preliminary taks

2024-11-14

Functions and Libraries

```
# Loading the libraries
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
require(tidytext)
```

```
## Loading required package: tidytext
```

```
require(textstem)
```

```
## Loading required package: textstem
## Loading required package: koRpus.lang.en
## Loading required package: koRpus
## Loading required package: sylly
## For information on available language packages for 'koRpus', run
##
##   available.koRpus.lang()
##
## and see ?install.koRpus.lang()
##
##
## Attaching package: 'koRpus'
##
## The following object is masked from 'package:readr':
##
##     tokenize
```

```r
require(rvest)
```

```
## Loading required package: rvest
##
## Attaching package: 'rvest'
##
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```r
require(qdapRegex)
```

```
## Loading required package: qdapRegex
##
## Attaching package: 'qdapRegex'
##
## The following object is masked from 'package:dplyr':
##
##     explain
##
## The following object is masked from 'package:ggplot2':
##
##     %+%
```

```r
require(stopwords)
```

```
## Loading required package: stopwords
```

```r
require(tokenizers)
```

```
## Loading required package: tokenizers
```

```r
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.2.0 --
## v broom        1.0.7     v rsample      1.2.1
## v dials        1.3.0     v tune         1.2.1
## v infer        1.0.7     v workflows    1.1.4
## v modeldata    1.4.0     v workflowsets 1.1.0
## v parsnip      1.2.1     v yardstick    1.3.1
## v recipes      1.1.0
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x qdapRegex::%+%()     masks ggplot2::%+%()
## x scales::discard()    masks purrr::discard()
## x qdapRegex::explain() masks dplyr::explain()
## x dplyr::filter()      masks stats::filter()
## x recipes::fixed()     masks stringr::fixed()
## x dplyr::lag()         masks stats::lag()
## x yardstick::spec()    masks readr::spec()
## x recipes::step()      masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(modelr)
```

```
##
## Attaching package: 'modelr'
##
## The following objects are masked from 'package:yardstick':
##
##     mae, mape, rmse
##
## The following object is masked from 'package:broom':
##
##     bootstrap
```

```
library(Matrix)
```

```
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
library(sparsesvd)
library(glmnet)
```

```
## Loaded glmnet 4.1-8
```

```
# Loading the Data
source('preprocessing.R')
load("../data/claims-raw.RData")
load("../data/claims-clean-example.RData")

# PCA project function
# Function
projection_fn <- function(.dtm, .prop){
  # Coerce feature matrix to sparse
  dtm_mx <- .dtm %>%
    as.matrix() %>%
    as('sparseMatrix')
  # Compute svd
  svd_out <- sparsesvd(dtm_mx)
  # Select number of projections
  var_df <- tibble(var = svd_out$d^2) %>%
    mutate(pc = row_number(),
           cumulative = cumsum(var)/sum(var))
  n_pc <- which.min(var_df$cumulative < .prop)
  # Extract loadings
  loadings <- svd_out$v[, 1:n_pc] %>% as.matrix()
  # Extract scores
  scores <- (dtm_mx %*% svd_out$v[, 1:n_pc]) %>% as.matrix()
  # Adjust names
```

```
  colnames(loadings) <- colnames(scores) <- paste('pc', 1:n_pc, sep = '')
  # Output
    out <- list(n_pc = n_pc,
                var = var_df,
                projection = loadings,
                data = as_tibble(scores))
  return(out)
}

# Reproject test data
reproject_fn <- function(.dtm, .projection_fn_out){
  as_tibble(as.matrix(.dtm) %*% .projection_fn_out$projection)
}
```

Task 1

Data with Headers

```
# Add headers
#claims_clean_headers <- claims_raw %>%
  #parse_data()
# Save the data into an RData file
#save(claims_clean_headers, file = '../data/claims-clean-headers.RData')
```

Change Data With Headers Into a TF-IDF

```
# Load the data
load('../data/claims-clean-headers.RData')
headers_clean <- claims_clean_headers %>%
  select(-c(1:5), -7)

# Convert to a TF-IDF
headers_tfidf <- headers_clean %>%
  unnest_tokens(output = token,
                input = text_clean,
                token = 'words',
                stopwords = str_remove_all(stop_words$word,
                                           '[[:punct:]]')) %>%
  mutate(token.lem = lemmatize_words(token)) %>%
  filter(str_length(token.lem) > 2) %>%
  count(.id, bclass, mclass, token.lem, name = 'n') %>%
  bind_tf_idf(term = token.lem,
              document = .id,
              n = n) %>%
  pivot_wider(id_cols = c('.id', 'bclass', 'mclass'),
              names_from = 'token.lem',
              values_from = 'tf_idf',
              values_fill = 0)
```

Partition the Data

```
# Partition data
set.seed(102722)
partitions <- headers_tfidf %>% initial_split(prop = 0.8)

# Separate DTM from labels
test_dtm <- testing(partitions) %>%
  select(-.id, -bclass, -mclass)
test_labels <- testing(partitions) %>%
  select(.id, bclass, mclass)

# Same, training set
train_dtm <- training(partitions) %>%
  select(-.id, -bclass, -mclass)
train_labels <- training(partitions) %>%
  select(.id, bclass, mclass)
```

PCA with Header Data

```
# Set seed for reproducibility
set.seed(102722)

# Find projections based on training data
proj_out <- projection_fn(.dtm = train_dtm, .prop = 0.7)
train_dtm_projected <- proj_out$data

# How many components were used?
proj_out$n_pc
```

```
## [1] 211
```

Fit Header Data into Logistic Regression

```
# Set seed for reproducibility
set.seed(102722)

# Regression on training data
train <- train_labels %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(train_dtm_projected)

# Fit the model
fit <- glm(bclass~., data = train, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Project onto Test Data and Calculate Metrics

```
# Set seed for reproducibility
set.seed(102722)

# Project test data
test_dtm_projected <- reproject_fn(.dtm = test_dtm, proj_out)

# Get predictions
preds <- predict(fit,
                 newdata = as.data.frame(test_dtm_projected),
                 type = 'response')

# Test-labels with predictions
pred_df <- test_labels %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(pred = as.numeric(preds)) %>%
  mutate(bclass.pred = factor(pred > 0.5,
                              labels = levels(bclass)))

# Evaluate errors on test set
class_metrics <- metric_set(sensitivity,
                            specificity,
                            accuracy,
                            roc_auc)

# Calculate metrics
pred_df %>% class_metrics(truth = bclass,
                  estimate = bclass.pred,
                  pred,
                  event_level = 'second')
```

```
## # A tibble: 4 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 sensitivity binary         0.804
## 2 specificity binary         0.744
## 3 accuracy    binary         0.776
## 4 roc_auc     binary         0.871
```

```
#save(pred_df, file = '../data/claims-clean-headers-metrics.RData')
```

Data without Headers

Change Data Without Headers Into a TF-IDF

```
# Turn into a TF-IDF
no_headers_tfidf <- claims_clean %>%
  unnest_tokens(output = token,
                input = text_clean,
```

```
                token = 'words',
                stopwords = str_remove_all(stop_words$word,
                                           '[[:punct:]]')) %>%
  mutate(token.lem = lemmatize_words(token)) %>%
  filter(str_length(token.lem) > 2) %>%
  count(.id, bclass, mclass, token.lem, name = 'n') %>%
  bind_tf_idf(term = token.lem,
              document = .id,
              n = n) %>%
  pivot_wider(id_cols = c('.id', 'bclass', 'mclass'),
              names_from = 'token.lem',
              values_from = 'tf_idf',
              values_fill = 0)
```

Partition the Data

```
# Partition data
  # Set seed for reproducibility
  set.seed(102722)
partitions1 <- no_headers_tfidf %>% initial_split(prop = 0.8)

# Separate DTM from labels
test_dtm1 <- testing(partitions1) %>%
  select(-.id, -bclass, -mclass)
test_labels1 <- testing(partitions1) %>%
  select(.id, bclass, mclass)

# Same, training set
train_dtm1 <- training(partitions1) %>%
  select(-.id, -bclass, -mclass)
train_labels1 <- training(partitions1) %>%
  select(.id, bclass, mclass)
```

PCA without Headers Data

```
# Find projections based on training data
proj_out1 <- projection_fn(.dtm = train_dtm1, .prop = 0.7)
train_dtm_projected1 <- proj_out1$data

# How many components were used?
proj_out1$n_pc
```

```
## [1] 153
```

Fit Data into Logistic Regression

```
#regression
train1 <- train_labels1 %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(train_dtm_projected1)


fit1 <- glm(bclass~., data = train1, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Project onto Test Data and Calculate Metrics

```
# Project test data
test_dtm_projected1 <- reproject_fn(.dtm = test_dtm1, proj_out1)

# Get predictions
preds1 <- predict(fit1,
                  newdata = as.data.frame(test_dtm_projected1),
                  type = 'response')

# Test-labels with predictions
pred_df1 <- test_labels1 %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(pred = as.numeric(preds1)) %>%
  mutate(bclass.pred = factor(pred > 0.5,
                              labels = levels(bclass)))

# Calculate metrics
pred_df1 %>% class_metrics(truth = bclass,
                   estimate = bclass.pred,
                   pred,
                   event_level = 'second')
```

```
## # A tibble: 4 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 sensitivity binary         0.847
## 2 specificity binary         0.786
## 3 accuracy    binary         0.820
## 4 roc_auc     binary         0.861
```

```
#save(pred_df1, file = '../data/claims-clean-no-headers-metrics.RData')
```

Task 2

Tokenize into Bigrams

```r
# Set seed for reproducibility
set.seed(102722)

# Tokenize into bigrams
headers_bigrams <- claims_clean_headers %>%
  select(.id, bclass, text_clean) %>%
  unnest_tokens(output = bigram,
                input = text_clean,
                token = 'ngrams',
                n = 2,
                stopwords = str_remove_all(stop_words$word, '[[:punct:]]'))
```

Change Bigram Data into a TF-IDF

```r
# Count bigrams and compute TF-IDF
headers_bigrams_tfidf <- headers_bigrams %>%
  count(.id, bclass, bigram, name = 'n') %>%
  bind_tf_idf(term = bigram,
              document = .id,
              n = n) %>%
  filter(n>=5) %>%
  pivot_wider(id_cols = c(.id, bclass),
              names_from = bigram,
              values_from = tf_idf,
              values_fill = 0)
```

Partition the Data

```r
# Partition data
partitions_bigrams <- headers_bigrams_tfidf %>% initial_split(prop = 0.8)

train_dtm_bigrams <- training(partitions_bigrams) %>%
  select(-.id, -bclass)
train_labels_bigrams <- training(partitions_bigrams) %>%
  select(.id, bclass)

test_dtm_bigrams <- testing(partitions_bigrams) %>%
  select(-.id, -bclass)
test_labels_bigrams <- testing(partitions_bigrams) %>%
  select(.id, bclass)
```

First Logistic Regression

```r
# Set seed for reproducibility
set.seed(102722)

# PCA projection for training bigram data
```

```
train_dtm_bigrams_sparse <- train_dtm_bigrams %>%
  as.matrix() %>%
  as('sparseMatrix')
svd_out_bigrams <- sparsesvd(train_dtm_bigrams_sparse, rank=173)

# Training PCs data frame
train_dtm_projected2 <- svd_out_bigrams$u %*% diag(svd_out_bigrams$d)

# Assign column names
colnames(train_dtm_projected2) <- paste0("PC", 1:ncol(train_dtm_projected2))

# Regression with training data
train2 <- train_labels_bigrams %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(train_dtm_projected2)

fit2 <- glm(bclass~., data = train2, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Re-projection of test data
reproject_fn1 <- function(.dtm, train_projected) {
  .dtm_sparse <- as(.dtm, "sparseMatrix")
  test_projected <- as.matrix(.dtm_sparse %*% train_projected$v %*% diag(1 / train_projected$d))
  colnames(test_projected) <- paste0("PC", 1:ncol(test_projected))
  return(test_projected)
}

# Test PCs data frame
test_dtm_projected2 <- reproject_fn1(.dtm = test_dtm_bigrams, svd_out_bigrams)
```

Creating Log-odds

```
# Put predicted probabilities into training dataset
train_bigram_preds <- train2 %>%
  bind_cols(pred = as.numeric(predict(fit2, type = 'response'))) %>%
  mutate(log_odds = log(pred / (1 - pred)), #pred into log-odds
    bclass.pred = factor(pred > 0.5,
                         labels = levels(bclass)))
```

Second Logistic Regression

```
# Regression on bclass with log-odds and 50 PC
train3 <- train_bigram_preds %>%
  select(bclass, log_odds, PC1:PC50)

fit3 <- glm(bclass~., data = train3, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Calculating Testing Data Metrics

```
# Creating preds2
preds2 <- predict(fit2,
                  newdata = as.data.frame(test_dtm_projected2),
                  type = 'response')

# Creating pred_df2
pred_df2 <- test_labels_bigrams %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(pred = as.numeric(preds2)) %>%
  mutate(bclass.pred = factor(pred > 0.5,
                              labels = levels(bclass)))

# Take projected test data and input log-odds
test <- cbind(pred_df2, test_dtm_projected2) %>%
  mutate(log_odds = log(pred / (1 - pred)), #pred into log-odds
    bclass.pred = factor(pred > 0.5,
                              labels = levels(bclass)))%>%
  select(bclass, log_odds, PC1:PC50)

# Add predictions on test data
test_pred <-predict(fit3,
                  newdata = as.data.frame(test),
                  type = 'response')
test_pred_df <- test %>%
  transmute(bclass = factor(bclass)) %>%
  bind_cols(pred = as.numeric(test_pred))%>%
  mutate(bclass.pred = factor(pred > 0.5,
                              labels = levels(bclass)))

# Metrics
test_pred_df %>% class_metrics(truth = bclass,
                  estimate = bclass.pred,
                  pred,
                  event_level = 'second')
```

```
## # A tibble: 4 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 sensitivity binary         0.969
## 2 specificity binary         0.149
## 3 accuracy    binary         0.565
## 4 roc_auc     binary         0.613
```

```
#save(test_pred_df, file = '../data/claims-clean-bigrams-metrics.RData')
```