Predicting the number of stars a yelp review has

Alice and Phillis

Data Source

- The data was freely available to download as a .TGZ file from the yelp website
- We had 5 files in JSON format:
  - Review data, tip data, business data, user data and check-in data
  - We only need to worry about the review and the user data

yelp_academic_dataset_business	21/03/2021 20:43	JSON File	121,466 KB
yelp_academic_dataset_checkin	21/03/2021 20:43	JSON File	388,938 KB
yelp_academic_dataset_review	21/03/2021 20:46	JSON File	6,774,100 KB
yelp_academic_dataset_tip	21/03/2021 20:46	JSON File	224,910 KB
yelp_academic_dataset_user	21/03/2021 20:48	JSON File	3,598,150 KB

Huge files!

Data Source - issues

Large files

JSON format

No missing data!

Only take the top million reviews

- Cross reference this with the user files
- Save the processed files as a .csv so there is no need to process them every time the code is run

Use package 'jsonlite' to turn the JSON file into an R dataframe

Data Processing - review data

- Separate the date into year, month, day
- Combine the total votes the user received
- Remove the useless columns

review_id	user_id	~~business_id~~	stars	useful	funny	cool	text	date
8bFej1QE5LXp4O05qjGqXA	YoVfDbnISlW0f7abNQACIg	~~RA4V8pr014UyUbDvI-LW2A~~	4	1	2	3	The store is...	2015-07-03 20:38:25

review_id	user_id	stars	votes_received	text	year	month	day
8bFej1QE5LXp4O05qjGqXA	YoVfDbnISlW0f7abNQACIg	4	6	The store is...	2015	07	03

Data Processing - user data

- Mutate the elite variable into a yes/no variable
- Summarise compliments the user received
- Add a “user_” prefix to help us distinguish between the user data and review data
- Remove the “average_stars” variable

```
colnames(user_data)
```

```
##  [1] "user_id"             "name"                "review_count"
##  [4] "yelping_since"       "useful"              "funny"
##  [7] "cool"                "elite"               "friends"
## [10] "fans"                "average_stars"       "compliment_hot"
## [13] "compliment_more"     "compliment_profile" "compliment_cute"
## [16] "compliment_list"     "compliment_note"     "compliment_plain"
## [19] "compliment_cool"     "compliment_funny"    "compliment_writer"
## [22] "compliment_photos"
```

```
colnames(user_data_tidy)
```

```
## [1] "user_id"                    "user_review_count"
## [3] "user_yelping_since"         "user_friend_count"
## [5] "user_fans"                  "user_average_stars"
## [7] "user_elite_status"          "user_votes_sent"
## [9] "user_compliments_received"
```

Tidytext package for R

- We use the bing dataset, part of the tidytext package for R
- It has a huge list of words and the sentiment (either positive or negative)
- We can:
  - Count the number of positive/negative words in each review text and use this as an explanatory variable
  - Use each word as a new column name - the entries in each row are the frequency of that word in the text

word	sentiment
abnormal	negative
abolish	negative
abominable	negative
abominably	negative
abominate	negative
abomination	negative
abort	negative
⋮	⋮

Data Processing - two dataframes!

- Both datasets include information about the user
  - Number of reviews they have given
  - The number of votes they have received
  - etc

no_freq_data:
- Includes columns - one for each positive/negative word
- The value in that column gives the frequency of that word in the review

freq_data:
- Includes one column for the total number of positive words included in the review
  - And another for the total number of negative words used in the review

Which one will create a better model?

Data Exploration - frequency of positive words

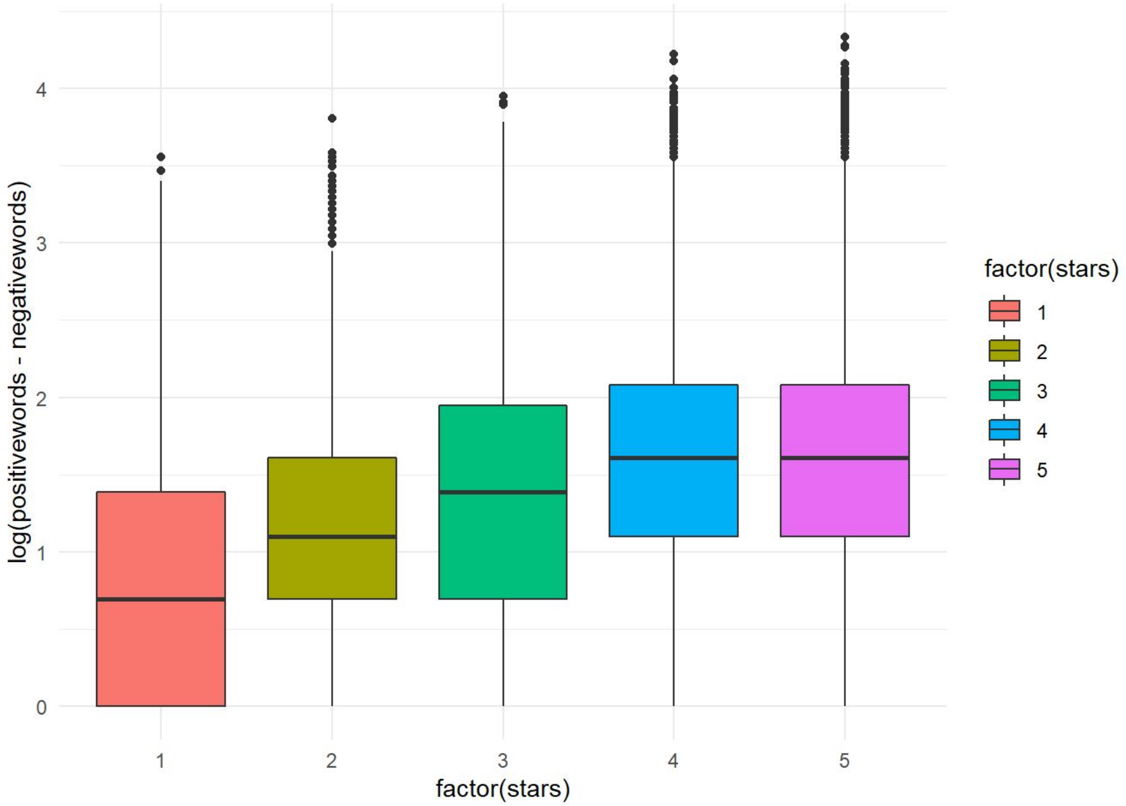
word	n
good	4835
great	4223
like	3492
well	1853
nice	1795
best	1781
love	1501
friendly	1455
delicious	1418
pretty	1287

Data Exploration - frequency of negative words

word	n
bad	893
fried ??	579
hard	469
disappointed	436
cold	369
wrong	353
problem	341
cheap ??	297
rude	289
expensive	284

Data Exploration

Stars against (positive words - negative words)



Analytical Plan

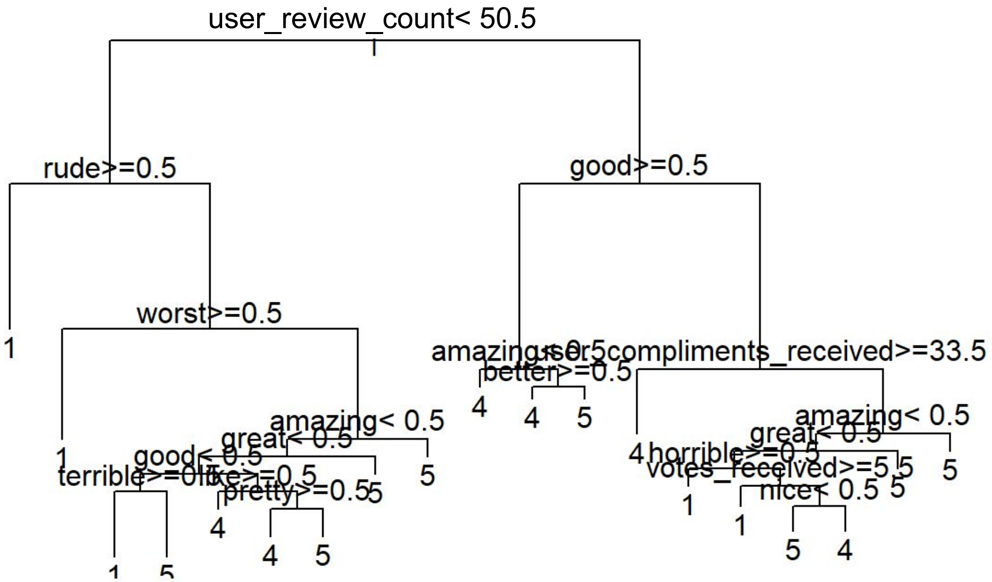
The plan is to use a random forest

- Good for classification data
- There is no underlying linear relationship between the response and explanatory variables
- Works fast on low RAM computers

First, I will try using a tree in rpart() to decide between the two datasets, before creating the final model

Analytical Plan

Let’s try creating a tree using freq_data



freq_data

pred → true ↓	1	2	3	4	5
1	28	11	3	8	6
2	1	4	6	8	9
3	4	6	6	11	7
4	4	14	18	47	52
5	15	15	27	58	132

43.4% correct

no_freq_data

pred → true ↓	1	2	3	4	5
1	32	11	7	4	7
2	7	5	8	6	11
3	4	5	6	13	10
4	8	11	18	40	54
5	8	15	20	61	123

42.4% correct

Results - predicting stars

- We create a random forest using the ranger package

```
## Ranger result
##
## Call:
##  ranger(factor(stars) ~ ., data = no_freq_data, mtry = p^0.5,      verbose = FA
LSE, min.node.size = 1, num.trees = 100, importance = "permutation",      classifi
cation = TRUE)
##
## Type:                             Classification
## Number of trees:                  100
## Sample size:                      1000000
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error:             45.62 %
```

Classifies 55% perfectly but 82% are correct to within one star!

Results - predicting stars

```
##                year                   month                     day
##        0.0132607592            0.0005751142            0.0003720364
##      votes_received       user_review_count      user_yelping_since
##        0.0121287778            0.0459241149            0.0058191642
##   user_friend_count               user_fans       user_elite_status
##        0.0070796757            0.0164257937            0.0062018406
##     user_votes_sent user_compliments_received           positivewords
##        0.0299881788            0.0302868661            0.0475714025
##       negativewords
##        0.0798317979
```

Results - stars as a high/low factor

```
## Ranger result
##
## Call:
##  ranger(factor(stars_f) ~ ., data = no_freq_data_f, mtry = p^0.5,      verbose
= FALSE, min.node.size = 1, num.trees = 100, importance = "permutation",      clas
sification = TRUE)
##
## Type:                             Classification
## Number of trees:                  100
## Sample size:                      1000000
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error:             20.37 %
```

Discussion

- Strengths
  - Large dataset - use different sets of training, validation and testing to build model.
  - Detailed information about the users.
  - No missing data.

- Limitations
  - Processing power!
  - Words misleading
  - Words not included in the package.
  - Misspellings of words
  - Words incorrectly classified (eg cheap)

Thank you!