Assignment 2: Classification for the Detection of Opinion Spam

Anouk van der Lee (6620590), Shu Zhao (6833519), Fleur Petit (5583837)

12, October, 2020

Contents

TO DO

- How to best handle sparsity in dtm?
- Normalise data before training
  - Check if necessary: https://stats.stackexchange.com/questions/29781/when-conducting-multiple-regression-when-should-you-center-your-predictor-varia
- Cross-validate/choose random forest parameters
  - ntree
  - mtry
- Visualise more of the results
- Write more explanations, figure captions, table captions, etcetera
  - figure captions are done in the chunk-header
  - kable captions are done in the 'kable()' function.
  - Latex code can be used for references and labels
    * for the tables: https://stackoverflow.com/questions/54082814/adding-label-in-kable-kableextra-latex-output
    * for the figures: https://cran.r-project.org/web/packages/officedown/vignettes/captions.html

0. Pre-processing

Load reviews

```
df <- NULL

file_names <-
  list.files(path = "negative_polarity", recursive = T)

for (name in file_names)
  df <- bind_rows(df,
    list(label = as.numeric(str_detect(name, "truthful")), # 0 for deceptive and 1 for truthful
      fold = as.numeric(str_extract(name, pattern = regex('\\d'))),
      hotel = str_extract(name, pattern = regex('[a-z]*_\\d+')),
      review = read_file(str_c("negative_polarity/", name))
    )
  )

df <- df %>%
  mutate(id = as.numeric(str_extract(hotel, pattern = regex('\\d+'))),
         hotel = str_extract(hotel, pattern = regex('[a-z]*'))
         )


head(df, 2) %>%
  kable()
```

label	fold	hotel	review	id
0	1	hilton	We stayed at the Schicago Hilton for 4 days and 3 nights for a conference. I have to say, normally I am very easy going about amenities, cleanliness, and the like… however our experience at the Hilton was so awful I am taking the time to actually write this review. Truly, DO NOT stay at this hotel. When we arrived in our room, it was clear that the carpet hadn't been vacuumed. I figured, "okay, it's just the carpet." Until I saw the bathroom! Although the bathroom had all the superficial indicators of housekeeping having recently cleaned (i.e., a paper band across the toilet, paper caps on the drinking glasses, etc., it was clear that no ACTUAL cleaning took place. There was a spot (probably urine!) on the toilet seat and, I kid you not, the remnants of a lip-smudge on the glass. I know people who have worked many years in the hotel industry and they always warned that lazy housekeeping will make things "appear" clean but in fact they make no effort to keep things sanitary. Well, the Hilton was proof. I called downstairs and complained, and they sent up a chambermaid hours later. Frankly, I found the room disgusting. The hotel itself, outside the rooms, was cavernous and unwelcoming, with an awful echo in the lobby area that created a migraine-inducing din. Rarely have I been so eager to leave a place as this. When I got home, I washed all my clothes whether I had worn them or not, such was the skeeviness of our accomodations. Please, do yourself a favor and stay at a CLEAN hotel.	1
0	1	hilton	Hotel is located 1/2 mile from the train station which is quite hike when you're traveling with luggage and/or kids. They seem to cash in on guests who arrive in private car by charging exorbitant parking/valet fees. Rooms feature either double or king sized beds; no queen beds at all. If you want a little extra leg room in your bed, the price jump from double- to king-sized is stiff. Rooms with any kind of view pay a healthy surcharge, too.	10

Prepare corpus

```
corpus <- VCorpus(VectorSource(df[["review"]])) %>%
  tm_map(., content_transformer(tolower)) %>% # no capital letters
  tm_map(., stripWhitespace) %>% # remove extra white space
  tm_map(., removeWords, stopwords("english")) %>% # remove stopwords
  tm_map(., stemDocument) # stem words
```

Get word frequency matrix

```
dtm <- DocumentTermMatrix(corpus) %>%
  removeSparseTerms(., sparse = 0.7) # those terms from x are removed which have at least a 70
# percent of empty (i.e., terms occurring 0 times in a document) elements. I.e., the resulting
# matrix contains only terms with a sparse factor of less than 70 percent


inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 800, terms: 10)>>
## Non-/sparse entries: 3754/4246
## Sparsity           : 53%
## Maximal term length: 7
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  chicago get hotel like one room servic staff stay will
##   128       2   3     3    1   2    6      1     2    1    0
##   18        0   3     3    3   1    5      0     2    6    2
```

```
##   247        1  0     9    4   2    3      1     0     0     1
##   311        1  2     5    0   3    4      1     0     6     0
##   390        1  1     4    0   5    6      0     0     4     3
##   412        3  0     2    0   4   11      0     0     2     1
##   514        0  4     4    2   4    4      0     2     1     1
##   549        3  4     4    5   1    6      4     0     3     0
##   610        0  5     8    0   3    9      3     0     4     1
##   7          1  5     4    3   3    5      0     0     1     0
```

Vector with labels

```r
labels <- df[["label"]]

labels[c(1:10, 790:800)]
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
```

Random split in train and test data

```r
train_ind <- sample(1:nrow(df), size = nrow(df)*0.75)

train_dtm <- as.matrix(dtm)[train_ind,]
train_labels <- labels[train_ind]

test_dtm <- as.matrix(dtm)[-train_ind,]
test_labels <- labels[-train_ind]
```

1. Multinomial naive Bayes (generative linear classifier)

Function for multinomial Bayes classifier

```r
# Function for multinomial naive Bayes

train.mnb <- function (dtm, labels) {
  call <- match.call()
  V <- ncol(dtm)
  N <- nrow(dtm)
  prior <- table(labels) / N
  labelnames <- names(prior)
  nclass <- length(prior)
  cond.probs <- matrix(nrow = V, ncol = nclass)
  dimnames(cond.probs)[[1]] <- dimnames(dtm)[[2]]
  dimnames(cond.probs)[[2]] <- labelnames
  index <- list(length = nclass)
  for (j in 1:nclass) {
    index[[j]] <- c(1:N)[labels == labelnames[j]]
  }

  for (i in 1:V) {
    for (j in 1:nclass) {
      cond.probs[i, j] <-
        (sum(dtm[index[[j]], i]) + 1) / (sum(dtm[index[[j]], ]) + V)
    }
```

```
  }
  list(call = call,
       prior = prior,
       cond.probs = cond.probs)
}

predict.mnb <- function (model, dtm) {
    classlabels <- dimnames(model$cond.probs)[[2]]
    logprobs <- dtm %*% log(model$cond.probs)
    N <- nrow(dtm)
    nclass <- ncol(model$cond.probs)
    logprobs <-
      logprobs + matrix(nrow = N,
                        ncol = nclass,
                        log(model$prior),
                        byrow = T)
    classlabels[max.col(logprobs)]
  }
```

Train

```
mnb_model <- train.mnb(train_dtm, train_labels)

mnb_model
```

```
## $call
## train.mnb(dtm = train_dtm, labels = train_labels)
##
## $prior
## labels
##         0         1
## 0.4933333 0.5066667
##
## $cond.probs
##                  0          1
## chicago 0.07937685 0.04006969
## get     0.05786350 0.07491289
## hotel   0.22589021 0.21472125
## like    0.05675074 0.04442509
## one     0.05637982 0.06925087
## room    0.22774481 0.25391986
## servic  0.05675074 0.05836237
## staff   0.04080119 0.04747387
## stay    0.15615727 0.14982578
## will    0.04228487 0.04703833
```

Predict

```
predicted_mnb <- predict.mnb(mnb_model, test_dtm)
```

Confusion matrix

```
conf_mnb <- table(test_labels, predicted_mnb)

conf_mnb
```

```
##            predicted_mnb
## test_labels  0  1
##           0 60 44
##           1 33 63
```

Performance metrics

```
performance <- function(confusion_matrix){
  tn <- confusion_matrix[1,1]
  fn <- confusion_matrix[2,1]
  n_pred <- tn+fn

  fp <- confusion_matrix[1,2]
  tp <- confusion_matrix[2,2]
  p_pred <- fp+tp

  n <- tn+fp
  p <- fn+tp

  performance_metrics <- tibble(metric = c("recall",
                                "miss-rate", # 1 - recall
                                "fall-out", # 1 - selectivity
                                "selectivity",
                                "prevalence",
                                "precision",
                                "false omission rate", # 1 - neg_predictive_value
                                "pos likelihood ratio",
                                "neg likelihood ratio",
                                "accuracy",
                                "false discovery rate", # 1 - precision
                                "neg predictive value",
                                "diagnostic odds ratio",
                                "F1"),
                      value = c(tp/p, # recall
                               fn/p, # 1 - recall
                               fp/n, # 1 - selectivity
                               tn/n, # selectivity
                               p/(n+p), # prevalence
                               tp/p_pred, # precision
                               fn/n_pred, # 1 - neg_predictive_value
                               (tp/p)/(fp/n), # positive likelihood ratio
                               (fn/p)/(tn/n), # negative likelihood ratio
                               (tp+tn)/(n+p), # accuracy
                               fp/p_pred, # 1 - precision
                               tn/n_pred, # negative predictive value
                               ((tp/p)/(fp/n))/((fn/p)/(tn/n)), # diagnostic odds ratio
                               2*(tp/p_pred)*(tp/p)/((tp/p_pred) + (tp/p))) # F1
  )
```

```
  performance_metrics
}

perf_mnb <-performance(conf_mnb)

perf_mnb %>% kable()
```

metric	value
recall	0.6562500
miss-rate	0.3437500
fall-out	0.4230769
selectivity	0.5769231
prevalence	0.4800000
precision	0.5887850
false omission rate	0.3548387
pos likelihood ratio	1.5511364
neg likelihood ratio	0.5958333
accuracy	0.6150000
false discovery rate	0.4112150
neg predictive value	0.6451613
diagnostic odds ratio	2.6033058
F1	0.6206897

2. Regularised logistic regression (discriminative linear classifier)

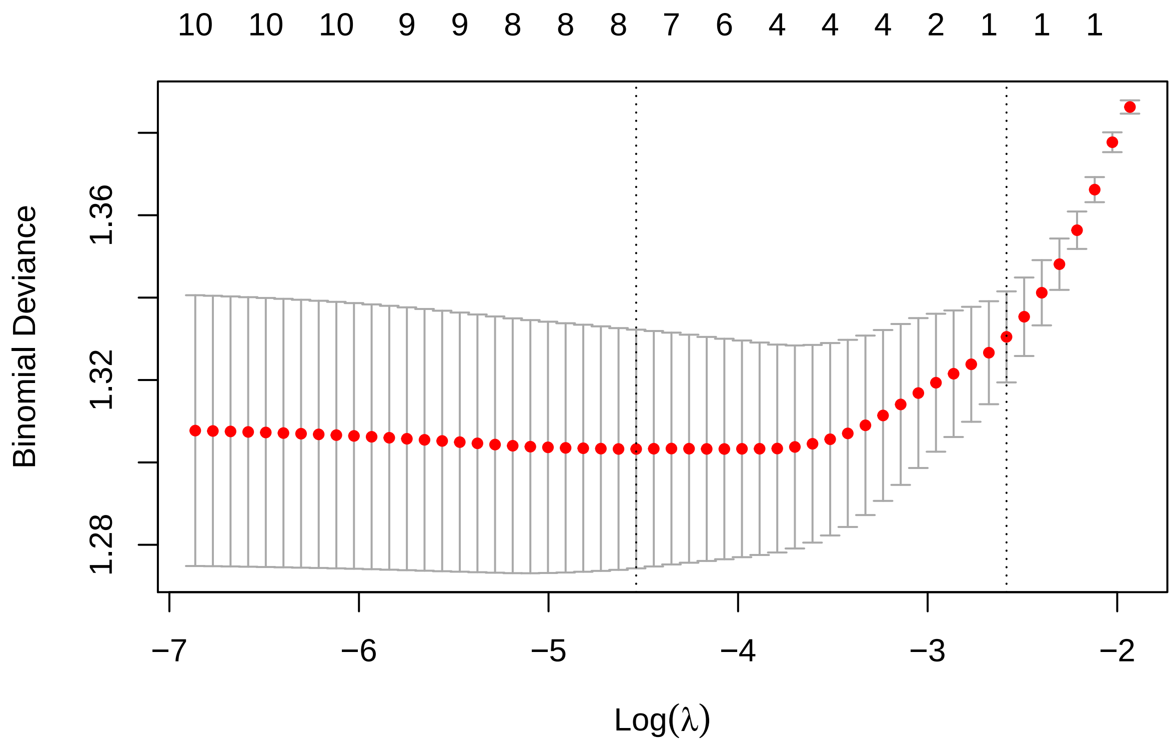
Choose lambda

```
# Use LASSO (penalising for number of parameters)
# Determine lambda by means of cross validation

set.seed(1)

cv_lasso <- cv.glmnet(train_dtm, train_labels, alpha = 1, family = "binomial")

plot(cv_lasso)
```



Train

```
model_glm <- glmnet(train_dtm, train_labels, alpha = 1, family = "binomial",
                    lambda = cv_lasso$lambda.1se) # I choose the largest lambda within 1se from the sma
```

Predict

```
#test_dtm_glm <- model.matrix(test_labels ~ test_dtm)[,-1]

probabilities_glm <- predict(model_glm,
                            newx = test_dtm,
                            type = "response", # Type "response" gives the fitted probabilities for "b
                            s= cv_lasso$lambda.1se)

predicted_glm <- ifelse(probabilities_glm > 0.5, 1, 0)
```

Confusion matrix

```
conf_glm <- table(test_labels, predicted_glm)

conf_glm
```

```
##              predicted_glm
## test_labels  0   1
```

```
##           0 66 38
##           1 21 75
```

Performance metrics

```r
perf_glm <- performance(conf_glm)

perf_glm %>% kable()
```

metric	value
recall	0.7812500
miss-rate	0.2187500
fall-out	0.3653846
selectivity	0.6346154
prevalence	0.4800000
precision	0.6637168
false omission rate	0.2413793
pos likelihood ratio	2.1381579
neg likelihood ratio	0.3446970
accuracy	0.7050000
false discovery rate	0.3362832
neg predictive value	0.7586207
diagnostic odds ratio	6.2030075
F1	0.7177033

3. Classification trees (flexible classifier)

Train

```r
model_tree <- randomForest(x = train_dtm, y = train_labels, ntree = 1, mtry = ncol(train_dtm))
```

```
## Warning in randomForest.default(x = train_dtm, y = train_labels, ntree = 1, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
```

Predict

```r
probabilities_tree <- predict(model_tree, test_dtm, type = "response")

predicted_tree <- ifelse(probabilities_tree > 0.5, 1, 0)
```

Confusion matrix

```r
conf_tree <- table(test_labels, predicted_tree)

conf_tree
```

```
##            predicted_tree
## test_labels  0  1
##           0 71 33
##           1 50 46
```

Performance metrics

```r
perf_tree <- performance(conf_tree)

perf_tree %>% kable()
```

metric	value
recall	0.4791667
miss-rate	0.5208333
fall-out	0.3173077
selectivity	0.6826923
prevalence	0.4800000
precision	0.5822785
false omission rate	0.4132231
pos likelihood ratio	1.5101010
neg likelihood ratio	0.7629108
accuracy	0.5850000
false discovery rate	0.4177215
neg predictive value	0.5867769
diagnostic odds ratio	1.9793939
F1	0.5257143

4. Random forests (flexible classifier)

```r
# TO DO: cross validate for mtry (caret package)

model_random <- randomForest(x = train_dtm, y = train_labels, ntree = 10, mtry = 5)
```

```
## Warning in randomForest.default(x = train_dtm, y = train_labels, ntree = 10, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
```

Predict

```r
probabilities_random <- predict(model_random, test_dtm, type = "response")

predicted_random <- ifelse(probabilities_random > 0.5, 1, 0)
```

Confusion matrix

```r
conf_random <- table(test_labels, predicted_random)

conf_random
```

```
##            predicted_random
## test_labels  0  1
##           0 66 38
##           1 37 59
```

Performance metrics

```
perf_random <- performance(conf_random)

perf_random %>% kable()
```

metric	value
recall	0.6145833
miss-rate	0.3854167
fall-out	0.3653846
selectivity	0.6346154
prevalence	0.4800000
precision	0.6082474
false omission rate	0.3592233
pos likelihood ratio	1.6820175
neg likelihood ratio	0.6073232
accuracy	0.6250000
false discovery rate	0.3917526
neg predictive value	0.6407767
diagnostic odds ratio	2.7695590
F1	0.6113990

Comparison

Performance metrics

```
perf_compare <- tibble(metric = perf_mnb[["metric"]],
                       mnb = perf_mnb[["value"]],
                       glm = perf_glm[["value"]],
                       tree = perf_tree[["value"]],
                       random = perf_random[["value"]])

perf_compare %>% kable()
```

metric	mnb	glm	tree	random
recall	0.6562500	0.7812500	0.4791667	0.6145833
miss-rate	0.3437500	0.2187500	0.5208333	0.3854167
fall-out	0.4230769	0.3653846	0.3173077	0.3653846
selectivity	0.5769231	0.6346154	0.6826923	0.6346154
prevalence	0.4800000	0.4800000	0.4800000	0.4800000
precision	0.5887850	0.6637168	0.5822785	0.6082474
false omission rate	0.3548387	0.2413793	0.4132231	0.3592233
pos likelihood ratio	1.5511364	2.1381579	1.5101010	1.6820175
neg likelihood ratio	0.5958333	0.3446970	0.7629108	0.6073232
accuracy	0.6150000	0.7050000	0.5850000	0.6250000
false discovery rate	0.4112150	0.3362832	0.4177215	0.3917526
neg predictive value	0.6451613	0.7586207	0.5867769	0.6407767
diagnostic odds ratio	2.6033058	6.2030075	1.9793939	2.7695590
F1	0.6206897	0.7177033	0.5257143	0.6113990

Logistic regression

```
predictions_per_model <- tibble(predictions = c(predicted_mnb,
                                      as.vector(predicted_glm),
                                      predicted_tree,
                                      predicted_random),
                      models = c(rep("mnb",
                                     length(predicted_mnb)),
                                 rep("logistic regression",
                                     nrow(predicted_glm)),
                                 rep("single tree",
                                     length(predicted_tree)),
                                 rep("random forest",
                                     length(predicted_random))),
                      ground_truth = c(rep(test_labels, 4)),
                      correct = ifelse(ground_truth == predictions, 1, 0)) %>%
  mutate(models = factor(models,levels=c("mnb", "logistic regression", "single tree", "random forest")))


glm(correct ~ models, family = "binomial", data = predictions_per_model) %>%
  tidy() %>%
  kable()
```

term	estimate	std.error	statistic	p.value
(Intercept)	0.4683789	0.1453172	3.2231482	0.0012679
modelslogistic regression	0.4028435	0.2125051	1.8956887	0.0580012
modelssingle tree	-0.1250456	0.2042359	-0.6122607	0.5403653
modelsrandom forest	0.0424467	0.2060350	0.2060169	0.8367777