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 -0.84282229 -0.9748338
## 2       hoch        50.00             OK           22.10  0.49018952 -0.1577680
## 3       hoch        11.11             OK           22.10  0.17469695 -1.1638943
## 4       hoch        50.00             OK           22.10 -1.27127496 -1.6572690
## 5       hoch        33.33             OK           22.10 -1.03427157  0.2177840
## 6     mittel        25.00             OK           22.10  0.05299106  0.9683255
##           V3           V4        V6         V7         V8          V9
## 1 -1.9175118  0.511867963 1.2806262  0.2646515  0.5896901 -0.02586405
## 2 -1.6116727  0.692813191 1.2516870 -0.1527890  0.1462077  0.06484574
## 3 -0.4684088 -0.238539752 1.2442378 -1.0349975  0.6875121  1.39700351
## 4 -0.9991738  0.605265155 0.6962658 -0.8326447 -0.8274708  0.26645309
## 5 -0.5433600 -0.008262809 1.0396007 -0.3611670  2.4004223 -0.90813945
## 6 -1.1787004  1.893247830 1.0985802  0.1639573  1.1991250 -0.97900765
##          V10
## 1  1.5140560
## 2  2.2111498
## 3  0.9695238
## 4 -1.1622594
## 5 -0.8497838
## 6  0.2347105
```

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.:0.5003
##  mittel:665   Median :0.1429   OPEN  :   3    22.20   :  12   Median :0.5003
##               Mean   :0.2320                   22.30   : 302   Mean   :0.5009
##               3rd Qu.:0.3333                   OLDERT21:   3   3rd Qu.:0.5003
##               Max.   :1.0000                                   Max.   :1.0000
##        V2               V3               V4               V6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.6114   1st Qu.:0.6996   1st Qu.:0.4776   1st Qu.:0.4549
##  Median :0.6114   Median :0.6996   Median :0.4776   Median :0.4549
##  Mean   :0.6080   Mean   :0.6892   Mean   :0.4768   Mean   :0.4615
##  3rd Qu.:0.6114   3rd Qu.:0.6996   3rd Qu.:0.4776   3rd Qu.:0.4549
##  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.:0.6344   1st Qu.:0.3768   1st Qu.:0.5106   1st Qu.:0.4928
##  Median :0.6344   Median :0.3768   Median :0.5106   Median :0.4928
##  Mean   :0.6314   Mean   :0.3807   Mean   :0.5130   Mean   :0.4938
##  3rd Qu.:0.6344   3rd Qu.:0.3768   3rd Qu.:0.5106   3rd Qu.:0.4928
##  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)
model_nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##    gering      hoch    mittel
## 0.1668107 0.3733794 0.4598099
##
## Conditional probabilities:
##         TF_ABDECKUNG
## Y            [,1]      [,2]
##   gering 0.352813 0.3035048
##   hoch   0.212209 0.2440976
##   mittel 0.201153 0.2133919
##
##         AKT_RES_STATUS
## Y             FAILED          OK        OPEN
##   gering 0.005181347 0.989637306 0.005181347
##   hoch   0.009259259 0.990740741 0.000000000
##   mittel 0.058270677 0.941729323 0.000000000
##
##         AKT_RES_RELEASE
## Y               21x      22.10      22.20      22.30   OLDERT21
##   gering 0.00000000 0.86010363 0.00000000 0.12953368 0.01036269
##   hoch   0.04861111 0.71296296 0.00462963 0.23379630 0.00000000
##   mittel 0.03571429 0.72744361 0.01503759 0.22180451 0.00000000
##
##         V1
## Y            [,1]       [,2]
##   gering 0.4967181 0.03552152
##   hoch   0.5006926 0.02231741
##   mittel 0.5042564 0.06886485
##
##         V2
## Y            [,1]       [,2]
##   gering 0.6083777 0.04407849
##   hoch   0.6103947 0.02405195
##   mittel 0.6079147 0.05450881
##
##         V3
## Y            [,1]       [,2]
##   gering 0.6948004 0.05713392
##   hoch   0.6958183 0.03863756
##   mittel 0.6856281 0.08250294
```

```
##
##         V4
## Y             [,1]       [,2]
##   gering 0.4754348 0.03907584
##   hoch   0.4791940 0.01833275
##   mittel 0.4760484 0.06000427
##
##         V6
## Y             [,1]       [,2]
##   gering 0.4568688 0.01881547
##   hoch   0.4583139 0.03307815
##   mittel 0.4633915 0.06552417
##
##         V7
## Y             [,1]        [,2]
##   gering 0.6332665 0.009672893
##   hoch   0.6335647 0.023851365
##   mittel 0.6294413 0.071919884
##
##         V8
## Y             [,1]       [,2]
##   gering 0.3765377 0.02660749
##   hoch   0.3780844 0.02797547
##   mittel 0.3825441 0.05868134
##
##         V9
## Y             [,1]       [,2]
##   gering 0.5116623 0.02077096
##   hoch   0.5114331 0.01910105
##   mittel 0.5142982 0.06786922
##
##         V10
## Y             [,1]       [,2]
##   gering 0.4920465 0.02297277
##   hoch   0.4953050 0.03858354
##   mittel 0.4937473 0.05826253
```

```
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     11    3      6
##     hoch       35  102    114
##     mittel      2    3     13
##
## Overall Statistics
##
##                Accuracy : 0.436
##                  95% CI : (0.378, 0.4953)
##     No Information Rate : 0.4602
##     P-Value [Acc > NIR] : 0.8119
```

```
##
##                   Kappa : 0.1122
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.55000      0.4064       0.72222
## Recall                     0.22917      0.9444       0.09774
## F1                         0.32353      0.5682       0.17219
## Prevalence                 0.16609      0.3737       0.46021
## Detection Rate             0.03806      0.3529       0.04498
## Detection Prevalence       0.06920      0.8685       0.06228
## Balanced Accuracy          0.59591      0.5606       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))
model_knn
```

```
## k-Nearest Neighbors
##
## 1157 samples
##   12 predictor
##    3 classes: 'gering', 'hoch', 'mittel'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 925, 925, 927, 927, 924
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.6109404  0.3547421
##   7  0.5962661  0.3313116
##   9  0.5876340  0.3188926
##
## 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     21    9     10
##     hoch       10   55      8
##     mittel     17   44    115
##
```

```
## Overall Statistics
##
##                Accuracy : 0.6609
##                  95% CI : (0.6032, 0.7153)
##     No Information Rate : 0.4602
##     P-Value [Acc > NIR] : 5.052e-12
##
##                   Kappa : 0.437
##
##  Mcnemar's Test P-Value : 6.514e-06
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                  0.52500      0.7534        0.6534
## Recall                     0.43750      0.5093        0.8647
## F1                         0.47727      0.6077        0.7443
## Prevalence                 0.16609      0.3737        0.4602
## Detection Rate             0.07266      0.1903        0.3979
## Detection Prevalence       0.13841      0.2526        0.6090
## Balanced Accuracy          0.67933      0.7049        0.7368
```