Software Internship Report

April 24, 2014

AUTHOR:

Sabrina Galasso
sabrina.galasso(at)student.uni-tuebingen.de
Department of Linguistics
University of Tübingen

SUPERVISOR:

Verena Henrich
verena.henrich(at)uni-tuebingen.de

COMPANY:

European Academy of Bozen/Bolzano
Institute for Specialised Communication and Multilingualism

PERSON OF CONTACT:

Egon Stemle
egon.stemle(at)eurac.edu

This paper reports a five-month Internship at the Institute for Specialised Communication and Multilingualism of the interdisciplinary European Academy (EURAC) in Bolzano, Italy. Its current researches focus on specialised terminology, translation, language acquisition, minority languages and language technologies. My task was to contribute to different linguistic and educational research projects by solving linguistic and computational problems. Some projects investigate first and second language skills of South Tyrolean students, whereas another project investigates South Tyrolean computer-mediated communication. It will be shown how part of speech tagging results, which were required within these projects, were visualized in a confusion matrix using R. Furthermore an ensemble tagging tool was implemented in Python, which combines the results of different part of speech taggers and aims at the reduction of manual correction work. Additionally a python tool was developed that improves the TreeTagger's tokenization results for emoticons. In cooperation with the University of Bozen/Bolzano I created an xml learner corpus for a set of university student texts, which helps to analyze their higher literacy skills.

1 Introduction

The research center EURAC was founded in 1992 and focuses on problems concerning autonomies, mountains, health and technologies. It is placed in South Tyrol, a multilingual region where Italian, German and Ladin are official languages. I chose Italy as a country for my Internship to improve my language skills and to gain experience abroad [1]. The Institute for Specialised Communication and Multilingualism [2] is one of EURAC's 11 institutes and consists of 23 employees. Within this research group different projects build different working teams. I was involved in the following projects:

- KOKO analyzes the language skills of German L1 learners based on a corpus. My task was to improve the accuracy of part of speech (PoS) tagging results. We created a gold standard for a small sample to evaluate existing tag assignments, which were based on the manual correction of the IMS TreeTagger [Schmid, 1995] results. The tagger's pre-annotations were suspected of having influenced the decisions of the annotator adversely. Thus, we decided to improve the PoS tags by another semi-automatic approach (Ensemble Tagging), where manual annotation work is also reduced, but not based on pre-annotations. By means of a confusion matrix we visualized in which cases the IMS TreeTagger failed. This was done to easily select the cases that should always be revised manually in future PoS tagging tasks.

- DIDI investigates South Tirolean computer-mediated communication in special consideration of the writer's age. The developed approach of Ensemble Tagging in KOKO could be adopted for this project. Furthermore we improved the tokenization results.

[1] At the University of Applied Sciences in Karlsruhe, a center coordinates all international internships for Baden-Württemberg: `http://www.hs-karlsruhe.de/internationales/koor/stipendienprogramme/erasmus-praktika.html`

[2] Information about the institute can be found via the following link: `http://www.eurac.edu/en/research/institutes/multilingualism/default.html`

- A project from researchers of the Faculty for Education (University of Bozen/Bolzano) aims to investigate higher literacy skills of university students with the help of corpus-based analysis. My task was to transform 148 student essays into one corpus on which analysis can be performed.

2 Creating a PoS Gold standard for the KOKO project

In the project called KOKO the language competence of especially South Tyrolean German L1 learners is analyzed on the basis of a corpus. The dataset contains 4842 tokens. Figure 1 shows how a PoS gold standard was developed for the KOKO dataset. A silver standard has already been developed by means of a semi-automatic approach that is based on revising manually a tagger's pre-annoation. These pre-annoations were suspected of having influenced the decisions of the annotator adversely. To investigate this assumption a gold standard was created for a small data sample and compared to the silver standard (Silver1). In the following subsection it will be shown how this sample lead to the decision of creating a second silver standard (Silver2) for the whole dataset. Silver2 was created by making use of ensemble tagging, where we avoided influences from pre-annotation results. In the end I created the gold standard by revising the differences between the two silver standards manually. We discussed ambiguous cases in a group to find a consent.

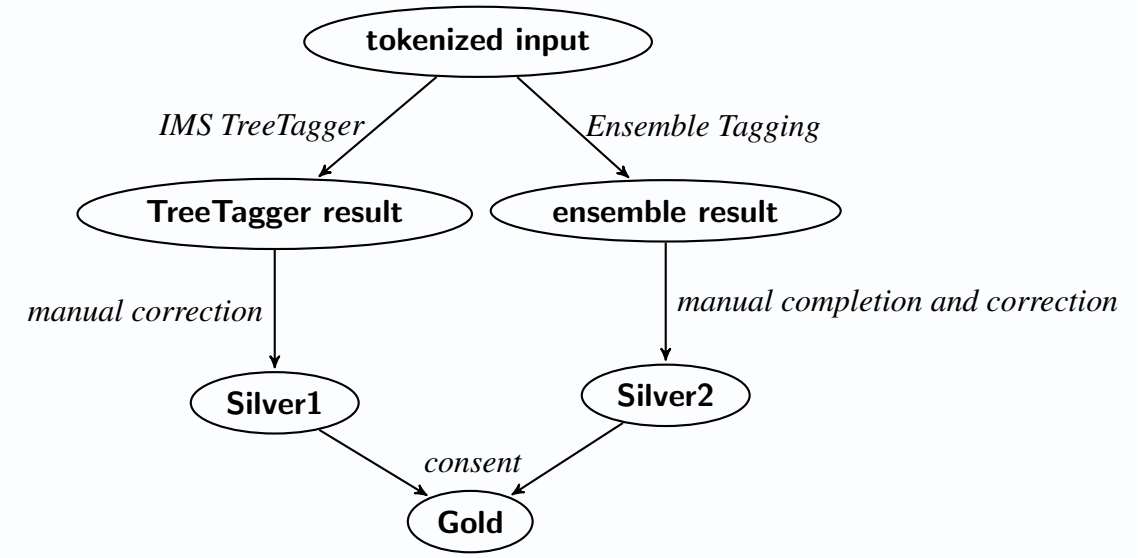

Figure 1: The consent of two different methods builds the gold standard. The first method is manual correcting the TreeTagger result, whereas the second method makes use of Ensemble Tagging and manual completion and correction.

2.1 Evaluation of a small sample

The assumption that pre-annotation can affect the annotators decisions adversely was examined by firstly annotating the first 20 sentences (384 tokens) manually from scratch and secondly

finding a consent in the disagreements with Silver1 to get a gold standard.

Table 1 shows that mistakes made during the revision of a tagger's result correspond to the mistakes made by the tagger itself. 6 out of 7 mistakes found in Silver1 correspond to the pre-annotation's result. Additionally, only 1 out of 8 mistakes made while annotating from scratch correspond to the tagger's result. Thus, annotators who are correcting a tagger's result tend to accept it too often.

	total number of mistakes	mistakes that correspond to tagger's result
annotating with pre-annotation	7	6
annotating from scratch	8	1

Table 1: Annotating with pre-annotation vs. annotating from scratch considering the overlap between the annotators' mistakes and those made by the TreeTagger.

Visualizing the results in a Confusion Matrix

Confusion matrices are used to visualize the problematic cases for a machine learning system. These matrices compare actual results to the system's predicted values. In PoS tagging it can be used to see which PoS tags differ often from a given gold standard. One axis refers to the actual correct PoS tag, whereas the other axis refers to the predicted values. Tags that have an accuracy of 100% or tags that never occur neither in the predicted nor in the actual result, might not be taken into account. The matrix in Figure 2 shows that the following cases tend to be hard to annotate correctly for the IMS TreeTagger. These cases might be revised in future manual corrections of IMS TreeTagger results.

- VVINF↔VVFIN (at the end of the sentence)
- ADV↔ADJD
- KOKOM↔KOUS
- PIAT↔PIDAT

Caret (Classification And REgression Training) [Kuhn, 2008] is a package for the programming language R which contains functions for training and plotting classification and regression models. Caret includes a function that creates a confusion matrix and calculates associated statistics such as the overall accuracy. The visualization package Ggplot2 [3] can be used to plot different *geoms* (geometric objects) such as histograms, scatterplots or line graphs. In figure 3 a tile plane was generated to visualize the confusion matrix. The brightness of each tile indicates how often a predicted value occurs for a certain actual tag on a percentage basis. The darker the diagonal line, the more accurate are the predicted results.

[3] Information about ggplot2 can be found on `www.ggplot2.org`

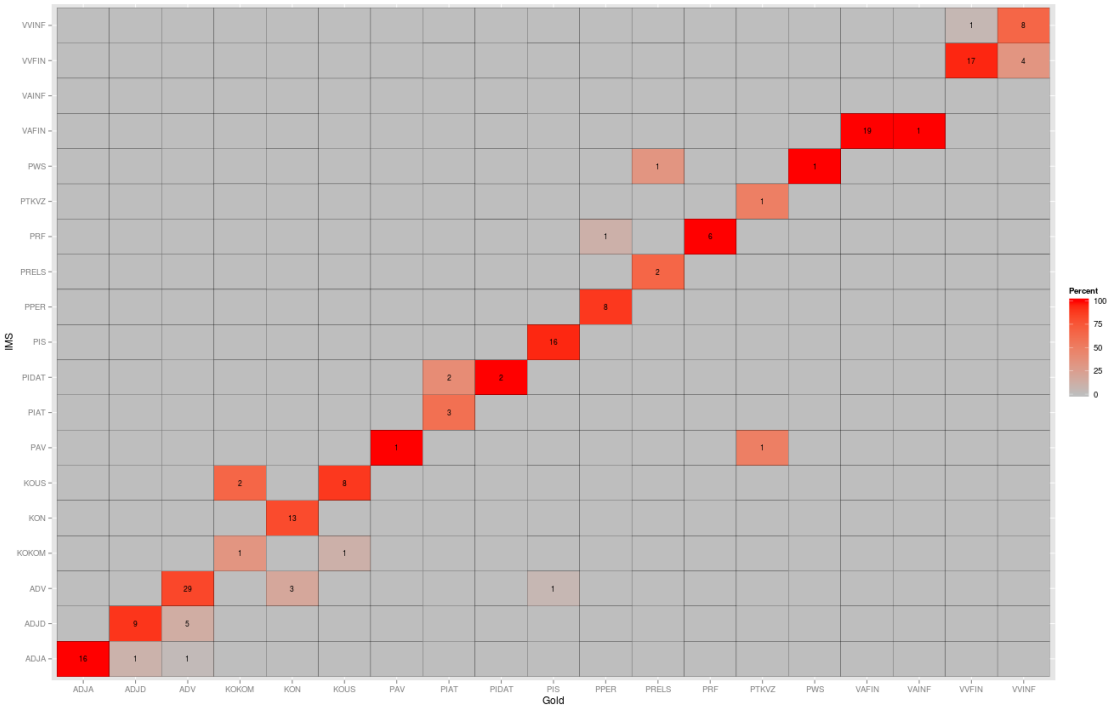

Figure 2: The IMS part of speech tagger with respect to the gold standard.

2.2 Tagging the whole dataset

To tag the whole dataset we first consider possible PoS tagging tools for the STTS tagset to use as a part of the tagging ensemble. Below it is documented which cases can occur when three different taggers are compared to each other with respect to the gold standard of the first 20 sentences. It will be explained how the second silver standard was created with the help of the ensemble results and manual revision. As shown in figure 1, the consent found among the two silver standards finally lead to the gold standard.

Ensemble Tagging and possible PoS tools

As shown previously pre-annoations by a PoS tagging tool can affect the decisions made by annotators adversely. The goal was to avoid errors caused by pre-annotations, without annotating the whole dataset from scratch. With ensemble tagging the tokens that should be tagged from scratch can be picked out. Having given the output of 3 different PoS tagging tools, it can look for those tokens which were not tagged identically. These tokens should then be tagged manually by an annotator who does not see any pre-annotation information. All the other tokens are considered to be tagged correctly by the ensemble with a high probability. In the end, all the tags that are known to be difficult for PoS-taggers, are revised manually.

Choosing the taggers. Ensemble tagging requires different PoS tagging tools which tag already tokenized data. Therefore it is important that the tokenization process is either not included or can be disabled. In addition, the underlying tagset should be STTS for all the three taggers and the output should have been tested resulting in high accuracy. The three chosen taggers are the

```
1  library(ggplot2)
2  library(caret)
3  # Read in the data
4  ...
5  # Create a confusion matrix using Caret
6  > caretMatrix <- confusionMatrix(data$Predicted, data$Actual)
7  # Get and remove all tags that are not interesting
8  > byClassPruned <- byClass[which((byClass$"Pos Pred Value"!=1) |
       (byClass$"Neg Pred Value"!=1) | (byClass$"Specificity"!=1) ),]
9  > data=data[which(data$Actual %in% interestingTags | data$Predicted
       %in% interestingTags),]
10 # Get the actual frequency for every tag
11 > actual <- as.data.frame(table(data$Actual))
12 ...
13 # Get all combinations and their frequencies
14 > confusion <- as.data.frame(table(data$Actual, data$Predicted))
15 ...
16 # Merge the two data frames by the common column "Actual"
17 > confusion <- merge(confusion, actual, by = c("Actual"))
18 # Calculate the percentages of agreement for every tag
19 > confusion$Percent <- confusion$Freq/confusion$ActualFreq*100
```

Figure 3: A code snippet that shows how a confusion matrix can be drawn in R using the packages Caret and Ggplot2.

IMS TreeTagger (Markov Model), the Berkeley Parser [4] and the Stanford PoS tagger (Maximum Entropy) [Toutanova et al., 2003].

Preparing the input. The input text which is already tokenized by the IMS TreeTagger is present in a file with one token per line as well as sentence boundary tags. The Berkeley Parser and the Stanford PoS Tagger accept an input file which contains one sentence per line with tokens seperated by whitespaces. Thus, the IMS tokenized output had to be transformed into the appropriate format. A python script was written to replace newlines by whitespaces and sentence boundary tags by newlines.

Transforming the output. The output format of the Stanford Tagger can be formatted in the same way as the TreeTagger output. With the Berkeley parser it is not possible to do only the PoS assignment step. Thus, the output is one fully parsed sentence per line represented in a bracketing structure. The output had to be reformatted into one sentence per line with sentence boundary tags included. This was done with the help of a python script. Furthermore other tags were used to avoid parentheses because of the bracketing structure (e.g. The tag $(became $*LRB*).

[4] Information about the Berkeley Parser can be found via the following link: `http://nlp.cs.berkeley.edu/Software.shtml`

Generating the ensemble column. The three different output columns are put into one file. Afterwards, a python script generates a fourth column. If the taggers agree with each other, the result is just copied to that fourth column. If they do not agree with each other, the line remains empty and needs to be filled out manually by an annotator.

3 Tokenization and PoS tagging in the Genre of Vernacular Computer-mediated Communication

The project DIDI (Digital Natives - Digital Immigrants) investigates linguistic phenomena of South Tyrolean writer's of various age on social network sites. As a pretest study, public data from a flirt page on Facebook called *Spotted Südtirol* was analyzed. The dataset consisting of about 5400 tokens was manually translated from the South Tirolean dialect into standard German according to certain conventions. The developed approach of ensemble tagging in KOKO could be adopted for the PoS tagging of this project. Written python scripts were adapted to achieve a high accuracy of PoS tags.

My task was to document which kind of linguistic phenomena seem typical for the underlying genre. The language is characterized by dialectal phenomena on word and on sentence level. Additionally there occurred phenomena that are typical for spoken language rather than typical for written language. Furthermore, phenomena such as the usage of hyperlinks and emoticons are caused by the type of communication, namely so called computer mediated communication (CMC). Some of these phenomena cause difficulties for automatic PoS taggers. One problem of the PoS tagging procedure is the TreeTagger's tokenization, which is used for the whole ensemble. Emoticons conceptually represent single tokens. The TreeTagger is not intended to be used for CMC data and tends to tokenize them wrong. As shown in the following examples, emoticons are either split up into separate tokens or are not separated from preceding or following tokens.

- haha^^ → haha^^ *it stays as one token*
- :)) → ;)) *is split up into three tokens*
- :** → : ** *is split up into two tokens*

To solve the problem I tried two different approaches in python. The first approach was looping through the file and making use of the file object's control functions. With the tell()-method you can store the position you are at, when there is a character that could be the start of an emoticon (e.g. at a colon). Afterwards you can check the following lines for matching characters (e.g. one or more closing parenthesis, what than forms the smiling emoticon ":))"). In case of a match, the newlines are removed and the reader continues with the following lines. In case of a mismatch the pointer goes back to the stored position with the seek()-method. With this approach I had to make sure that the longest match is caught and the script had to know the maximal length of an emoticon. That is why I decided for another approach using regular expressions. The code snippet in figure 3 shows how a regular expression can be written automatically haven given a

list of typical emoticons. This regular expression can be used to delete newlines if a word ends with an emoticon. An advantage of this approach is that regular expressions can easily allow for modifications of emoticons such as using more than one bracket for one emoticon. As shown in the snippet in line 11, the uppercase Ds and round brackets can be repeated infinitely by just adding the sign "+" to the regular expression. In this case, this is done by calling the helper function *makeIterable()*. As an example, the expression ":(D)+" matches the emoticon ":D" as well as the emoticon ":DDDDD". The *breakdown()* method is needed to take into account every possible type of error or in other words every way of splitting the emoticons in multiple partitions. Furthermore it is an advantage that the expressions always catch the longest match. The result was tested with the diff-command in the linux bash. It can show in which lines two files differ from each other.

```
1  import re
2  [...]
3  #Create a regular expression as a string to merge symbols of emoticons
4  mergeRegexAsString="("
5  for e in emoList:
6    #handle all possible splitting errors: ";\n)\n)", ";)\n)", ...
7    for combination in breakdown(e):
8      #escape character if needed and make "D", "(" and ")" iterable
           with the "+" sign
9      for character in combination:
10       character = re.escape(character)
11       character =
             re.sub('\\\\\\\(\\\n|\\\\\\\)\\\n|D\\\n',makeIterable,character)
12       mergeRegexAsString+=character+"\n"
13     mergeRegexAsString+="|"
14 mergeRegexAsString=mergeRegexAsString[:-1]+")"
15
16 #Helper: Replace all the linebreaks in the matching object and put one
       at its end. This is done if the mergeRegex in  line 23 matches
17 def mergerepl(matchobj):
18   num=len(matchobj.group(0))
19   return matchobj.group(0).replace('\n','') + "\n"
20
21 #compile and apply regexes
22 mergeRegex=re.compile(mergeRegexAsString)
23 text=re.sub(mergeRegex, mergerepl, text)
```

Figure 4: This code snippet is taken from a python script that improves the tokenization of emoticons. A list of typical emoticons is required. The script will additionally recognize modified forms of these emoticons.

4 Building a Learner Corpus

For a project of the Institute for Education of the University of Bozen/Bolzano I was asked to transform documents written by university students into a corpus on which linguistic analysis can be performed. The underlying data were 148 essays as answers to 3 different assignments. The files containing the essays were of different document types, such as docx files, odt files and others. My work was consisting of the following steps:

- Manually assigning a basic xml structure
- Anonymizing the files
- Adapting the xml structure EURAC's typical learner corpus format

Manually assigning a basic structure. The data was different to other EURAC corpora because the original essays were not handwritten but typed in using different text editors. The format of the files was formatting settings. Furthermore the structure of the content was different. I manually inserted basic xml tags, such as <text>, <title> or <exercsie>.

Anonymizing the files. A python script replaces all the names found in a certain list by a code. In the end it makes sure that all the names contained in that list do not occur in the xml structure anymore. If a certain name is still left anywhere in the data, the script prints a warning message. This is important because the students do not put their names in a consistent and remarkable way.

Adapting the xml structure to EURAC's typical learner corpus format. To be able to reuse existing tools for learner corpora, the format of the data should be similar to existing corpora. An existing xsd file contains an XML Schema and is used to check if an xml document is valid against that schema. As already mentioned, the format differs in some points from the typical structure, thus the xsd-file had to be modified. To reformat the data I used python's minidom package. The codes snippet in figure 5 shows how a tree can be built reading an xml document and then be modified. Due to the fact that the data was not handwritten there where two types of paragraph markers. As seen in the example (figure 5), this information will be stored as an attribute of the paragraph element. Afterwards we validated the file against the modified xsd file.

5 Conclusion and Personal Statement

During the internship I could contribute independently to ongoing research projects. Programming tutorials but especially explanations from my tutor and other colleagues helped me improving the following programming skills:

- programming in python (basics, file I/O, xml parsing with mindom and etree, argument handling, regular expression handling, understanding data structures while playing with ipython)
- programming in R (basic data structures, visualization with the package ggplot)

```
import xml.dom.minidom
[...]
#parse the xml-file and store the information in a minidom tree
tree = xml.dom.minidom.parseString(string.encode('utf8'))
[...]
#rename all the hard paragraph skips to "par" with the attribute
    type="hard"
for hardS in tree.getElementsByTagName("hardSkip"):
  hardS.tagName = "par"
  hardS.setAttribute("type", "hard")
#rename all the soft paragraph skips to "par" with the attribute
    type="soft"
for softS in tree.getElementsByTagName("manualLineBreak"):
  softS.tagName = "par"
  softS.setAttribute("type", "soft")
[...]
#print the modified tree to the output file
tree.writexml(fileWriter,'    ','\n')
```

Figure 5: This snippet shows parts of a script that is used to reformat an xml-file using python's minidom package

- experiences with LINUX as operation system and using the bash shell (file I/O, more, diff, grep, ...)
- experiences with basic tasks in computational linguistics (e.g. encoding problems, documentation of workflows, debugging, installing/downloading packages or IDEs, deciding about importance, ...)

Apart from the improvement of specialised skills, I could gain experience in everyday working life. In the beginning of the internship I could inform the colleagues about my interests and about what I have done so far in a presentation. I could always propose my own ideas and got constructive criticism. The colleagues were open to answer any type of technical questions, but also questions concerning the free time and everyday life in Bolzano.

References

Max Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28 (5):1–26, 2008.

Helmut Schmid. Improvements in part-of-speech tagging with an application to german. In *In Proceedings of the ACL SIGDAT-Workshop*. Citeseer, 1995.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.