Using Machine Learning to Annotate Data for NLP Tasks Semi-Automatically

Gerhard B van Huyssteen **Martin J Puttkammer** **Suléne Pilon** **Handré J Groenewald**

CTexT (Centre for Text Technology), North-West University,
Potchefstroom, 2520
{Gerhard.VanHuyssteen; Martin.Puttkammer; Sulene.Pilon; Handre.Groenewald}@nwu.ac.za

Abstract

Developing digital resources is an expensive and time-consuming endeavour, especially in the case of less-resourced languages. We developed TurboAnnotate in an attempt to accelerate the annotation of linguistic data by means of bootstrapping linguistic data for machine-learning purposes. The design and functionality of the tool is given to show how machine learning is used in the annotation process. It is shown that TurboAnnotate does not only promise to help increase the accuracy of human annotators, but also to save enormously on human effort in terms of time.

Keywords
Machine learning, bootstrapping, linguistic data, TurboAnnotate, Afrikaans, Setswana

1. Introduction

Less-resourced (a.k.a. resource-scarce) languages can generally be defined as "languages for which few digital resources exist; and thus, languages whose computerization poses unique challenges. [They] are languages with limited financial, political, and legal resources…" [4] Digital resources are very important for a less-resourced language with a growing HLT industry, and a useful and effective way of developing such resources semi-automatically, is by means of bootstrapping.

However, less-resourced languages also often lack human resources, but this aspect seldom receives the necessary attention in research or discussions. It is often hard to find computational linguists working on these languages and therefore the task of annotating training data and training a machine-learning algorithm often falls to linguists or mother-tongue speakers who are computer literate. Unfortunately, these individuals often lack the necessary skills to collaborate in the computerization of less-resourced languages.

This lack in skills could cause delays in projects, since large amounts of data need to be annotated manually because the annotators do not have the necessary skills to use bootstrapping and engineers are too busy to constantly train new, improved versions of an application. The manually annotation process also necessitates extra quality control, since the large amounts of data being annotated often lead to annotation errors.

Since South Africa has ten official less-resourced languages, we had to find a way to empower linguists and mother-tongue speakers with the necessary tools to deliver high quality annotated data in the shortest possible time. We conducted interviews with several annotators to determine what they required and then investigated ways to create user-friendly environments (i.e. software with graphical user interfaces - GUIs) for escalating the annotation of linguistic data by mother-tongue speakers with little or no experience in computational linguistics. We also looked at ways of automating machine learning and annotation of new data by means of bootstrapping. Similar approaches include Abdennadher, *et al*. [9] who uses bootstrapping to expedite the annotation of speech data for emotion detection and Brants and Plaehn [10] who uses Markov Models in a bootstrapping process to annotate syntactic data.

The interface presented here allows users to annotate data, train a machine-learning algorithm, annotate new data with the trained algorithm and check the annotated data manually. All that is required of the user is to click on the correct buttons. An algorithm is trained and new data is annotated automatically. The newly annotated data is then presented to the user who, in turn, can correct mistakes made by the machine learner.

We developed *TurboAnnotate* (version 1.0.0) to simplify the process of creating a machine learning classifier for NLP tasks. We tested *TurboAnnotate* by developing hyphenated data for Afrikaans and Setswana as well as data for an Afrikaans compound analyser. In Section 2 we discuss some general points of departure, based on our requirement analyses. Section 3 describes the system in detail, while Section 4 presents some results. This article concludes with ideas on future implementations.

2. Requirements

Our central point of departure in this project is that, for any given language, annotators (e.g. linguists, mother-tongue speakers with a linguistic sensitivity, or student assistants) are invaluable resources towards the creation of digital resources for that language. Based on our experiences in the past with less-resourced languages, it is often the case that such annotators mostly have word processing skills in a GUI-based environment, and not necessarily advanced skills in a computational or programming environment. In worst cases, annotators sometimes have difficulties with file management, unzipping, proper encoding of text files, etc. The aim of this project is thus to maximize their experience, by enabling them to focus on what they are good at: enriching data with their expert linguistic knowledge, while processes that can aid them in the annotation (i.e. bootstrapping) and eventually training a machine learner with the annotated data should occur automatically.

In order to determine how we could enable annotators (i.e. determine end-user requirements) we conducted unstructured interviews with four annotators who are currently working on some of our data annotation projects (usually working in simple text editors or spreadsheet programs). We asked two basic questions: (1) What do you find unpleasant about your work as an annotator?; and (2) What will make your life as an annotator easier?

All of them find the repetitiveness of annotation work rather tedious and boring. Although they have a passion for linguistics and "enjoy working with language", they often feel "useless" because they do not "see the bigger picture". Also related to this, is the fact that they do not see results: "We send our work off to the developers, only to hear weeks or months later what the results were."

With regard to question (2), the respondents described their ideal working environment as a "friendly" environment (i.e. GUI-based, and not lists of words) and they prefer to work in chunks, finishing off "bite-sizes of data" rather than "endless lists". They find it easier and quicker to correct data (i.e. verification), than to annotate from scratch. They just want to click or drag: "…no moving around with arrows, or inserting symbols with difficult hand-eye-coordination". Reference works (such as Google, online dictionaries, etc.) need to be readily available – preferably without starting up various other applications. They also have difficulty with managing their work in terms of version control, saving, etc.

The above dislikes and preferences were analysed and rendered into end-user requirements, which steered the design and development of our annotating system, called *TurboAnnotate*.

3. Using TurboAnnotate

TurboAnnotate is a user-friendly annotating environment (i.e. tool) for bootstrapping linguistic data for machine-learning purposes, or for manually creating gold standards or other annotated lists. In as such, the design of *TurboAnnotate* was inspired by two other annotating tools, *viz. DictionaryMaker* [8] and *Alchemist* [1].

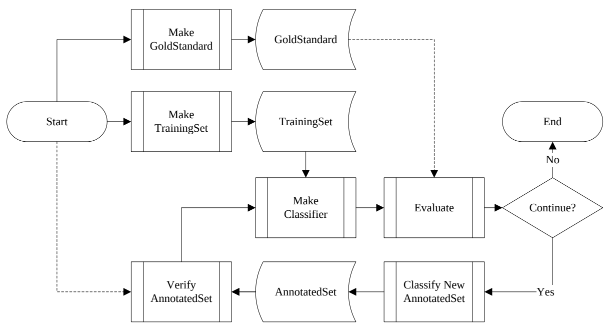
DictionaryMaker is a pronunciation dictionary bootstrapping system, allowing "a speaker fluent in the target language to develop a pronunciation dictionary without requiring expert linguistic knowledge or programming expertise" [8]. The backbone of the bootstrapping system is the *Default&Refine* algorithm, which is used for rule extraction [6]. Positive results are reported in [7], indicating "[f]or a 10,000 word dictionary, the bootstrapping approach requires 23% of the effort of the manual approach."

Alchemist is a GUI-based tool, designed to create morphological gold standards for text data [3]. While it was initially conceptualised as a tool to create data for machine learning purposes, its wider applicability for field linguists and in teaching was soon realized. In a computational linguistic environment, "*Alchemist* increases the efficiency of the human researcher and shortens the time of creating grammars, interlinear texts, and dictionaries" [3].

Drawing on the experiences of the aforementioned programs, as well as our end-user requirements, we developed a first version of *TurboAnnotate*, specifically with the task of hyphenation and compound analyses for South African languages in mind.

Essentially, *TurboAnnotate* implements three steps, which are next discussed in more detail (see Figure 1).

Figure 1: Simplified workflow of TurboAnnotate



3.1. Create gold standard

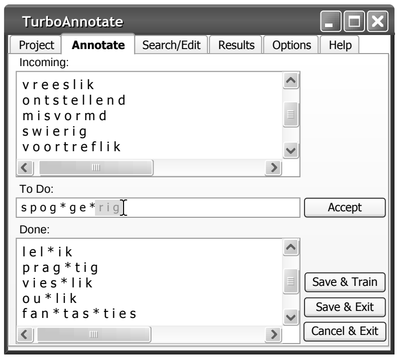
After a new project has been created, the first step is to create a gold standard – i.e. an independent test set that will be used for evaluating the performance of classifiers. Note that an annotator only has to select one file (the data file/base list); the rest of file management is handled by the tool. A random list of instances for the gold standard is automatically extracted from a base list (not indicated in Figure 1).

The size of the gold standard is by default set on a 1,000 words, but could be changed under the "Options" tab. Although this might seem like a relatively small evaluation set, it was decided to keep it rather small in order to make it more manageable for annotators. More-over, an exploratory experiment with a training set of 42,967 hyphenated Afrikaans words indicated an accuracy of 99.20% on an evaluation set of 2,200 words, and 99.60% accuracy on an evaluation set of 1,000 words. It thus led us to believe that, at least for the hyphenation task, a gold standard of a 1,000 words would suffice, without presenting a completely skewed view of the actual performance of the system.

3.2. Create training set

Before the bootstrapping process can commence (step 3), a training set to train a first classifier has to be created. Once again, based on the annotators request for "bite-sizes of data", we decided to experiment with very small sets of training data. The default size is therefore set on 200 words, which could also be changed under "Options". Data is being stored automatically both in annotated format (together with the original word-form, in two columns), and in a format suitable for the machine learning system.
The annotation GUI (see Figure 2) is to some extent inspired by *Alchemist*: the annotator simply drags the mouse over the part of the word to be annotated, and on release of the mouse button, the selection changes colour, and an asterisk is inserted after the selection (in the case of data for hyphenation). Alternatively, there is also an option available to use the keyboard only for annotation. When annotating data for purposes of compound analyses, the user clicks on a location in the word and presses F8 to insert a plus (+) to indicate a word boundary, or F9 to insert an underscore to indicate a valence morpheme. In the "Search" tab (not shown here), s/he can also search for words in training sets that have already been annotated.

Figure 2: Screenshot: annotation environment



v r e e s l i k
o n t s t e l l e n d
m i s v o r m d
s w i e r i g
v o o r t r e f l i k

s p o g * g e * r i g

l e l * i k
p r a g * t i g
v i e s * l i k
o u * l i k
f a n * t a s * t i e s

The machine learning system that we use in our system is the well-known Tilburg Memory-Based Learner (TiMBL;[12]). The choice for TiMBL is based on its wide success and applicability in the field of natural language processing, its availability for research purposes, as well as its relative ease to use. On the down-side, it is widely known that memory-based machine learning systems perform best with large quantities of data, which is an undesirable characteristic for purposes of developing data for less-resourced languages; however, it will be indicated that, at least for the tasks of hyphenation and compound analyses, TiMBL performs quite well, even with little data available.
After a classifier has been trained, it is evaluated against the complete gold standard to determine its accuracy. For hyphenation and for compound analyses, accuracy is determined on word-level (i.e. all hyphens, or word and morpheme boundaries in a word), and not per correct instance; however, the user has the option to change this way of evaluation under the "Option" tab. Simplified results are displayed in the "Results" tab (not shown here), and the user has the choice to save data and exit, or to continue with the annotation process.

The features that are used when training a machine learning algorithm will be discussed briefly in the following subsection, after which we will indicate how parameter optimisation is done without any input from the user.

3.2.1. Features

In order to train and test the classifier, all input words are converted into feature vectors. This is done by splitting the word into letters and using a shifting window method with a context of three positions to the left and three to the right. This means that every instance contains seven values. The first six values denote the context and the final value indicates the class. In this case, the class is an indication of whether or not a break is present. In the case of compound analysis, there are three possible classes: a plus sign indicates a word boundary, an underscore indicates a valence morpheme and an equals sign indicates that no break is present. The Afrikaans word *besigheidsure* (business hours) will be annotated as follows:

besigheid _ s + ure
'business' _ 'valency' + 'hours'.

The compound *eksamen + lokaal* ('examination room') is converted into the feature vectors depicted in Table 1 (using a context of three positions to the left and three to the right). Note that the equals sign in the context region denotes a missing value, whilst the equals sign in the class region denotes that no break is present at the current position. Table 1 shows that there should only be a breakpoint between -men- and -lok- (indicated by "+"). No additional breaks should be inserted at any position between the other characters in the word.

Ambiguity in compound boundaries can be divided into five groups. The first group contains compounds with truly ambiguous boundaries, e.g. *nagaap* which can be analysed as *nag* + *aap* ('night' + 'monkey') or *na* + *gaap* ('after' + 'gawk'). Compounds with one correct, lexicalised analysis and one uncommon analyses fall into the second group.

An example of a compound belonging to this group is *wolfrokkie* with lexicalised analysis *wol* + *frokkie* (‘wool’ + ‘vest’) and uncommon analysis *wolf* + *rokkie* (‘wolf’ + ‘little dress’). The third group contains stems that can erroneously be analysed as compounds, e.g. *malaria* (‘malaria’) which can be analysed into the words *mal* (‘crazy’) and *aria* (‘aria’). The fourth group of ambiguous compounds are actually inflectional forms that can erroneously be analysed into two parts. An example is the derivative *brander* + *tjie* (‘wave’ + ‘DIM’) which can be analysed into the words *brand* (‘burn’) + *ertjie* (‘pea’). The last group contains derivatives that appear to be compounds, e.g. *gunsteling* (‘favourite’) can be divided into the words *guns* (‘favour’) and *teling* (‘breeding’).

The words belonging to groups two to five should not be analysed and should pose no problem for the annotators checking the data. Thus, group one (compounds with more than one correct and frequently used analysis) is the only ambiguous group that could be problematic. Since we use memory based learning to train the classifiers, it is not desirable to have a string with the same features and different classes, e.g. the third position in the string *nagaap* (i.e. after the letter *g*) cannot have a class ‘+’ and a class ‘=’ for analysing the word as *nag* + *aap* and *na* + *gaap* respectively. Therefore, only one of these possible analyses can be included in the data. We decided to give the classifier the benefit of the doubt in these cases and if it presents the annotator with a correct and acceptable analysis, the analysis should not be changed. If such a compound is not correctly analysed, the annotator should analyse the word in the first manner that comes to mind.

Table 1: Feature vector for *eksamenlokaal*

Left context			**⬇ Right Context**			**Class**
=	=	=	e	k	s	=
=	=	e	k	s	a	=
=	e	k	s	a	m	=
e	k	s	a	m	e	=
k	s	a	m	e	n	=
s	a	m	e	n	l	=
a	m	e	n	l	o	=
m	e	n	l	o	k	+
e	n	l	o	k	a	=
n	l	o	k	a	a	=
l	o	k	a	a	l	=
o	k	a	a	l	=	=
k	a	a	l	=	=	=
a	a	l	=	=	=	=

For the task of hyphenation, the same approach is followed, the only difference being that there are only two classes in the training data, an asterisk indicating a break and an equals sign indicating that no break is present. The example above will be hyphenated as *be * sig * heids * u * re*.

3.2.2. Parameter optimisation

Large fluctuations in generalization accuracy can be observed when changing parameter settings of memory-based learning algorithms like TiMBL [11]. One way of finding the best algorithmic parameter settings is to perform an exhaustive search throughout all of the valid combinations of parameter settings. This approach is however mostly not desirable, as it is computationally expensive and time-consuming to experiment with all the combinations on the full dataset.

As an alternative approach to an exhaustive search, Van den Bosch [2] developed a tool (Paramsearch) that produces combinations of algorithmic parameters that are estimated to deliver best results. In order to determine the algorithmic parameter combinations that deliver the best performance in applications using machine learning algorithms, we developed and use PSearch. PSearch basically operates on the same principles as the original Paramsearch, with the major difference being the way that the sizes of the training and evaluation sets are generated. In addition, PSearch supports all of the TiMBL classification algorithms, as well as C4.5 [5]. PSearch is only implemented after all the data has been annotated, ensuring the best classifier possible for the specific set of training data available and took 2 hours and 21 minutes using PSearch to complete on the compound data.

3.3. Verify annotated text

If the annotation GUI was inspired by *Alchemist*, then this step was inspired by *DictionaryMaker*, since it represents the iterative bootstrapping phase in the design.

New data is sourced from the base list (not shown in Figure 1), in chunks determined under the "Options" tab (default is 200 words). This data is automatically annotated by the trained classifier (created in the previous step), and presented to the human annotator in the "Annotate" tab.

Next, the annotator has to verify whether the annotated word is correct by clicking on "Accept". Alternatively, s/he can correct the input in the same manner as explained before. Verification, compared to annotating from scratch, seems to have a positive effect on the human effort required (in terms of time, at least), as well as on the accuracy of the human annotators – see Table 3 below.

The verified data next serves as training data for developing a subsequent classifier. However, all previously created and verified training sets (i.e. excluding the gold standard) are automatically merged to form one single training set. The training data for each subsequent classifier is thus incrementally more, with a predicted increase in accuracy.

TurboAnnotate runs under the Linux environment and requires Perl 5.8 and Gtk+ 2.10 or later. TiMBL (version 5.1) also needs to be installed, and the bin directory needs to be added to the PATH environment variable. Source code is freely available under an open-source license at http://www.nwu.ac.za/ctext.

4. Evaluation

We evaluate our project in terms of two general criteria; *viz.* accuracy, and human effort (in terms of time). All evaluations are done on Afrikaans and Setswana (in the case of hyphenation), involving four human annotators in total; two of the annotators are well-experienced in annotating, while the other two could be considered novices in the field. Our results and main findings are discussed in the following subsections.

4.1. Accuracy

For evaluating our effort, we are interested in two kinds of accuracy: (1) classifier accuracy (i.e. how accurate is a classifier, and how much data is needed to approach the gold standard?), and (2) human accuracy (i.e. does TurboAnnotate contribute towards more accurate annotations by human annotators?). Accuracy is expressed as a percentage of correctly annotated words over the total number of words (either the gold standard, or another portion of data, depending on the evaluation). In all evaluations of classifiers, the gold standard was excluded as training data.

Table 2: Classifier accuracy (Hyphenation)

Training data	Accuracy (Afrikaans)	Accuracy (Setswana)
200	38.60%	94.50%
600	54.00%	98.30%
1000	58.30%	98.80%
2000	68.50%	98.90%

Table 2 clearly shows a huge difference between hyphenation for Afrikaans and Setswana: with only 200 training words, the Setswana classifier already reaches an accuracy of 94.50%, while the Afrikaans classifier only reaches 38.60%. This can be ascribed to the highly systematic CV-syllable structure of Setswana, thus proving that hyphenation for Setswana is rather trivial (a rule-based hyphenator with even less human effort would probably result in even better accuracy scores). For Afrikaans, however, the improvement is significant: after only 2,000 annotated words, a human annotator would have to correct only 3 out of every 10 words, thus promising less human effort towards the development of a state-of-the-art hyphenator (even more so when larger training sets will be used).

To determine human accuracy, we created two previously unseen datasets of 200 words each for each language. The first dataset was annotated by each annotator in an ordinary text editor, by inserting a chosen symbol between syllables. The second dataset was annotated using TurboAnnotate. For the hyphenation task (Hyph), the mean accuracy of the annotators on the first dataset was 93.25%, while on the second dataset 98.34% - a difference of more than 5% (see Table 3). With regards to compound analysis (CA), the mean accuracy of the annotators was 91.5% on the first dataset and 94% on the second. While this may seem insignificant, it could have an important impact when annotating large datasets for machine learning purposes.

It thus proves that TurboAnnotate could not only ensure higher accuracy in human annotations, but could also save on human effort required (at least in the case of Afrikaans). The latter aspect is discussed in the next section.

Table 3: Comparison of human accuracy & effort

Annotation Tool	Accuracy (Hyph)	Time (s) (Hyph)	Accuracy (CA)	Time (s) (CA)
Text Editor (200 words)	93.25%	1325	91.50%	802
TurboAnnotate (200 words)	98.34%	1258	94.00%	748

4.2. Human effort

To determine whether *TurboAnnotate* would be beneficial in terms of time saved on annotation, we asked two questions: (1) Is it faster to annotate with *TurboAnnotate*, than with a text editor?; and if so, (2) What would be the predicted saving on human effort on a large dataset?

From Table 3 it is evident that it is more than 1 minute faster to annotate 200 words for hyphenation with *TurboAnnotate*, than with a text editor. On a larger dataset, say of 40,000 words, that would mean a small difference of only circa 3.5 uninterrupted human hours.

Table 4: Human effort using *TurboAnnotate*

Training Data	Time (s) (Hyph)	Time (s) (CA)
0	1258	748
600	663	614
2000	573	582

If one takes the effect of bootstrapping into consideration, a different picture arises. In the fourth iteration of the bootstrapping process for hyphenation (i.e. after the classifier was trained with 600 words), annotation time was measured, as well as after the last iteration (i.e. after the classifier was trained with 2,000 words). Annotators were not made aware of time measurements, thus not influencing their focus on accuracy. Table 4 represents the mean results, indicating that the bootstrapping process has a significant positive impact on human effort.

If we extrapolate this to a dataset of 42,967 words then using *TurboAnnotate* could save almost 51 uninterrupted human hours when annotating data for an Afrikaans hyphenator – a saving of almost 68%. In the case of compound analysis, the bootstrapping process in *TurboAnnotate* already leads to a saving of more than 9 uninterrupted human hours on a dataset of 40,000 words - a saving of more than 41%.

5. Conclusion

In this paper, we have presented the design and functionality *TurboAnnotate,* our free and open-source tool, which is used for creating gold standards and annotating lexical data through bootstrapping. We indicated that empowering linguists, mother-tongue speakers and student assistants by giving them access to this technique saves a large amount of valuable time and improves accuracy.

Future work includes extending the tool's capability to be scaled easily for other lexical annotation tasks, such as creating lexicons for spelling checkers (by using n-grams instead of machine learning to predict whether a word is correctly spelled), and creating data for other forms of morphological analysis. We will also focus on further improving the GUI, as well as a network solution for multi-users.

We will also extend *TurboAnnotate* through active learning in order to decrease the annotation effort even further. This will imply the selection of certain training instances to act as training data in order to optimise prediction accuracy, rather than relying on random samples.

Instead of TiMBL, we will also experiment with C5.0, since commercial licenses for the latter is rather affordable, an aspect that is relevant for the commercial exploitation of HLTs, especially in developing countries.

6. References

[1] Alchemist v2.0: A GUI-based tool for analyzing morphemes and creating morphological gold-standards in XML format. [Online document], [cited Apr 26, 2004], Available HTTP: http://linguistica.uchicago.edu/alchemistv2_0_final.pdf.

[2] A. van den Bosch. Paramsearch 1.0 Beta Patch 24. [Online document], Mar 2002, [cited Mar 20, 2007], Available HTTP: http://ilk.uvt.nl/software.html/paramsearch.

[3] C. Sprague. Creating Morphological Descriptions with Alchemist. [Online document], [cited Jul 20, 2006], Available HTTP: http://linguistica.uchicago.edu/Sprague_2006.pdf.

[4] E. Garrett. 2nd Call for papers: Resource-scarce language engineering at ESSLLI0 2006. Feb 20, 2006, http://pvs.csl.sri.com/mail-archive/pvs/msg02605.html.

[5] J. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[6] M. Davel and E. Barnard. A default-and-refinement approach to pronunciation prediction. Proc. of the Symposium of the Pattern Recognition Assoc. of South Africa, South Africa, Nov 2004, pp. 119-123.

[7] M. Davel and E. Barnard. Bootstrapping for language resource generation. Proc. of the Symposium of the Pattern Recognition Assoc. of South Africa, South Africa, Nov 2003, pp. 97–100.

[8] M. Davel and M. Peche. DictionaryMaker User Manual, Version 2.0(i). Sep 2006 [Online document], [cited Nov 21, 2006], Available HTTP: http://dictionarymaker. sourceforge.net/.

[9] S. Abdennadher, M. Aly, D. Bühler, W. Minker, J. Pittermann. BECAM Tool – A semi-automatic tool for bootstrapping emotion corpus annotation and management. Proc. of Interspeech 2007 - Eurospeech, 10th European Conference on Speech Communication and Technology. Antwerp, Belgium, Aug 27-31, 2007.

[10] T, Brants and O. Plaehn**.** Interactive Corpus Annotation. Proc. of the Second International Conference on Language Resources and Evaluation (LREC-2000), Athens, Greece, 2000.

[11] W. Daelemans and A. Van den Bosch. Memory-Based Language Processing. Cambridge University Press, New York, 2005.

[12] W. Daelemans, A. Van den Bosch, J. Zavrel and K. Van der Sloot. TiMBL: Tilburg Memory Based Learner, Version 5.1, Reference Guide. ILK Technical Report, Feb 4, 2004.