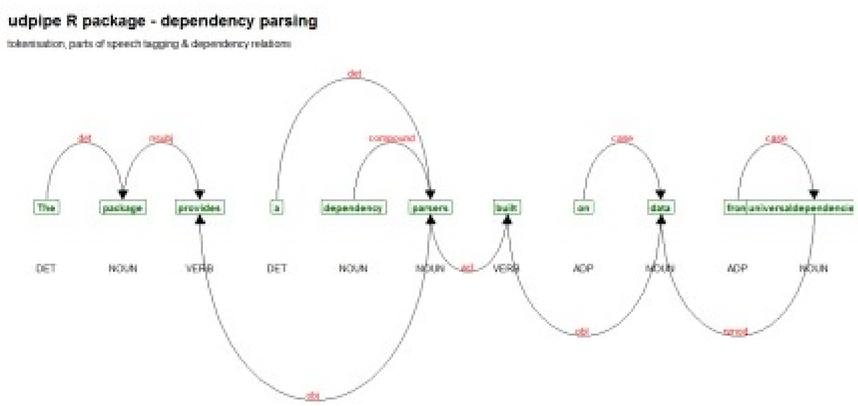
An update of the udpipe R package (https://bnosac.github.io/udpipe/en) landed safely on CRAN last week. Originally the udpipe R package was put on CRAN in 2017 wrapping the UDPipe (v1.2 C++) tokeniser/lemmatiser/parts of speech tagger and dependency parser. It now has many more functionalities next to just providing this parser.

The current release (0.8.4-1 on CRAN: https://cran.r-project.org/package=udpipe) makes sure default models which are used are the ones trained on version 2.5 of universal dependencies. Other features of the release are detailed in the NEWS item. This is what dependency parsing looks like on some sample text.

```
library(udpipe)
x <- udpipe("The package provides a dependency parsers built on data
from universaldependencies.org", "english")
View(x)
library(ggraph)
library(ggplot2)
library(igraph)
library(textplot)
plt <- textplot_dependencyparser(x, size = 4, title = "udpipe R package
- dependency parsing")
plt
```



During the years, the toolkit has now also incorporated many functionalities for commonly used data manipulations on texts which are enriched with the output of the parser. Namely functionalities and algorithms for collocations, token co-occurrence, document term matrix handling, term frequency inverse document frequency calculations, information retrieval metrics, handling of multi-word expressions, keyword detection (Rapid Automatic Keyword Extraction, noun phrase extraction, syntactical patterns) sentiment scoring and semantic similarity analysis.

Many add-on R packages

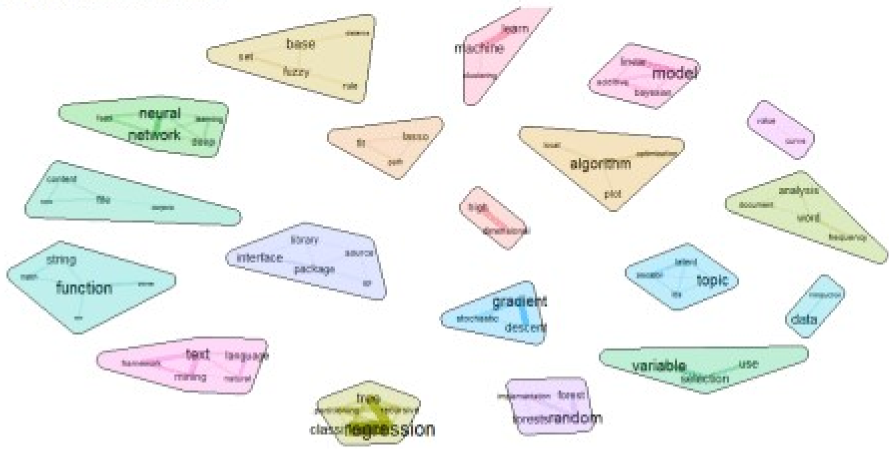
The udpipe package is loosely coupled with other NLP R packages which BNOSAC released in the last 4 years on CRAN. Loosely coupled means that none of the packages have hard dependencies of one another making it easy to install and maintain and allowing you to use only the packages and tools that you want.

> Hereby a small list of loosely coupled packages by BNOSAC which contain functions and documentation where the udpipe package is used as a

preprocessing step.

- BTM: Biterm Topic Modelling
- crfsuite: Build named entity recognition models using conditional random fields
- nametagger: Build named entity recognition models using markov models
- torch.ner: Named Entity Recognition using torch
- word2vec: Training and applying the word2vec algorithm
- ruimtehol: Text embedding techniques using Starspace
- textrank: Text summarisation and keyword detection using textrank
- brown: Brown word clustering on texts
- sentencepiece: Byte Pair Encoding and Unigram tokenisation using sentencepiece
- tokenizers.bpe: Byte Pair Encoding tokenisation using YouTokenToMe
- text.alignment: Find text similarities using Smith-Waterman
- textplot: Visualise complex relations in texts

Example from the textplot package



Model building example

To showcase the loose integration, let's use the udpipe package alongside the word2vec package to build a udpipe model by ourselves on the German GSD treebank which is described at https://universaldependencies.org/treebanks/de_gsd/index.html and contains a set of CC BY-SA licensed annotated texts from news articles, wiki entries and reviews.
More information at https://universaldependencies.org.

Download the treebank.

```
library(utils)
settings <- list()
settings$ud.train    <- "https://raw.githubusercontent.com/
UniversalDependencies/UD_German-GSD/r2.6/de_gsd-ud-train.conllu"
settings$ud.dev      <- "https://raw.githubusercontent.com/UniversalDependencies/UD_
German-GSD/r2.6/de_gsd-ud-dev.conllu"
settings$ud.test     <- "https://raw.githubusercontent.com/
UniversalDependencies/UD_German-GSD/r2.6/de_gsd-ud-test.conllu"
## Download the conllu files
download.file(url = settings$ud.train, destfile = "train.conllu")
download.file(url = settings$ud.dev,   destfile = "dev.conllu")
download.file(url = settings$ud.test,  destfile = "test.conllu")
```

Build a word2vec model using out R package word2vec

- Create wordvectors on the downloaded training dataset as these are used for training the dependency parser
- Save the word vectors to disk
- Inspect a bit the word2vec model by showing similarities to some German words

```
library(udpipe)
library(word2vec)
txt <- udpipe_read_conllu("train.conllu")
txt <- paste.data.frame(txt, term = "token", group = c("doc_id",
"paragraph_id", "sentence_id"), collapse = " ")
txt <- txt$token
w2v <- word2vec(txt, type = "skip-gram", dim = 50, window = 10,
min_count = 2, negative = 5, iter = 15, threads = 1)
write.word2vec(w2v, file = "wordvectors.vec", type = "txt", encoding =
"UTF-8")
predict(w2v, c("gut", "freundlich"), type = "nearest", top = 20)
```

And train the model

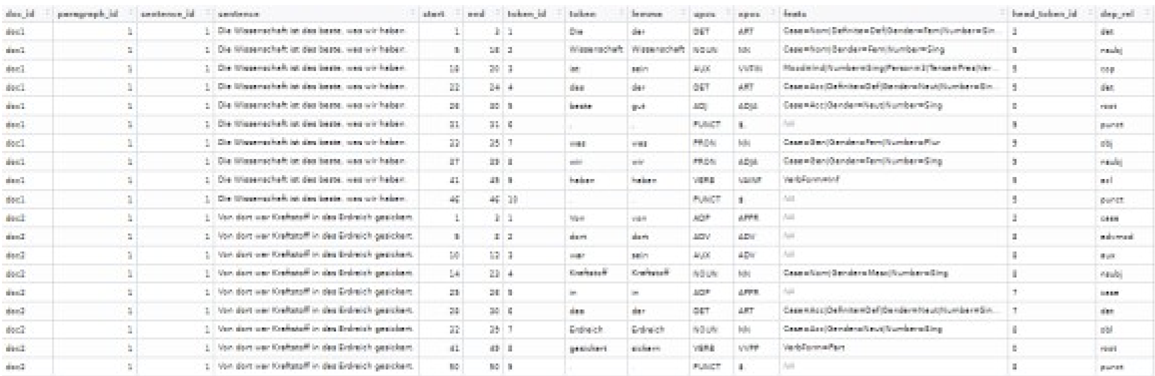
- Using the hyperparameters for the tokeniser, parts of speech tagger & lemmatizer and the dependency parser as shown here: https://github.com/bnosac/udpipe/tree/master/inst/models-ud-2.5
- Note that model training takes a while (8hours up to 3days) depending on the size of the treebank and your hyperparameter settings. This example was run on a Windows i5 CPU laptop with 1.7Ghz, so no GPU needed, which makes this model building process still accessible for anyone with a simple PC.

```
print(Sys.time())
m <- udpipe_train(file = "de_gsd-ud-2.6-20200924.udpipe",
                  files_conllu_training = "train.conllu",
                  files_conllu_holdout  = "dev.conllu",
                  annotation_tokenizer = list(dimension = 64, epochs =
100, segment_size=200, initialization_range = 0.1,
                                              batch_size = 50,
learning_rate = 0.002, learning_rate_final=0, dropout = 0.1,
                                              early_stopping = 1),
                  annotation_tagger = list(models = 2,
                                           templates_1 = "lemmatizer",
                                           guesser_suffix_rules_1 = 8,
guesser_enrich_dictionary_1 = 4,
                                           guesser_prefixes_max_1 = 4,
                                           use_lemma_1 =
1,provide_lemma_1 = 1, use_xpostag_1 = 0, provide_xpostag_1 = 0,
                                           use_feats_1 = 0,
provide_feats_1 = 0, prune_features_1 = 1,
                                           templates_2 = "tagger",
                                           guesser_suffix_rules_2 = 8,
guesser_enrich_dictionary_2 = 4,
                                           guesser_prefixes_max_2 = 0,
                                           use_lemma_2 = 1,
provide_lemma_2 = 0, use_xpostag_2 = 1, provide_xpostag_2 = 1,
                                           use_feats_2 = 1,
```

```
provide_feats_2 = 1, prune_features_2 = 1),
                  annotation_parser = list(iterations = 30,
                                           embedding_upostag = 20,
embedding_feats = 20, embedding_xpostag = 0,
                                           embedding_form = 50,
embedding_form_file = "wordvectors.vec",
                                           embedding_lemma = 0,
embedding_deprel = 20, learning_rate = 0.01,
                                           learning_rate_final = 0.001,
l2 = 0.5, hidden_layer = 200,
                                           batch_size = 10,
transition_system = "projective", transition_oracle = "dynamic",
                                           structured_interval = 8))
print(Sys.time())
```

You can see the logs of this run here. Now your model is ready, you can use it on your own terms and you can start using it to annotate your text.

```
model <- udpipe_load_model("de_gsd-ud-2.6-20200924.udpipe")
texts <- data.frame(doc_id = c("doc1", "doc2"), text = c("Die
Wissenschaft ist das beste, was wir haben.", "Von dort war Kraftstoff
in das Erdreich gesickert."), stringsAsFactors = FALSE)
anno  <- udpipe(texts, model, trace = 10)
View(anno)
```



Enjoy!