Important commands

Mateusz Żółtak

Rstudio / R

- To run your code:
  - set cursor in the right line and hit ***CTRL+ENTER*** (cursor will be automatically moved to the next line)
  - or highlight code you want to run and hit ***CTRL+ENTER***
- To change working directory (directory in which stylo package will save all graphs, files, etc.):
  - go to the right directory in the *Files* pane (bottom-right part of the window)
  - choose *Session->Set Working Directory->To Files Pane Location* from the menu.
- To read a table saved by the stylo package in a .txt file

```
myTable = read.table('pathToFile', header = TRUE, sep = " ", stringsAsF
actors = FALSE)
```

- To save a table (e.g. features frequency table) to an Excel-compatible CSV file

```
write.csv2('pathToFile.csv', row.names = FALSE, encoding = 'Windows-125
2')
```

Initialization

```
# Load required libraries
library('styloWorkshop')

# Connect to our texts database
db.connect()

# Fetch information about all available texts
texts = get.texts()
```

Corpora preparation

(assuming that texts meta data were fetched to the *texts* variable)

- See available meta data for first 10 texts

```
texts
```

- See frequency of values for a given meta data attribute (here docsrc)

```
texts %>%
  get.values('docsrc')
```

- Choose SPORTZTG texts from 2013-08-01 and store them in the “sportztg” variable

```
sportztg = texts %>%
  filter(docsrc %in% 'SPORTZTG' & date %in% '2013-03-12')
```

- Derive a year variable

```
textsWithYear = texts %>%
  mutate(year = substr(date, 1, 4))
```

- Write the whole British Fiction corpora to the “corpus” directory grouping by author and title

```
texts %>%
  filter(source == 'British Fiction') %>%
  write.corpus('author', 'title', 'corpus', 0.9)
```

- Complex example:
  - Filter all AMC data
  - derive a year variable
  - Write 70% sample to the primary_set directory grouping them by docsrc and year
  - Write the rest to the secondary_set directory grouping them by docsrc and year
  - In both cases limit single file size to ~5 MB

```
texts %>%
  filter(source %in% 'amc') %>%
  mutate(year = substr(date, 1, 4)) %>%
  write.corpus('docsrc', 'year', 'primary_set', 0.7, limit = 5*10^6) %>
%
  write.corpus('docsrc', 'year', 'secondary_set', limit = 5*10^6)
```

Features frequency table

Features parsing

Parse texts from directory “corpus” into bi-grams of words neglecting characters case and assuming they are in German.

```
features = load.corpus.and.parse(
  corpus.dir = 'corpus',
  language = 'German',
  features = 'w',
  ngram.size = 2,
  preserve.case = FALSE
)
```

Feautures frequency table computation

Compute frequency table using relative frequencies from parsed features

```
freqTab = count.freqs(features, relative = TRUE)
```

Features frequency table manipulation

(assuming that *freqTab* denotes a variable with computed features frequency table)

- Inspect features frequency table
  - see 20 more frequent features frequencies

    ```
    freqTab[, 1:20]
    ```

  - see 20 less frequent features frequencies

    ```
    fNumber = ncol(freqTab)
    freqTab[, (fNumber - 20):fNumber]
    ```

  - see frequent features frequencies from 200 to 210

    ```
    freqTab[, 200:210]
    ```

  - see frequency of a given feature

    ```
    freqTab[, 'my Feature']
    ```

- Limit features frequency table to a given number of features
  - take 200 most frequent ones

    ```
    freqTab[, 1:200]
    ```

  - take 50 less frequent one

    ```
    fNumber = ncol(freqTab)
    freqTab[, (fNumber - 50):fNumber]
    ```

  - take ones from 50 to 250

    ```
    freqTab[, 50:250]
    ```

- (Culling) Drop features present in less then 30% of texts (limit number of features in your table first if you don't want to wait ages!)

  ```
  freqTabAdj = perform.culling(freqTab, 30)
  ```

- Delete given features from the features frequency table

  ```
  freqTabAdj = delete.stop.words(freqTab, c('sampleFeature', 'my Feature'
  ))
  ```

  - Delete pronouns for a given language from the features frequency table (makes sense only if you are using single words as a feature)

    ```
    freqTabAdj = delete.stop.words(freqTab, stylo.pronouns('English'))
    ```

Exploratory analysis

(assuming that *freqTabAdj* variable holds final features frequency table)

Cluster analysis

```
results = stylo(
  gui = FALSE,
  frequencies = freqTabAdj,
  mfw.min = ncol(freqTabAdj), mfw.max = ncol(freqTabAdj),
  analysis.type = 'CA',
  distance.measure = 'delta', # distance metric delta/argamon/eder/simple/manhattan/canberra/euclidean/cosine
  linkage = 'ward' # method of linking texts into tree ward/nj/single/complete/average/mcquitty/median/centroid
)
```

- accessing normalized (z-scale) features frequencies:

```
as.data.frame(results$table.with.all.zscores)
```

- accessing computed distances:

```
as.data.frame(results$distance.table)
```

Multidimensional scaling

```
results = stylo(
  gui = FALSE,
  frequencies = freqTabAdj,
  mfw.min = ncol(freqTabAdj), mfw.max = ncol(freqTabAdj),
  analysis.type = 'MDS',
  distance.measure = 'delta', # distance metric delta/argamon/eder/simple/manhattan/canberra/euclidean/cosine
  text.id.on.graphs = 'both',
  label.offset = 4 # adjust if labels intersect points on the plot
)
```

- accessing normalized (z-scale) features frequencies:

```
as.data.frame(results$table.with.all.zscores)
```

- accessing computed distances:

```
as.data.frame(results$distance.table)
```

Principal Component Analysis

```
results = stylo(
  gui = FALSE,
  frequencies = freqTabAdj,
  mfw.min = ncol(freqTabAdj),
  mfw.max = ncol(freqTabAdj),
  analysis.type = 'PCV', # PCV/PCR
  pca.visual.flavour = 'classic', # classic/loadings/technical/symbols
  text.id.on.graphs = 'both',
  label.offset = 4 # adjust if labels intersect points on the plot
)
```

- to restore correct graphs drawing after using *pca.visual.flavour = 'technical'*:

```
par(mfrow = c(1, 1))
```

Classification

(assuming that *trainFreqTab* variable holds features frequency table for the training set and *testFreqTab* holds one for the test set)

Delta

```
results = classify(
  gui = FALSE,
  training.frequencies = trainFreqTab,
  test.frequencies = testFreqTab,
  mfw.min = ncol(trainFreqTab),
  mfw.max = ncol(trainFreqTab),
  classification.method = 'delta',
  distance.measure = 'delta', # distance metric delta/argamon/eder/simple/ma
nhattan/canberra/euclidean/cosine
)
```

k-Nearest Neighbors

```
results = classify(
  gui = FALSE,
  training.frequencies = trainFreqTab,
  test.frequencies = testFreqTab,
  mfw.min = ncol(trainFreqTab),
  mfw.max = ncol(trainFreqTab),
  classification.method = 'knn',
  k.value = 1 # number of neighbors taken into account
)
```

SVM

```
results = classify(
  gui = FALSE,
  training.frequencies = trainFreqTab,
  test.frequencies = testFreqTab,
  mfw.min = ncol(trainFreqTab),
  mfw.max = ncol(trainFreqTab),
  classification.method = 'svm',
  svm.kernel = 'linear', # linear/polynomial/radial
  svm.degree = 3,
  svm.coef0 = 0,
  svm.cost = 1
)
```

Naive Bayes

```
results = classify(
  gui = FALSE,
  training.frequencies = trainFreqTab,
  test.frequencies = testFreqTab,
  mfw.min = ncol(trainFreqTab),
  mfw.max = ncol(trainFreqTab),
  classification.method = 'naivebayes'
)
```

Nearest Shrunken Centroid

```
results = classify(
  gui = FALSE,
  training.frequencies = trainFreqTab,
  test.frequencies = testFreqTab,
  mfw.min = ncol(trainFreqTab),
  mfw.max = ncol(trainFreqTab),
  classification.method = 'nsc'
)
```