Text Analytics - Predicting AirBnB review scores

Suzana Iacob

01/12/2019

Data Exploration

The dataset is from AirBnb and contains ratings and reviews from New York between March 2011 and March 2018. We will use NLP techniques to process and predict the ratings.

```
reviews = read.csv("airbnb-small.csv", stringsAsFactors = F)
reviews$X = NULL
table(reviews$review_scores_rating)
```

```
##
##    1    2    3    4    5
##   62   56  156  708 3191
```

We see above the number of reviews for each rating. And the average length:

```
one = reviews  %>% filter(review_scores_rating == 1)
two = reviews  %>% filter(review_scores_rating == 2)
three = reviews  %>% filter(review_scores_rating == 3)
four = reviews  %>% filter(review_scores_rating == 4)
five = reviews  %>% filter(review_scores_rating == 5)
sum(nchar(reviews$comments))/nrow(reviews)
```

```
## [1] 297.6779
```

```
sum(nchar(one$comments))/nrow(one)
```

```
## [1] 464.5968
```

```
sum(nchar(two$comments))/nrow(two)
```

```
## [1] 597.7321
```

```
sum(nchar(three$comments))/nrow(three)
```

```
## [1] 388.8013
```

```
sum(nchar(four$comments))/nrow(four)
```

```
## [1] 276.476
```

```
sum(nchar(five$comments))/nrow(five)
```

```
## [1] 289.4184
```

Many reviews have positive ratings. The reviewes are mixed length, some very short, some longer. We would expect a very negative review to be long, but also a very positive one could be if the tenant was extremely satisfied.

Corpus processing

```
corpus = Corpus(VectorSource(reviews$comments))
corpus = tm_map(corpus, tolower)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords("english"))
corpus = tm_map(corpus, removeWords, c("airbnb", "next", "for", "while"))
corpus = tm_map(corpus, stemDocument)
```

Frequencies

```
frequencies = DocumentTermMatrix(corpus)
sparse = removeSparseTerms(frequencies, 0.99)
```

Most reviews are positive

```
reviews$positive = as.factor(reviews$review_scores_rating > 3)
table(reviews$positive)
```

```
##
## FALSE  TRUE
##   274  3899
```

Document term matrix

```
document_terms = as.data.frame(as.matrix(sparse))
ncol(document_terms)
```

```
## [1] 404
```

```
document_terms$positive = reviews$positive
document_terms$review_length = nchar(reviews$comments)
```

How many words in the corpus: 404

Training and test set.

```
split1 = (reviews$date < "2017-12-31")
split2 = (reviews$date  >= "2018-01-01")
train = document_terms[split1,]
test = document_terms[split2,]
table(train$positive)
```

```
##
## FALSE  TRUE
##   228  2918
```

```
table(test$positive)
```

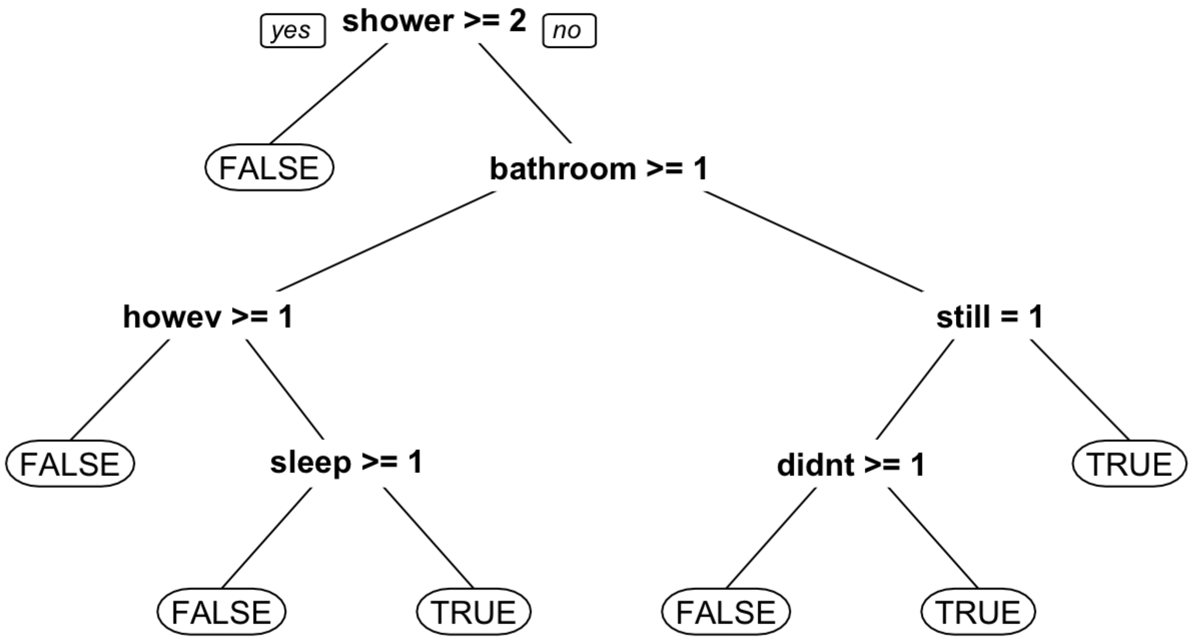
```
##
## FALSE  TRUE
##    41   944
```

The proportion of positive/negative from the original is preserved in the train and test. The prediction problem will be quite difficult since the overwhelming majority is positive reviews, so we will have a high accuracy by always predicting positive. This is not very helpful, we would like to accurately detect the negative ones.

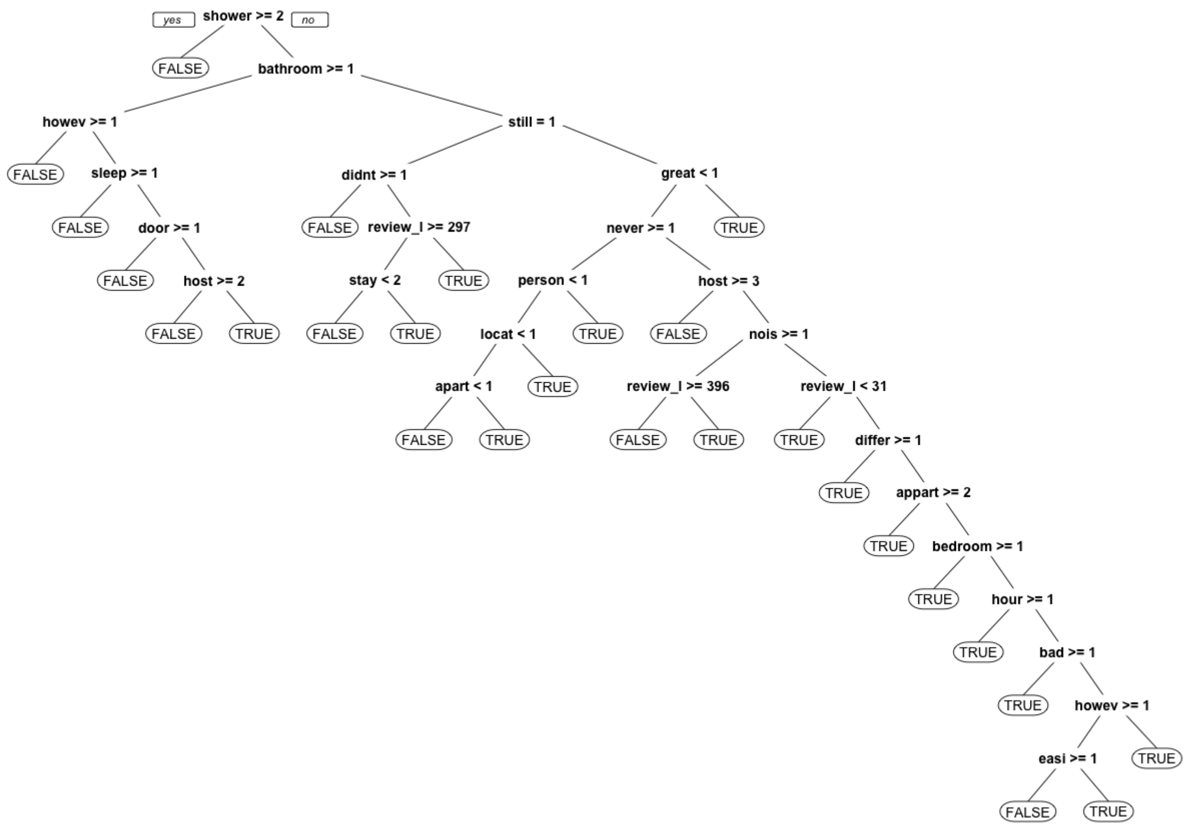
The data processing cut the majority (99%) of the words in our set. The remaining words are usually two types: negative/positive classifiers (good, issue, kind) or booking-specific things (neighborhood, room, respond). This is very interesting because 1) **the prediction problem becomes difficult since "room" may be great in one property and bad in another** and 2) if we were to look at a specific property along our prediction we could tell why that property receives positive and negative reviews based on the words that are predictors.

Preicting using Decision Trees

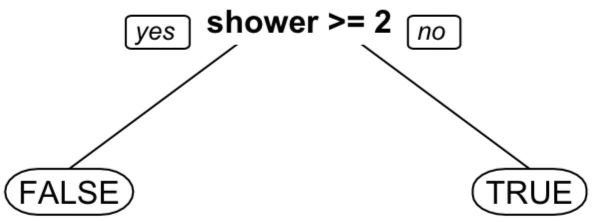
```
tree1 = rpart(positive ~., data=train,cp=0.01)
prp(tree1)
```



```
tree2 = rpart(positive ~., data=train,cp=0.0005)
prp(tree2)
```



```
tree3 = rpart(positive ~., data=train,cp=0.02)
prp(tree3)
```



The three trees are similar at the beginning but have different complexities. We note the most important splits are shower (if shower is mentioned more than 2 times the review is negative which makes sense as people are more linkely to be dissatisfied rather than satisfied with basic amenities like the shower). Other key predictors are bathroom, sleep and other such property-related words.

Random Forest

```
rf = randomForest(positive ~ ., data = train)
```

```
importance.rf <- data.frame(imp=importance(rf))
importance.rf$position <- seq(from = 1, to = 405, by = 1)
importance.rf.ordered <- importance.rf[order(-importance.rf$MeanDecreaseGini), ,drop
 = FALSE]
head(importance.rf.ordered,7)
```

```
##               MeanDecreaseGini position
## review_length        18.419335      405
## shower                6.909219      161
## great                 5.868980       29
## bathroom              5.386000      154
## host                  5.208257       12
## clean                 4.828461       43
## room                  4.550523       18
```

Review length is the most important feature. Ideally we should fit another model to see if longer reviews are more positive or more negative because it is not intuitive and we could argue both ways. However we know from the inital analysis that negative reviews are longer so we interpret this accordingly.

Shower, clean and host are the stronger word predictors, which intuitively makes sense, customers want a communicative host and that is one of the aspects that determine the quality of their stay.

As expected we see sentiment classifiers (didn't, great) as strong predictors of sentiment. We also see booking-specific words like shower. We infer that if someone mentions the shower they did not have a good experience (people are less likely to say "the shower was great" vs "the shower didn't work").

Baseline Model

A baseline model always predicts a positive review since this is the most common.

```
table(test$positive)
```

```
##
## FALSE  TRUE
##    41   944
```

```
accuracy_baseline_train = 944/nrow(test)
accuracy_baseline_train
```

```
## [1] 0.9583756
```

True positive rate

```
tpr_baseline_test = 1
fpr_baseline = 1
```

```
predict_tree1 = predict(tree1, newdata = test,  type="class")
predict_tree2 = predict(tree2, newdata = test, type="class")
predict_tree3 = predict(tree3, newdata = test, type="class")
predict_rf = predict(rf, newdata = test,type="class")

table(predict_tree1, test$positive)
```

```
##
## predict_tree1 FALSE TRUE
##         FALSE     5    5
##         TRUE     36  939
```

```
table(predict_tree2, test$positive)
```

```
##
## predict_tree2 FALSE TRUE
##         FALSE     7   18
##         TRUE     34  926
```

```
table(predict_tree3, test$positive)
```

```
##
## predict_tree3 FALSE TRUE
##         FALSE     1    0
##         TRUE     40  944
```

```
table(predict_rf, test$positive)
```

```
##
## predict_rf FALSE TRUE
##      FALSE     1    0
##      TRUE     40  944
```

Accuracy:

```
accuracy_tree1 = (939+5)/nrow(test)
accuracy_tree2 = (926+7)/nrow(test)
accuracy_tree3 = (944+1)/nrow(test)
accuracy_rf = (943+1)/nrow(test)

print(accuracy_tree1)
```

```
## [1] 0.9583756
```

```
print(accuracy_tree2)
```

```
## [1] 0.9472081
```

```
print(accuracy_tree3)
```

```
## [1] 0.9593909
```

```
print(accuracy_rf)
```

```
## [1] 0.9583756
```

True positive rate:

```
tpr_tree1 = 939/(944)
tpr_tree2 = 926/(944)
tpr_tree3 = 944/(944)
tpr_rf = 943/(944)

print(tpr_tree1)
```

```
## [1] 0.9947034
```

```
print(tpr_tree2)
```

```
## [1] 0.9809322
```

```
print(tpr_tree3)
```

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

```
print(tpr_rf)
```

```
## [1] 0.9989407
```

False positive rate:

```
fpr_tree1 = 36/41
fpr_tree2 = 34/41
fpr_tree3 = 40/41
fpr_rf = 40/41

print(fpr_tree1)
```

```
## [1] 0.8780488
```

```
print(fpr_tree2)
```

```
## [1] 0.8292683
```

```
print(fpr_tree3)
```

```
## [1] 0.9756098
```

```
print(fpr_rf)
```

```
## [1] 0.9756098
```

As expected, the baseline performs very well, one model performs worse than the baseline, and the other models perform similar or slightly above the baseline. We find that a CART that is very complex (tree2) performes the best because it predicts the most true negatives.

Despite high accuracy, these are not very encouraging results, simply because the proportion of positives is so high. We may consider sub-sampling to exclude some of the positive reviews and have a more balanced dataset, or include a loss matrix and force the model to predict negative reviews.

As we mentioned in the beginning, most words are not good predictors because the booking-specific words (neighborhood, room, respond) can be positive or negative depending on which review we look at. What we are doing is building a sentiment classifier and relying on our model to learn words like “good”, “issue”, etc.

As an extention we should consider ** using a sentiment dictionary which already has all these positive and negative words and look for those in our document matrix ** exclude 3 star reviews. They are not really positive, and they are not really negative. They are also not neutral (so including a 3rd class would not help).

This is because neutral language typically expresses no feelings, but in reviews 3-stars mean both positive and negative aspects, which may confuse the model. We might get better accuracy if we exclude them.

We should also think about the business scenario for why we are predicting positive/negative. On Airbnb reviews always come with raiting, which means we will never get just the review text and no rating. Perhaps we can use the model to predict whether a comment is positive/negative on some other hotel review website. My suggestion is to use the model to look at an individual Airbnb and trying to predict positive and negative aspect of that particular house using the model. Airbnb could then make suggestions to hosts on what to improve.