should we treat `date` as continuous variables or categorical ones?
  - Highly depends on the context and the goal of the study.
- In our situation, we are interested in knowing if people tend to leave reviews over the weekend and if those reviews are better?
- Let us use functions in `tidyverse` to format the dates and extract weekdays

```
weekdays <- weekdays(as.Date(data$date)) # get weekdays for each review
months <- months(as.Date(data$date))    # get months
```

How to handle date

Do people tend to leave a review over weekends? (months?)

```
par(mfrow=c(1,2))
pie(table(weekdays), main="Prop of reviews") # Pretty much evenly distributed
pie(table(months))
```



How to handle date

Proportion of Good reviews: Don't really see any patterns.

```
prop.table(table(data$rating, weekdays), 2)  # prop of the columns
prop.table(table(data$rating, weekdays), 1)  # prop of the rows
```

```
##    weekdays
##      Friday Monday Saturday Sunday Thursday Tuesday Wednesday
##   0  0.415  0.405    0.431  0.400    0.415   0.319     0.416
##   1  0.585  0.595    0.569  0.600    0.585   0.681     0.584
##    weekdays
##      Friday Monday Saturday Sunday Thursday Tuesday Wednesday
##   0  0.123  0.166    0.156  0.141    0.141   0.131     0.143
##   1  0.115  0.161    0.136  0.140    0.131   0.184     0.133
```

- How should we use a review as predictors?
  - Sentences, words, and sentiments are all informative.
- We will turn a text into a vector of features, each of which represents the words that are used.
  - We collect all possible words (referred to as a library or bag of all words).
  - We will then record frequency of each word used in the review/text.

Word term frequency table using ‘tm‘

- First form a `bag of words`: all the words appeared in the documents say `N` (in general, very large)
- For each document (row), record the frequency (count) of each word in the bag which gives us `N` values (notice: most of the entries are 0, as most words will *not* occur in every document)
- Output the document term matrix (`dtm`) as an input to a later model

Word term frequency table using ‘tm‘

Corpus: a collection of text

- `VCorpus()`: create Volatile Corpus
- `inspect()`: display detailed info of a corpus

```
data1.text <- data$text     # data1.text[1:5]
mycorpus1 <- VCorpus(VectorSource(data1.text))
mycorpus1
typeof(mycorpus1)   ## It is a list
# inspect the first corpus
inspect(mycorpus1[[1]])
# or use `as.character` to extract the text
#as.character(mycorpus1[[1]])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1000
## [1] "list"
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 253
##
## Super cute shop with great jewelry and gifts.  I also really love their baby stuff: bibs, clothes, toys.  It
## A lot of their art and gifts are made by local artists!
```

Word term frequency table using ‘tm‘

Data cleaning using `tm_map()`

- Before transforming the text into a word frequency matrix, we should transform the text into a more standard format and clean the text by removing punctuation, numbers and some common words that do not have predictive power (a.k.a. stopwords)
  - e.g. pronouns, prepositions, conjunctions).
- We use the `tm_map()` function with different available transformations
  - `removeNumbers()`
  - `removePunctuation()`
  - `removeWords()`
  - `stemDocument()`
  - `stripWhitespace()`.

Word term frequency table using ‘tm‘

Data cleaning using `tm_map()`

```
# Converts all words to lowercase
mycorpus_clean <- tm_map(mycorpus1, content_transformer(tolower))
# Removes common English stopwords (e.g. "with", "i")
mycorpus_clean <- tm_map(mycorpus_clean, removeWords, stopwords("english"))
# Removes any punctuation
# NOTE: This step may not be appropriate if you want to account for differences
#       on semantics depending on which sentence a word belongs to if you end up
#       using n-grams or k-skip-n-grams.
#       Instead, periods (or semicolons, etc.) can be replaced with a unique
#       token (e.g. "[PERIOD]") that retains this semantic meaning.
mycorpus_clean <- tm_map(mycorpus_clean, removePunctuation)
# Removes numbers
mycorpus_clean <- tm_map(mycorpus_clean, removeNumbers)
# Stem words
mycorpus_clean <- tm_map(mycorpus_clean, stemDocument, lazy = TRUE)
lapply(mycorpus_clean[4:5], as.character)
```

```
## $‘4‘
## [1] "asia bar choic go year now know everyon work now even occasion work doorman owner pretti cool peopl alw
##
## $‘5‘
## [1] "star red velvet fanat sad red velvet live standard cupcak will make come back sever choic recommend tri
```

Word term frequency table using ‘tm‘

Word frequency matrix

Now we transform each review into a word frequency matrix using the function `DocumentTermMatrix()`.

```
dtm1 <- DocumentTermMatrix( mycorpus_clean )   ## library = collection of words from all documents
class(dtm1)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

Word term frequency table using ‘tm‘

Word frequency matrix

```
inspect(dtm1) # typeof(dtm1)  #length(dimnames(dtm1)$Terms)
```

```
## <<DocumentTermMatrix (documents: 1000, terms: 7161)>>
## Non-/sparse entries: 51588/7109412
## Sparsity           : 99%
## Maximal term length: 73
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  food get good great just like one place realli time
##   113    3   1    3     0    2    1   2     2      2    1
##   129    1   3    3     0    1    9   6     3      3    0
##   184    0   5    7     1    1    3   2     0      2    1
##   216    0   4    0     0    0    1   0     0      0    0
##   269    3   1    2     5    3    3   0     1      2    2
##   336    3   1    3     0    2    5   2    11      0    2
##   404    0   0    0     0    2    4   2     1      0    1
##   454    0   3    4     3    3    6   5     4      4    4
##   735    3   2    0     0    3    1   2    11      2    2
##   92     5   6    6     1    8    8   4     6      4    4
```

Word term frequency table using ‘tm‘

Word frequency matrix

Take a look at the dtm.

```
colnames(dtm1)[7150:7161] # the last a few words in the bag
```

```
##  [1] "zest"     "zillion"  "zing"     "zip"      "zippi"    "zoc"
##  [7] "zod"      "zoe"      "zoltar"   "zone"     "zoo"      "zucchini"
```

```
# another way to get list of words
# dimnames(dtm1)$Terms[7000:7161]
dim(as.matrix(dtm1))  # we use 7161 words as predictors
```

```
## [1] 1000 7161
```

Word term frequency table using 'tm'

Word frequency matrix

Document 1, which is row1 in the dtm.

```
inspect(dtm1[1,])  #Non-/sparse entries: number of non-zero entries vs. number of zero entries
```

```
## <<DocumentTermMatrix (documents: 1, terms: 7161)>>
## Non-/sparse entries: 25/7136
## Sparsity           : 100%
## Maximal term length: 73
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs also art artist babi bib brows chao cloth gift great
##    1    1   1      1    1   1     1    1     1    3     2
```

It has 25 distinctive words; in other words, there 25 non-zero cells out of 7161 bag of words.

```
as.matrix(dtm1[1, 500:525])  # most of the cells are 0
```

```
##     Terms
## Docs baseballs baselin basi basic basil basket basketbal bass bastard
##    1         0       0    0     0     0      0         0    0       0
##     Terms
## Docs bastid bat bataclan batch bath bathroom bathtub batman batter battl
##    1      0   0        0     0    0        0       0      0      0     0
##     Terms
## Docs bay bball bbb bbq bbqi beach bead
##    1   0     0   0   0    0     0    0
```

Word term frequency table using ‘tm‘

Word frequency matrix

This is because review 1 only consists of 28 words after all the cleansing. (some words are repeated)

```
sum(as.matrix(dtm1[1,]))
```

```
## [1] 28
```

We may

```
colnames(as.matrix(dtm1[1, ]))[which(as.matrix(dtm1[1, ]) != 0)]
```

```
##  [1] "also"    "art"     "artist"  "babi"    "bib"     "brows"   "chao"
##  [8] "cloth"   "compet"  "cute"    "gift"    "grab"    "great"   "jewelri"
## [15] "local"   "lot"     "love"    "made"    "place"   "realli"  "shop"
## [22] "stuff"   "super"   "toy"     "villag"
```

```
as.character(mycorpus1[[1]]) #original text
```

```
## [1] "Super cute shop with great jewelry and gifts.  I also really love their baby stuff: bibs, clothes, toys
```

Word term frequency table using 'tm'

Reduce the size of the bag

Many words do not appear nearly as often as others. If your cleaning was done appropriately, it will hopefully not lose much of the information if we drop such rare words. So, we first cut the bag to only include the words appearing at least 1% (or the frequency of your choice) of the time. This reduces the dimension of the features extracted to be analyzed.

```
threshold <- .01*length(mycorpus_clean)   # 1% of the total documents
words.10 <- findFreqTerms(dtm1, lowfreq=threshold)  # words appearing at least among 1% of the documents
length(words.10)  # dim reduces to 1128
```

```
## [1] 1128
```

```
words.10[580:600]
```

```
##  [1] "luck"      "lunch"     "mac"       "macaron"   "machin"
##  [6] "made"      "magic"     "main"      "major"     "make"
## [11] "man"       "manag"     "mango"     "mani"      "margarita"
## [16] "mark"      "market"    "masala"    "mash"      "matter"
## [21] "may"
```

Word term frequency table using ‘tm‘

Reduce the size of the bag

```
dtm.10<- DocumentTermMatrix(mycorpus_clean, control = list(dictionary = words.10))
dim(as.matrix(dtm.10))
```

```
## [1] 1000 1128
```

```
colnames(dtm.10)[40:50]
```

```
##  [1] "anyway"  "anywher" "apart"   "apolog"  "appar"   "appet"   "appl"
##  [8] "appoint" "appreci" "arbor"   "area"
```

Word term frequency table using ‘tm‘

Reduce the size of the bag `removeSparseTerms()`:

Another way to reduce the size of the bag is to use `removeSparseTerms`

```
dtm.10.2 <- removeSparseTerms(dtm1, 1-.01)  # control sparsity < .99
inspect(dtm.10.2)
```

```
## <<DocumentTermMatrix (documents: 1000, terms: 929)>>
## Non-/sparse entries: 38204/890796
## Sparsity           : 96%
## Maximal term length: 12
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  food get good great just like one place realli time
##   113    3   1    3     0    2    1   2     2      2    1
##   129    1   3    3     0    1    9   6     3      3    0
##   216    0   4    0     0    0    1   0     0      0    0
##   269    3   1    2     5    3    3   0     1      2    2
##   336    3   1    3     0    2    5   2    11      0    2
##   404    0   0    0     0    2    4   2     1      0    1
##   454    0   3    4     3    3    6   5     4      4    4
##   459    3   4    2     1    2    5   3     0      1    0
##   735    3   2    0     0    3    1   2    11      2    2
##   92     5   6    6     1    8    8   4     6      4    4
```

```
# colnames(dtm.10.2)[1:50]
# words that are in dtm.10 but not in dtm.10.2
```

Word term frequency table using ‘tm‘

Reduce the size of the bag We end up with two different bags because

- `findFreqTerms()`: counts a word multiple times if it appears multiple times in one document.
- `removeSparseTerms()`: keep words that appear at least once in X% of documents.

Word term frequency table using ‘tm‘

One step to get `DTM`

We consolidate all possible processing steps to the following clean `R-chunk`, turning texts (input) into `Document Term Frequency` which is a sparse matrix (output) to be used in the down-stream analyses.

All the `tm_map()` can be called inside `DocumentTermMatrix` under parameter called `control`. Here is how.

Word term frequency table using ‘tm‘

One step to get DTM

```
# Turn texts to corpus
mycorpus1  <- VCorpus(VectorSource(data1.text))
# Control list for creating our DTM within DocumentTermMatrix
# Can tweak settings based off if you want punctuation, numbers, etc.
control_list <- list( tolower = TRUE,
                      removePunctuation = TRUE,
                      removeNumbers = TRUE,
                      stopwords = stopwords("english"),
                      stemming = TRUE)
# dtm with all terms:
dtm.10.long  <- DocumentTermMatrix(mycorpus1, control = control_list)
#inspect(dtm.10.long)
# kick out rare words
dtm.10<- removeSparseTerms(dtm.10.long, 1-.01)
#inspect(dtm.10)
# look at the document 1 before and after cleaning
# inspect(mycorpus1[[1]])
# after cleaning
# colnames(as.matrix(dtm1[1, ]))[which(as.matrix(dtm1[1, ]) != 0)]
inspect(dtm.10)   # 950 words retained
```

```
## <<DocumentTermMatrix (documents: 1000, terms: 950)>>
## Non-/sparse entries: 39501/910499
## Sparsity           : 96%
## Maximal term length: 12
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  food get good great just like one place realli time
##   113    3   1    3     0    2    1   2     2      2    1
```

For this lecture, we are focusing on just word frequency. There are ways of dealing with things like word order using methods like n-grams. We will skip those today but if you are interested please look at the full lecture on Canvas.

Once we have turned a text into a vector, we can then apply any methods suitable for the settings. In our case we will use logistic regression models and LASSO to explore the relationship between `ratings` and `text`.

Note: For data preparation see full lecture on CANVAS. We have processed the entire data set into a word frequency matrix and written out all 100,000 documents into "YELP_tm_freq.csv". We will use that for subsequent analyses.

Splitting data

Let's first read in the processed data with text being a vector.

```
data2 <- fread("data/YELP_tm_freq.csv")  #dim(data2)
names(data2)[1:20] # notice that user_id, stars and date are in the data2
```

```
##  [1] "user_id"   "stars"     "date"      "rating"    "abl"
##  [6] "absolut"   "accept"    "accommod"  "across"    "actual"
## [11] "add"       "addit"     "admit"     "afford"    "after"
## [16] "afternoon" "age"       "ago"       "agre"      "ahead"
```

```
dim(data2)
```

```
## [1] 100000   1076
```

```
data2$rating <- as.factor(data2$rating)
table(data2$rating)
```

```
##
##     0     1
## 37042 62958
```

```
#str(data2)  object.size(data2)  435Mb!!!
```

Splitting data

As one standard machine learning process, we first split data into two sets one training data and the other testing data. We use training data to build models, choose models etc and make final recommendations. We then report the performance using the testing data.

Reserve 10000 randomly chosen rows as our test data (`data2.test`) and the remaining 90000 as the training data (`data2.train`)

```
set.seed(1)  # for the purpose of reporducibility
n <- nrow(data2)
test.index <- sample(n, 10000)
# length(test.index)
data2.test <- data2[test.index, -c(1:3)] # only keep rating and the texts
data2.train <- data2[-test.index, -c(1:3)]
dim(data2.train)
```

```
## [1] 90000  1073
```

Analysis 1: LASSO

We first explore a logistic regression model using LASSO. The regularization techniques used in linear regression are readily applied to logistic regression (see the appendix for details). The following R-chunk runs a LASSO model with $\alpha = .99$. The reason we take an elastic net is to enjoy the nice properties from both `LASSO` (impose sparsity) and `Ridge` (computationally stable).

LASSO takes sparse design matrix as an input. So make sure to extract the sparse matrix first as the input in cv.glm(). It takes about 1 minute to run cv.glm() with sparse matrix or 11 minutes using the regular design matrix.

```
---
## may give it a try to run LASSO here
y <- data2.train$rating
X1 <- sparse.model.matrix(rating~., data=data2.train)[, -1]
set.seed(2)
result.lasso <- cv.glmnet(X1, y, alpha=.99, family="binomial") # notice alpha = .99.
# 1.25 minutes in my MAC
plot(result.lasso)
# this this may take you long time to run, we save result.lasso
saveRDS(result.lasso, file="data/TextMining_lasso.RDS")
# result.lasso can be assigned back by
# result.lasso <- readRDS("data/TextMining_lasso.RDS")
# number of non-zero words picked up by LASSO when using lambda.1se
coef.1se <- coef(result.lasso, s="lambda.1se")
lasso.words <- coef.1se@Dimnames[[1]] [coef.1se@i][-1] # non-zero variables without intercept.
```

Analysis 1: LASSO

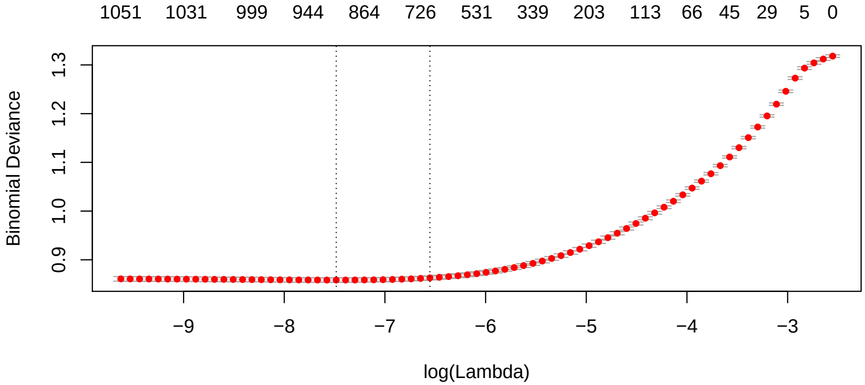
Try to kick out some not useful words (Warning: this may crash your laptop!!!) Because of the computational burden, I have saved the LASSO results and other results into `TextMining_lasso.RDS` and `TextMining_glm.RDS`.

```
# or our old way to extract non-zero coefficients
coef.1se <- coef(result.lasso, s="lambda.1se")
coef.1se <- coef.1se[which(coef.1se !=0),]
lasso.words <- rownames(as.matrix(coef.1se))[-1]
summary(lasso.words)
---
```

Analysis 1: LASSO

We resume our analyses by loading the LASSO results here. We extract useful variables using `lambda.1se`

```
result.lasso <- readRDS("data/TextMining_lasso.RDS")
plot(result.lasso)
```



Analysis 1: LASSO

```
coef.1se <- coef(result.lasso, s="lambda.1se")
coef.1se <- coef.1se[which(coef.1se !=0),]
lasso.words <- rownames(as.matrix(coef.1se))[-1]  #length(lasso.words)
summary(lasso.words) # return about 700 words
```

```
##    Length     Class      Mode
##       700 character character
```

Analysis 2: Relaxed LASSO

As an alternative model we will run our relaxed LASSO. Input variables are chosen by LASSO and we get a regular logistic regression model. Once again it is stored as `result.glm` in `TextMining.RData`. The code is available the full lecture on CANVAS.

```
## codes to run glm. Give it a try
# sel_cols <- c("rating", lasso.words)
#     # use all_of() to specify we would like to select variab
# data_sub <- data2.train %>% select(all_of(sel_cols))
# result.glm <- glm(rating~., family=binomial, data_sub) # tak
#     ## glm() returns a big object with unnecessary informati
# saveRDS(result.glm,
#      file = "data/TextMining_glm.RDS")
```

Analysis 3: Word cloud! (Sentiment analysis)

Logistic regression model connects the chance of `being good` given a text/review. What are the `nice` (or positive) words and how much it influence the chance being good? In addition to explore the set of good words we also build word clouds to visualize the correlation between positive words and negative words.

1. Order the glm positive coefficients (positive words). Show them in a word cloud. The size of the words indicates the strength of positive correlation between that word and the chance being a good rating.
2. Order the glm negative coefficients (negative words)

TIME TO PLOT A WORD CLOUD!! Plot the world clouds, the size of the words are prop to the logistic reg coef's

Analysis 3: Word cloud! (Sentiment analysis)

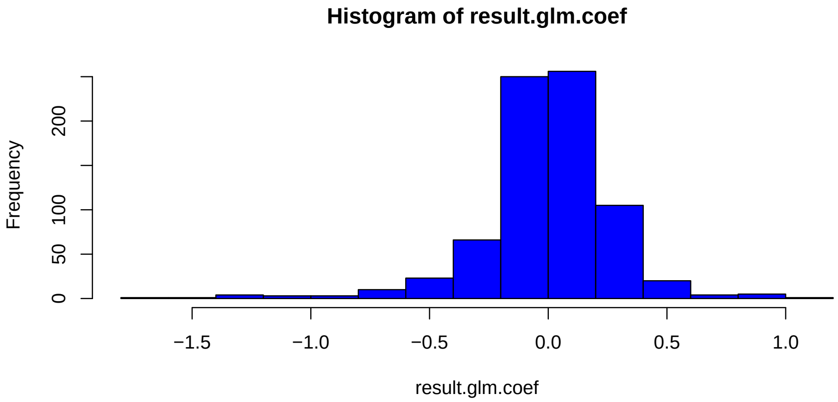
Positive word cloud:

```
# load the glm results
# result.glm <- readRDS("data/result.glm.RDS")
result.glm <- readRDS("data/TextMining_glm_small.RDS")  # also have a smaller version
result.glm.coef <- coef(result.glm)
result.glm.coef[200:250]
```

```
##    either      els elsewher  employe    empti      end    enjoy   enough
##   -0.2116  -0.0635  -0.7028  -0.3244  -0.1820  -0.0645   0.1763  -0.0464
##     entir   especi espresso      etc    event     ever    everi  everyon
##   -0.1275   0.0450   0.0564   0.0644   0.0482   0.1213   0.2182   0.2033
##   everyth    exact    excel   except    excit   expect   expens  explain
##    0.2329   0.0800   0.9018   0.0951  -0.3869  -0.1750  -0.2951   0.2601
##     extra   extrem      eye    fabul     fact     fall   famili      fan
##    0.2036   0.0994   0.0713   0.8936  -0.0790  -0.1739   0.3020   0.2157
##     fanci  fantast      far     fast  favorit     felt    figur     find
##    0.3052   0.8570   0.1260   0.2074   0.7734  -0.1711  -0.0952   0.0709
##      fine    first     fish     five      fix     folk     food     for.
##   -0.4441   0.0557  -0.0214   0.4874   0.1453   0.1013  -0.1364  -0.0918
##     forev   forget   forgot
##   -0.4770   0.1005  -0.1052
```

Analysis 3: Word cloud! (Sentiment analysis)

```r
hist(result.glm.coef, col = 'blue')
```



Analysis 3: Word cloud! (Sentiment analysis)

```r
# pick up the positive coef's which are positively related to the prob of being a good review
good.glm <- result.glm.coef[which(result.glm.coef > 0)]
good.glm <- good.glm[-1]  # took intercept out
names(good.glm)[1:20]  # which words are positively associated with good ratings
```

```
##  [1] "abl"      "absolut"  "accommod" "add"      "admit"    "afford"
##  [7] "age"      "ahead"    "all"      "allow"    "along"    "alreadi"
## [13] "also"     "alway"    "amaz"     "and"      "ann"      "anyway"
## [19] "anywher"  "appl"
```

```r
good.fre <- sort(good.glm, decreasing = TRUE) # sort the coef's
round(good.fre, 4)[1:20] # leading 20 positive words, amazing!
```

```
##   heaven    excel    fabul     amaz  fantast   delici   awesom  perfect
##    1.174    0.902    0.894    0.868    0.857    0.804    0.796    0.777
##  favorit knowledg     best      yum     beat     glad     love    great
##    0.773    0.617    0.594    0.568    0.560    0.552    0.550    0.508
##     five  delight   wonder      die
##    0.487    0.482    0.479    0.476
```

```r
length(good.fre)  # 390 good words
```

```
## [1] 390
```

```r
# hist(as.matrix(good.fre), breaks=30, col="red")
good.word <- names(good.fre)  # good words with a decreasing order in the coeff's
```

Analysis 3: Word cloud! (Sentiment analysis)

The above chunk shows in detail about the weight for positive words. We only show the positive word-cloud here. One can tell the large positive words are making sense in the way we do expect the collection of large words should have a positive tone towards the restaurant being reviewed.

Analysis 3: Word cloud! (Sentiment analysis)

```r
cor.special <- brewer.pal(8,"Dark2")  # set up a pretty color scheme
wordcloud(good.word[1:300], good.fre[1:300],  # make a word cloud
          colors=cor.special, ordered.colors=F)
```

Concern: Many words got trimmed due to stemming? We may redo dtm without stemming?

Analysis 3: Word cloud! (Sentiment analysis)

Negative word cloud:

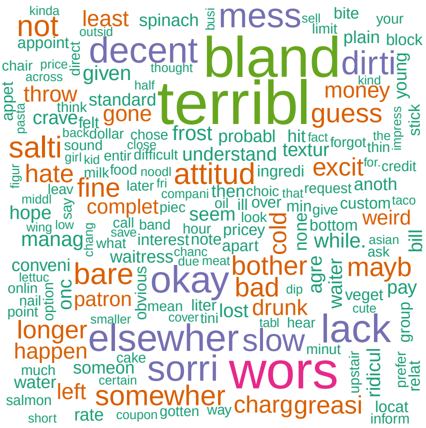
Similarly to the negative coef's which is positively correlated to the prob. of being a bad review

```
bad.glm <- result.glm.coef[which(result.glm.coef < 0)]
# names(bad.glm)[1:50]
cor.special <- brewer.pal(6,"Dark2")
bad.fre <- sort(-bad.glm, decreasing = TRUE)
round(bad.fre, 4)[1:40]
```

```
##      worst    mediocr       rude    terribl    horribl     overpr
##      1.642      1.554      1.360      1.286      1.253      1.204
##      bland       wors    alright   unfortun      gross       wast
##      1.188      1.073      1.000      0.917      0.908      0.817
##       poor       lack       okay     averag   elsewher      sorri
##      0.750      0.728      0.717      0.707      0.703      0.681
##     decent       noth       mess disappoint        sad      dirti
##      0.646      0.637      0.615      0.615      0.585      0.575
##        dri       slow      howev       paid    attitud       bare
##      0.573      0.551      0.546      0.530      0.525      0.512
##       suck      salti     suppos        not      forev        whi
##      0.505      0.503      0.493      0.489      0.477      0.464
##   somewher      guess       fine     bother
##      0.458      0.446      0.444      0.442
```

Analysis 3: Word cloud! (Sentiment analysis)

```
# hist(as.matrix(bad.fre), breaks=30, col="green")
bad.word <- names(bad.fre)
wordcloud(bad.word[1:300], bad.fre[1:300],
          color=cor.special, ordered.colors=F)
```



Analysis 4: Predictions

We have obtained two sets of models one from `LASSO` the other from `relaxed LASSO`. To compare the performance as classifiers we will evaluate their `mis-classification error` using `testing data`.

Analysis 4: Predictions

1. **How does glm do in terms of classification?**

```
predict.glm <- predict(result.glm, data2.test, type = "response")
class.glm <- ifelse(predict.glm > .5, "1", "0")
# length(class.glm)
testerror.glm <- mean(data2.test$rating != class.glm)
testerror.glm   # mis classification error is 0.19
```

```
## [1] 0.193
```

Analysis 4: Predictions

2) **LASSO model using `lambda.1se`**

Once again we evaluate the testing performance of `LASSO` solution.

```
predict.lasso.p <- predict(result.lasso, as.matrix(data2.test[, -1]), type = "response", s="lambda.1se")
  # output lasso estimates of prob's
predict.lasso <- predict(result.lasso, as.matrix(data2.test[, -1]), type = "class", s="lambda.1se")
  # output majority vote labels
# LASSO testing errors
mean(data2.test$rating != predict.lasso)   # .19
```

```
## [1] 0.193
```

Analysis 4: Predictions

Comparing the two predictions through testing errors we do not see much of the difference. We could use either final models for the purpose of the prediction.

Conclusion

In this lecture, we apply LASSO to classify good/bad review based on the text. The core technique for text mining is a simple bag of words, i.e. a word frequency matrix. The problem becomes a high-dimensional problem. Using LASSO, we reduce dimension and train a model with high predictive power. Based on the model, we find out the positive/negative words and build a word cloud.

LASSO for classification

The regularization techniques used in regression are readily applied to classification problems. Here we will penalize the coefficients while maximizing the likelihood function or minimizing the -loglikelihood function.

LASSO for classification

For a given lambda we minimize -loglikelihood. Here is the LASSO solutions:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} -\frac{1}{n}\log(\mathcal{L}\rangle\|) + \lambda\{|\beta_1| + |\beta_2|, \ldots + |\beta_p|\}$$

Similarly we obtain the solution for elastic net using the general penalty functions:

$$\Big(\frac{1-\alpha}{2}\Big)\|\beta\|_2^2 + \alpha\|\beta\|_1$$

LASSO for classification

For the remaining lecture:

- Do EDA as usual.
- Digitize the reviews into a large dimension of word frequency vectors.
- Use`glm` and `LASSO` methods to build models of rating based on the reviews
- Report testing errors comparing different models.