DSP_word2vec_ds3

Melanie Weissenboeck

2022-11-28

Laden von Bibliotheken und Daten

```
library("xlsx")
library(word2vec)
library(udpipe)
library(SnowballC)
library(ggplot2)
library(tm)
library(wordcloud)
library(tidytext)
library(tidyr)
library(mlbench)
library(e1071)
library(caret)
library(class)
```

Vorverarbeiten der Texte

Bereinigen der Texte

Im ersten Schritt werden die Beschreibungstexte mit der Funktion `txt_clean_word2vec` für die weitere Verarbeitung vorbereitet. Der Rest des Dataframes bleibt unverändert.

```
# Funktion fuer Text Bereinigung
ds3$ANF_BESCHREIBUNG <- txt_clean_word2vec(ds3$ANF_BESCHREIBUNG,
                                           ascii = FALSE, alpha = TRUE,
                                           tolower = TRUE, trim = TRUE)
```

Erstellen einer Worteinbettung

Als Vorbereitung für das spätere Modell wird zunächst eine Einbettung erstellt. Dazu wird in 15 Iterationen für alle Texte gemeinsam eine Darstellung gesucht.

```
# Modell trainieren fuer Einbettung
model_ds3 <- word2vec(ds3$ANF_BESCHREIBUNG, dim = 10, iter = 15)
embedding_ds3 <- as.matrix(model_ds3)
```

```
# Dimension der Einbettung
dim(embedding_ds3)
```

```
## [1] 1391   10
```

Generieren von numerischen Prädiktoren

In diesem Schritt werden die einzelnen Texte zu einem Vektor der Länge 10 transformiert. Dazu wird die Einbettung aus dem vorherigen Abschnitt verwendet.

```
# # aufteilen der Texte in einzelne Token
ds3$token <- tokenizers::tokenize_words(ds3$ANF_BESCHREIBUNG)
```

```
# Vektor der Laenge 10 fuer jedes Dokument
features3 <- matrix(nrow = 0, ncol = 10)
for (i in (1:length(ds3$ANF_BESCHREIBUNG))){
  vec_doc1 <- doc2vec(model_ds3, ds3$token[1][[1]][i], split = " ")
  features3 <- rbind(features3, vec_doc1)
}
```

Zusammenführen mit anderen Prädiktoren

Im Folgenden werden alle Prädiktoren in einem Dataframe zusammengefasst. Dieser stellt die Ausgangslage für die Klassifikation dar.

```
features3 <- as.data.frame(features3)
ds3_all <- cbind(ds3, features3)
ds3_all <- as.data.frame(ds3_all)
```

```
df <- ds3_all[ , c(6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 20)]
df[is.na(df)] <- 0
head(df)
```

```
##   ANF_RISIKO TF_ABDECKUNG AKT_RES_STATUS AKT_RES_RELEASE         V1          V2
## 1     mittel        50.00             OK           22.30  0.1082783 -0.22955474
## 2       hoch        50.00             OK           22.10  0.6893682 -0.01903756
## 3       hoch        11.11             OK           22.10  0.6955095  0.04437103
## 4       hoch        50.00             OK           22.10 -1.4780217 -0.04170451
## 5       hoch        33.33             OK           22.10  0.2241467 -1.59535608
## 6     mittel        25.00             OK           22.10 -0.2627254 -0.48272706
##           V3          V4          V6          V7        V8         V9
## 1 -0.2270607 -0.03225575  0.22715447 -1.05858192 1.9339696  1.2006974
## 2 -0.8091563 -0.47359440 -0.91980196 -0.45551009 1.7035148  0.8212010
## 3  0.8261509  0.36486636  0.12074435  0.06650966 1.9586064 -1.5816054
## 4 -0.6926929  0.13639020  1.90286419 -0.41398117 1.4759568 -0.9611224
## 5  0.4883708  1.15640924 -0.04091396 -1.07871065 0.6136647  0.9541643
## 6 -0.7465809 -0.48740471 -1.05438872 -1.00145753 0.9438817  0.6093275
##          V10
## 1  1.7431670
## 2  1.9259528
## 3  1.2263606
## 4 -0.6217698
## 5 -1.5353033
## 6 -0.7009105
```

Normalisieren numerischer Spalten

Mittels `min-max`-Normalisierung werden die numerischen Spalten auf eine gemeinsame Skalierung gebracht. Zur besseren Übersicht wird am Ende nochmal eine Zusammenfassung ausgegeben.

```
set.seed(1234)
```

```r
# definiere normalisierungsfunktion
min_max_norm <- function(x) {
(x - min(x)) / (max(x) - min(x))
}
# alle spalten normalisieren
df[, 5:13] <- as.data.frame(lapply(df[, 5:13], min_max_norm))
df[2] <- as.data.frame(lapply(df[2], min_max_norm))

df$ANF_RISIKO <- as.factor(df$ANF_RISIKO)
df$AKT_RES_STATUS <- as.factor(df$AKT_RES_STATUS)
df$AKT_RES_RELEASE <- as.factor(df$AKT_RES_RELEASE)
summary(df)
```

```
##   ANF_RISIKO   TF_ABDECKUNG    AKT_RES_STATUS AKT_RES_RELEASE       V1
##  gering:241   Min.   :0.0000   FAILED:  48    21x     :  48   Min.   :0.0000
##  hoch  :540   1st Qu.:0.0667   OK    :1395    22.10   :1081   1st Qu.:0.4970
##  mittel:665   Median :0.1429   OPEN  :   3    22.20   :  12   Median :0.4970
##               Mean   :0.2320                   22.30   : 302   Mean   :0.4979
##               3rd Qu.:0.3333                   OLDERT21:   3   3rd Qu.:0.4970
##               Max.   :1.0000                                   Max.   :1.0000
##        V2               V3               V4               V6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.5396   1st Qu.:0.5257   1st Qu.:0.4922   1st Qu.:0.4413
##  Median :0.5396   Median :0.5257   Median :0.4922   Median :0.4413
##  Mean   :0.5351   Mean   :0.5269   Mean   :0.4915   Mean   :0.4431
##  3rd Qu.:0.5396   3rd Qu.:0.5257   3rd Qu.:0.4922   3rd Qu.:0.4413
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##        V7               V8               V9              V10
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.5849   1st Qu.:0.3337   1st Qu.:0.5658   1st Qu.:0.5632
##  Median :0.5849   Median :0.3337   Median :0.5658   Median :0.5632
##  Mean   :0.5804   Mean   :0.3437   Mean   :0.5662   Mean   :0.5652
##  3rd Qu.:0.5849   3rd Qu.:0.3337   3rd Qu.:0.5658   3rd Qu.:0.5632
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Klassifikation

Erstellen von Train- / Test-Split

Die vorliegenden Daten werden in Trainings- und Testdaten aufgeteilt im Verhältnis 80:20.

```r
# partition erstellen
part <- createDataPartition(df$ANF_RISIKO, times = 1, p = 0.80)
# extract training set
X_train <- df[part$Resample1, ]
# extract testing set
X_test <- df[-part$Resample1, ]
# extract target
y_train <- df[part$Resample1, 1]
y_test <- df[-part$Resample1, 1]
```

NaiveBayes Klassifikation

Ein Naive-Bayes Klassifikator wird erstellt und mit den Trainingsdaten trainiert. Anhand der Testdaten wird das Modell evaluiert. Die Ergebnisse werden in einer Confusionmatrix angegeben.

```
model_nb = naiveBayes(ANF_RISIKO ~ ., data = X_train)
```

```
pred_nb <- predict(model_nb, X_test)
mat.nb <- confusionMatrix(pred_nb, X_test$ANF_RISIKO, mode = "prec_recall")
mat.nb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction gering hoch mittel
##     gering      6    1      2
##     hoch       40  104    118
##     mittel      2    3     13
##
## Overall Statistics
##
##                Accuracy : 0.4256
##                  95% CI : (0.3679, 0.4849)
##     No Information Rate : 0.4602
##     P-Value [Acc > NIR] : 0.8926
##
##                   Kappa : 0.0844
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.66667      0.3969       0.72222
## Recall                     0.12500      0.9630       0.09774
## F1                         0.21053      0.5622       0.17219
## Prevalence                 0.16609      0.3737       0.46021
## Detection Rate             0.02076      0.3599       0.04498
## Detection Prevalence       0.03114      0.9066       0.06228
## Balanced Accuracy          0.55628      0.5450       0.53285
```

KNN Klassifikation

Analog zum Naive-Bayes Klassifikator wird auch ein KNN Modell trainiert. Auch hier wird das Ergebnis anhand einer Confusionmatrix gezeigt.

```
model_knn <- train(ANF_RISIKO ~ ., data = X_train, "knn",
trControl = trainControl(method = "cv", number = 5))
```

```
pred_knn <- predict(model_knn, X_test, type = "raw")
mat.knn <- confusionMatrix(pred_knn, X_test$ANF_RISIKO, mode = "prec_recall")
mat.knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction gering hoch mittel
```

```
##     gering     23   11      9
##     hoch        7   54      7
##     mittel     18   43    117
##
## Overall Statistics
##
##                Accuracy : 0.6713
##                  95% CI : (0.6138, 0.7252)
##     No Information Rate : 0.4602
##     P-Value [Acc > NIR] : 3.844e-13
##
##                   Kappa : 0.4557
##
##  Mcnemar's Test P-Value : 1.514e-06
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.53488      0.7941        0.6573
## Recall                     0.47917      0.5000        0.8797
## F1                         0.50549      0.6136        0.7524
## Prevalence                 0.16609      0.3737        0.4602
## Detection Rate             0.07958      0.1869        0.4048
## Detection Prevalence       0.14879      0.2353        0.6159
## Balanced Accuracy          0.69809      0.7113        0.7443
```