DSP_word2vec_ds2

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
ds2$ANF_BESCHREIBUNG <- txt_clean_word2vec(ds2$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_ds2 <- word2vec(ds2$ANF_BESCHREIBUNG, dim = 10, iter = 15)
embedding_ds2 <- as.matrix(model_ds2)
```

```
# Dimension der Einbettung
dim(embedding_ds2)
```

```
## [1] 5305   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
ds2$token <- tokenizers::tokenize_words(ds2$ANF_BESCHREIBUNG)
```

```
# Vektor der Laenge 10 fuer jedes Dokument
features2 <- matrix(nrow = 0, ncol = 10)
for (i in (1:length(ds2$ANF_BESCHREIBUNG))){
  vec_doc1 <- doc2vec(model_ds2, ds2$token[1][[1]][i], split = " ")
  features2 <- rbind(features2, vec_doc1)
}
```

Zusammenführen mit anderen Prädiktoren

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

```
features2 <- as.data.frame(features2)
ds2_all <- cbind(ds2, features2)
ds2_all <- as.data.frame(ds2_all)
```

```
df <- ds2_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         50.0             OK           22.30 -0.32154928  1.5886789
## 2     mittel         50.0             OK           22.30  0.00000000  0.0000000
## 3     mittel         50.0             OK           22.20  0.00000000  0.0000000
## 4     mittel         50.0             OK           22.20  1.69181746 -0.4741313
## 5     gering          0.0             OK             21x -0.72511362  0.7625329
## 6     mittel          0.8         FAILED           22.30  0.09097556  2.1890372
##            V3         V4          V6          V7          V8         V9
## 1 -0.54065718 -0.4628298 -0.03377985  0.06681724  1.06705634 -2.1194228
## 2  0.00000000  0.0000000  0.00000000  0.00000000  0.00000000  0.0000000
## 3  0.00000000  0.0000000  0.00000000  0.00000000  0.00000000  0.0000000
## 4 -0.09308541 -1.1222361 -0.30554955  0.46509964 -0.27934741 -1.4483209
## 5 -0.21299150 -1.7164870  0.35718434 -0.43193369 -1.76328680 -1.2511996
## 6  0.19405335 -0.3362219  0.83137856 -0.76583335  0.06005758  0.7565307
##           V10
## 1 -0.23476289
## 2  0.00000000
## 3  0.00000000
## 4 -0.52109216
## 5  0.76657807
## 6 -0.08763688
```

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
##  gering: 633   Min.   :0.00000   FAILED: 577    21x     :1146
##  hoch  :1122   1st Qu.:0.04121   OK    :2363    22.10   : 434
##  mittel:1366   Median :0.17229   OPEN  : 181    22.20   : 727
##                Mean   :0.30279                  22.30   : 326
##                3rd Qu.:0.50348                  OLDERT21: 488
##                Max.   :1.00000
##        V1               V2               V3               V4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.4304   1st Qu.:0.3957   1st Qu.:0.5285   1st Qu.:0.6523
##  Median :0.4304   Median :0.3957   Median :0.5285   Median :0.6523
##  Mean   :0.4339   Mean   :0.3993   Mean   :0.5280   Mean   :0.6468
##  3rd Qu.:0.4304   3rd Qu.:0.3957   3rd Qu.:0.5285   3rd Qu.:0.6523
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##        V6               V7               V8               V9
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.5087   1st Qu.:0.4268   1st Qu.:0.5208   1st Qu.:0.4513
##  Median :0.5087   Median :0.4268   Median :0.5208   Median :0.4513
##  Mean   :0.5107   Mean   :0.4270   Mean   :0.5186   Mean   :0.4548
##  3rd Qu.:0.5087   3rd Qu.:0.4268   3rd Qu.:0.5208   3rd Qu.:0.4513
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       V10
##  Min.   :0.0000
##  1st Qu.:0.4822
##  Median :0.4822
##  Mean   :0.4857
##  3rd Qu.:0.4822
##  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, ]
```

```r
# 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.

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

```r
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     13   11      3
##     hoch       73  193    231
##     mittel     40   20     39
##
## Overall Statistics
##
##                Accuracy : 0.3933
##                  95% CI : (0.3547, 0.4329)
##     No Information Rate : 0.4382
##     P-Value [Acc > NIR] : 0.9896
##
##                   Kappa : 0.0442
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.48148      0.3883        0.3939
## Recall                     0.10317      0.8616        0.1429
## F1                         0.16993      0.5354        0.2097
## Prevalence                 0.20225      0.3596        0.4382
## Detection Rate             0.02087      0.3098        0.0626
## Detection Prevalence       0.04334      0.7978        0.1589
## Balanced Accuracy          0.53750      0.5499        0.4857
```

KNN Klassifikation

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

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

```r
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     46   15     27
##     hoch       11  122     34
##     mittel     69   87    212
##
## Overall Statistics
##
##                Accuracy : 0.61
##                  95% CI : (0.5704, 0.6485)
##     No Information Rate : 0.4382
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.367
##
##  Mcnemar's Test P-Value : 3.629e-09
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.52273      0.7305        0.5761
## Recall                     0.36508      0.5446        0.7766
## F1                         0.42991      0.6240        0.6615
## Prevalence                 0.20225      0.3596        0.4382
## Detection Rate             0.07384      0.1958        0.3403
## Detection Prevalence       0.14125      0.2681        0.5907
## Balanced Accuracy          0.64029      0.7159        0.6654
```