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  0.03469865 -2.0330779
## 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 -1.41458112  1.0161230
## 5     gering          0.0             OK             21x -1.55033128 -0.8415447
## 6     mittel          0.8         FAILED           22.30  0.79025176 -1.0669475
##            V3         V4          V6         V7         V8        V9
## 1  0.65497564  1.2960864  1.73520442  0.1694185 -0.4113724 0.1410621
## 2  0.00000000  0.0000000  0.00000000  0.0000000  0.0000000 0.0000000
## 3  0.00000000  0.0000000  0.00000000  0.0000000  0.0000000 0.0000000
## 4  0.04195164  0.6624943  0.12968210  1.6377786 -1.1399058 0.2477629
## 5 -0.42474223 -0.3348293 -0.99013432 -0.7045494 -0.9925252 0.7997408
## 6 -1.18397612 -0.2470517 -0.02052553  0.9134457 -1.7328399 1.3559976
##           V10
## 1 -0.40510514
## 2  0.00000000
## 3  0.00000000
## 4  0.06746044
## 5 -1.00307491
## 6  0.26797082
```

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.:0.4469   1st Qu.:0.5451   1st Qu.:0.6139   1st Qu.:0.3845
##  Median :0.4469   Median :0.5451   Median :0.6139   Median :0.3845
##  Mean   :0.4476   Mean   :0.5448   Mean   :0.6085   Mean   :0.3886
##  3rd Qu.:0.4469   3rd Qu.:0.5451   3rd Qu.:0.6139   3rd Qu.:0.3845
##  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.:0.5020   1st Qu.:0.4573   1st Qu.:0.5714   1st Qu.:0.5254
##  Median :0.5020   Median :0.4573   Median :0.5714   Median :0.5254
##  Mean   :0.4977   Mean   :0.4571   Mean   :0.5720   Mean   :0.5275
##  3rd Qu.:0.5020   3rd Qu.:0.4573   3rd Qu.:0.5714   3rd Qu.:0.5254
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       V10
##  Min.   :0.0000
##  1st Qu.:0.4476
##  Median :0.4476
##  Mean   :0.4526
##  3rd Qu.:0.4476
##  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)
model_nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##    gering      hoch    mittel
## 0.2029624 0.3594876 0.4375500
##
## Conditional probabilities:
##         TF_ABDECKUNG
## Y             [,1]      [,2]
##   gering 0.4976361 0.3581394
##   hoch   0.2125114 0.2608936
##   mittel 0.2763715 0.3248865
##
##         AKT_RES_STATUS
## Y            FAILED         OK       OPEN
##   gering 0.16962525 0.76134122 0.06903353
##   hoch   0.16592428 0.78507795 0.04899777
##   mittel 0.20402562 0.73559012 0.06038426
##
##         AKT_RES_RELEASE
## Y               21x      22.10      22.20      22.30   OLDERT21
##   gering 0.42998028 0.08678501 0.11637081 0.13412229 0.23274162
##   hoch   0.34966592 0.20712695 0.24053452 0.07572383 0.12694878
##   mittel 0.36962489 0.10338518 0.28179323 0.10064044 0.14455627
##
##         V1
## Y             [,1]       [,2]
##   gering 0.4492933 0.07319039
##   hoch   0.4462600 0.03776641
##   mittel 0.4477408 0.04762433
##
##         V2
## Y             [,1]       [,2]
##   gering 0.5439576 0.05839234
##   hoch   0.5453284 0.02341427
##   mittel 0.5439728 0.04306611
##
```

```
##         V3
## Y             [,1]       [,2]
##   gering 0.5991114 0.08696158
##   hoch   0.6134476 0.03157953
##   mittel 0.6110923 0.04201616
##
##         V4
## Y             [,1]       [,2]
##   gering 0.3928440 0.06493946
##   hoch   0.3867280 0.03374744
##   mittel 0.3888233 0.04552037
##
##         V6
## Y             [,1]       [,2]
##   gering 0.4951425 0.06586051
##   hoch   0.4987203 0.03858665
##   mittel 0.5003124 0.04333781
##
##         V7
## Y             [,1]       [,2]
##   gering 0.4546220 0.04777649
##   hoch   0.4581153 0.02497373
##   mittel 0.4584831 0.04107727
##
##         V8
## Y             [,1]       [,2]
##   gering 0.5728132 0.04469908
##   hoch   0.5715426 0.03609546
##   mittel 0.5721168 0.02859834
##
##         V9
## Y             [,1]       [,2]
##   gering 0.5343039 0.05426083
##   hoch   0.5243991 0.03934541
##   mittel 0.5274360 0.03401781
##
##         V10
## Y             [,1]       [,2]
##   gering 0.4560856 0.06275822
##   hoch   0.4515037 0.04069660
##   mittel 0.4516067 0.03687919
```

```
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       79  199    233
##     mittel     34   14     37
##
## Overall Statistics
```

```
##
##                Accuracy : 0.3997
##                  95% CI : (0.361, 0.4393)
##     No Information Rate : 0.4382
##     P-Value [Acc > NIR] : 0.9764
##
##                   Kappa : 0.0569
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.48148      0.3894       0.43529
## Recall                     0.10317      0.8884       0.13553
## F1                         0.16993      0.5415       0.20670
## Prevalence                 0.20225      0.3596       0.43820
## Detection Rate             0.02087      0.3194       0.05939
## Detection Prevalence       0.04334      0.8202       0.13644
## Balanced Accuracy          0.53750      0.5532       0.49919
```

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))
model_knn
```

```
## k-Nearest Neighbors
##
## 2498 samples
##   12 predictor
##    3 classes: 'gering', 'hoch', 'mittel'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1999, 1997, 1997, 1999, 2000
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.5828533  0.3319275
##   7  0.5788461  0.3239148
##   9  0.5760557  0.3183413
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 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     45   16     27
##     hoch       11  121     35
##     mittel     70   87    211
##
## Overall Statistics
##
##                Accuracy : 0.6051
##                  95% CI : (0.5655, 0.6437)
##     No Information Rate : 0.4382
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3592
##
##  Mcnemar's Test P-Value : 3.725e-09
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.51136      0.7246        0.5734
## Recall                     0.35714      0.5402        0.7729
## F1                         0.42056      0.6189        0.6583
## Prevalence                 0.20225      0.3596        0.4382
## Detection Rate             0.07223      0.1942        0.3387
## Detection Prevalence       0.14125      0.2681        0.5907
## Balanced Accuracy          0.63531      0.7124        0.6622
```