doc2vec with R

Brussels

Jan Wijffels

BNOSAC - jwijffels@bnosac.be

www.bnosac.be

Overzicht

1. doc2vec
2. example 1
3. example 2
4. BNOSAC R consultancy

bnosac

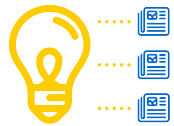
open analytical helpers

doc2vec

doc2vec

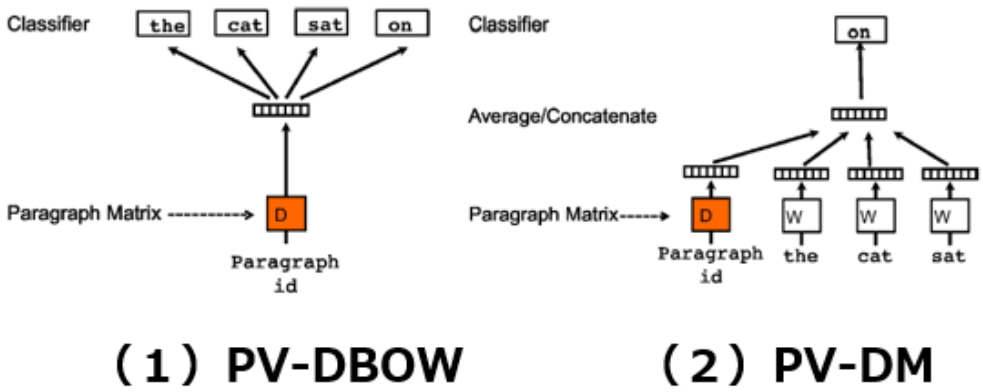
- doc2vec is an algorithm by Mikolov et al. (2014) which allows to **map paragraphs** of texts **to numbers** which are called embeddings
  - the algorithm is explained in the paper 'Distributed Representations of Sentences and Documents' available at `https://arxiv.org/pdf/1405.4053.pdf`
  - the algorithm is an extension of the word2vec algorithm
  - these models are known as paragraph vector models but are also commonly labelled as doc2vec
- For R users you can compute Paragraph Vector models using the **doc2vec** R package
- Note that paragraph vector algorithms (also known as doc2vec) are different than the word2vec algorithm
  - It is good to have an understanding of wordvectors to understand the doc2vec algorithm.
  - `https://www.bnosac.be/index.php/blog/100-word2vec-in-r` showcases the *word2vec* R package

- Common **applications** of document vectors are:
  - Use the embeddings to find similar documents, sentences or words
  - Use the vectors as input to predictive models to do specific NLP tasks - typically classification or regression modelling (glmnet / SVM)
  - Text summarisation alongside R package *textrank*
  - Perform text clustering by applying *DBSCAN* on the embeddings to find document topics



In these slides I cover *R package doc2vec* of which I am the R package author. The package uses a C++ implementation of the algorithm, following closely the original C implementation available here

- **Document vectors** are a set of numbers assigned to each document - commonly applied to paragraphs or a set of sentences
- R package doc2vec learns such vectors on your own texts to obtain
  - **Distributed Memory** paragraph vectors (**PV-DM**)
    - PV-DM learns an embedding vector which in combination with word vectors predicts the next word in a sentence
  - **Distributed Bag Of Words** paragraph vectors (**PV-DBOW**)
    - PV-DBOW learns an embedding vector which allows to predict a set of words in the neighbourhood of one another



- Just install the following NLP packages from CRAN for showcasing doc2vec.

```
install.packages("doc2vec")
install.packages("udpipe")
install.packages("word2vec")
```

Make sure you have the latest versions (doc2vec >= 0.1, word2vec >= 0.3.3, udpipe >= 0.8.5)

bnosac

open analytical helpers

example 1

Example - train & similarities

In this example, we will showcase how doc2vec can be used as a similarity tool

- Extract all R packages on CRAN
- Create a data.frame with columns *doc_id* and *text*
  - The text column: combine the R package title and R package description
  - The R package name is used as a document identifier

```
library(tools)
db      <- CRAN_package_db()
db      <- data.frame(doc_id = sprintf("%s_", db$Package),
                      text   = paste(db$Title, db$Description, sep = " "),
                      stringsAsFactors = FALSE)
str(db)
```

```
'data.frame':   16868 obs. of  2 variables:
 $ doc_id: chr  "A3_" "aaSEA_" "AATtools_" "ABACUS_" ...
 $ text  : chr  "Accurate, Adaptable, and Accessible Error Metrics for Predictive\nModels Supplies tools for tabulating and anal"| __tr
```

Notes on data preparation before modelling

- Make sure the document identifier (doc_id) is not a word present in the text (we added an underscore to it)
- R package doc2vec learns the document as well as the word vectors. For this it assumes words are separated by spaces.
  - We standardise text below by lowercasing, putting all text in ASCII, keeping only alphanumeric text and trimming multiple white spaces
  - Below, this is done using function txt_clean_word2vec from the word2vec package
- docvec requires text to be shorter than 1000 words
  - Below, this filtering is done using function txt_count from the udpipe package

```
library(word2vec)
library(udpipe)
db$text   <- txt_clean_word2vec(db$text, ascii = TRUE, alpha = TRUE, tolower = TRUE, trim = TRUE)
db$nwords <- txt_count(db$text, pattern = " ")
db        <- subset(db, nwords < 1000)
```

Build doc2vec model

- Build a paragraph vector model
  - train the PV-DBOW algorithm in parallel on 2 threads
  - dimension of the word/paragraph vectors: 150
  - words should occur at least 3 times
  - loop 10 times over the data, updating the embeddings
  - default window with DBOW is 10 using negative sampling

```
library(doc2vec)
model <- paragraph2vec(x = db, type = "PV-DBOW",
                       dim = 150, iter = 10, min_count = 3, lr = 0.05, threads = 2)
```

- Each word and each document has an embedding with 150 columns

```
embedding <- as.matrix(model, which = "words")
embedding <- as.matrix(model, which = "docs")
dim(embedding)
```

```
[1] 16840   150
```

```
pkgs      <- c("ggplot2", "glmnet", "udpipe")
embedding[sprintf("%s_", pkgs), 1:5]
```

```
               [,1]        [,2]        [,3]        [,4]        [,5]
ggplot2_  0.12813878 -0.01083748  0.17510046 -0.08056564  0.04552916
glmnet_  -0.03324074  0.07839789 -0.05853574 -0.10381376  0.10422201
udpipe_   0.03802395 -0.02050885 -0.10659005  0.07485859 -0.01740843
```

```
embedding <- predict(model, newdata = c("linear", "regression", "topic"), type = "embedding")
embedding[, 1:7]
```

```
          [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

Use cases

The learned embeddings can be used to find similarities by calculating inner products between document/word embeddings. Examples:

1. Look for R packages which are similar to the following words **(word2doc)**

```
similar <- predict(model,
                   newdata = c("embedding", "email"),
                   type = "nearest", which = "word2doc", top_n = 7)
similar
```

```
[[1]]
      term1          term2 similarity rank
1 embedding           tpe_  0.5992507    1
2 embedding    dyndimred_  0.5963292    2
3 embedding    sleepwalk_  0.5810528    3
4 embedding           loe_  0.5795738    4
5 embedding partitionMap_  0.5730132    5
6 embedding          tsne_  0.5663707    6
7 embedding         hydra_  0.5650934    7

[[2]]
  term1                      term2 similarity rank
1 email                      mail_  0.6888300    1
2 email                      edeR_  0.6674483    2
3 email                 sendmailR_  0.6285636    3
4 email                   emayili_  0.6235993    4
5 email                    whoami_  0.6197692    5
6 email                     mailR_  0.6156900    6
7 email paws.customer.engagement_  0.6043246    7
```

2 Look for R packages which are similar to the following sentences **(sent2doc)**

```
sentences <- c("automate processes", "linear model")
sentences <- setNames(sentences, sentences)
sentences <- strsplit(sentences, split = " ")
similar   <- predict(model,
                     newdata = sentences,
                     type = "nearest", which = "sent2doc", top_n = 7)
similar
```

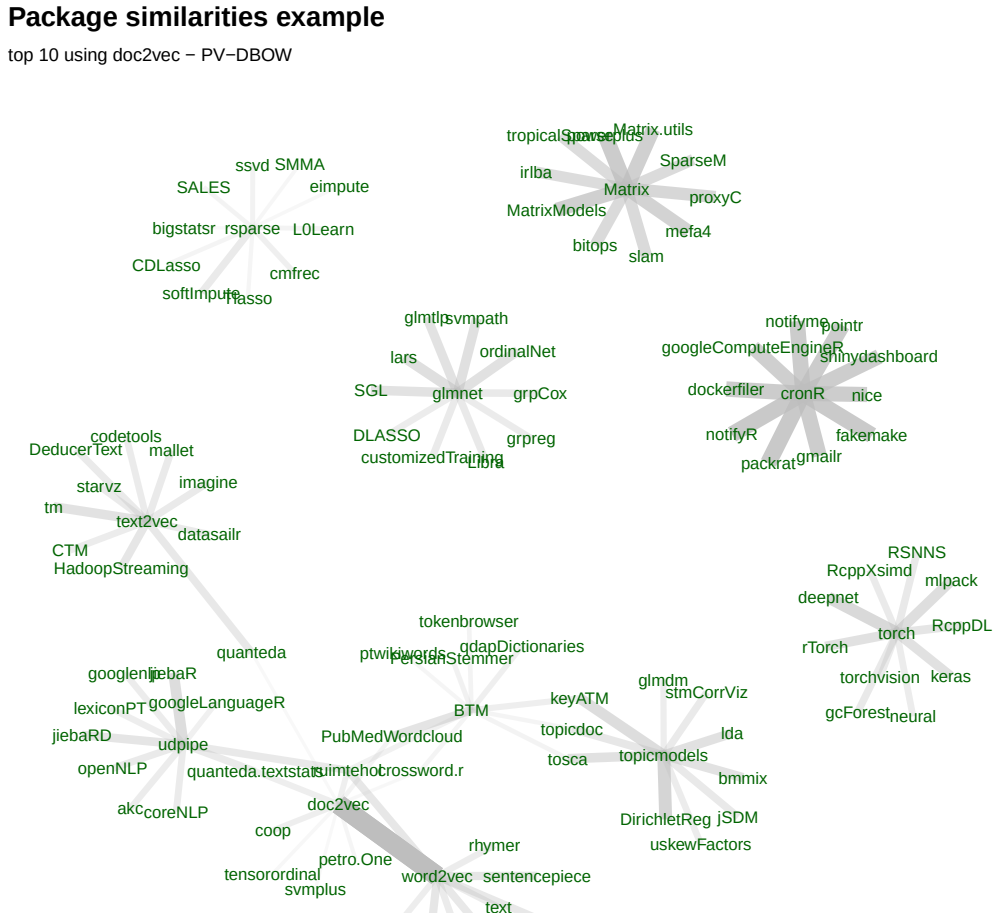
```
$`automate processes`
               term1          term2 similarity rank
1 automate processes taskscheduleR_  0.6681057    1
2 automate processes          SAPP_  0.6550066    2
3 automate processes      geotools_  0.5920188    3
4 automate processes          mmpp_  0.5734100    4
5 automate processes     codetools_  0.5643786    5
6 automate processes          nice_  0.5603570    6
7 automate processes            ps_  0.5598558    7

$`linear model`
         term1         term2 similarity rank
1 linear model         BGLR_  0.8396766    1
2 linear model          BLR_  0.8119417    2
3 linear model         nlme_  0.7823018    3
4 linear model        dynlm_  0.7815973    4
5 linear model          ljr_  0.7556728    5
6 linear model MixedPoisson_  0.7549467    6
7 linear model     MCMCglmm_  0.7528549    7
```

3 Look which R packages are similar to the following set of packages and visualise these similarities **(doc2doc)**

```
pkgs    <- c("glmnet", "udpipe", "word2vec", "doc2vec", "text2vec", "topicmodels", "BTM", "rsparse", "Matrix", "torch", "cronR")
similar <- predict(model, newdata = sprintf("%s_", pkgs), type = "nearest", which = "doc2doc", top_n = 10)
```

```
library(textplot)
library(ggraph)
x <- do.call(rbind, similar)
x <- data.frame(term1 = gsub("_", "", x$term1), term2 = gsub("_", "", x$term2), cooc = x$similarity * 100)
textplot_cooccurrence(x = x, top_n = +Inf, title = "Package similarities example", subtitle = "top 10 using doc2vec - PV-DBOW")
```



Save

- We can write the model to disk and read the model back in again

```
write.paragraph2vec(model, "mymodel.bin")
model   <- read.paragraph2vec("mymodel.bin")
```

- And use it again

```
similar <- predict(model, newdata = c("mixed", "lm"),
                   type = "nearest", which = "word2word", top_n = 7)
similar
```

```
[[1]]
  term1   term2 similarity rank
1 mixed  linear  0.7319272    1
2 mixed   gemma  0.6269846    2
3 mixed effects  0.5966737    3
4 mixed densely  0.5920510    4
5 mixed    reml  0.5897493    5
6 mixed  models  0.5858206    6
7 mixed    lmms  0.5790625    7

[[2]]
  term1    term2 similarity rank
1    lm      glm  0.7063262    1
2    lm      rlm  0.6972583    2
3    lm  survreg  0.6484824    3
4    lm    coxph  0.6309385    4
5    lm quantreg  0.6010337    5
6    lm    ivreg  0.5986212    6
7    lm speedglm  0.5706592    7
```

bnosac

open analytical helpers

example 2

Use the embeddings as predictive features

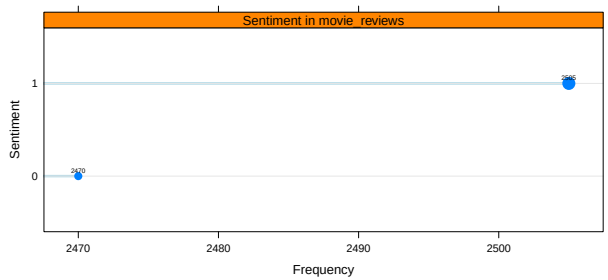
- Use the document embeddings for predictive modelling purposes
- **Classical example: Sentiment scoring based on text**

Example data:

```
set.seed(123456789)
library(text2vec)
data("movie_review", package = "text2vec")
x   <- data.frame(doc_id    = sprintf("movie_%s", movie_review$id),
                  text      = txt_clean_word2vec(movie_review$review),
                  train     = ifelse(runif(nrow(movie_review)) < 0.7, "train", "test"),
                  sentiment = movie_review$sentiment,
                  stringsAsFactors = FALSE)
x   <- subset(x, txt_count(text, pattern = " ") < 1000)
idx <- which(x$train == "train")
str(x)
```

```
'data.frame':   4975 obs. of  4 variables:
 $ doc_id   : chr  "movie_5814_8" "movie_2381_9" "movie_7759_3" "movie_3630_4" ...
 $ text     : chr  "with all this stuff going down at the moment with mj i ve started listening to his music watching the odd docum"| _
 $ train    : chr  "train" "train" "train" "test" ...
 $ sentiment: int  1 1 0 0 1 1 0 0 0 1 ...
```

```
library(textplot)
textplot_bar(table(x$sentiment), xlab = "Frequency", ylab = "Sentiment", panel = "Sentiment in movie_reviews", col.panel = "#FF8500")
```



- **Build the doc2vec model, extract the document embeddings**

```
library(doc2vec)
model     <- paragraph2vec(x = x, type = "PV-DBOW", dim = 100, iter = 10, min_count = 1, lr = 0.05, threads = 4)
embedding <- as.matrix(model, which = "docs")
dim(embedding)
```

```
[1] 4975  100
```

- **Combine the embeddings with the training dataset**
  - aligning the rownames of the embedding matrix with the doc_id
  - using dtm_align from the udpipe package
  - making sure the predictors and the target is in the same order
- **Train a penalised ridge regression**

```
library(glmnet)
library(udpipe)
traindata       <- dtm_align(x = embedding,
                             y = setNames(x$sentiment, x$doc_id)[idx])
model           <- cv.glmnet(x = traindata$x, y = traindata$y, family = "binomial", nfolds = 5, alpha = 1)
```

- Score the model + add the scoring column to the existing data.frame for evaluation later on

```
scores          <- predict(model, embedding, type = "response", s = "lambda.1se")
scores          <- drop(scores)
x$score_doc2vec <- txt_recode(x$doc_id, from = names(scores), scores, na.rm = TRUE)
```

Done!

- We can also **use another predictive model** using the doc2vec features
- Below: SVM using radial basis kernel with as input the doc2vec embeddings

```
library(e1071)
model_optim     <- tune.svm(x = traindata$x, y = factor(traindata$y),
                            type = "C-classification", kernel = "radial",
                            gamma = c(0.001, 0.005, 0.01, 0.1), cost = c(0.1, 1, 1.5),
                            tunecontrol = tune.control(sampling = "cross", cross = 3))
model_optim$best.parameters
```

```
  gamma cost
5 0.001    1
```

```
model           <- svm(x = traindata$x, y = traindata$y,
                       type = "C-classification", kernel = "radial",
                       gamma = 0.001, cost = 1.5,
                       probability = TRUE)
scores          <- predict(model, embedding, probability = TRUE)
scores          <- data.frame(doc_id = names(scores), score = attr(scores, which = "probabilities")[, "1"])
x$score_svm     <- txt_recode(x$doc_id, from = scores$doc_id, to = scores$score, na.rm = TRUE)
```

How does this compare to a traditional approach?

- Compare to just using the term frequency counts
- Using the udpipe package for creating the document-term-matrix and tfidf features but many other R packages do this as well

```
dtm <- document_term_frequencies(x = x$text, document = x$doc_id, split = " ")
dtm <- document_term_frequencies_statistics(dtm)
dtm <- document_term_matrix(dtm, weight = "tf_idf")
dim(dtm)
```

```
[1]  4975 38858
```

- traditional penalised regression on the term frequency counts weighted by inverse document frequency

```
traindata     <- dtm_align(x = dtm,
                           y = setNames(x$sentiment, x$doc_id)[idx])
model         <- cv.glmnet(x = traindata$x, y = traindata$y, family = "binomial", nfolds = 5, alpha = 1)
```

```
scores        <- predict(model, dtm, type = "response", s = "lambda.1se")
scores        <- drop(scores)
x$score_tfidf <- txt_recode(x$doc_id, from = names(scores), scores, na.rm = TRUE)
```

Can we combine the doc2vec embedding with the term frequencies?

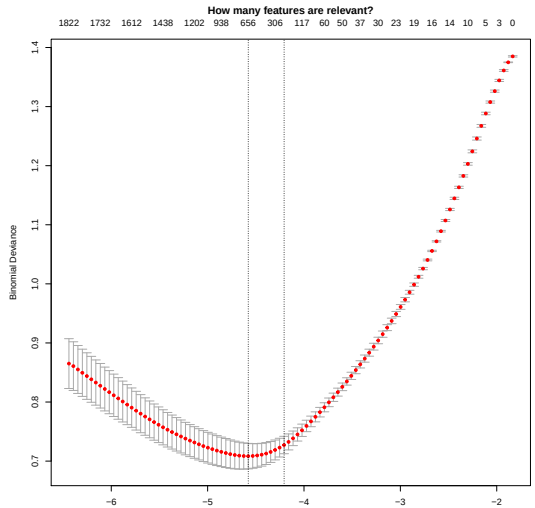
- combine sparse dtm term frequency count matrix with dense embedding matrix, using dtm_cbind from the udpipe package

```
dtm             <- dtm_cbind(dtm, embedding)
dim(dtm)
```

```
[1]  4975 38958
```

- build the model on the term frequencies + the document embeddings

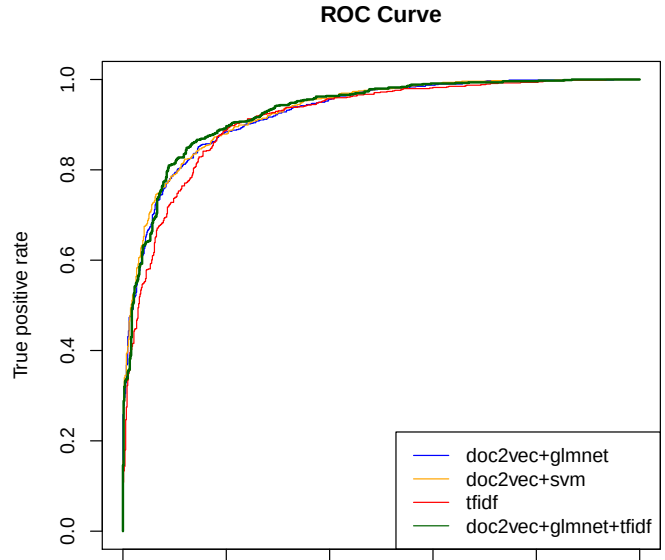
```
traindata       <- dtm_align(x = dtm,
                             y = setNames(x$sentiment, x$doc_id)[idx])
model           <- cv.glmnet(x = traindata$x, y = traindata$y, family = "binomial", nfolds = 5, alpha = 1)
scores          <- predict(model, dtm, type = "response", s = "lambda.1se")
scores          <- drop(scores)
x$score_combined <- txt_recode(x$doc_id, from = names(scores), scores, na.rm = TRUE)
plot(model, main = "How many features are relevant?\n")
```



Evaluation

- See how these models compare on the holdout test dataset
- doc2vec models provide better performance for larger volumes of texts (generally a few 100000 text documents), but for this small set, there is already a small performance gain

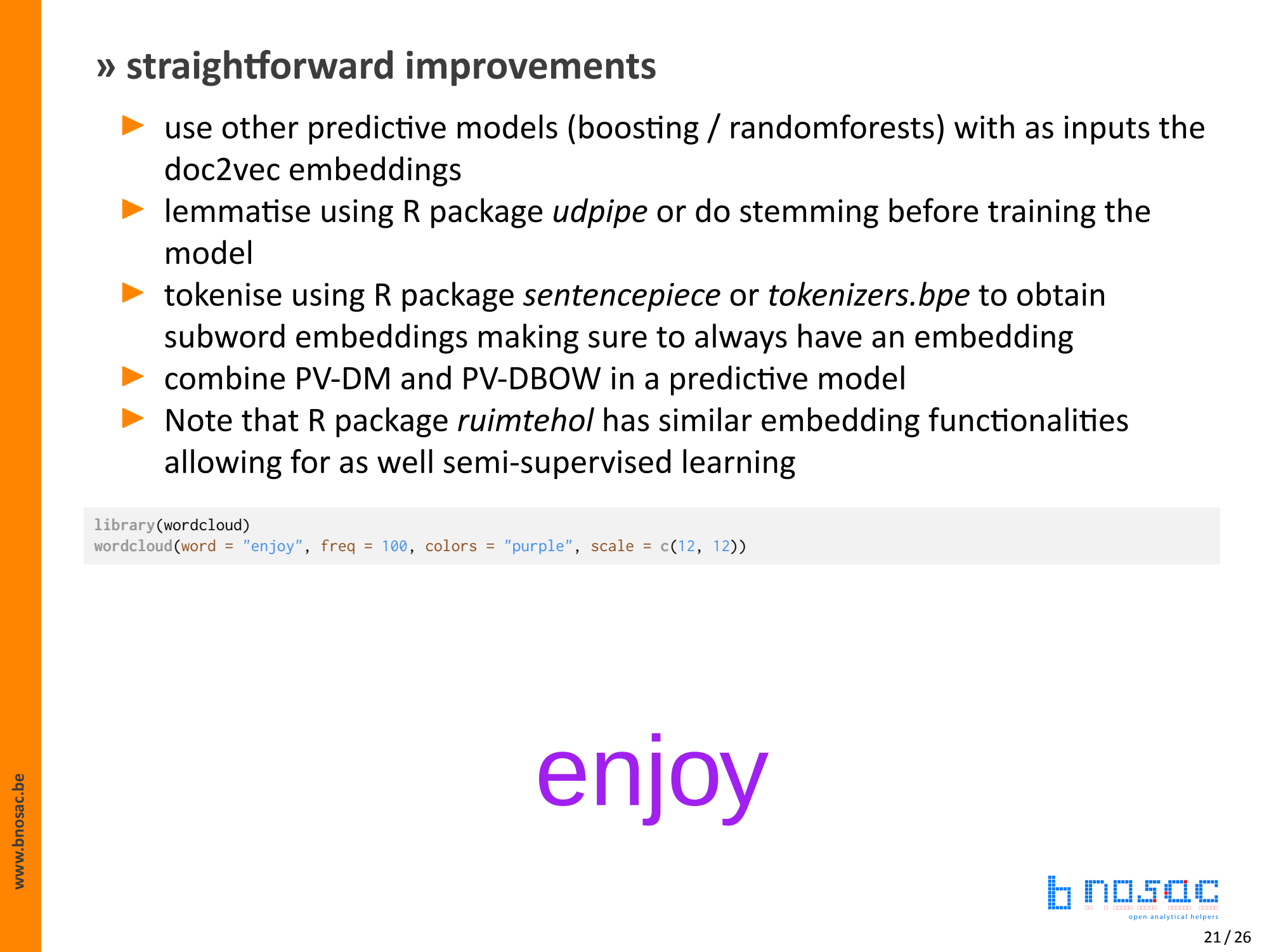
```
library(ROCR)
evaluation    <- subset(x, train == "test")
pred_doc2vec  <- prediction(evaluation$score_doc2vec, evaluation$sentiment)
pred_svm      <- prediction(evaluation$score_svm, evaluation$sentiment)
pred_tfidf    <- prediction(evaluation$score_tfidf, evaluation$sentiment)
pred_combined <- prediction(evaluation$score_combined, evaluation$sentiment)
plot(performance(pred_doc2vec,  "tpr", "fpr"), col = "blue", main = "ROC Curve")
plot(performance(pred_svm,      "tpr", "fpr"), col = "orange", add = TRUE)
plot(performance(pred_tfidf,    "tpr", "fpr"), col = "red", add = TRUE)
plot(performance(pred_combined, "tpr", "fpr"), col = "darkgreen", add = TRUE, lwd = 2)
legend(legend = c("doc2vec+glmnet", "doc2vec+svm", "tfidf", "doc2vec+glmnet+tfidf"),
       col = c("blue", "orange", "red", "darkgreen"), x = "bottomright", lwd = 1)
```



» straightforward improvements

- use other predictive models (boosting / randomforests) with as inputs the doc2vec embeddings
- lemmatise using R package *udpipe* or do stemming before training the model
- tokenise using R package *sentencepiece* or *tokenizers.bpe* to obtain subword embeddings making sure to always have an embedding
- combine PV-DM and PV-DBOW in a predictive model
- Note that R package *ruimtehol* has similar embedding functionalities allowing for as well semi-supervised learning

```
library(wordcloud)
wordcloud(word = "enjoy", freq = 100, colors = "purple", scale = c(12, 12))
```



bnosac

open analytical helpers

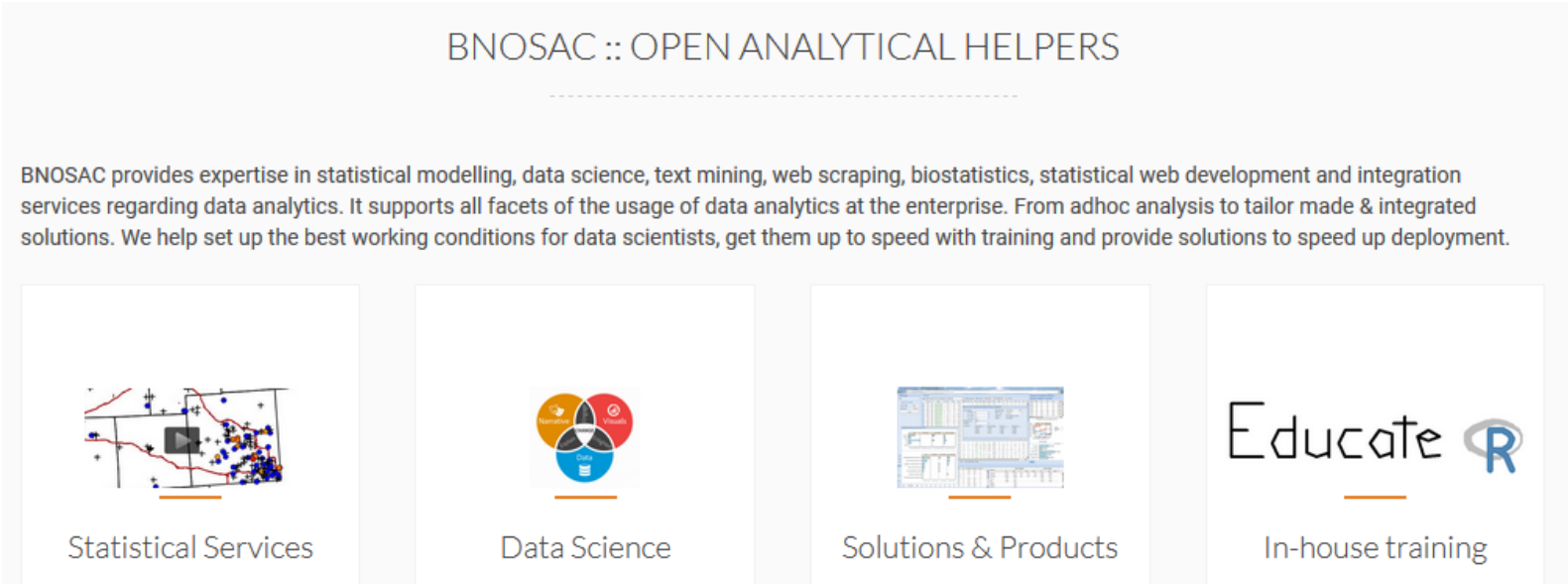
BNOSAC R consultancy

www.bnosac.be

www.bnosac.be: open source analytics experts

Providing consultancy services in **open source analytical engineering**

- Support for R / Oracle R Enterprise / Microsoft R / PostgreSQL / Python / ExtJS / Bigdata / Real-time processing / …
- Expertise in predictive data mining, biostatistics, geostats, R + Python programming, GUI building, artificial intelligence, process automation, analytical web development
- R/Python implementations, application maintenance & training/consulting
- RStudio Server Pro / Shiny Pro / RStudio Connect reseller
- Hosting CRAN at www.datatailor.be
- Contributing to the R community with R packages / R training



R support by BNOSAC

WE SUPPORT R USAGE AT YOUR ENTERPRISE

BNOSAC gives support for all facets of the usage of R at your enterprise. We help set up the best working conditions for R users, get them up to speed with training and provide solutions to speed up deployment. For small startups to big corporations or governments.

RStudio Pro & Shiny Pro reseller

Oracle R Enterprise support

Continuous integration & deployment for R users

Inhouse R training & package development

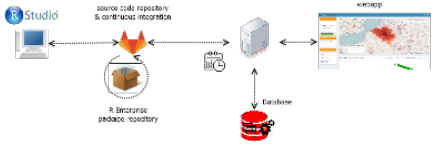
Training by BNOSAC: www.bnosac.be/training

R Training

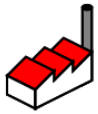
Data viz &
data munging

2a. Common data manipulation & Programming in R
2b. Visualisation with R
2c. Reporting with R
2d. Data connectivity with R

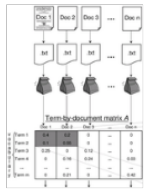
1. R for starters



Deploying



4a. Creating R packages and R repositories
4b. git/SVN + continuous integration with R
4c. Managing R processes
4d. Integration of R in web applications
4e. Shiny
4f. Big data analytics with R (Spark, HAWQ & PL/R)



Analytics

3a. Statistical Machine learning with R
3b. Text mining with R
3c. Applied Spatial Analysis with R
3d. Computer Vision with R and Python

bnosac
open analytical helpers

www.bnosac.be

Need support, send a message at www.bnosac.be/index.php/contact/get-in-touch

OFFICE ADDRESS

The bnosac office is located in

- la Lustrerie Business Center - http://www.lalustrerie.be
- Paleizenstraat 153, 1030 Brussels, Belgium
- close to Gare du Nord in Brussels
- email: info [at] bnosac [dot] be