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 -0.8003765  0.25678300
## 2     mittel         16.7             OK             21x -2.0562674 -0.24471149
## 3     gering         50.0             OK             21x -1.5874573  0.03242045
## 4     gering        100.0             OK             21x -1.8581314  0.22139942
## 5     gering        100.0             OK           22.10  0.3463377 -0.10300854
## 6       hoch         20.0             OK             21x -1.5874573  0.03242045
##            V3        V4         V6          V7        V8         V9         V10
## 1 -1.22818180 0.6894092 0.54377024 -0.66773842 0.4857516 -0.2674228 -2.4961874
## 2 -0.07507532 1.8370696 0.04684066 -0.06300781 0.6013248 -0.4731791 -1.2916497
## 3 -0.50426832 1.1812533 0.32485866 -0.04006239 1.6522530 -1.1603935 -1.2016314
## 4  1.05255329 1.7805005 0.45203909  1.06978428 0.3265966  0.7489142  0.4229912
## 5 -1.36255752 0.8725399 0.11455244 -1.09412049 1.8437377 -0.5753562 -1.4981779
## 6 -0.50426832 1.1812533 0.32485866 -0.04006239 1.6522530 -1.1603935 -1.2016314
```

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.:0.6319
##  mittel:136   Median :1.0000   OPEN  :  7     22.20   :129    Median :0.6319
##               Mean   :0.8056                  22.30   :  3    Mean   :0.6105
##               3rd Qu.:1.0000                  OLDERT21: 84    3rd Qu.:0.6319
##               Max.   :1.0000                                  Max.   :1.0000
##        V2               V3               V4               V6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.1550   1st Qu.:0.5002   1st Qu.:0.1822   1st Qu.:0.00000
##  Median :0.1550   Median :0.5002   Median :0.1822   Median :0.00000
##  Mean   :0.1634   Mean   :0.5042   Mean   :0.2157   Mean   :0.02731
##  3rd Qu.:0.1550   3rd Qu.:0.5002   3rd Qu.:0.1822   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##        V7               V8               V9              V10
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.5078   1st Qu.:0.1251   1st Qu.:0.4398   1st Qu.:0.7258
##  Median :0.5078   Median :0.1251   Median :0.4398   Median :0.7258
##  Mean   :0.5077   Mean   :0.1481   Mean   :0.4489   Mean   :0.7195
##  3rd Qu.:0.5078   3rd Qu.:0.1251   3rd Qu.:0.4398   3rd Qu.:0.7258
##  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)
model_nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
```

```
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##    gering      hoch    mittel
## 0.4177632 0.2236842 0.3585526
##
## Conditional probabilities:
##         TF_ABDECKUNG
## Y             [,1]      [,2]
##   gering 0.9763780 0.1065004
##   hoch   0.5109706 0.4309255
##   mittel 0.8081101 0.2836062
##
##         AKT_RES_STATUS
## Y             FAILED          OK        OPEN
##   gering 0.039370079 0.952755906 0.007874016
##   hoch   0.029411765 0.926470588 0.044117647
##   mittel 0.018348624 0.963302752 0.018348624
##
##         AKT_RES_RELEASE
## Y                21x       22.10       22.20       22.30    OLDERT21
##   gering 0.047244094 0.503937008 0.401574803 0.007874016 0.039370079
##   hoch   0.102941176 0.132352941 0.294117647 0.014705882 0.455882353
##   mittel 0.055045872 0.311926606 0.321100917 0.000000000 0.311926606
##
##         V1
## Y             [,1]       [,2]
##   gering 0.6315250 0.05291128
##   hoch   0.6040347 0.11096132
##   mittel 0.6077035 0.12291990
##
##         V2
## Y             [,1]       [,2]
##   gering 0.1591272 0.04325616
##   hoch   0.1564275 0.05374757
##   mittel 0.1664143 0.08725981
##
##         V3
## Y             [,1]       [,2]
##   gering 0.4954395 0.06831511
##   hoch   0.5120594 0.08568198
##   mittel 0.5020945 0.04478814
##
##         V4
## Y             [,1]      [,2]
##   gering 0.1923237 0.0785546
##   hoch   0.2101993 0.1154235
##   mittel 0.2173670 0.1570890
##
##         V6
## Y               [,1]       [,2]
```

```
##   gering 0.005269953 0.04043246
##   hoch   0.035870315 0.15238105
##   mittel 0.024888020 0.10873305
##
##         V7
## Y             [,1]       [,2]
##   gering 0.5047844 0.07091077
##   hoch   0.5059608 0.08247463
##   mittel 0.5108842 0.02206234
##
##         V8
## Y             [,1]       [,2]
##   gering 0.1367567 0.08563133
##   hoch   0.1473446 0.12706433
##   mittel 0.1480079 0.10596163
##
##         V9
## Y             [,1]       [,2]
##   gering 0.4401778 0.03172662
##   hoch   0.4486013 0.10515736
##   mittel 0.4439268 0.07729015
##
##         V10
## Y             [,1]       [,2]
##   gering 0.7206001 0.05078283
##   hoch   0.7233807 0.04759087
##   mittel 0.7206013 0.05459928
```

```
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        2    6      6
##     mittel      3    4      5
##
## Overall Statistics
##
##                Accuracy : 0.5
##                  95% CI : (0.3814, 0.6186)
##     No Information Rate : 0.4189
##     P-Value [Acc > NIR] : 0.09807
##
##                   Kappa : 0.2041
##
##  Mcnemar's Test P-Value : 0.01023
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                   0.5417     0.42857       0.41667
```

```
## Recall                      0.8387     0.37500       0.18519
## F1                          0.6582     0.40000       0.25641
## Prevalence                  0.4189     0.21622       0.36486
## Detection Rate              0.3514     0.08108       0.06757
## Detection Prevalence        0.6486     0.18919       0.16216
## Balanced Accuracy           0.6635     0.61853       0.51812
```

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
##
## 304 samples
##  12 predictor
##   3 classes: 'gering', 'hoch', 'mittel'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 243, 244, 243, 243, 243
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.5822951  0.3149617
##   7  0.5790164  0.3107654
##   9  0.5724590  0.3000309
##
## 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     27    7     16
##     hoch        0    6      1
##     mittel      4    3     10
##
## Overall Statistics
##
##                Accuracy : 0.5811
##                  95% CI : (0.4606, 0.6949)
##     No Information Rate : 0.4189
##     P-Value [Acc > NIR] : 0.003584
##
##                   Kappa : 0.3162
##
```

```
##  Mcnemar's Test P-Value : 0.001653
##
## Statistics by Class:
##
##                      Class: gering Class: hoch Class: mittel
## Precision                   0.5400     0.85714        0.5882
## Recall                      0.8710     0.37500        0.3704
## F1                          0.6667     0.52174        0.4545
## Prevalence                  0.4189     0.21622        0.3649
## Detection Rate              0.3649     0.08108        0.1351
## Detection Prevalence        0.6757     0.09459        0.2297
## Balanced Accuracy           0.6680     0.67888        0.6107
```