Thiemo Fetzer

A SIMPLE PIPELINE TO ACCESS SENNA NLP TOOLKIT THROUGH R

For my paper Social Insurance and Conflict : Evidence from India, I was using the SENNA Natural Language processing toolkit to automatically generate a conflict dataset at the district level, based on raw newspaper clippings collected from the SATP (South Asia Terrorism Portal) daily listings. SENNA is a standalone solution written in C, that uses a multi-layer Neural Network for semantic/syntactic extraction.

Semantic extraction fundamentally involves extracting features that make up the grammatical structure of sentences: Subject, Predicate and Object. On top of Semantic Role labeling, SENNA also performs simpler language processing tasks such as Named Entity Recognition or Part of Speech tagging.

There are now quite a few solutions for doing natural language processing analysis. In particular, the Stanford CORE NLP package has the potential to develop into a major tool for users trying to use natural language processing not just for purposes of topic modelling, grouping or clustering but rather for information retrieval.

For my purpose I wanted to use a simple standalone package that is able to do semantic role labelling as the text, from which I want to retrieve information is quite simple text snippets, as for example the following paragraph out of an SATP daily report:

> *Two unidentified terrorists massacred six members of a family and left a seventh injured at Mangnar Top, Poonch district, on December 31, 2001. Local residents refused to cremate the bodies of the slain victims, insisting that a Union Minister should visit the area and take notice of the increasing terrorist violence there.*

The challenge consists of the dimensionality in the data. In total, there are 69,317 individual news items, consisting of multiple paragraphs and sentences up to date. Hand-coding such an array of data is still feasible – and in fact – multiple authors have attempted to hand code parts of the data (see among others Shrivastava, 2014; Hoelscher et al., 2012; Khanna and Zimmermann, 2014; Vanden Eynde, 2011). However, it provides a natural setting in which semi-automated machine coding techniques can be tried and assessed relative to human coding.

For my paper Social Insurance and Conflict : Evidence from India, I was using the SENNA Natural Language processing toolkit to automatically generate a conflict dataset at the district level, based on raw newspaper clippings collected from the SATP (South Asia Terrorism Portal) daily listings. SENNA is a standalone solution written in C, that uses a multi-layer Neural Network for semantic/syntactic extraction.

Semantic extraction fundamentally involves extracting features that make up the grammatical structure of sentences: Subject, Predicate and Object. On top of Semantic Role labeling, SENNA also performs simpler language processing tasks such as Named Entity Recognition or Part of Speech tagging.

There are now quite a few solutions for doing natural language processing analysis. In particular, the Stanford CORE NLP package has the potential to develop into a major tool for users trying to use natural language processing not just for purposes of topic modelling, grouping or clustering but rather for information retrieval.

For my purpose I wanted to use a simple standalone package that is able to do semantic role labelling as the text, from which I want to retrieve information is quite simple text snippets, as for example the following paragraph out of an SATP daily report:

Two unidentified terrorists massacred six members of a family and left a seventh injured at Mangnar Top, Poonch district, on December 31, 2001. Local residents refused to cremate the bodies of the slain victims, insisting that a Union Minister should visit the area and take notice of the increasing terrorist violence there.

The challenge consists of the dimensionality in the data. In total, there are 69,317 individual news items, consisting of multiple paragraphs and sentences up to date. Hand-coding such an array of data is still feasible – and in fact – multiple authors have attempted to hand code parts of the data (see among others Shrivastava, 2014; Hoelscher et al., 2012; Khanna and Zimmermann, 2014; Vanden Eynde, 2011). However, it provides a natural setting in which semi-automated machine coding techniques can be tried and assessed relative to human coding.

ENTER SENNA

SENNA is a standalone executable that can be called from the command line (terminal), after it was downloaded. SENNA performs a range of classical NLP tasks together in one framework.

- Part of Speech Tagging (POS): aims at labeling each word with a unique tag that indicates its syntactic role, for example, plural noun, adverb
- Chunking:chunking aims at labeling segments of a sentence with syntactic constituents such as noun or verb phrases (NP or VP)
- Named Entity Recognition (NER): NER labels atomic elements, typically nouns, in sentences into categories such as “PERSON”, “LOCATION”, “ORGANIZATION”, …
- Semantic Role Labeling (SRL) aims at giving a semantic role to a syntactic constituent of a sentence; this basically involves identifying subject, verb and object dependency relations between parts of the sentence.

The fact that SENNA provides an integrated framework for performing these tasks together is appealing, as for the information retrieval task to extract information building a conflict event level dataset, we need — at least — the outputs from SRL and NER, to answer the questions: Who? What? Where?.

Getting SENNA up and running is not difficult. You only need to download the SENNA zip archive with its executibles and store it somewhere in your system. If you are on a Mac or *nix based machine, you can simply add the path to the senna executible as part of your PATH environment, which allows you to call the executible using simply its name, e.g. “senna” or by typing “./senna -h” to start the executable in the Terminal App.

If you pass it the help flag, you should see the following interface:


Since most of the data processing I have done is in R, I coded a very simple interface, which effectively just passes data to and from SENNA using R’s system()

command. The heart of the function is `pushSenna()`, which passes individual sentences to and from SENNA, writing to a temporary file and reading from it.

```
library(data.table) pushSenna<-function(ss,
opts="",sennapath="/Applications/senna") { tempin<-tempfile() writeLines(ss,
tempin) tempout<-tempfile() opts<-ifelse(nchar(opts)==0, "",paste("-
",opts,sep="")) temp<-system(paste("cd ",sennapath,"; ./senna ",opts," <
",tempin,">",tempout,sep=""), intern=TRUE) temp<-read.table(tempout)
unlink(tempin) unlink(tempout) temp }
```

The function uses R's tempfile() command and a specified path to the directory in which the SENNA executable is located. The function may need to be adjusted in order to run on Windows machines. A simple function call returns the following:

```
pushSenna("Two unidentified terrorists massacred six members of a family and
left a seventh injured at Mangnar Top, Poonch district, on December 31, 2001.")
## V1 V2 V3 V4 V5 V6 V7 V8 V9 ## 1 Two CD B-NP O - B-A0 B-A0 O (S1(S(S(NP* ##
2 unidentified JJ I-NP O - I-A0 I-A0 O * ## 3 terrorists NNS E-NP O - E-A0 E-A0 O *)
## 4 massacred VBD S-VP O massacred S-V O O (VP(VP* ## 5 six CD B-NP O - B-A1
O O (NP(NP* ## 6 members NNS E-NP O - I-A1 O O *) ## 7 of IN S-PP O - I-A1 O O
(PP* ## 8 a DT B-NP O - I-A1 O O (NP* ## 9 family NN E-NP O - E-A1 O O *)))) ## 10
and CC O O - O O O * ## 11 left VBD S-VP O left O S-V O (VP* ## 12 a DT B-NP O - O
B-A1 B-A1 (S(NP* ## 13 seventh JJ E-NP O - O I-A1 E-A1 *) ## 14 injured VBN S-VP
O injured O I-A1 S-V (VP* ## 15 at IN S-PP O - O I-A1 B-AM-LOC (PP* ## 16
Mangnar NNP B-NP B-LOC - O I-A1 I-AM-LOC (NP(NP* ## 17 Top NNP E-NP E-LOC
- O I-A1 I-AM-LOC *) ## 18 , , O O - O I-A1 I-AM-LOC * ## 19 Poonch NNP B-NP S-
LOC - O I-A1 I-AM-LOC (NP* ## 20 district NN E-NP O - O I-A1 E-AM-LOC *) ## 21 ,
, O O - O I-A1 O *)))) ## 22 on IN S-PP O - O I-A1 O (PP* ## 23 December NNP B-NP
O - O I-A1 O (NP* ## 24 31 CD E-NP O - O I-A1 O *))))) ## 25 , , O O - O I-A1 O * ##
26 2001. . O O - O E-A1 O (NP*)))
```

The first column is the sentence, the second colum are the results are the results from Part of Speech Tagging, the third column represent the results from chunking, the fourth provides the results from NER; from the fifth column onwards, the output of the interrelationships between subject verb and object triplets is presented, while the last column presents the parse tree

A second of functions extracts sub-parts of information that one may be interested in.

```
returnSVO<-function(temp, col.num=5, type="") { verbs<-
grep("^[^-]",temp[,col.num]) if(length(verbs)>0) { if(type=="") { SVO<-
lapply(1:length(verbs), function(x) data.frame("verb"=temp[grep("-
V{1}",temp[,col.num+x]),1], "subject"=paste(temp[grep("-
A0",temp[,col.num+x]),1], collapse=" "), "object"=paste(temp[grep("-A[1-
9]+",temp[,col.num+x]),1], collapse=" "))) SVO<-rbindlist(SVO) } else
if(type=="list") { SVO<-lapply(1:length(verbs), function(x)
data.frame("id"=x,rbind(cbind("type"="verb", "token"=temp[grep("-
V{1}",temp[,col.num+x]),1]),cbind(type="subject", "token"=paste(temp[grep("-
A0",temp[,col.num+x]),1], collapse=" ")),
cbind(type="object","token"=paste(temp[grep("-A[1-9]+",temp[,col.num+x]),1],
collapse=" "))))) SVO<-rbindlist(SVO) } SVO } } returnNER<-
function(temp,col.num=4) { temp$nertype<-gsub("([A-Z]{1})-([A-
Z]*)","\\2",temp[,col.num]) ##define start and end pairs start<-
grep("^B",temp[,col.num]) end<-grep("^E",temp[,col.num]) singleton<-
grep("^S",temp[,col.num]) index<-data.frame(rbind(cbind(start,end),
cbind(singleton,singleton))) index<-index[order(index[,1]),] if(nrow(index)>0) {
index$ner<-1:nrow(index) NER<-lapply(1:nrow(index), function(x)
cbind(index[x,],ner=paste(temp[index[x,1]:index[x,2],1], collapse=" "),type=
temp[index[x,1], "nertype"])) NER<-rbindlist(NER) NER } }
```

The result from calling these functions may look like this:

```
temp<-pushSenna("Two unidentified terrorists massacred six members of a family and left a seventh injured at Mangnar Top, Poonch district, on December 31, 2001.") returnSVO(temp)

## verb subject ## 1: massacred Two unidentified terrorists ## 2: left Two unidentified terrorists ## 3: injured ## object ## 1: six members of a family ## 2: a seventh injured at Mangnar Top , Poonch district , on December 31 , 2001. ## 3: a seventh
```

Presenting the subject-verb-object triplets, or presenting the named entities contained in the data.

```
returnNER(temp)

## start end ner ner type ## 1: 16 17 1 Mangnar Top LOC ## 2: 19 19 2 Poonch LOC
```

Having performed semantic role labeling and named entity recognition on the roughly 60,000 news reports resulted in close to 1 million subject-verb-object triplets. The performance of SENNA is quite remarkable, given that the newspaper language is quite simple with short sentences describing factual information. For the vast majority of triplets, both subject and object are identified. However, identifying the subject is generally more challenging due to the nested structure of sentences.

I perform a range of refinements to reduce the dimensionality. For example based on the whole set of triplets, one can refine the set of potential acts to consider ex-post, having analyzed all the text in the raw data. In the simple example, there are three verbs. The verb “left” is unlikely to be carrying a lot of information about an individual conflict event that is not already contained in the verbs “massacre” and “injure”.

In case a researcher wants to extract only events with fatalities, the researcher may refine the set of verbs to include only verbs that are indicative of lethal events; in manual coding of conflict data, this is typically done right from the start, by throwing away all text in which certain keywords do not appear. An example is to only study text in which the word “kill”x appears. However, an analysis of the set of subject-verb-object pairs suggests that there are a lot more ways of saying â€œto killâ€x. Some further examples are â€œto killâ€x, â€œto shoot deadâ€x, â€œto massacreâ€x, â€œto slayâ€x, â€œto succumb [to injuries]â€x.

The refinement step results in much fewer words being considered, they are represented as the following word cloud, with the size being relative to their respective frequency of occurence in the raw text.

The next processing step involves geographic matching, the natural starting point being the detected Named Entities and a database or gazetteer of location names. This needs to be fine tuned to the specific context, as in the raw SATP data, typically location information is provided.

BUILDING A CLASSIFIER

In the third step, we want to collapse data to the sentence level and provide a classification for the different conflict objects. Two classifiers are needed:

- Human versus non-human object classification
- Classification of (human) actor types: terrorist, civilian and security forces.

For this, we need to build our own classifier. This is done, collecting human judgements from a crowd-sourcing platform and squaring these human judgements with a trained support vector machine learning algorithm. The focus is particularly on the object of a verb. In the above case, we want to label the perpetrator (“two unidentified terrorists”) of the act of “massacring” to be “terrorists” and the object of the verb massacre (“six members of a family”) to be “civilians.”

A common machine learning method to train a classifier are support vector machines. The idea is to represent the vector of words as a word feature count vector, with the number of columns for a word feature vector being the different words in a vocabulary for a specific corpus. Since typically thousands of words are considered, these vectors are extremely sparse. Support vector machines than try to fit a separating hyperplane in the space spanned by the data that separates the types.

The training data, which is a random subset of the actual data, was coded up collecting human judgements. The idea is now to use the training data and learn t

```
set.seed(2016) TRAINING<-
data.table(read.csv(file="/Users/thiemo/Dropbox/Research/Matteo and
Thiemo/senna/classification-tree.csv")) PERSON<-
TRAINING[objecttype=="person" & label1!=""] PERSON$label1 <-
factor(PERSON$label1) PERSON$label1num <-
as.numeric(factor(PERSON$label1)) head(PERSON[label1
=="civilian",paste(verb,objectcleanpp,sep=" ")])
```

```
## [1] "abducted civilians village morian" ## [2] "killed civilians" ## [3]
"kidnapped civilian dingdongpara" ## [4] "killed civilian who had reportedly
functioned counter insurgent few years ago" ## [5] "killed civilian" ## [6]
"abducted civilians"
```

```
head(PERSON[label1 =="terrorist",paste(verb,objectcleanpp,sep=" ")])
```

```
## [1] "killed maoist cpi aoist cadre" ## [2] "raided maoist hideout village forest
area abuj mad" ## [3] "surrendered maoists" ## [4] "killed terrorists united
liberation front asom ulfa" ## [5] "injuring least militants" ## [6] "arrested cpi
aoist cadres village"
```

```
head(PERSON[label1 =="security",paste(verb,objectcleanpp,sep=" ")])
```

```
## [1] "injured soldiers" "killed soldiers" ## [3] "killed policeman" "injured
security force personnel" ## [5] "injured police constable" "killed security force
personnel"
```

The actual training step is .

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(RTextTools)
```

```
## Loading required package: SparseM ## ## Attaching package: 'SparseM' ## ##
The following object is masked from 'package:base': ## ## backsolve
```

```
set.seed(2016) #a validation set valid<-sample(1:nrow(PERSON), 500)
PERSON$validation<- 0 PERSON[valid]$validation<-1 PERSON<-
PERSON[order(validation)] DOC<-
create_matrix(c(PERSON[,paste(objectcleanpp,sep=" ")]),language="english",
removeNumbers=TRUE,stemWords=TRUE,removePunctuation=TRUE,removeSp
arseTerms=0.9999) DOCCONT<-create_container(DOC,PERSON$label1num,
trainSize=1:1200, testSize=1201:nrow(PERSON), virgin=TRUE) MOD <-
train_models(DOCCONT, algorithms=c("SVM")) RES <- classify_models(DOCCONT,
MOD) analytics <- create_analytics(DOCCONT, RES) res<-
data.table(analytics@document_summary) VALID<-
cbind(PERSON[validation==1],res)
table(VALID$label1,factor(VALID$SVM_LABEL, labels=levels(VALID$label1)))
```

```
## ## civilian security terrorist ## civilian 111 3 12 ## security 8 86 1 ## terrorist
18 6 255

##overall accuracy sum(diag(3) *table(VALID$SVM_LABEL,VALID$label1))/500

## [1] 0.904
```

Overall the prediction accuracy on the validation set is around 90%, indicating that we correctly classify the correct labels. This means that the classifier will have good out of sample performance in getting the labels right.
