DSP_word2vec_ds1

Melanie Weissenboeck

2022-11-28

Laden von Bibliotheken und Daten

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

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
ds1$ANF_BESCHREIBUNG <- txt_clean_word2vec(ds1$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_ds1 <- word2vec(ds1$ANF_BESCHREIBUNG, dim = 10, iter = 15)
embedding_ds1 <- as.matrix(model_ds1)
```

```
# Dimension der Einbettung
dim(embedding_ds1)
```

```
## [1] 632  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
ds1$token <- tokenizers::tokenize_words(ds1$ANF_BESCHREIBUNG)
```

```
# Vektor der Laenge 10 fuer jedes Dokument
features <- matrix(nrow = 0, ncol = 10)
for (i in (1:length(ds1$ANF_BESCHREIBUNG))){
  vec_doc1 <- doc2vec(model_ds1, ds1$token[1][[1]][i], split = " ")
  features <- rbind(features, vec_doc1)
}
```

Zusammenführen mit anderen Prädiktoren

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

```
features <- as.data.frame(features)
ds1_all <- cbind(ds1, features)
```

```
ds1_all <- as.data.frame(ds1_all)
```

```
df <- ds1_all[ , c(4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 18)]
df[is.na(df)] <- 0
head(df)
```

```
##   ANF_RISIKO TF_ABDECKUNG AKT_RES_STATUS AKT_RES_RELEASE          V1         V2
## 1     mittel         16.6             OK             21x -0.12927520 -1.4531809
## 2     mittel         16.7             OK             21x  0.06060153 -1.0110397
## 3     gering         50.0             OK             21x -0.95149344 -1.3297342
## 4     gering        100.0             OK             21x  0.28836251  0.2428824
## 5     gering        100.0             OK           22.10 -0.03376552 -1.1876963
## 6       hoch         20.0             OK             21x -0.95149344 -1.3297342
##           V3         V4        V6         V7          V8         V9        V10
## 1 -2.5284983 -0.3233660 0.5117046  0.4845300 -0.06991537 -0.1807021 -0.8534227
## 2 -2.0735570 -0.7024613 1.2250285 -0.2208301 -0.31186931  1.3915337 -0.5682574
## 3 -1.7319206 -1.1400864 1.0051885 -1.0180418 -0.84655725  0.1292439 -0.1662838
## 4 -0.5439908 -0.5579829 0.6758794 -0.8196268 -1.54321770  2.0214255 -1.0743244
## 5 -1.6718457 -0.2575216 1.3125226  0.1760638 -0.99687140 -1.4618888  0.1239714
## 6 -1.7319206 -1.1400864 1.0051885 -1.0180418 -0.84655725  0.1292439 -0.1662838
```

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)

# 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:158   Min.   :0.0000   FAILED: 12     21x     : 30    Min.   :0.0000
##  hoch  : 84   1st Qu.:0.5000   OK    :359     22.10   :132    1st Qu.:0.5178
##  mittel:136   Median :1.0000   OPEN  :  7     22.20   :129    Median :0.5178
##               Mean   :0.8056                  22.30   :  3    Mean   :0.5251
##               3rd Qu.:1.0000                  OLDERT21: 84    3rd Qu.:0.5178
##               Max.   :1.0000                                  Max.   :1.0000
##        V2               V3               V4              V6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.5765   1st Qu.:0.8893   1st Qu.:0.495   1st Qu.:0.1802
##  Median :0.5765   Median :0.8893   Median :0.495   Median :0.1802
##  Mean   :0.5742   Mean   :0.8649   Mean   :0.484   Mean   :0.2078
##  3rd Qu.:0.5765   3rd Qu.:0.8893   3rd Qu.:0.495   3rd Qu.:0.1802
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
##        V7               V8               V9              V10
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.7216   1st Qu.:0.8074   1st Qu.:0.3620   1st Qu.:0.9331
##  Median :0.7216   Median :0.8074   Median :0.3620   Median :0.9331
##  Mean   :0.6995   Mean   :0.7934   Mean   :0.3806   Mean   :0.9011
##  3rd Qu.:0.7216   3rd Qu.:0.8074   3rd Qu.:0.3620   3rd Qu.:0.9331
##  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.

```
# 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     26    6     16
##     hoch        1    8      6
##     mittel      4    2      5
##
## Overall Statistics
##
##                Accuracy : 0.527
##                  95% CI : (0.4075, 0.6443)
##     No Information Rate : 0.4189
##     P-Value [Acc > NIR] : 0.039373
##
##                   Kappa : 0.2495
##
##  Mcnemar's Test P-Value : 0.005158
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                   0.5417      0.5333       0.45455
## Recall                      0.8387      0.5000       0.18519
## F1                          0.6582      0.5161       0.26316
## Prevalence                  0.4189      0.2162       0.36486
## Detection Rate              0.3514      0.1081       0.06757
## Detection Prevalence        0.6486      0.2027       0.14865
## Balanced Accuracy           0.6635      0.6897       0.52876
```

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     27    7     16
##     hoch        0    6      2
##     mittel      4    3      9
##
## Overall Statistics
##
##                Accuracy : 0.5676
##                  95% CI : (0.4472, 0.6823)
##     No Information Rate : 0.4189
```

```
##     P-Value [Acc > NIR] : 0.007026
##
##                   Kappa : 0.2965
##
##  Mcnemar's Test P-Value : 0.002408
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                   0.5400     0.75000        0.5625
## Recall                      0.8710     0.37500        0.3333
## F1                          0.6667     0.50000        0.4186
## Prevalence                  0.4189     0.21622        0.3649
## Detection Rate              0.3649     0.08108        0.1216
## Detection Prevalence        0.6757     0.10811        0.2162
## Balanced Accuracy           0.6680     0.67026        0.5922
```