Neural Text Models with R package ruimtehol

Jan Wijffels

Abstract

Ruimtehol is a comprehensive R package which wraps the StarSpace C++ library (https://github.com/facebookresearch/StarSpace). Starspace is a neural natural language modelling toolkit which allows you to calculate word, sentence, article, document, webpage, link and entity 'embeddings'. By using the 'embeddings', you can perform text based multi-label classification, find similarities between texts and categories, do collaborative-filtering based recommendation as well as content-based recommendation, find out relations between entities, calculate graph 'embeddings' as well as perform semi-supervised learning and multi-task learning on plain text. The techniques are explained in detail in the paper: *StarSpace: Embed All The Things!*, available at (https://arxiv.org/abs/1709.03856).

Keywords: Starspace, NLP, embed, embedding, neural, text.

1. Ground control to ruimtehol

1.1. Overview

The ruimtehol R package which wraps the StarSpace C++ library (https://github.com/facebookresearch/StarSpace), focussing on building and utilising embedding models for natural language. It allows you to do the following Natural Language Processing tasks

- Text classification
- Learning word, sentence or document level embeddings
- Finding sentence or document similarity
- Ranking web documents
- Content-based recommendation (e.g. recommend text/music based on the content)
- Collaborative filtering based recommendation (e.g. recommend text/music based on interest)
- Identification of entity relationships

Source code repository

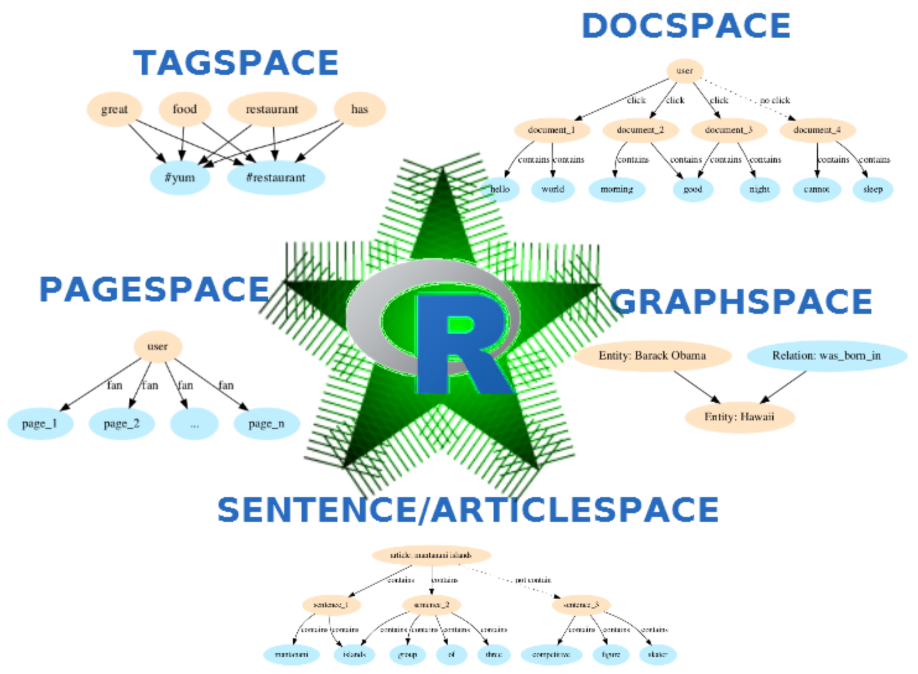
The source code of the package is on github at https://github.com/bnosac/ruimtehol.
It uses Starspace version 'STARSPACE-2017-2' cloned in September 2018 to https://github.

com/bnosac-dev/StarSpace. Modifications were done in order to make it CRAN compliant and to make the usage more straightforward for R users. The R package is distributed under the Mozilla Public License version 2.0.

1.2. Functionalities

The R package allows you to have a low-level access to the C++ library for fine-grained control and provides as well high-level interfaces for commonly used tasks. Functionalities are also included for saving and loading models, using the models to predict and general embedding similarity and ranking functions.

starspace	Low-level interface to build a Starspace model
starspace_load_model	Load a pre-trained model or a tab-separated file
starspace_save_model	Save a Starspace model
starspace_embedding	Get embeddings of documents/words/ngrams/labels
starspace_knn	Find k-nearest neighbouring information for new text
starspace_dictonary	Get words/labels part of the model dictionary
predict.textspace	Get predictions along a Starspace model
as.matrix	Get words and label embeddings
embedding_similarity	Cosine/dot product similarity between embeddings - top-n most similar text
embed_wordspace	Build a Starspace model which calculates word/ngram embeddings
embed_sentencespace	Build a Starspace model which calculates sentence embeddings
embed_articlespace	Build a Starspace model for embedding articles - sentence-article similarities
embed_tagspace	Build a Starspace model for multi-label classification
embed_docspace	Build a Starspace model for content-based recommendation
embed_pagespace	Build a Starspace model for interest-based recommendation
embed_entityrelationspace	Build a Starspace model for entity relationship completion



1.3. Example data

In what follows below, we showcase some of the use cases of the R package. In order to do that, we will use text with questions and answers from the Belgian parliament. This dataset was collected as open data under the CC0 license from `http://data.dekamer.be`

The data contains Dutch questions asked in the year 2017 by members in the National Belgian parliament. Each question was categorised alongside several themes and the dataset also contains the answer which was given by the department or minister who was responsible for that topic. We also know to which political party the person belonged to who was asking the question.

```
 library(ruimtehol)
 data("dekamer", package = "ruimtehol")
 str(dekamer)
```

```
'data.frame':        3811 obs. of  12 variables:
 $ doc_id              : int  184501 184502 184503 184504 184505 184506 184507 184508 1845
 $ depotdat            : Date, format: "2017-01-04" "2017-01-04" ...
 $ aut_party           : Factor w/ 16 levels "CD&V","CDH","DEFI",..: 6 6 9 9 1 6 6 7 11 9
 $ aut_person          : Factor w/ 156 levels "Almaci, Meyrem",..: 10 17 87 87 86 12 12 15
 $ aut_language        : Factor w/ 2 levels "dutch","french": 2 2 1 1 1 2 2 1 2 1 ...
 $ question            : chr  "Lijn Brussel-Luxemburg. - Technisch incident. \n\n Op maand
 $ question_theme_main : Factor w/ 45 levels "ARBEID","BEGROTING",..: 43 43 39 39 15 NA NA
 $ question_theme      : chr  "VERVOERSLIJN,VERVOERBELEID,VERVOER PER SPOOR,NMBS,DIENSTREG
 $ answer              : chr  "1. De noodrem werd geactiveerd; het ging om kwaad opzet. 2.
 $ answer_deptpres     : int  13 13 6 3 9 4 16 13 13 16 ...
 $ answer_department   : chr  "Minister van Mobiliteit, belast met Belgocontrol en de Nati
 $ answer_subdepartment: chr  "Mobiliteit, Belgocontrol en NMBS" "Mobiliteit, Belgocontrol
```

2. Text classification

In this example, we will perform text classification. Each question in parliament can be labelled with one or more tags. A tagspace model is constructed which can be used to tag new questions with the learned tagset.

2.1. Data preparation

The following data preparation is done first

- make sure all the text is separated by spaces, this is the format which Starspace needs
- make sure the response is a list of all categories in case you do multi-label classification and each category should not contain spaces (in the below example spaces are replaced with a dash)

The Starspace C++ library internally uses spaces as a separator, make sure each token is separated by a space if you want to include it in the model and that the labels do not contain spaces.

```
dekamer$x <- strsplit(dekamer$question, "\\W")
dekamer$x <- lapply(dekamer$x, FUN = function(x) setdiff(x, ""))
dekamer$x <- sapply(dekamer$x, FUN = function(x) paste(x, collapse = " "))
dekamer$x <- tolower(dekamer$x)
dekamer$y <- strsplit(dekamer$question_theme, split = ",")
dekamer$y <- lapply(dekamer$y, FUN=function(x) gsub(" ", "-", x))
dekamer$x[1:2]
```

```
[1] "lijn brussel luxemburg technisch incident op maandag 19 december 2016 kondigde de nmb
[2] "mystery shopping resultaten van de onderzoeken op een eerdere parlementaire vraag ove
```

```
dekamer$y[1:2]
```

```
[[1]]
[1] "VERVOERSLIJN"                    "VERVOERBELEID"
[3] "VERVOER-PER-SPOOR"               "NMBS"
[5] "DIENSTREGELING-VAN-HET-VERVOER"

[[2]]
[1] "FRAUDE"             "CONTROLEORGAAN"    "VERVOERBELEID"
[4] "PLAATSBEWIJS"       "VERVOER-PER-SPOOR" "NMBS"
```

2.2. Model building

Next a model is constructed, in this case a tagspace model which is a model which can be used for simple classification as well as multi-label classification (which is the case here).

```
set.seed(321)
model <- embed_tagspace(x = dekamer$x, y = dekamer$y,
                        early_stopping = 0.8, validationPatience = 10,
                        dim = 50,
                        lr = 0.01, epoch = 40, loss = "softmax", adagrad = TRUE,
                        similarity = "cosine", negSearchLimit = 50,
                        ngrams = 2, minCount = 2,
                        maxTrainTime = 2 * 60)

model
```

```
Object of class textspace
 dimension of the embedding: 50
 training arguments:
      loss: softmax
      margin: 0.05
      similarity: cosine
      epoch: 40
      adagrad: TRUE
      lr: 0.01
      termLr: 1e-09
      norm: 1
      maxNegSamples: 10
      negSearchLimit: 50
      p: 0.5
      shareEmb: TRUE
      ws: 5
      dropoutLHS: 0
      dropoutRHS: 0
      initRandSd: 0.001
```

The code above has trained a model with the following relevant arguments

- **early_stopping**: data is split in a training (80 %) and a validation set (20 %)
- **dim**: the dimension of the embedding is set to 50
- optimisation is done with **adagrad**, during 40 **epochs**, starting with a learning rate (**lr**) of 0.01 and each time decreasing the learning rate by 1/epoch.
- The **loss** which is optimised is softmax loss. If the loss has not decreased during 10 epochs as set with **validationPatience**, training is stopped.
- Similarity between positive and negative labels is done by **cosine** similarity

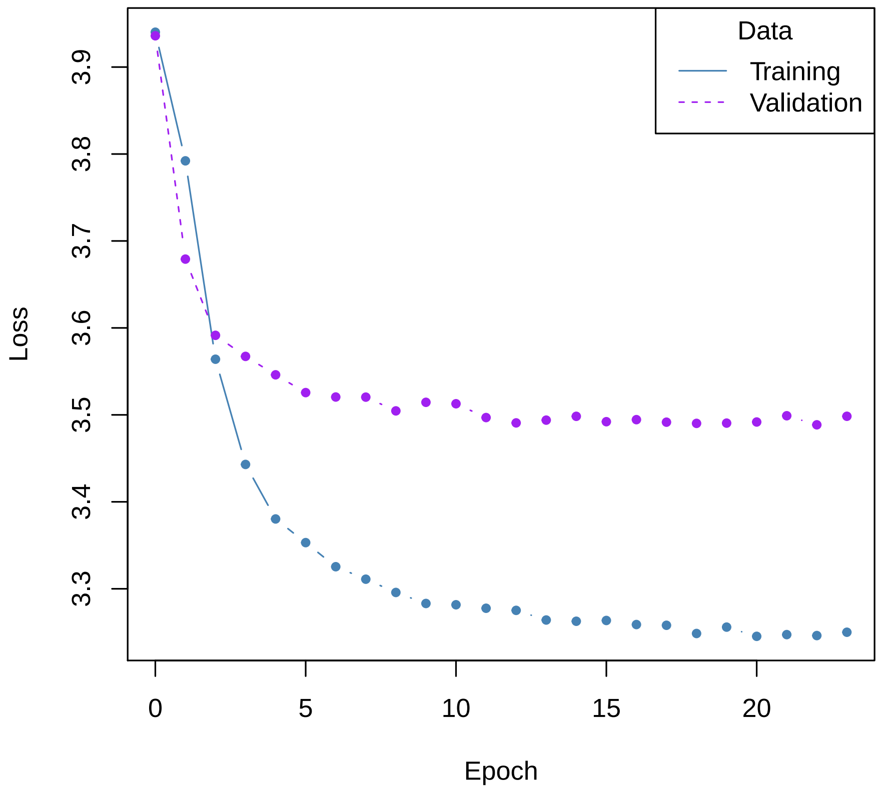
- **negSearchLimit** indicates the number of negative samples which are taken (for each question we know which labels were given (positive) and we sample from the list of labels which were not given a bunch of negatives)
- model is trained on bigrams (argument **ngrams**) and these should occur at least twice (argument **minCount**)
- model is trained for maximum 120 seconds (argument **maxTrainTime**))

2.3. Model inspection

Loss evolution

The plot shows the evolution of the loss over the epochs on the training set and the validation set. This should generally decrease steadily and stabilise on the validation set which indicates it has learned the embeddings well.

```
plot(model)
```



Dictionary

Starspace models the embeddings of the labels and the words in the same embedding space allowing you to compute similarities across labels and text. To get the dictionary of all terms in the model (words and labels), you can do the following.

```
dict <- starspace_dictionary(model)
str(dict)
```

```
List of 6
 $ ntokens        : int 368118
 $ nwords         : int 14144
 $ nlabels        : int 1344
 $ labels         : chr [1:1344] "__label__VERVOERBELEID" "__label__OVERHEID" "__label__GE
 $ dictionary_size: int 15488
 $ dictionary     :'data.frame':	15488 obs. of  3 variables:
  ..$ term    : chr [1:15488] "de" "in" "het" "van" ...
  ..$ is_word : logi [1:15488] TRUE TRUE TRUE TRUE TRUE TRUE ...
  ..$ is_label: logi [1:15488] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
length(dict$labels)
```

```
[1] 1344
```

The dictionary element contains a data.frame with all words as well as all labels which were found in the data and for which embeddings are calculated.
Note that bigrams or ngrams are not stored and you'll also see that labels are prefixed with the text __label__, allowing them to be distinguished from plain words.

2.4. Embeddings of the dictionary

You can get the learned embeddings of the words and the labels which are part of the dictionary as follows.

```
emb <- as.matrix(model)
dim(emb)
```

```
[1] 15488    50
```

If you only want embeddings of the words or of the labels, you can set the *type* argument in the function *as.matrix* or you can use the function *starspace_embedding* directly. Below the embedding of the word 'geld' is retrieved as well as the embedding of the label 'VERVOERBELEID'.

```
emb_words  <- as.matrix(model, type = "words")
emb_labels <- as.matrix(model, type = "labels", prefix = FALSE)
e <- starspace_embedding(model, x = c("__label__VERVOERBELEID", "geld"), type = "ngram")
```

If you trained a model with *ngrams* > 1, you can get the embedding of a bigram/ngram also. For this Starspace uses a hashing trick from *fastText* and gets - based on the words which define the bigram/ngram - the embedding of the hashed bucket of the combined term. If you specify to retrieve this type of embedding, you can not have more words than you specified *ngrams* when doing the training.

```
e <- starspace_embedding(model, c("nationale loterij"), type = "ngram")
```

2.5. Embeddings of full text

For retrieving embeddings of full text, use *type = 'document'* which is the default of *starspace_embedding*. It aggregates the embeddings of the words which are part of the text. Note that this aggregation is governed by the normalization parameter p which you can set when you build the model. The sum of the embeddings of the words which are in the text is normalised by dividing by $\#words^p$. If $p = 1$, this is equivalent to taking the average of the embeddings. When $p = 0$, this is equivalent to taking sum of the embeddings. The default is $p = 0.5$ indicating a mixture of both.

```
text <- c("de nmbs heeft het treinaanbod uitgebreid via onteigening ...",
          "de migranten komen naar europa de asielcentra ...")
emb_text <- starspace_embedding(model, text)
dim(emb_text)
```

```
[1]  2 50
```

2.6. Predict/Similarities/Ranking

You can extract predictions and get embedding similarities for information retrieval and ranking. Function *predict* gets the document embeddings of the text and find the closest among the labels.

```
predict(model, "de migranten komen naar europa de asielcentra ...")
```

```
[[1]]
[[1]]$doc_id
[1] 1

[[1]]$text
[1] "de migranten komen naar europa de asielcentra ..."

[[1]]$prediction
           label           label_starspace similarity
1          LIBIE           __label__LIBIE  0.8524756
2    VLUCHTELING     __label__VLUCHTELING  0.8037947
3   MENSENHANDEL    __label__MENSENHANDEL  0.7843831
```

```
4 MIDDELLANDSE-ZEE __label__MIDDELLANDSE-ZEE  0.7739475
5   MIGRATIEBELEID   __label__MIGRATIEBELEID  0.7410468

[[1]]$terms
[[1]]$terms$basedoc_index
[1] 566  23 553 506   9

[[1]]$terms$basedoc_terms
character(0)
```

The same can also be achieved with the function *embedding_similarity* which provides cosine and dot product similarities. They give the same numbers as the predict functionality.

```
embedding_similarity(emb_text, emb_labels, type = "cosine", top_n = 5)
```

```
                                                          term1
1  de nmbs heeft het treinaanbod uitgebreid via onteigening ...
2  de nmbs heeft het treinaanbod uitgebreid via onteigening ...
3  de nmbs heeft het treinaanbod uitgebreid via onteigening ...
4  de nmbs heeft het treinaanbod uitgebreid via onteigening ...
5  de nmbs heeft het treinaanbod uitgebreid via onteigening ...
6             de migranten komen naar europa de asielcentra ...
7             de migranten komen naar europa de asielcentra ...
8             de migranten komen naar europa de asielcentra ...
9             de migranten komen naar europa de asielcentra ...
10            de migranten komen naar europa de asielcentra ...
                           term2 similarity rank
1                 VERVOERBELEID  0.9347046    1
2                  STADSVERVOER  0.9343675    2
3                          NMBS  0.9256427    3
4             VERVOER-PER-SPOOR  0.9241741    4
5  DIENSTREGELING-VAN-HET-VERVOER  0.9237946    5
6                         LIBIE  0.8524755    1
7                   VLUCHTELING  0.8037945    2
8                  MENSENHANDEL  0.7843830    3
9              MIDDELLANDSE-ZEE  0.7739474    4
10               MIGRATIEBELEID  0.7410468    5
```

Shorthands for the knn Starspace functionality is also provided. This function allows you to answer things like 'What does this look like'. It shows the nearest neighbour of text to the dictionary.

```
starspace_knn(model, "de migranten komen naar europa de asielcentra ...", k = 5)
```

```
$input
[1] "de migranten komen naar europa de asielcentra ..."
```

```
$prediction
              label similarity rank
1         migranten  0.8872196    1
2   migratiestromen  0.8735061    2
3             libie  0.8708380    3
4 mensensmokkelaars  0.8655367    4
5               iom  0.8619062    5
```

2.7. Customising

As the words and labels are in the same embedding space, you can interpret the predict and similarity functionalities in a broad way. This is shown below where the target documents to compare with are changed.

```
targetdocs <- c("__label__FISCALITEIT",
                "__label__OVERHEIDSADMINISTRATIE",
                "__label__MIGRATIEBELEID",
                "__label__POLITIE",
                "__label__BUITENLANDS-BELEID",
                "__label__ECONOMISCH-BELEID",
                "de migranten komen naar europa ZZZ",
                "__label__PERSONEEL")
predict(model, "de migranten komen naar europa de asielcentra ...",
          basedoc = targetdocs)
embedding_similarity(
    starspace_embedding(model, "de migranten komen naar europa de asielcentra ..."),
    starspace_embedding(model, targetdocs), top_n = 3)
```

2.8. Save/Load model

Saving the model consists of saving the embeddings of the words and the labels as well as storing the model parameters. The saved model can next be loaded back in and used for information retrieval. The following approach is the advised approach to save and reload models.

```
starspace_save_model(model, file = "textspace.ruimtehol")
model <- starspace_load_model("textspace.ruimtehol")
```

3. Other models

The ruimtehol package contains many more models. It is advised to just inspect the help of the functions listed up in section 1.2.

- Embeddings of word, sentences, articles, documents, webpages, links and entities: See the examples in the package.
- Ranking and information retrieval: See the examples in the package.
- Collaborative filtering: See the examples in the package.

4. Semi-supervised learning

In the example in this vignette on classification modelling (embed_tagspace shown in section 2.2), Starspace was used in a completely supervised setting. If you have a look at the documentation of the functions embed_wordspace, embed_sentencespace, embed_articlespace, you will notice that these are completely unsupervised. Starspace also allows to do a combination of both, namely you can perform semi-supervised learning. This is shown below where we randomly remove some data with text and some data of the labels as well and still learn on the full data. As long as we have information on the labels or on the terms or both, we can learn embeddings.

```r
set.seed(321)
dekamer <- dekamer[order(rnorm(n = nrow(dekamer))), ]
X <- dekamer$x
Y <- dekamer$y
X[1:250]   <- NA
Y[251:500] <- NA
model <- embed_tagspace(x = X, y = Y,
                          early_stopping = 0.8, validationPatience = 10,
                          dim = 50,
                          lr = 0.01, epoch = 40, loss = "softmax", adagrad = TRUE,
                          similarity = "cosine", negSearchLimit = 50,
                          ngrams = 2, minCount = 2,
                          maxTrainTime = 2 * 60)
```

5. Transfer learning

The ruimtehol R package also allows to do transfer learning. In transfer learning, knowledge gained while learning on other data can be transferred to new data. The typical use case of this is the case where we have already pretrained embeddings available. Several authors have provided embeddings which were trained on different sources (e.g. Wikipedia / Open databases / Google or Bing queries / Gigaword / Open CONLLU corpora). These are mostly monolingual resources but also cross-lingual embeddings are common now. There are 2 use cases of such pretrained embeddings, namely.

1. You can use these embedding as is
2. You can use these embedding as starting point to train and customise them on your data (transfer learning)

In the examples below, both will be shown. In either case, we need to provide the argument *embeddings* which should be a pretrained embedding matrix where the rownames of the matrix are the terms or labels from the model.

5.1. Transfer learning - use embeddings as is

In order to show the first use case we generate some random embedding matrix and feed it to function *starspace* which is the main workhorse behind all embed_ ... functions. The important arguments that you need to give are

- embeddings: a matrix of embeddings where the row names indicate the terms or label
- similarity: how you want to calculate similarities between embeddings and documents
- ngrams: when calculating similarities, shall we consider more than unigrams only
- p: normalisation parameter as explained in section 2.5
- trainMode: either 0 (tagspace), 1 (pagespace), 2 (articlespace), 3 (sentence space), 4 (multi-relational graphspace), 5 (word embeddings)

```
pretrained <- matrix(data = rnorm(1000 * 100), nrow = 1000, ncol = 100,
                     dimnames = list(term = sprintf("word%s", 1:1000)))
model <- starspace(embeddings = pretrained,
                   similarity = "cosine", p = 0.5, ngrams = 1, trainMode = 5)
predict(model, newdata = c("word5 word1 word5 word3"), type = "knn")
```

```
[[1]]
  doc_id   label similarity rank
1      1   word5  0.8257448    1
2      1   word1  0.4352573    2
3      1   word3  0.3640119    3
4      1  word71  0.3003811    4
5      1 word832  0.2960142    5
```

5.2. Transfer learning - use embeddings as starting values for training

Now for the second use case: **transfer learning**. Below we build a simple wordspace model to extract word embeddings. These embeddings will be passed on as starting values for another tagspace model.

```
set.seed(321)
model <- embed_wordspace(dekamer$x,
                           dim = 50, ws = 7, epoch = 5, ngrams = 2, adagrad = FALSE,
                           margin = 0.8, negSearchLimit = 10,
                           maxTrainTime = 2 * 60)
pretrained_words  <- as.matrix(model)
```

It's important to note that we need to pass on pretrained embeddings **for all terms as well as all labels which we want to keep training upon**. As we haven't got the embeddings of the labels, in the example below, we assign some random starting values to the embedding of the labels. (Small note: in business settings it makes sense to let these labels start from a sensible combination of the embeddings of words which you think are similar to that label). Starspace uses the default prefix __label__ for identifying a label, which we need to add as prefix to the rownames from the label embeddings.

```
labels            <- sort(unique(unlist(dekamer$y)))
pretrained_labels <- matrix(data = rnorm(n = length(labels) * 50,
                                          mean = mean(pretrained_words),
                                          sd = sd(pretrained_words)),
                             nrow = length(labels),
                             ncol = 50,
                             dimnames = list(term = sprintf("__label__%s", labels)))
pretrained        <- rbind(pretrained_words, pretrained_labels)
```

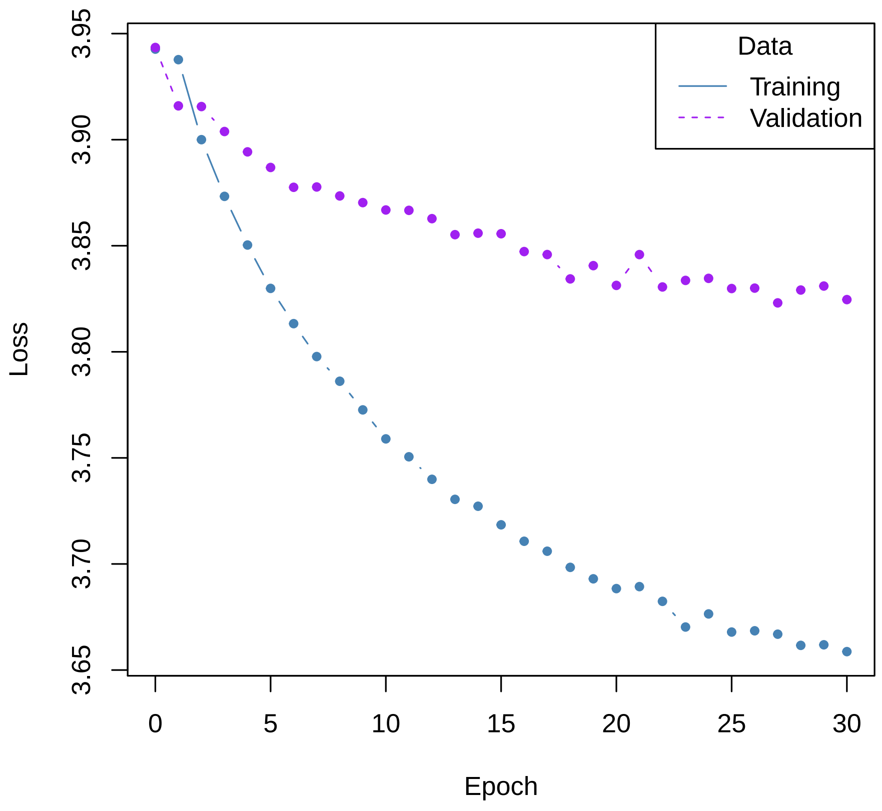
Once we have these pretrained embeddings, we can use them as starting values for the training. Note that we do not need to provide the dim argument as this argument is governed by the dimension of the pretrained embedding matrix.

```
set.seed(321)
model <- embed_tagspace(x = dekamer$x, y = dekamer$y,
                          embeddings = pretrained,
                          early_stopping = 0.8, validationPatience = 10,
                          dim = 50,
                          lr = 0.01, epoch = 40, loss = "softmax", adagrad = TRUE,
                          similarity = "cosine", negSearchLimit = 50,
                          ngrams = 2, minCount = 2,
                          maxTrainTime = 2 * 60)
embedding <- as.matrix(model)

plot(model)
starspace_knn(model, "__label__FISCALITEIT", k = 10)
```

```
$input
[1] "__label__FISCALITEIT"

$prediction
                   label similarity rank
1   __label__FISCALITEIT  1.0000001    1
2             winkelier  0.9194673    2
3               accijns  0.9167983    3
4               valorem  0.9166589    4
5           prijsbeleid  0.9163721    5
6      tabaksfabrikanten  0.9155601    6
7        kwijtgescholden  0.9089543    7
8        accijnsbedragen  0.9067204    8
9            ingevorderd  0.9031077    9
10                 17675  0.8970568   10
```



And now we have the model ready for doing further work based on the enhanced embedding matrix.

Affiliation:
BNOSAC - Open Analytical Helpers
E-mail: `jwijffels@bnosac.be`
URL: `http://www.bnosac.be`