Assignment 2: Classification for the Detection of Opinion Spam

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

21, 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
- Add analyses of a bi-grams dtm

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, "deceptive")), # 1 for deceptive and 0 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
1	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
1	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

Unigrams

```
length(corpus)
```

```
## [1] 800
```

```
dtm <- DocumentTermMatrix(corpus)  %>%
  removeSparseTerms(., sparse = 0.95) # 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

dim(dtm)
```

```
## [1] 800 260
```

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 800, terms: 260)>>
## Non-/sparse entries: 25228/182772
## Sparsity           : 88%
```

```
## Maximal term length: 11
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  call check chicago get hotel like one room servic stay
##   143    2     2       0   1     4    0   0   10      2    0
##   18     1     0       0   3     3    3   1    5      0    6
##   21     0     2       1   1     5    1   2    6      1    1
##   498    0     1       0   1     3    2   2    3      2    5
##   544    2     2       1   2     4    0   3    1      0    0
##   549    2     1       3   4     4    5   1    6      4    3
##   610    1     1       0   5     8    0   3    9      3    4
##   650    2     2       2   1     2    0   3    8      1    0
##   651    9     0       0   3     5    0   2    3      0    0
##   7      3     1       1   5     4    3   3    5      0    1
```

Bigrams

```r
BigramTokenizer <-
  function(x)
    unlist(lapply(ngrams(words(x), 2), paste, collapse = " "), use.names = FALSE)

dtm2 <- DocumentTermMatrix(corpus, control = list(tokenize = BigramTokenizer))  %>%
  removeSparseTerms(., sparse = 0.99)

dim(dtm2)
```

```
## [1] 800 311
```

```r
inspect(dtm2)
```

```
## <<DocumentTermMatrix (documents: 800, terms: 311)>>
## Non-/sparse entries: 5057/243743
## Sparsity           : 98%
## Maximal term length: 22
## Weighting          : term frequency (tf)
## Sample             :
##      Terms
## Docs  call front check . front desk hotel chicago look like recent stay
##   134          0       0          1             2         0           0
##   137          2       0          3             1         0           1
##   143          0       0          1             0         0           0
##   166          0       1          1             0         0           0
##   198          0       0          1             0         0           1
##   330          1       0          2             0         0           0
##   349          0       0          0             1         2           0
##   354          0       0          0             1         1           0
##   650          1       1          1             0         0           0
##   7            0       0          0             0         0           0
##      Terms
## Docs  room servic stay . stay hotel will never
##   134           0      0          0          0
##   137           0      1          0          0
##   143           1      0          0          0
##   166           0      0          0          0
```

```
##   198          0      1          0          0
##   330          0      0          0          1
##   349          0      1          0          0
##   354          0      1          0          0
##   650          0      0          0          0
##   7            0      0          0          0
```

Vector with labels

```
labels <- as.factor(df[["label"]])

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

```
##  [1] 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

Random split in train and test data

Unigrams

```
set.seed(123)

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]
```

Bigrams

```
train_dtm2 <- as.matrix(dtm2)[train_ind,]
train_labels <- labels[train_ind]

test_dtm2 <- as.matrix(dtm2)[-train_ind,]
test_labels <- labels[-train_ind]
```

1. Multinomial naive Bayes (generative linear classifier)

Function for multinomial Bayes classifier

```
# 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$cond.prob %>% head(10) %>% kable()
```

	0	1
abl	0.0016576	0.0020600
actual	0.0024034	0.0022889
air	0.0016576	0.0023651
almost	0.0019062	0.0019074
alreadi	0.0013260	0.0016022
also	0.0057185	0.0045014
also,	0.0027350	0.0011444
although	0.0015747	0.0026703
anoth	0.0042268	0.0054170
anyth	0.0011603	0.0022889

Predict

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

Confusion matrix

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

conf_mnb
```

```
##              predicted_mnb
## test_labels  0  1
##            0 86 11
##            1 26 77
```

Performance metrics

```r
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	0.7475728
miss-rate	0.2524272
fall-out	0.1134021
selectivity	0.8865979
prevalence	0.5150000
precision	0.8750000
false omission rate	0.2321429
pos likelihood ratio	6.5922330
neg likelihood ratio	0.2847144
accuracy	0.8150000
false discovery rate	0.1250000
neg predictive value	0.7678571
diagnostic odds ratio	23.1538462
F1	0.8062827

2. Regularised logistic regression (discriminative linear classifier)

Unigram

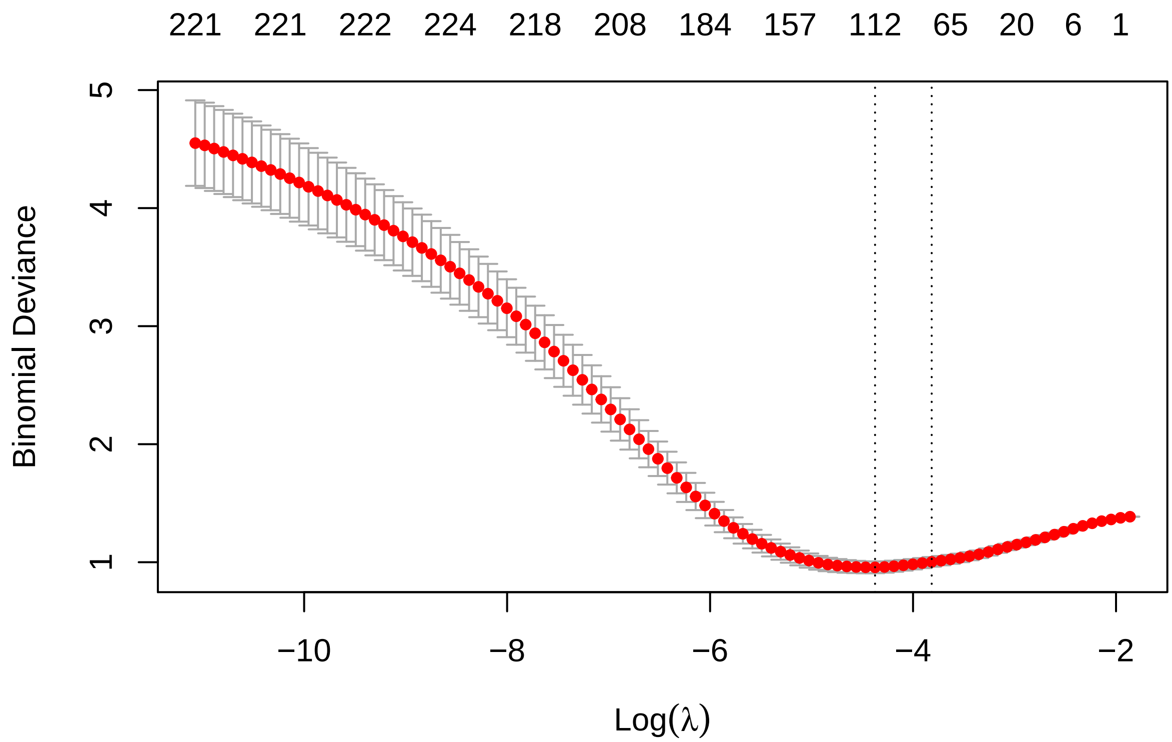
Choose lambda

```
# 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)
```




```r
coefs <- as.data.frame(as.matrix(coef(cv_lasso, s="lambda.1se"))) %>%
  rownames_to_column(var = "term") %>%
  rename(coefficient = `1`) %>%
  filter(coefficient != 0)

coefs %>% head(10) %>% kable()
```

term	coefficient
(Intercept)	-0.5531754
area	-0.3169211
arriv	0.1395486
away	-0.0069208
bathroom	-0.2405532
bed.	-0.1049680
better	-0.0706174
breakfast	-0.1276098
call	-0.2749966
charg	-0.1871259

Train

```r
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 89  8
##            1 29 74
```

Performance metrics

```
perf_glm <- performance(conf_glm)

perf_glm %>% kable()
```

metric	value
recall	0.7184466
miss-rate	0.2815534
fall-out	0.0824742
selectivity	0.9175258
prevalence	0.5150000
precision	0.9024390
false omission rate	0.2457627
pos likelihood ratio	8.7111650
neg likelihood ratio	0.3068616
accuracy	0.8150000
false discovery rate	0.0975610
neg predictive value	0.7542373
diagnostic odds ratio	28.3879310
F1	0.8000000

Bigram

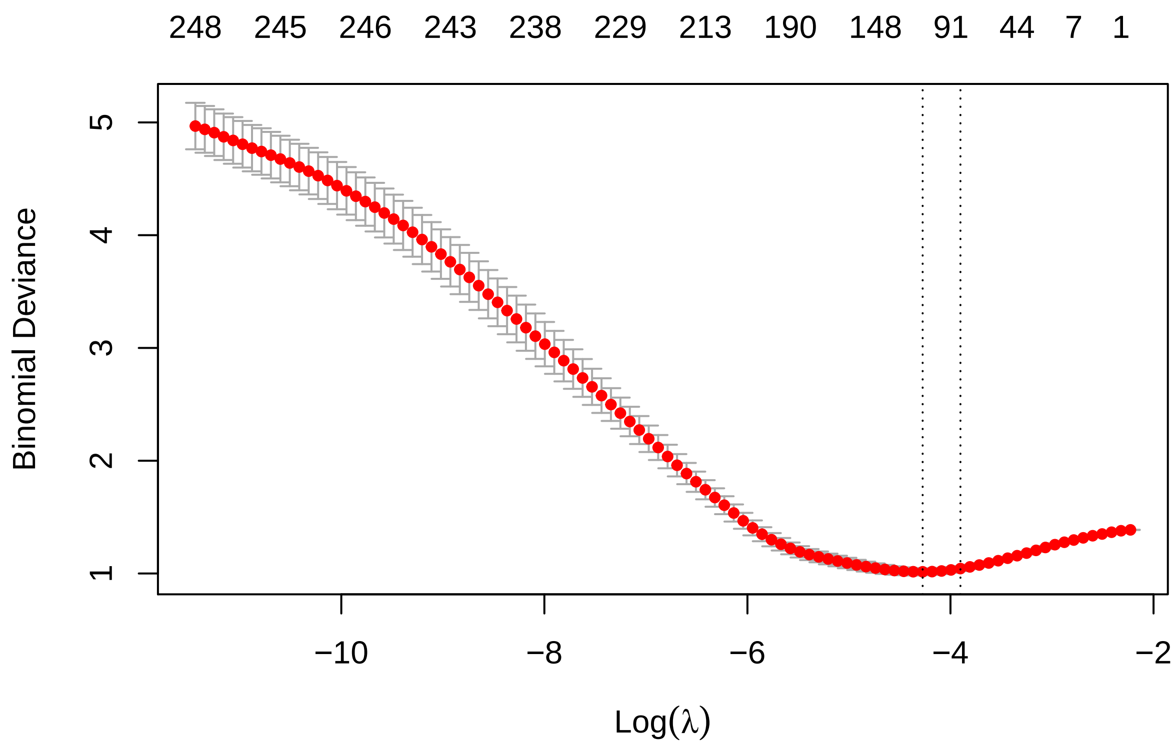
Choose lambda

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

set.seed(1)
```

```r
cv_lasso <- cv.glmnet(train_dtm2, train_labels, alpha = 1, family = "binomial")

plot(cv_lasso)
```




```r
coefs <- as.data.frame(as.matrix(coef(cv_lasso, s="lambda.1se"))) %>%
  rownames_to_column(var = "term") %>%
  rename(coefficient = `1`) %>%
  filter(coefficient != 0)

coefs %>% head(10) %>% kable()
```

term	coefficient
(Intercept)	-0.7714320
, hotel	0.6887584
, stay	-0.5590063
. got	0.3139568
4 star	-0.2290675
5 star	-1.1802660
air condition	0.0193255
amalfi hotel	0.5387076
ambassador east	0.1255046
arriv room	0.3661980

Train

```
model_glm <- glmnet(train_dtm2, 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_dtm2,
                             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 82 15
##           1 46 57
```

Performance metrics

```
perf_glm2 <- performance(conf_glm)

perf_glm2 %>% kable()
```

metric	value
recall	0.5533981
miss-rate	0.4466019
fall-out	0.1546392
selectivity	0.8453608
prevalence	0.5150000
precision	0.7916667
false omission rate	0.3593750
pos likelihood ratio	3.5786408
neg likelihood ratio	0.5282974
accuracy	0.6950000
false discovery rate	0.2083333
neg predictive value	0.6406250
diagnostic odds ratio	6.7739130
F1	0.6514286

3. Classification trees (flexible classifier)

Train

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

Predict

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

Confusion matrix

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

conf_tree
```

```
##             predicted_tree
## test_labels  0  1
##           0 60 37
##           1 43 60
```

Performance metrics

```
perf_tree <- performance(conf_tree)

perf_tree %>% kable()
```

metric	value
recall	0.5825243
miss-rate	0.4174757
fall-out	0.3814433
selectivity	0.6185567
prevalence	0.5150000
precision	0.6185567
false omission rate	0.4174757
pos likelihood ratio	1.5271582
neg likelihood ratio	0.6749191
accuracy	0.6000000
false discovery rate	0.3814433
neg predictive value	0.5825243
diagnostic odds ratio	2.2627278
F1	0.6000000

4. Random forests (flexible classifier)

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

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

Predict

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

Confusion matrix

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

conf_random
```

```
##            predicted_random
## test_labels  0  1
##           0 70 27
##           1 49 54
```

Performance metrics

```
perf_random <- performance(conf_random)

perf_random %>% kable()
```

metric	value
recall	0.5242718
miss-rate	0.4757282
fall-out	0.2783505
selectivity	0.7216495
prevalence	0.5150000
precision	0.6666667
false omission rate	0.4117647
pos likelihood ratio	1.8834951
neg likelihood ratio	0.6592233
accuracy	0.6200000
false discovery rate	0.3333333
neg predictive value	0.5882353
diagnostic odds ratio	2.8571429
F1	0.5869565

Comparison

Performance metrics

```
perf_compare <- tibble(metric = perf_mnb[["metric"]],
                       mnb = perf_mnb[["value"]],
                       glm = perf_glm[["value"]],
                       glm2 = perf_glm2[["value"]],
                       tree = perf_tree[["value"]],
                       random = perf_random[["value"]])

perf_compare %>% kable()
```

metric	mnb	glm	glm2	tree	random
recall	0.7475728	0.7184466	0.5533981	0.5825243	0.5242718
miss-rate	0.2524272	0.2815534	0.4466019	0.4174757	0.4757282
fall-out	0.1134021	0.0824742	0.1546392	0.3814433	0.2783505
selectivity	0.8865979	0.9175258	0.8453608	0.6185567	0.7216495
prevalence	0.5150000	0.5150000	0.5150000	0.5150000	0.5150000
precision	0.8750000	0.9024390	0.7916667	0.6185567	0.6666667
false omission rate	0.2321429	0.2457627	0.3593750	0.4174757	0.4117647
pos likelihood ratio	6.5922330	8.7111650	3.5786408	1.5271582	1.8834951
neg likelihood ratio	0.2847144	0.3068616	0.5282974	0.6749191	0.6592233
accuracy	0.8150000	0.8150000	0.6950000	0.6000000	0.6200000
false discovery rate	0.1250000	0.0975610	0.2083333	0.3814433	0.3333333
neg predictive value	0.7678571	0.7542373	0.6406250	0.5825243	0.5882353
diagnostic odds ratio	23.1538462	28.3879310	6.7739130	2.2627278	2.8571429
F1	0.8062827	0.8000000	0.6514286	0.6000000	0.5869565

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)	-2.8438517	0.3101544	-9.169149	0.000000
modelslogistic regression	0.3315461	0.4102045	0.808246	0.418949
modelssingle tree	3.2493168	0.3420951	9.498285	0.000000
modelsrandom forest	3.3334000	0.3426633	9.727916	0.000000