Problem Set PS_6

Table of Contents

Social Media Analytics - Problem Set Nr. 6 1
Exercise 11 -- Textklassifikation 1
a) Naive Bayes-Klassifikation am Beispiel - deutsch vs. italienisch 2
b) k-Nearest-Neighbor-Klassifikation zur Sentiment Analysis von Filmkritiken - positiv vs. negativ 9
Award: Textklassifikation 16
Exercise LV -- Literaturverzeichnis 16
Packages: 16
Packagebeschreibungen/Hilfeseiten: 17
Buch-, Paper- und Onlinequellen: 17

Social Media Analytics - Problem Set Nr. 6

Stand: 03.07.2017

Willkommen zum sechsten RTutor-Problem Set in diesem Modul!
Sie finden hier die letzte Aufgabe in diesem Semester (*Exercise 11*) zu Inhalten aus Kapitel 4 *Text Mining* - konkret zu Teil 4.2.2 *Textklassifikation*.

Exercise 11 -- Textklassifikation

Nachdem in der vorherigen *Exercise 10* Text Clustering von Textklassifikation abgegrenzt wurde, soll nun konkret gezeigt werden, wie man Texte mit den in R zur Verfügung stehenden Befehlen klassifiziert. Hierfür wird auf die Document-Term-Matrix (DTM) als Repräsentationsmodell für den Text zurückgegriffen.

Im Skript haben Sie bereits die **Naive Bayes-Klassifikation** kennengelernt, um Textdokumente einer von mehreren vordefinierten Klassen zuzuweisen. Es handelt sich hierbei um eine statistische Klassifikationsmethode, die zu den Bayes'schen Klassifikationsverfahren gehört. Ein weiteres Ihnen vertrautes Klassifikationsverfahren ist die **k-Nearest-Neighbour-Klassifikation**, die zu den instanzbasierten Klassifikationsverfahren zählt. Die Grundidee besteht bei diesem Klassifikator darin, Textdokumente auf Basis der *k* ihnen ähnlichsten Trainingsdokumente zu klassifizieren.
Im Folgenden möchten wir Ihnen nun die Anwendung dieser beiden Klassifikatoren in R näher bringen und ihre Güte mit den Ihnen bekannten Evaluationsmetriken ermitteln.
Zunächst führen wir die *Naive Bayes-Klassifikation* und *Evaluationsmetriken* in R anhand eines überschaubaren manuell erzeugten Datensatzes ein. Anschließend wird Ihnen die R-

Funktion zur *k-Nearest-Neighbour-Klassifikation* erläutert, mit welcher sie eine Sentiment Analysis der Ihnen bekannten Filmkritiken von Pang und Lee (2004) durchführen. Auch hier werden die Ergebnisse evaluiert.

Wie Sie im Skript gelernt haben, besteht das Verfahren der Textklassifikation aus zwei Schritten:
1. Modellkonstruktion ("Lernen"): Basierend auf Trainingsdaten, deren Zuordnung zu Klassen bereits bekannt ist, leitet das Klassifikationsverfahren ein Klassifikationsmodell ab.
2. Modellvalidierung und -nutzung (Klassifikation): Das Klassifikationsmodell wird zunächst auf Testdokumente angewendet, deren Zuordnung zu Klassen ebenfalls im Vorhinein bekannt ist. Diese Klassifikationsergbnisse werden evaluiert. Bei ausreichender Validität des Modells wird dieses anschließend auf die eigentlich zu klassifizierenden Dokumente angewandt.
(Zhai und Massung, 2016)

Im Rahmen dieser Übungsaufgabe werden wir für jedes der beiden betrachteten Klassifikationsverfahren die Modellkonstruktion und die Modellvalidierung durchführen. Auf die Modellnutzung wollen wir hier verzichten. Das Vorgehen wäre hierbei jedoch analog zur Anwendung des Klassifikationsmodells auf die Testdokumente - mit dem Unterschied, dass Dokumente verwendet werden, deren Klassenzuordnung noch nicht bekannt ist.

a) Naive Bayes-Klassifikation am Beispiel - deutsch vs. italienisch

Sie haben bereits auf Übungsblatt 4 gelernt, wie Daten in R per Hand erzeugt und eingelesen werden können, um Code auf kleinen Datenmengen zu testen. Um die R-Funktion zur Klassifikation nach dem **Naive Bayes-Klassifikator** einzuführen, verwenden wir wieder als Beispiel die Posts über die Essgewohnheiten von Restaurantbesuchern, die Sie bereits auf dem Übungsblatt 4 mit R in Cluster unterteilen sollten, wobei hier die Menge an Posts etwas erhöht wurde. Ein Teil der Posts dient dabei als Trainingsdokumente, die restlichen Posts als Testdokumente. Das Ziel dieser Klassifikation soll sein, die Verfasser von Posts nach ihren Essgewohnheiten unterscheiden zu können. Konkret soll danach unterschieden werden, ob jemand die deutsche oder die italienische Küche bevorzugt. Alle insgesamt 14 Posts wurden dazu basierend auf ihrem Inhalt per Hand bereits einer der beiden Klassen *deutsche Küche (D)* oder *italienische Küche (I)* zugeordnet. Insgesamt ergeben sich dadurch sieben Posts der Klasse *D* und weitere sieben der Klasse *I*.

Die Naive Bayes-Klassifikation kann in R mit dem Package *'e1071'* in zwei Schritten durchgeführt werden:
A. Basierend auf dem Textinhalt der Trainingsdokumente und ihrer Zuordnung zu den Klassen wird ein **Naive Bayes-Klassifikator gebildet**.
B. Der Naive Bayes-Klassifikator wird auf Textdokumente angewendet um die einzelnen **Textdokumente den Klassen zuzuordnen**.

Hinweis: Schritt B kann sowohl auf Testdokumente (mit bekannter Klasse) als auch auf Textdokumente mit unbekannter Klasse angewendet werden. Wir beschränken uns in dieser Aufgabe auf die Anwendung auf Testdokumente.

Für den ersten Schritt zu unserem Ziel, die Testdokumente gemäß eines Naive Bayes-Klassifikators den Klassen *D* und *I* zuzuordnen, müssen die Daten zunächst eingelesen und aufbereitet werden. Lesen Sie dazu bitte zunächst alle Posts des eben beschriebenen Beispiels ein und erzeugen Sie wie gewohnt ein Objekt der Klasse *VCorpus* aus den Textinhalten, indem Sie den folgenden Code ausführen.
Hinweis: Mit einem Klick auf den Button data über dem Code Chunk nach dem Ausführen des Codes können Sie im Data Explorer den Data Frame posts einsehen.

```
#Laden des Packages 'tm', um Methoden des Text Mining verwenden zu können
library(tm)
#Laden der Posts mit der zugehörigen Klasse
posts <-  rbind(
  c('Wenn ich meine Freundin zum Essen einlade, gehen wir meistens
     zum Italiener um die Ecke. Dort gibt es nicht nur die beste
     Pizza der Stadt, sondern auch einen super leckeren Wein','I'),
  c('Mein Lieblingsrestaurant hat eine gut bürgerliche Küche.
     Hier esse ich am liebsten Schnitzel und trinke dazu ein
     Bier.','D'),
  c('Pizza und Wein und alles ist fein!','I'),
  c('Ich mag italienische Pizza und vor allem Wein','I'),
  c('Für mich besteht ein gutes Essen aus einem großen Schnitzel
     und dazu gehört auch ein Bier','D'),
  c('Stimme ich zu, ein gutes Restaurant hat für mich Schnitzel
     und Bier','D'),
  c('Pizza, Pasta, Salat, egal wichtig ist italienische Küche!
     Ein guter Wein darf auch nicht fehlen.','I'),
  c('Das seh ich genau so. Wenn ich im Restaurant bin braucht
     es auf jeden Fall einen guten Wein und einen Salat als Vorspeise.
     Ob dann Pizza oder Pasta ist mir egal','I'),
  c('Ich gehe nur deutsch Essen. Wenn ich im Restaurant bin dann
     braucht es Schnitzel und Bier, das reicht!','D'),
  c('Schnitzel in allen Varianten und Bier vom Fass, das muss ein
     gutes Restaurant zu bieten haben','D'),
  c('Wein ist super und nichts geht über Pizza! Nur mit Salat kann
     ich nichts anfangen','I'),
  c('In einem deutschen Restaurant bestelle ich mir zu einem panierten
     Schnitzel ein Bier vom Fass','D'),
  c('Am liebsten sitze ich im Sommer bei meinem Lieblingsitaliener
     draußen bei einem Glas Wein und einer Pizza','I'),
  c('Ach was, Bier und Schnitzel das ist es! Hauptsache deutsch!','D')
  )

#Einlesen der Posts in einen Corpus
#zur Erzeugung der DTM
posts_corpus <- VCorpus(VectorSource(posts[,1]))
```

Analog zum Text Clustering muss auch für die Textklassifikation der Textinhalt in **maschinenlesbarer Form** dargestellt werden. Hierzu wird wieder die Funktion *DocumentTermMatrix()* verwendet, die Sie bereits in Problem Set Nr. 5 kennengelernt haben. Ihr werden die Posts in Form des eben erzeugten *VCorpus*-Objekts *posts_corpus*

übergeben. An dieser Stelle wird auf das Entfernen von Stoppwörtern verzichtet, um in einem späteren Schritt die Auswirkung auf die Klassifikation mit und ohne Stoppwörter zu vergleichen. Nachdem die Document Term Matrix (DTM) mit der absoluten Termfrequenz (tf) als Gewichtung erzeugt wurde, wird - wie bereits in Problem Set Nr. 5 - ein Überblick über die DTM mit der Funktion *inspect()* ausgegeben. Führen Sie bitte den folgenden Code Chunk aus, um die eben erläuterten Schritte zu realisieren.

```
#Erstellen der Document Term Matrix
DTM_tf <- DocumentTermMatrix(x=posts_corpus,
                             control=list(
                               stopwords= FALSE,
                               removeNumbers=TRUE,
                               removePunctuation=TRUE,
                               tolower=TRUE,
                               stripWhitespace=TRUE,
                               stemming=FALSE))

#Begutachten der DTM
inspect(x=DTM_tf)
```

```
## <<DocumentTermMatrix (documents: 14, terms: 111)>>
## Non-/sparse entries: 202/1352
## Sparsity           : 87%
## Maximal term length: 19
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs bier das ein ich ist pizza restaurant schnitzel und wein
##   1     0   0   0   1   0     1          0         0   0    1
##   10    1   1   1   0   0     0          1         1   1    0
##   11    0   0   0   1   1     1          0         0   1    1
##   12    1   0   1   1   0     0          1         1   0    0
##   13    0   0   0   1   0     1          0         0   1    1
##   2     1   0   1   1   0     0          0         1   1    0
##   5     1   0   2   0   0     0          0         1   1    0
##   7     0   0   1   0   1     1          0         0   0    1
##   8     0   1   0   2   1     1          1         0   1    1
##   9     1   1   0   2   0     0          1         1   1    0
```

Die erzeugte DTM stellt die absolute Häufigkeit aller Terme in den einzelnen Posts dar. Wie Sie sowohl am Wert der *Sparsity* von 87% als auch an dem dargestellten Beispielausschnitt der DTM erkennen können, enthält die DTM mehr Einträge größer als Null als die DTM der Filmkritiken von Problem Set Nr. 5, die wir später erneut betrachten werden.

Auf Basis der Zeilen der eben erzeugten DTM soll nun der *Naive Bayes-Klassifikator* **trainiert** und später **evaluiert** werden.
Anders als beim Clustering kann die DTM jedoch nicht direkt an die Funktion aus dem Package *'e1071'* zur Bildung des *Naive Bayes-Klassifikators* übergeben werden, da diese Funktion nicht mit dem Klassentyp von Sparse-Matrizen in R umgehen kann. Daher muss zunächst die DTM mit der Funktion *as.matrix(x)* in eine **dichte Matrix** umgewandelt

werden. Dies hat zur Folge, dass alle Einträge in der DTM, die den Wert Null besitzen, explizit auch mit dem Wert 0 gespeichert werden und sich dadurch der benötigte Speicherplatz für die DTM erhöht. Nachteile in Bezug auf die Rechenzeit entstehen dadurch erst bei größeren Datenmengen.
Nachdem die DTM entsprechend in die richtige Typklasse konvertiert ist, werden **Trainings- und Testdokumente** zusammen mit ihrem Klassenlabel **ausgewählt**. Letzteres beschreibt, welcher Klasse das jeweilige Dokument zugeordnet ist. In diesem Fall werden die ersten sechs Posts als Trainingsdaten und die verbleibenden acht Posts als Testdaten festgelegt. Üblicherweise würde man mehr Trainings- als Testdaten wählen, sodass zum Beispiel ein Verhältnis von 4:1 zwischen Trainings- und Testdaten entsteht. Für dieses einführende Beispiel ist mit der vorliegenden Aufteilung jedoch trotzdem eine genügend genaue Klassifikation gewährleistet. Neben den Posts werden auch die zugehörigen Klassenlabels in entsprechend gekennzeichneten Trainings- und Testlabels gespeichert. Hierbei müssen die Trainingslabels wieder entsprechend auf die Funktion zur Erzeugung des Naive Bayes-Klassifikators aus dem Package *'e1071'* angepasst werden. D. h. sie werden mit der Funktion *as.factor(x)* von Strings in kategorische Werte umgewandelt.
Führen Sie bitte zur Realisierung der Schritte den folgenden Code aus.

```
#Format der DTM umwandeln
DTM_denseMatrix = as.matrix(x=DTM_tf)

#Aufteilung in Trainingsdaten und Testdaten
posts_training <- DTM_denseMatrix[1:6,]
posts_test <- DTM_denseMatrix[7:14,]
#Abspeichern der Trainingslabels im richtigen Format
posts_traininglabel <- as.factor(posts[1:6,2])
#Abspeichern der Testlabels
posts_testlabel <- posts[7:14,2]
```

Nachdem nun die Textdaten entsprechend zur Klassifikation aufbereitet sind, können diese nun für Schritt **A** der *Naive Bayes-Klassifikation* mit dem Package *'e1071'* verwendet werden. Die Funktion *naiveBayes(x,y,laplace)* aus dem Package *'e1071'* erzeugt einen *Naive Bayes-Klassifikator* auf Basis der Trainingsdaten im Eingapeparameter *x* und den zu diesen Daten zugehörigen Klassenlabels im Eingabeparameter *y*. Der Eingabeparameter *laplace* ermöglicht die Anwendung eines Glättungsverfahrens, welches defaultmäßig nicht verwendet wird (*laplace=0*). Dieser Parameter ist an dieser Stelle nur der Vollständigkeit halber aufgeführt und kann im Folgenden vernachlässigt werden.
Um nun mit dem resultierenden *Naive Bayes-Klassifikator* Daten klassifizieren zu können (Schritt **B**), wird die Funktion *predict(object, newdata, type, threshold, eps)* verwendet. Diese nimmt für den Parameter *object* einen *Naive Bayes-Klassifikator* entgegen und klassifiziert auf dieser Basis die Daten, die für den Parameter *newdata* übergeben werden. Der Parameter *type* erlaubt es, zu steuern, ob lediglich die vorhergesagten Klassen ausgegeben werden (Defaulteinstellung) oder auch die Wahrscheinlichkeiten der Daten, dass sie zu den jeweiligen Klassen gehören. Da die Defaulteinstellung für die Klassifikation genügt, kann dieser Parameter ebenso wie die Parameter *treshold* und *eps*, die zur Optimierung bezüglich der Glättung dienen, ebenfalls vernachlässigt werden.
- - - *Aufgabe*: Erzeugen Sie mit der Funktion *naiveBayes()* einen *Naive Bayes-Klassifikator* auf Basis der Trainingsdokumente *posts_training* und den entsprechenden Klassenlabeln

posts_traininglabel. Verwenden Sie nun diesen Klassifikator in der Funktion *predict()*, um die entsprechenden Klassen für die Testdokumente *posts_test* vorherzusagen und geben Sie das Ergebnis der Klassifikation aus. - - -

```
#Verwenden Sie die Funktion naiveBayes(), um mit den Daten aus
#den Variablen posts_training und posts_traininglabel den Naive Bayes
#-Klassifikator zu bilden und speichern Sie das Ergebnis in der Variablen
#nB_classifier
nB_classifier <- naiveBayes(x=posts_training,y=posts_traininglabel)
#Verwenden Sie Ihren eben erzeugten Naive Bayes-Klassifikator
#in der Funktion predict() um die Klassen für die Daten in
#der Variable posts_test vorherzusagen und speichern Sie Ihr
#Ergebnis in der Variablen nB_predicted
nB_predicted <- predict(object=nB_classifier,newdata=posts_test)
#Lassen Sie sich das Ergebnis der Klassifikation ausgeben
nB_predicted
```

```
## [1] I I D D I D D D
## Levels: D I
```

In der Variable *nB_predicted* befinden sich nun die vom Naive Bayes-Klassifikator *nB_classifier* vorhergesagten Klassen für die Posts Nr. 7 bis Nr. 14. Im Output können Sie sehen, dass die Posts 7, 8 und 11 der Klasse *I*, alle restlichen Posts der Klasse *D* zugeordnet wurden.

Um nun die Güte des *Naive Bayes-Klassifikators* mit den aus dem Skript bekannten **Evaluationsmetriken** zu prüfen, vergleichen wir die vorhergesagten Klassen in *nB_predicted* mit den tatsächlichen (d. h. den vorher manuell zugeordneten) Klassen, die in der Variable *posts_testlabel* festgehalten wurden. Hierzu bilden wir zunächst die *Confusion Matrix*, um auf dieser Basis die Metriken *Precision*, *Recall* und *F-Measure* zu berechnen. Zur Erstellung der *Confusion Matrix* kann die Funktion *table("Spaltenname1"=Vektor1, "Spaltenname2"=Vektor2)* verwendet werden. Sie prüft die Gleichheit der Einträge von zwei eingegebenen Vektoren und generiert daraus eine Tabelle.

A uf gabe: Verwenden Sie den Befehl* table()* mit den Dimensionen *"Klasse"* und *"Klassifiziert als"* um eine Confusion Matrix für die Klassifikationsergebnisse in *nB_predicted* zu erhalten. Lassen Sie sich die Confusion Matrix anschließend bitte ausgeben.

```
#Erzeugen Sie eine Confusion Matrix mit dem Befehl table()
#und speichern Sie diese in der Variable nB_confMat
nB_confMat <- table("Klasse" = posts_testlabel,"Klassifiziert als" =
nB_predicted)
#Lassen Sie sich nB_confMat ausgeben
print(nB_confMat)
```

```
##       Klassifiziert als
## Klasse D I
##      D 4 0
##      I 1 3
```

Bevor wir auf die Interpretation der Confusion Matrix eingehen, sollen zunächst auf Grundlage der Confusion Matrix für die Klasse *D*, analog zu den Formeln im Skript, die Metriken *Precision*, *Recall* und die *F-Measure* mit *beta=1* berechnet werden. Führen Sie bitte den folgenden Code aus, betrachten Sie die Ergebnisse und vollziehen Sie die Berechnungen nach.

```
#Berechnung von Precision und Recall für die Klasse D
#Auf Basis der Confusion Matrix
nB_precision_D <- nB_confMat[1,1]/sum(nB_confMat[,1])
print(c("Precision für die Klasse Deutsch:", nB_precision_D))

## [1] "Precision für die Klasse Deutsch:" "0.8"

nB_recall_D <- nB_confMat[1,1]/sum(nB_confMat[1,])
print(c("Recall für die Klasse Deutsch:", nB_recall_D))

## [1] "Recall für die Klasse Deutsch:" "1"

#Berechnen der F-Measure mit beta = 1 für die Klasse D
#Auf Basis von Precision und Recall
beta <- 1
nB_F1_D <-
((beta^2+1)*nB_precision_D*nB_recall_D)/(beta^2*nB_precision_D+nB_recall_D)
print(c("F1-Measure für die Klasse Deutsch:", nB_F1_D))

## [1] "F1-Measure für die Klasse Deutsch:"
## [2] "0.888888888888889"
```

Wie Sie (zunächst einmal an der Confusion Matrix) sehen, wurde ein Großteil der Posts korrekt klassifiziert. Allerdings gibt es einen Post, der fälschlicherweise der Klasse *D* zugewiesen wurde. Dies spiegelt sich in der *Precision* der Klasse *D* wieder. Für die *Precision* ergibt sich ein Wert von 80%, was bedeutet, dass 80% aller als *D* klassifizierten Posts korrekt klassifiziert wurden. Dem Gegenüber steht der *Recall* der Klasse *D* mit 100%, was bedeutet, dass alle Posts, die zur Klasse *D* gehören, korrekt dieser Klasse zugeordnet wurden. *Precision* und *Recall* miteinander verrechnet ergeben dann die *F-Measure*.

Insgesamt ist das Ergebnis der Klassifikation relativ zufriedenstellend. Es stellt sich aber die Frage, ob sich dieses gute Ergebnis noch weiter verbessern lässt, indem man **Stoppwörter entfernt**. Zu diesem Zweck werden nochmal alle bisherigen Schritte in einem Code Chunk wiederholt, mit dem Unterschied, dass die Stoppwörter aus der DTM entfernt werden. Hierbei ist zu beachten, dass das Package *'tm'* Stoppwortlisten zu mehreren Sprachen enthält (darunter auch deutsch), defaultmäßig jedoch von der englischen Sprache ausgeht und für die Eingabe *stopwords = TRUE* alle englischen Stoppwörter aus den Texten filtert. Da in unserem Fall deutschsprachige Texte vorliegen, muss dementsprechend die Liste der Stoppwörter angepasst werden. Das heißt, der Eingabeparameter zur Entfernung der deutschen Stoppwörter bei der Erzeugung der DTM muss auf *stopwords=stopwords(kind="de")* gesetzt werden. Die restlichen Befehle sind analog zu den vorherigen Schritten. Führen Sie den folgenden Code aus und betrachten Sie in der erzeugten *Confusion Matrix* das Ergebnis der *Naive Bayes-Klassifikation* auf Basis der von Stoppwörtern bereinigten Posts.

```r
#Erzeugen der DTM
DTM_tf <- DocumentTermMatrix(x=posts_corpus,
                             control=list(
                               stopwords= stopwords(kind="de"),
                               removeNumbers=TRUE,
                               removePunctuation=TRUE,
                               tolower=TRUE,
                               stripWhitespace=TRUE,
                               stemming=FALSE))

#Begutachten der DTM
inspect(x=DTM_tf)
```

```
## <<DocumentTermMatrix (documents: 14, terms: 64)>>
## Non-/sparse entries: 107/789
## Sparsity           : 88%
## Maximal term length: 19
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs bier braucht deutsch essen gutes pizza restaurant salat schnitzel
##   1     0       0       0     1     0     1          0     0         0
##   10    1       0       0     0     1     0          1     0         1
##   11    0       0       0     0     0     1          0     1         0
##   12    1       0       0     0     0     0          1     0         1
##   13    0       0       0     0     0     1          0     0         0
##   2     1       0       0     0     0     0          0     0         1
##   5     1       0       0     1     1     0          0     0         1
##   7     0       0       0     0     0     1          0     1         0
##   8     0       1       0     0     0     1          1     1         0
##   9     1       1       1     1     0     0          1     0         1
##     Terms
## Docs wein
##   1     1
##   10    0
##   11    1
##   12    0
##   13    1
##   2     0
##   5     0
##   7     1
##   8     1
##   9     0
```

```r
#Format der DTM umwandeln
DTM_denseMatrix = as.matrix(DTM_tf)

#Aufteilen in Trainingsdaten und Testdaten
posts_training <- DTM_denseMatrix[1:6,]
posts_test <- DTM_denseMatrix[7:14,]
#Abspeichern der Trainingslabels im richtigen Format
```

```r
posts_traininglabel <- as.factor(posts[1:6,2])
#Abspeichern der Testlabels
posts_testlabel <- posts[7:14,2]

#Erzeugen des Naive Bayes-Klassifikators
nB_classifier <- naiveBayes(x=posts_training,y=posts_traininglabel)
#Anwenden des Naive Bayes-Klassifikators auf die Testdaten
nB_predicted <- predict(object=nB_classifier,newdata=posts_test)
#Ausgabe der Ergebnisse der Klassifikation
nB_predicted
```

```
## [1] I I D D I D I D
## Levels: D I
```

```r
#Berechnung und Ausgabe der Confusion Matrix
nB_confMat <- table("Klasse" = posts_testlabel,"Klassifiziert als" =
nB_predicted)
print(nB_confMat)
```

```
##       Klassifiziert als
## Klasse D I
##      D 4 0
##      I 0 4
```

Wie Sie sehen, hat das Entfernen der Stoppwörter zu einem zu 100% korrekten Klassifikationsergebnis geführt!

b) k-Nearest-Neighbor-Klassifikation zur Sentiment Analysis von Filmkritiken - positiv vs. negativ

Das Ziel dieser Teilaufgabe ist es, die bereits im Skript kennengelernte **k-Nearest Neighbour-Klassifikation** auf den Ihnen bereits bekannten Datensatz von Pang and Lee (2004) bestehend aus Filmkritiken im Rahmen einer **Sentiment Analysis** anzuwenden. Dabei sollen *positive* von *negativen* Filmkritiken möglichst gut unterschieden werden. Zu diesem Zweck wird in einem zweiten Schritt eine iterative Evaluationsmethode, die **k-Fold Cross-Validation** eingeführt. Damit kann der verwendete Datensatz mehrfach in Trainings- und Testdaten unterteilt und dadurch die Güte des Klassifikators bzw. der Sentiment Analysis besser bestimmt werden, als es mit nur einem Trainingsdaten- und Testdatensatz der Fall wäre.

Zur Klassifikation nach der *k-Nearest-Neighbor*-Methode verwenden wir hier die Funktion *knn(train,test,cl,k)* aus dem R-Package *'class'*. Sie nimmt für den Eingabeparameter *train* die Trainingsdaten, für *test* die zu klassifizierenden Daten (in unserem Fall wieder die Testdaten) und für den Eingabeparameter *cl* die entsprechenden Klassenlabels der Trainingsdaten entgegen. Anschließend werden die *k* ähnlichsten Dokumente aus den Trainingsdaten zu den Testdaten im Eingabeparameter *test* gesucht, um diese nach der Mehrheit der Klassenlabels der Trainingsdokumente zu klassifizieren.

Um die Funktion verwenden zu können, müssen die Textdaten zunächst eingelesen und anschließend aufbereitet werden. Widmen wir uns zunächst dem **Einlesen des**

Datensatzes, der die Filmkritiken enthält. Die Vorgehensweise hierbei unterscheidet sich nicht zu der in Problem Set Nr. 5. Allerdings wird in dieser Aufgabe aus Laufzeitgründen nicht auf den gesamten Datensatz mit den 2000 Filmkritiken zurückgegriffen, da die Rechenzeiten störend für den Lesefluss wären. Stattdessen wurde der Datensatz auf 100 positive und 100 negative Bewertungen beschränkt, anhand derer die Sentiment Analysis mit dem *k-Nearest Neighbour-Klassifikator* veranschaulicht und evaluiert werden soll.
Nachdem die Textdateien eingelesen und die *VCorpus*-Objekte der positiven und negativen Dateien in einem Vektor aneinandergehängt wurden, muss die Klasse des Sentiments zu jeder Textdatei ergänzt werden. Da bekannt ist, dass Dateien aus dem Ordner *"pos"* positive Filmkritiken sowie Dateien aus dem Ordner *"neg"* negative Filmkritiken darstellen, werden dementsprechend die zugehörigen Klassenlabel *positiv* und *negativ* in einem separaten Vektor *sentiment_label* gespeichert. Hierzu wird die Funktion *rep(x,times)* verwendet, die den für *x* übergebenen Wert so oft repliziert wie in *times* angegeben. Dadurch ergeben sich im Vektor *sentiment_label*, entsprechend der zugehörigen Filmkritiken, 100 Einträge des Klassenlabels *"positive"* und 100 Einträge des Klassenlabels *"negative"*.
Führen Sie den folgenden Code aus, um die Daten in R einzulesen, sie für die spätere Erstellung einer DTM aufzubereiten und den verwendeten Daten in einem zusätzlichen Vektor ihr entsprechendes Klassenlabel zuzuordnen.

```
#Separates Einlesen der Datensätze für positiv und negativ bewertete Filme
pos_movie<-DirSource(directory = "./pos")
neg_movie<-DirSource(directory = "./neg")
#Speichern der Daten jeweils in einem VCorpus ("Volatile Corpus")
pos_corpus<-VCorpus(x=pos_movie)
neg_corpus<-VCorpus(x=neg_movie)
#Verwendete Trainings- und Testdaten in einem  Vektor zusammenfassen
#(100 positive und 100 negative Bewertungen)
full_corpus<-c(pos_corpus[401:500],neg_corpus[401:500])
#Den Daten das entsprechende Klassenlabel zuordnen
sentiment_label <- c(rep(x = "positive", times = 100),
                     rep(x = "negative", times = 100))
```

Nachdem die Daten in R eingelesen wurden, wird wie gewohnt die **DTM** erstellt. Auch hierbei erfolgt das Vorgehen analog zu Problem Set Nr. 5. In diesem Fall verwenden wir die invertierte Termfrequenz tf-idf, wobei Sie auch gerne weitere Gewichtungsverfahren zur Klassifikation testen dürfen.
Erzeugen Sie bitte durch Ausführung des Codes die DTM und betrachten Sie die Ausgabe.

```
#Erzeugen der DTM mit tfidf-Gewichtung
DTM_tfidf <- DocumentTermMatrix(full_corpus,
                                control = list(
                                  stopwords = TRUE,
                                  removeNumbers = TRUE,
                                  removePunctuation = TRUE,
                                  tolower = TRUE,
                                  stripWhitespace = TRUE,
                                  stemming = FALSE,
                                  weighting=function(x)
                                   weightTfIdf(x)))
```

```
#Begutachtung des Aufbaus der DTM_tfidf
print(DTM_tfidf)

## <<DocumentTermMatrix (documents: 200, terms: 14469)>>
## Non-/sparse entries: 53201/2840599
## Sparsity           : 98%
## Maximal term length: 54
## Weighting          : term frequency - inverse document frequency
(normalized) (tf-idf)
```

Sie sehen, dass auch für diesen Ausschnitt an 200 Textdokumenten der Wert der Sparsity relativ hoch ist, viele Matrixeinträge also nicht besetzt sind.

Die Funktion *knn()*, welche wir im Folgenden verwenden werden, klassifiziert auf Basis des *euklidischen Distanzmaßes*. Daher müssen mithilfe der aus Problem Set 5 bereits bekannten Funktion *norm_eucl()* die Zeilen der DTM entsprechend normiert werden. Da die Funktion *knn()* - wie schon die Funktion zur Erzeugung des Naive Bayes-Klassifikators - nur auf einer dicht besetzten Matrix ausgeführt werden kann, in der auch Einträge mit Null gespeichert sind, muss auch hier - wie in Aufgabenteil *a)* - das Dateiformat der *DTM_tfidf* mit der Funktion *as.matrix(x)* in eine **dichte Matrix** umgewandelt werden.

A uf gabe: Wenden Sie die Funktion* norm_eucl* auf die *DTM_tfidf* an, um diese zu normieren. Wandeln Sie anschließend die DTM mit dem Befehl *as.matrix()* in eine dichte Matrix um.

```
#Mithilfe der Funktion norm_eucl soll die DTM mit dem euklidischem
#Distanzmaß normiert werden
norm_eucl <- function(x){
   x/apply(x,1,function(x) sum(x^2)^.5)
}

#Wenden Sie die Funktion norm_eucl() auf die DTM_tfidf an und
#speichern Sie das Ergebnis erneut in der Variable DTM_tfidf
#Um die alte DTM_tfidf zu überschreiben
DTM_tfidf<-norm_eucl(x=DTM_tfidf)

#Wenden Sie die Funktion as.matrix() auf die neue DTM_tfidf an und
#speichern Sie das Ergebnis erneut in der Variable DTM_tfidf
#Um die alte DTM_tfidf zu überschreiben
DTM_tfidf = as.matrix(DTM_tfidf)
```

Um die Funktion *knn* anwenden zu können, muss nun definiert werden, welche Daten Trainingsdaten und welche Daten die zu klassifizierenden Daten sind. Diesmal soll die **Zuordnung zu Trainings- und Testdokumenten** zufällig erfolgen. Hierzu verwenden wir die Funktion *createFolds(y,k)* aus dem Package *'caret'*, welche zufällig *k* gleiche Indizes für den in *y* übergebenen Vektor erzeugt. Bei uns ist *y* der Vektor, der die Sentimentlabels der Textdokumente enthält. Die Indizes, die die Funktion *createFolds()* ausgibt, können dann zur Auswahl der Trainings- und Testdaten in die DTM und den Klassenlabelvektor zur eigentlichen Unterteilung der Daten eingesetzt werden. Für die Anzahl *k* an Teilen, in die die Daten aufgeteilt werden, soll hier ein Wert von 3 verwendet werden, um ein Verhältnis von 2:1 der Trainingsdaten zu den Testdaten zu erhalten. Für den Parameter *seed* wird ein

Wert von *1111* festgesetzt. Er sorgt dafür, dass die Ergebnisse aller Nutzer dieses Problem Sets trotz Zufall identisch sind.
Führen Sie bitte den folgenden Code Chunk aus, um aus den aufbereiteten Daten Trainings- und Testdaten zu erhalten.
Hinweis: Es wird eine Besonderheit von R ausgenutzt, die es erlaubt, mit dem "-" Operator all diejenigen Einträge einer Matrix oder eines Vektors zu erhalten, die nicht den eingegebenen Indizes entsprechen.

```
#Laden des benötigten Packages 'caret', um die nachfolgenden Methoden
#verwenden zu können
library(caret)

#Der Seed wird gesetzt um reproduzierbare Aufteilungen zu erzeugen
set.seed(1111)

#Elemente des Sentimentlabel-Vektors zufällig in drei Gruppen aufteilen
indxs<-createFolds(y=sentiment_label,k=3)

#Testdaten gemäß der Aufteilung definieren
testset<-DTM_tfidf[indxs[[1]],]
#Labels der Testdaten
testlabels <- sentiment_label[indxs[[1]]]

#Trainingsdaten gemäß der Aufteilung definieren
trainingset<- DTM_tfidf[-indxs[[1]],]
#Labels der Trainingsdaten
traininglabels <- sentiment_label[-indxs[[1]]]
```

t kann die Funktion *knn()* verwendet werden, um auf Basis der Trainingsdaten eine k-Nearest-Neighbor-Klassifikation der Testdaten durchzuführen. Hierbei braucht es einen möglichst guten Wert für *k*, um zu bestimmen, wie viele Nachbarn bei der Klassifikation mit einbezogen werden sollen. Tests bei der Erstellung dieses Problem Sets haben ergeben, dass ein Wert von *k = 12* gute Ergebnisse für die Sentiment Analysis der vorliegenden Filmkritiken liefert. Daher soll dieser Wert im folgenden Funktionsaufruf verwendet werden. Zusätzlich wird wieder der *seed* gesetzt, um im Fall, dass die *k* ähnlichsten Dokumente zur einen Hälfte der "positiven" und zur anderen Hälfte der "negativen" Klasse angehören, die selbe

Je tz Klassifikation zu erhalten.

A uf gabe: Wenden Sie die Funktion* knn(train,test,cl,k)* auf die eben definierten Trainingsdaten *trainingset* mit ihren Klassenlabels *traininglabels* und auf die eben definierten Testdaten *testset* an. Wählen Sie als Parameter *k* der Anzahl der ähnlichsten Dokumente, die gesucht werden, bitte *k=12*.

```
#Der Seed wird gesetzt um im Falle gleich vieler ähnlicher Dokumente
#verschiedener Klassen die selbe Klassifikation zu erhalten
set.seed(1111)
#Wenden Sie die Funktion knn wie oben beschrieben an
#und speichern Sie das Ergebnis in der Variable knn_predicted_single
knn_predicted_single <- knn(train = trainingset, test =testset,
                            cl =  traininglabels, k = 12)
```

Da es sich um einen weitaus größeren Datensatz handelt als in Teilaufgabe *a)*, ist es hier nicht empfehlenswert, sich die Klassifikationsergebnisse direkt ausgeben zu lassen. Daher wollen wir uns auf die Auswertung der Evaluationsmetriken beschränken.
Führen Sie bitte den folgenden Code Chunk aus, um die *Confusion Matrix* sowie *Precision*, *Recall* und *F-Measure* für die Klasse "positive" zu bestimmen.

```
#Tabelle mit "richtigen" und vorhergesagten Werten ausgeben
knn_confMat_single <- table("Klasse" = testlabels,
                            "Klassifiziert als" = knn_predicted_single)
print(knn_confMat_single)
```

```
##           Klassifiziert als
## Klasse     negative positive
##   negative       21       12
##   positive       10       23
```

```
#Precision der Klasse "positive" berechnen
knn_precision_pos_single <-
knn_confMat_single[2,2]/sum(knn_confMat_single[,2])
print(c("Precision:", knn_precision_pos_single))
```

```
## [1] "Precision:"        "0.657142857142857"
```

```
#Recall der Klasse "positive" berechnen
knn_recall_pos_single <- knn_confMat_single[2,2]/sum(knn_confMat_single[2,])
print(c("Recall:", knn_recall_pos_single))
```

```
## [1] "Recall:"           "0.696969696969697"
```

```
#F-1 Measure mit beta = 1 der Klasse "positive" berechnen
beta <- 1
knn_F1_pos_single <-
((beta^2+1)*knn_precision_pos_single*knn_recall_pos_single)/(beta^2*knn_preci
sion_pos_single+knn_recall_pos_single)
print(c("F1-Measure:", knn_F1_pos_single))
```

```
## [1] "F1-Measure:"       "0.676470588235294"
```

Wie Sie sehen, sind die Evaluationsergebnisse hier weiter entfernt vom Optimum als im Beispiel von Teilaufgabe *a)* bei der Anwendung der Naive Bayes-Klassifikation. Nur etwa zwei Drittel der Textdaten wurden korrekt klassifiziert. Dabei treten falsche Einordnungen laut der Confusion Matrix eher bei den positiven als bei den negativen Dokumenten auf. Dies spiegelt sich auch am Vergleich zwischen *Precision* und *Recall* der Klasse "positive" wieder. Die *Precision* dieser Klasse liegt etwa vier Prozentpunkte unter dem Wert des *Recalls* der selbigen Klasse. Allerdings sind sowohl Precision und Recall als auch die F-Measure für die positiven Dokumente ein ganzes Stück vom optimalen Wert 1 entfernt. Gleichzeitig heißt das nicht, dass der Klassifikator grundlegend schlecht ist, da die Ergebnisse immer noch über der Zufallsrate von 50% liegen.

Um die Güte des Klassifikators weitreichender zu evaluieren, werden Sie nun abschließend in das Verfahren der **k-Fold Cross-Validation** eingeführt:
Bei diesem Verfahren wird für *mehrere* Trainings- und Testsets jeweils die Klassifikation

durchgeführt und anschließend werden die Evaluationsmetriken ausgewertet. Die Idee dahinter ist, den vorhandenen Datensatz zufällig in *k* Teile gleicher Größe aufzutrennen, um von diesen *k* Teilen immer einen Teil als Testdaten zurückzubehalten und die anderen *k-1* Teile als Trainingsdaten für die Klassifikation zu verwenden. Der gesamte Cross-Validation-Prozess wird insgesamt *k*-mal durchgeführt, sodass jeder Teil der Daten genau einmal als Testdaten verwendet wird. Mithilfe dieser Vorgehensweise kann sichergestellt werden, dass die Güte der Ergebnisse nicht mit der zufälligen Wahl der Testdaten zusammenhängt, sondern allein das verwendete Klassifikationsmodell zu den Resultaten führt.

Um dieses Vorgehen mit dem *k-Nearest-Neighbour-Klassifikator* umzusetzen wird im folgenden Code Chunk die Funktion *crossvalidate(DTM,label,numberFolds,k,seed)* implementiert. Diese nimmt eine DTM (*DTM*) mit den entsprechenden Klassenlabels (*label*) entgegen und teilt dabei die DTM in die in *numberFolds* spezifizierte Anzahl an Teilen für die *k-Fold Cross Validation*. Der Paramater *k* steht für die *k* nächsten Nachbarn, die für die *k-Nearest-Neighbour-Klassifikation* herangezogen werden sollen. Da an dieser Stelle die Benennungen ein wenig irreführend sein kann, nochmal kurz zusammengefasst: der Parameter *numberFolds* steht für das *k* aus der *k-Fold Cross-Validation* und der Parameter *k* für das *k* zur *k-Nearest-Neighbour-Klassifikation*. Der *seed* sorgt an dieser Stelle wieder dafür, dass Sie mit Ihrem Code dieselbe Zufallseinteilung erhalten wie wir bei der Erstellung des Problem Sets.
Die zufällige Aufteilung erfolgt auch hier mit der Funktion *createFolds()* analog zu der eben beispielhaft durchgeführten Aufteilung. Für jede mögliche Wahl eines dieser Teile als Testdatensatz wird nun dasselbe Verfahren angewendet, das sie in den vergangenen drei Code Chunks bereits im Rahmen der k-Nearest-Neighbor-Klassifikation kennengelernt haben: Die Trainings- und Testdatensätze mit ihren Sentimentlabels werden definiert, die Funktion *knn()* auf diese angewendet und abschließend werden für diese Klassifikationsergebnisse die bekannten Evaluationsmetriken bestimmt. Letztere stellen den Output der Funktion *crossvalidate()* dar.
Lesen Sie bitte durch die Ausführung des folgenden Codes die eben beschriebene Funktion ein, um Sie verwenden zu können.

```
#Laden der benötigten Packages 'class' und 'caret', um die nachfolgenden
Methoden verwenden zu können
library(class)
library(caret)
#Cross-Validation für k Nearest Neighbors machen
crossvalidate <- function(DTM,label,numberFolds,k,seed){
  #Der Seed wird gesetzt um reproduzierbare Aufteilungen zu erzeugen
  set.seed(seed)
  #createFolds benötigt das package "caret"
  indxs<-createFolds(y=label,k=numberFolds)
  for(i in 1:numberFolds){
    #Trainingsdaten
    trainingset<- DTM[-indxs[[i]],]
    #Labels der Trainingsdaten
    traininglabels <- label[-indxs[[i]]]
    #Testdaten
    testset<-DTM[indxs[[i]],]
    #Labels der Testdaten
```

```
    testlabels <- label[indxs[[i]]]

    ##k Nearest Neighbor Classifier
    #knn ausführen
    knn_predicted <- knn(train = trainingset, test =testset, cl =
trainlabels, k = k)

    #Tabelle mit "richtigen" und vorhergesagten Werten ausgeben
    knn_confMat <- table("Klasse" = testlabels, "Klassifiziert als" =
knn_predicted)
    print(knn_confMat)

    #Precision der Klasse "positive" für jeden Durchlauf berechnen
    knn_precision_pos <- knn_confMat[2,2]/sum(knn_confMat[,2])
    print(c("Precision:", knn_precision_pos))

    #Recall der Klasse "positive"  für jeden Durchlauf berechnen
    knn_recall_pos <- knn_confMat[2,2]/sum(knn_confMat[2,])
    print(c("Recall:", knn_recall_pos))

    #F-1 Measure mit beta = 1 der Klasse "positive" für jeden Durchlauf
berechnen
    beta <- 1
    knn_F1_pos <-
((beta^2+1)*knn_precision_pos*knn_recall_pos)/(beta^2*knn_precision_pos+knn_r
ecall_pos)
    print(c("F1-Measure:", knn_F1_pos))
  }
}
```

Nun kann die Funktion *crossvalidate()* mit den passenden Parametern aufgerufen werden, um die Sentiment Analysis nach der Cross Validation-Methode durchzuführen. Auch hier soll mit derselben Begründung wie im Beispiel oben der Parameter *k = 12* gewählt werden. Für die Anzahl *numberFolds* soll erneut ein Wert von 3 verwendet werden, um ein Verhältnis von 2:1 der Trainingsdaten zu den Testdaten zu erhalten. Für den Parameter *seed* wird wieder der Wert von *1111* festgesetzt.

A uf gabe: Führen Sie die Funktion* crossvalidate()* mit den eben beschriebenen Parameterwerten aus und betrachten Sie das Ergebnis.

```
#Rufen Sie die Funktion crossvalidate() mit den beschriebenen Parametern auf
crossvalidate(DTM = DTM_tfidf, label = sentiment_label, numberFolds = 3, k =
12, seed = 1111)
```

```
##           Klassifiziert als
## Klasse     negative positive
##   negative       22       11
##   positive        9       24
## [1] "Precision:"        "0.685714285714286"
## [1] "Recall:"           "0.727272727272727"
## [1] "F1-Measure:"       "0.705882352941176"
```

```
##           Klassifiziert als
## Klasse     negative positive
##   negative       21       13
##   positive        7       27
## [1] "Precision:" "0.675"
## [1] "Recall:"           "0.794117647058823"
## [1] "F1-Measure:"      "0.72972972972973"
##           Klassifiziert als
## Klasse     negative positive
##   negative       18       15
##   positive        7       26
## [1] "Precision:"        "0.634146341463415"
## [1] "Recall:"           "0.787878787878788"
## [1] "F1-Measure:"       "0.702702702702703"
```

Wie Sie sehen, ergeben sich drei verschiedene *Confusion Matrices* zusammen mit den dazugehörigen Evaluationsmetriken. In allen drei Testsets wird ein Großteil der Filmkritiken korrekt klassifiziert. Allerdings zeigen die Ergebnisse, dass es noch Raum zur Optimierung gibt. Sie können es gerne als Herausforderung betrachten, diese Werte zu überbieten, indem Sie verschiedene Klassifikatoren und dazugehörige Parameterkombinationen austesten. Viel Spaß dabei!

Award: Textklassifikation

Toll! Sie haben das gesamte Problem Set gemeistert! Sie sind nun in der Lage, unstrukturierte Textdaten zu klassifizieren, beispielsweise im Rahmen einer Sentiment Analysis! Ihr Wissen im Text Mining-Bereich ist nun schon soweit fortgeschritten, dass Sie das selbstständige und souveräne Arbeiten mit Textdaten beherrschen.

Anmerkungen:
** Dieses Problem Set wurde mit dem Package 'RTutor' von Prof. Dr. Sebastian Kranz von der Universität Ulm erstellt. Weitere Informationen zu diesem Package und den RTutor-Problem Sets finden Sie auf der GitHub-Seite* https://github.com/skranz/RTutor.
** Informationen zu den einzelnen Befehlen sind - falls nicht anders angegeben - den jeweiligen R-Hilfeseiten entnommen. Dort finden Sie auch weitere Informationen zu den einzelnen Befehlen. * Inspirationen zu den Aufgaben kamen außerdem unter anderem aus Pang et. al (2002), Pang und Lee(2004) und https://rstudio-pubs-static.s3.amazonaws.com/132792_864e3813b0ec47cb95c7e1e2e2ad83e7.html.* Alle Quellenangaben finden Sie im Literaturverzeichnis (Exercise LV).*

Exercise LV -- Literaturverzeichnis

Packages:

- David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2015). e1071: Misc Functions of the Department of Statistics, Probability

Theory Group (Formerly: E1071), TU Wien. R package version 1.6-7, URL: https://CRAN.R-project.org/package=e1071

- Ingo Feinerer, Kurt Hornik, and David Meyer (2008). Text Mining Infrastructure in R. Journal of Statistical Software 25(5): 1-54. URL: http://www.jstatsoft.org/v25/i05/.
- Kranz, Sebastian (2015): RTutor: R problem sets with automatic test of solution and hints. R package version 2015.12.16.
- Kross, Sean; Carchedi, Nick; Bauer, Bill; Grdina, Gina (2016): swirl: Learn R, in R. R package version 2.4.2, URL: https://CRAN.R-project.org/package=swirl.
- R Core Team (2016): R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, URL: https://www.R-project.org/.
- Stephens, Jeremy (2016): yaml: Methods to Convert R Data to YAML and Back. R package version 2.1.14, URL: https://CRAN.R-project.org/package=yaml.
- Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

Packagebeschreibungen/Hilfeseiten:

- tm (2017): Package 'tm', URL: https://cran.r-project.org/web/packages/tm/tm.pdf
- class (2015): Package 'class', URL: https://cran.r-project.org/web/packages/class/class.pdf
- e1071 (2017): Package 'tm', URL: https://cran.r-project.org/web/packages/e1071/e1071.pdf

Buch-, Paper- und Onlinequellen:

- Pang, B., and L. Lee (2004): A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. Proceedings of the 42nd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics

- C. X. Zhai und S. Massung (2016): Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining. Association for Computing Machinery und Morgan & Claypool Publishers