Variable-Length Markov Models and Ambiguous Words in Portuguese*

Fabio Natanael Kepler
Institute of Mathematics and Statistics
University of Sao Paulo
Sao Paulo, SP, Brazil
kepler@ime.usp.br

Marcelo Finger
Institute of Mathematics and Statistics
University of Sao Paulo
Sao Paulo, SP, Brazil
mfinger@ime.usp.br

Abstract

Variable-Length Markov Chains (VLMCs) offer a way of modeling contexts longer than trigrams without suffering from data sparsity and state space complexity. However, in Historical Portuguese, two words show a high degree of ambiguity: *que* and *a*. The number of errors tagging these words corresponds to a quarter of the total errors made by a VLMC-based tagger. Moreover, these words seem to show two different types of ambiguity: one depending on non-local context and another on right context. We searched ways of expanding the VLMC-based tagger with a number of different models and methods in order to tackle these issues. The methods showed variable degrees of success, with one particular method solving much of the ambiguity of *a*. We explore reasons why this happened, and how everything we tried fails to improve the precision of *que*.

1 Introduction

In the Computational Linguistics area, the task of *part-of-speech tagging* (POS tagging) consists in assigning to words in a text the grammatical class they belong. Since the same word may belong to more than one class, models for POS tagging have to look at the context where each word occurs to try to solve the ambiguity.

Previous and current work have developed a wide range of models and methods for tagging. The vast majority uses supervised learning methods, which need an already tagged corpus as input in order to train the model, calculating relations, weights, probabilities etc.

Among the various models for tagging, there are Maximum Entropy models (dos Santos et al., 2008; de Almeida Filho, 2002; Ratnaparkhi, 1996), Hidden Markov Models (HMMs) (Brants, 2000), Transformation Based Learning (Brill, 1993), and other succesful approaches (Toutanova et al., 2003; Tsuruoka and Tsujii, 2005; Shen et al., 2007).

Current state-of-the-art precision in tagging is achieved by supervised methods. Although precision is pretty high – less than 3% error rate for English – the disavantage is exactly the need of a tagged corpus, usually built manually. This is a very restrictive issue for languages with lack of resources such as linguistic especialists, corpora projects etc.

The Portuguese language falls in between resourceful languages, such as English, and languages with restricted resources. There have been initiatives both in Brazil and in Portugal, which include modern Brazilian Portuguese corpora (ICMC-USP, 2010), European Portuguese corpora (Flo, 2008), and historical Portuguese corpora (IEL-UNICAMP and IME-USP, 2010). Also, some supervised POS taggers have already been developed for Portuguese (dos Santos et al., 2008; Kepler and Finger, 2006; Aires, 2000) with a good degree of success. And finally, there has also been increasing effort and interest in Portuguese annotation tools, such as E-Dictor[1] (de Sousa et al., 2009).

Despite these advances, there is still lack of material and resources for Portuguese, as well as research

*During the course of this work Fabio received support from Brazilian funding agencies CAPES and CNPq.

[1] See http://purl.org/edictor.

Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas, pages 15–23, Los Angeles, California, June 2010.

in unsupervised methods to bootstrap text annotation.

Our work focuses on further improvement of the current state-of-the-art in Portuguese tagging. For this, we focus on the *Tycho Brahe* (IEL-UNICAMP and IME-USP, 2010) corpus for testing and benchmarking, because of its great collaboration potential: it is easily accessible[2]; is under continuous development; and has recently started using E-Dictor, which also offers a great collaboration potential.

1.1 Previous Works

One popular approach to tagging is to use HMMs of order 2. Order 2, or *trigram*, means the tagger considers the previous two words/tags when tagging a word. This adds context to help disambiguation. The drawback is that this context may not be sufficient. Increasing the order does not help, since this incurs in too many model parameters and suffers from the data sparsity problem.

In (Kepler and Finger, 2006), we developed a tagger for Portuguese that uses Markov chains of variable length, that is, orders greater than 2 can be used conditioned on certain tags and sequences of tags. This approach is better at avoiding the sparsity and complexity problems, while being able to model longer contexts. However, one interesting conclusion from that work is that, even using longer contexts, some words stay extremely hard to disambiguate. Apparently, those words rely on flexible contexts not captured by pure VLMCs.

Motivated by this problem, we improve over the previous work, and developed a set of tagger models based on Variable-Length Markov Chains (VLMCs) extended with various other approaches in order to try to tackle the problem.

In the next section we describe the VLMC theory, the results it achieves, and the problems with two common words. Then, in Section 3, we explain in summary the set of models and approaches we tried to mix with VLMCs, and the different types of results they give. Conclusions are drawn in Section 4. Finally, Section 5 describes how this work can be incorporated in other projects, and Section 6 presents ideas for future work.

[2]More information at `http://www.tycho.iel.unicamp.br/~tycho/corpus/en/index.html`.

2 Variable-Length Markov Chains

The idea is to allow the memory of a Markov chain to have variable length, depending on the observed past values. (Bühlmann and Wyner, 1999) give a formal description of VLMCs, while here we will explain them in terms of the POS-tagging task.

Consider a Markov chain with a finite, large order k. Let t_i be a tag, and $t_{i-k,i-1}$ be the k tags preceding t_i. Variable length memory can be seen as a cut of irrelevant states from the $t_{i-k,i-1}$ history. We call the set of these states the *context* of t_i. Given a tag t_i, its context $t_{i-h,i-1}$, $h \leq k$, is given by the *context function* $c(t_{i-k,i-1})$.

A *context tree* is a tree in which each internal node has at most $|\mathcal{T}|$ children, where $\mathcal{T}$ is the tagset. Each value of a context function $c(\cdot)$ is represented as a branch of such tree. For example, the context given by $c(t_{i-k,i-1})$ is represented as a branch whose subbranch at the top is determined by t_{i-1}, the next subbranch by t_{i-2}, and so on, until the leaf, determined by t_{i-h}.

The parameters of a VLMC are the underlying functions $c(\cdot)$ and their probabilities. To obtain these parameters we use a version of the context algorithm of (Rissanen, 1983). First, it builds a big context tree, using a training corpus. For a tag t_i, its maximal history $t_{i-k,i-1}$ is placed as a branch in the tree. Then, the algorithm uses a pruning function considering a local decision criterion. This pruning cuts off the irrelevant states from the tags' histories. For each leaf u in the context tree, and branch v that goes from the root to the parent node of u, u is pruned from the tree if

$$\Delta_{vu} = \sum_{t \in \mathcal{L}} P(t|vu) \log \left(\frac{P(t|vu)}{P(l|v)} \right) C(vu) < K,$$

where $C(vu)$ is the number of occurrences of the sequence vu in the training corpus, and K is a threshold value, called the *cut value* of the context tree,

If the probability of a tag does not change much between considering the entire branch together with the leaf (all past history) and considering only the branch (the history without the furthest tag), then the leaf does not need to be considered, and can be removed from the tree.

We want to find the best sequence of tags $t_1 \dots t_n$ for a given sequence of words $w_1 \dots w_n$ of size n,

that is,

$$\arg\max_{t_1 \ldots t_n} \left[\prod_{i=1}^{n} P(t_i | c(t_{i-k,i-1})) P(w_i | t_i) \right].$$

Probabilities are computed from a tagged training corpus using maximum likelihood estimation from the relative frequencies of words and sequences of tags. The context tree is built with sequences of tags of maximum length k and then pruned, thus defining the context functions. For decoding, the Viterbi Algorithm is used (Viterbi, 1967).

2.1 Initial Results

We used the tagged texts available by the Tycho Brahe Corpus of Historical Portuguese (IEL-UNICAMP and IME-USP, 2010). The Tycho Brahe project uses 377 POS and inflectional tags, and contains annotated texts written by authors born between 1380 and 1845. We have selected 19 texts for composing our corpus, which contains 1035593 tagged words and has 262 different tags. This corpus was then randomly divided into 75% of the sentences for generating a training corpus and 25% for a testing corpus. The training corpus has 775602 tagged words, while the testing corpus has 259991 tagged words. The Tycho Brahe project is undergoing rapid development, so as for today there are more texts available which are not present in the corpus we used[3].

Because of some of the approaches explained below, we also created a new training corpus and a new testing corpus by segmenting contracted words from the original corpus. Contracted words are words like *da*, which has the tag `P+D-F` and is a contraction of the preposition *de* (`P`) with the feminine determiner *a* (`D-F`).

Using the original corpus, our VLMC implementation, which we will call VLMM TAGGER[4] (from *Variable Length Markov Model*), and which better implements under- and overflow control, achieves 96.29% of precision[5], while the VLMC TAGGER from (Kepler and Finger, 2006) achieves 95.51%. Table 1 shows the numbers for both taggers, where P and E means Precision and Error, respectively. The difference in precision is mainly due to a 21.64% error reduction in known words tagging[6]. That, combined with 6.82% error reduction in unknown words, results in 17.50% total error reduction. With the segmented corpus the VLMM TAGGER achieved 96.54% of precision.

TAGGER	WORDS	P (%)	ERR. / OCURR.
VLMC	Unknown	69.53	2713 / 8904
	Known	96.39	9065 / 251087
	Total	95.51	11674 / 259991
VLMM	Unknown	71.60	2528 / 8904
	Known	97.17	7102 / 251087
	Total	96.29	9630 / 259991

Table 1: Precision of VLMC-based taggers.

Table 2 shows numbers for the two words that present the most number of errors made by the VLMM TAGGER. Note that they are not necessarily the words with the highest error percentage, since there are known words that appear only a couple of times in the testing corpus and may get wrong tags half of this times, for example.

WORDS	P (%)	E (%)	ERR. / OCURR.
que	84.7413	15.2586	1687 / 11056
a	90.9452	9.0547	661 / 7300

Table 2: Results for words with the most number of errors using the VLMM TAGGER with the normal corpus.

These two words draw attention because together they correspond to almost 25% of the errors made by the tagger, where most confusion for each of these words is between two different tags:

- The word *que* is, most of the times, either a relative pronoun – denoted by the tag `WPRO` and

[3] We can provide the training and testing corpus if requested by email.

[4] A package containing the VLMM TAGGER will be available at `http://www.ime.usp.br/~kepler/vlmmtagger/`, but requests for the raw source code can be made by email. Currently, there is only an automake bundle ready for download containing the VLMC TAGGER.

[5] Precision is given by the number of correctly assigned tags to the words in the testing corpus over the total number of words in the testing corpus.

[6] Known words are words that appear both in the training and the testing corpus.

equivalent to the word *which* in English –, or a subordinating conjunction – denoted by the tag `C` and equivalent, in English, to the words *that* or *than*;

- The word *a* is, usually, either a feminine determiner (tag `D-F`), or a preposition (tag `P`).

As a baseline, assigning the most common tag to *que* yields a precision of 55.64%, while *a* gets a precision of 58.09%. Also, these words seem to show two different types of ambiguity: one that needs context to the right, and one that needs non-local context. The VLMM model does not have parameters for these contexts, since it tags from left to right using context immediately to the left.

2.2 Objectives

It seems that *a* could be better disambiguated by looking at words or tags following it: for example, if followed by a verb, *a* is much more likely to be a preposition. For *que*, it seems that words occuring not immediately before may add important information. For example, if *que* follows *mais* (*more than*, in English), it is more likely that *que* has tag `C`. However, like in the English expression, it is possible to have various different words in between *mais* and *que*, as for example: "*mais provável que*" ("*more likely than*"); "*mais caro e complexo que*" ("*more expensive and complex than*"); and so on. Thus, it may yield better results if non-local context could be efficiently modeled.

In order to develop these ideas about *que* and *a* and prove them right or wrong, we searched ways of expanding the VLMM tagger with a number of different models and methods that could help solving these two issues. Those models are described next.

3 Auxiliary Approaches

3.1 Syntactic Structure

The first idea we had was to generalize nodes in the VLMM's context tree, that is, to model a way of abstracting different sequences of tags into the same node. This could make it possible to have branches in the context tree like `ADV * C`, that could be used for *mais* * *que*.

One way of doing this is to use sequences of tags that form phrases, like noun phrases (`NP`), prepositional phrases (`PP`), and verbal phrases (`VP`), and use them in the context tree in place of the sequences of tags they cover. The context tree will then have branches like, say, `P VP N`.

In order to train this mixed model we need a treebank, preferably from the texts in the Tycho Brahe corpus. However, it does not have a sufficiently large set of parsed texts to allow efficient supervised learning. Moreover there is not much Portuguese treebanks available, so we were motivated to implement an unsupervised parsed for Portuguese.

Based on the work of (Klein, 2005), we implemented his CCM model, and used it over the Tycho Brahe corpus. The CCM model tries to learn constituents based on the contexts they have in common. We achieved 60% of f-measure over a set of texts from the Tycho Brahe project that were already parsed.

Using the CCM constituents learned, we extended the VLMM TAGGER to use this extra information. It yielded worse results, so we restricted the use of constituents to *que* (the VLMM+SPANS-QUE TAGGER). This yielded a precision of 96.56%, with a *que* precision increase of 3.73% and an *a* precision reduction of 0.67%. A comparison with the plain VLMM TAGGER over the segmented corpus can be seen in Table 3. We use the segmented corpus for comparison because the constituents only use segmented tags. Even after many tries and variations in the way the VLMM TAGGER could use constituents, the result did not improve. This led us to a new approach, shown in the next section.

WORDS	P (%)	ERR. / OCURR.
que	*84.50*	*1715 / 11063*
	85.18	1651 / 11063
a	*94.52*	*745 / 13597*
	94.49	750 / 13597
Total	*96.5433*	*9559 / 276541*
	96.5636	9503 / 276541

Table 3: Comparison of precision using the VLMM TAGGER (in italics) and the VLMM+SPANS-QUE TAGGER (upcase) with the segmented corpus.

3.2 Chunks

Since induced syntactic structure did not help, a new idea was to, this time, begin with the already parsed and revised texts from the Tycho Brahe, even with they summing only a little more than 300 thousand words. To ease the problem of sparsity, the trees were flattened and merged in such a way that only NPs, PPs and VPs remained. Then the bracketed notation was converted to the IOB notation, now forming a chunked corpus.

Chunking, or *shallow parsing*, divides a sentence into non-overlapping phrases (Manning and Schütze, 1999). It is used in information extraction and in applications where full parsing is not necessary, offering the advantage of being simpler and faster.

We made a small experiment with the chunked corpus: divided the sentences randomly into 90% and 10% sets, the former for training and the later for testing. Then we ran the VLMM TAGGER with these chunked sets, and got a precision in chunking of 79%.

A model for chunks processing was mixed into the VLMM model, similar but not equal to the mixed model with CCM. The chunked corpus uses segmented words, because the parsed texts available in Tycho Brahe only use segmented words. Thus, we ran the VLMM TAGGER with the segmented training corpus and the chunked corpus, testing over the segmented test corpus. The precision yielded with this VLMM+CHUNKS TAGGER was 96.55%.

Table 4 shows the results for the segmented corpus with the VLMM TAGGER and the VLMM+CHUNKS TAGGER. Interestingly, results did not change much, in spite of the VLMM+CHUNKS TAGGER achieving a higher precision. Interestingly, the word *a* error rate is reduced by around 13% with the help of chunks, while the *que* error rate increases almost 3%.

WORDS	P (%)	ERR. / OCURR.
que	*84.50*	*1715 / 11063*
	84.05	1764 / 11063
a	*94.52*	*745 / 13597*
	95.26	644 / 13597
Total	*96.5433*	*9559 / 276541*
	96.5506	9539 / 276541

Table 4: Comparison of precision using the VLMM TAGGER (in italics) and the VLMM+CHUNKS TAGGER (upcase) with the segmented corpus.

3.3 Bidirectional

Another approach was to follow the intuition about *a*: that the right context should help solving some ambiguities. The problem that makes this approach non trivial is that a right tag context is not yet available when tagging a word, due to the natural left-to-right order the tagger follows when tagging a sentence. A right context that is available is the context of words to the right, but this presents the problem of sparsity and will probably not yield good results.

Our approach was then to model a right context of tags when the words to the right were not ambiguous, that is, if they could be assigned only one specific tag. During training, a new context tree is built for the right context, where, for each word in a sentence, a continuous but variable-length sequence of tags from unambiguous words to the right is added as a branch to the right context tree. That is, if k words to right of a given word are not ambiguous, then the sequence of the k tags these words will have is added to the right tree. The right context tree is also prunned like the left context tree and the Viterbi algorithm for tagging is adapted to consider these new parameters.

WORDS	P (%)	ERR. / OCURR.
que	*84.74*	*1687 / 11056*
	84.80	1680 / 11056
a	*90.94*	*661 / 7300*
	92.15	573 / 7300
Total	*96.29*	*9630 / 259991*
	96.33	9544 / 259991

Table 5: Comparison of precision using the VLMM TAGGER (in italics) and the VLMM+A-RIGHT TAGGER (upcase) with the normal corpus.

After various tests with different options for the right context tree, the result over the original VLMM tagger did not improve. We then experimented building the right context tree only for the word *a*,

resulting in the VLMM+RIGHT-A TAGGER. Table 5 shows what happens with the normal corpus.The error rate of *a* is decreased almost 5% with this bidirectional approach.

3.4 Perceptron

The Perceptron algorithm was first applied to POS-tagging by (Collins, 2002). It is an algorithm for supervised learning that resembles Reinforcement Learning, but is simpler and easier to implement.

(Collins, 2002) describes the algorithm for trigram HMM taggers. Here, we will describe it for the VLMM tagger, adapting the notation and explanation.

Instead of using maximum-likelihood estimation for the model parameters, the perceptron algorithm works as follows. First, the model parameters are initialized to zero. Then, the algorithm iterates a given number of times over the sentences of the training corpus. For each sentence s, formed by a sequence of words w^s paired with a sequence of tags t^s, the Viterbi decoding is ran over w^s, returning z^s, the predicted sequence of tags. Then, for each sequence of tags o of length at most k, k the maximum order of the VLMC, seen c_1 times in t^s and c_2 times in z^s, we make $\alpha_{c(o)} = \alpha_{c(o)} + c_1 - c_2$. $c(o)$ is the context function defined in Section 2 applied to the tag sequence o, which returns the maximum subsequence of o found in the context tree. $\alpha_{c(o)}$ represents the parameters of the model associated to $c(o)$, that is, the branch of the context tree that contains $c(o)$.

The above procedure effectively means that parameters which contributed to errors in z^s are penalized, while parameters that were not used to predict z^s are promoted. If $t^s = z^s$ then no parameter is modified. See (Collins, 2002) for the proof of convergence.

Implementing the perceptron algorithm into the VLMM tagger resulted in the VLMM+PERCEPTRON TAGGER. Table 6 shows the results obtained. Note that no prunning is made to the context tree, because doing so led to worse results. Training and predicting with a full context tree of height 10 achieved better precision. The numbers reported were obtained after 25 iterations of perceptron training. The total precision is lower than the VLMM TAGGER's precision, but it is interesting to note that the precision for *que* and *a* actually increased.

WORDS	P (%)	ERR. / OCURR.
que	*84.74*	*1687 / 11056*
	85.15	1641 / 11056
a	*90.94*	*661 / 7300*
	92.41	554 / 7300
Total	*96.29*	*9630 / 259991*
	95.98	10464 / 259991

Table 6: Comparison of precision using the VLMM TAGGER (in italics) and the VLMM+PERCEPTRON TAGGER (upcase) with the normal corpus.

3.5 Guided Learning

(Shen et al., 2007) developed new algorithms based on the easiest-first strategy (Tsuruoka and Tsujii, 2005) and the perceptron algorithm. The strategy is to first tag words that show less ambiguity, and then use the tags already available as context for the more difficult words. That means the order of tagging is not necessarily from left to right.

The inference algorithm works by maintaining hypotheses of tags for spans over a sequence of words, and two queues, one for accepted spans and one for candidate spans. Beam search is used for keeping only a fixed number of candidate hypotheses for each accepted span. New words from the queue of candidates are tagged based on their scores, computed by considering every possible tag for the word combined with all the available hypotheses on the left context and on the right context. The highest scoring word is selected, the top hypotheses are kept, and the two queues are updated. At each step one word from the queue of candidates is selected and inserted in the queue of accepted spans.

The core idea of Guided Learning (GL) training is to model, besides word, tag, and context parameters, also the order of inference. This is done by defining scores for hypotheses and for actions of tagging (actions of assigning a hypothesis). The score of a tagging action if computed by a linear combination of a weight vector and a feature vector of the action, which also dependes on the context hypotheses. The score of a given span's hypothesis is the sum of the scores of the top hypothesis of the left and right con-

texts (if available) plus the score of the action that led to this hypothesis.

The GL algorithm estimates the values of the weight vector. The procedure is similar to the inference algorithm. The top scoring span is selected from the queue of candidate spans and, if its top hipothesis matches the gold standard (the tags from the training corpus), the queues of accepted and candidate spans are updated as in the inference algorithm. Otherwise, the weight vector is updated in a perceptron style by promoting the features of the gold standard action and demoting the features of the top hypothesis' action. Then the queue of candidate spans is regenerated based on the accepted spans.

This model uses trigrams for the left and right contexts, and so it could be potentially extended by the use of VLMCs. It is our aim to develop a tagger combining the VLMM and the GL models. But as for today, we have not yet finished a succesful implementation of the GL model in C++, in order to combine it with the VLMM TAGGER's code (current code is crashing during training). Original GL's code is written in Java, which we had access and were able to run over our training and testing corpora.

Table 7 shows the result over the normal corpus. The first thing to note is that the GL model does a pretty good job at tagging. The precision means a 10% error reduction. However, the most interesting thing happens with our two words, *que* and *a*. The precision of *que* is not significantly higher. However, the error rate of *a* is reduced by half. Such performance shows that the thought about needing the right context to correctly tag *a* seems correct. Table 8 shows the confusion matrix of the most common tags for *a*.

WORDS	P (%)	ERR. / OCURR.
que	*84.74*	*1687 / 11056*
	84.90	1670 / 11056
a	*90.94*	*661 / 7300*
	95.49	329 / 7300
Total	*96.29*	*9630 / 259991*
	96.67	8650 / 259991

Table 7: Comparison of precision using the VLMM TAGGER (in italics) and the GUIDED LEARNING TAGGER (upcase) with the normal corpus.

	D-F	P	CL
D-F	<4144>	92	5
P	189	<2528>	2
CL	26	9	<294>

Table 8: Confusion matrix for *a* with the most common tags in the normal corpus (line: reference; column: predicted).

4 Conclusions

In almost all extended versions of the VLMM TAGGER, *que* and *a* did not suffer a great increase in precision. With the approaches that tried to generalize context – by using syntactic structure – and capture longer dependencies for *que*, the results did not change much. We could see, however, that the right context does not help disambiguating *que* at all. Training the VLMM model with a long context (order 10) helped a little with *a*, but showed overall worse results. Modeling a right context for *a* in a simple manner did also help a little, but not significantly. The model that gave good results for *a* was the one we still have not finished extending with VLMM. It looks promising, but a way of better disambiguating *que* was not found. A better approach to generalize contexts and to try to capture non-local dependencies is needed. Some further ideas for future work or work in progress are presented in Section 6.

5 Oportunities for Collaboration

Tycho Brahe is a corpus project undergoing continuous development. Since there is already a good amount of resource for supervised tagging, our tagger can be used for boosting new texts annotation. Furthermore, the project has started using E-Dictor, an integrated annotation tool. E-Dictor offers a range of easy to use tools for corpora creators: from transcription, philological edition, and text normatization, to morphosyntactic annotation. This last tool needs an integrated POS-tagger to further ease the human task of annotation. Besides, an increasing number of projects is starting and willing to start using E-Dictor, so the need for an automatic tagger

is getting urgent. We have already been contacted by the E-Dictor developers for further collaboration, and should integrate effors during this year.

Another project that can benefit from a good POS-tagger is the *Brasiliana Digital Library*, from the University of Sao Paulo[7]. It started last year digitalizing books (and other pieces of literature) about Brazil from the 16th to the 19th century, making them available online. Many books have been OCRed, and a side project is already studying ways of improving the results. Since the library is an evolving project, the texts will soon be of reasonable size, and will be able to form another corpus of historical Portuguese A POS-tagger will be of great help in making it a new resource for Computational Linguistics research. We are already negotiating a project for this with the Brasiliana directors.

There is a tagger for Portuguese embedded in the CoGrOO[8] gramatical corrector for Open Office. They seem to implement some interesting rules for common use Portuguese that maybe would help some of our disambigation problems. Besides inspecting the available open source code, we have contacted the current maintainer for further conversation. A possibility that has appeared is to integrate the VLMM TAGGER with CoGrOO.

Using different data would be interesting in order to check if the exactly same problems arise, or if other languages show the same kind of problems. We will try to get in contact with other projects having annotated resources available, and seek for further collaboration. Currently, we got in touch with people working on another corpus of Portuguese[9]. Both sides are hoping to form a partnership, with us providing a POS tagger and them the annotated corpora.

6 Future Work

Short term future work includes implementing Guided Learning in C++ and mixing it with VLMCs. This looks promising since the current GL implementation uses a fixed trigram for contexts to the left and to the right. Also, there is a need for fast execution in case our tagger is really integrated into E-Dictor, so converting GL to C++ seems more natural than implementing the VLMM TAGGER in Java.

To try to tackle the difficulty in tagging *que* there are some ideas about using context trees of non-local tags. It seems a potentialy good model could be achieved by mixing such context trees with the Guided Learning approach, making a hypothesis consider non adjacent accepted spans. This is still a fresh idea, so further investigation on maybe other approaches should be done first.

Further investigation involves analyzing errors made by POS taggers over modern Portuguese and other romance languages like Spanish in order to verify if *que* and *a* continue to have the same degree of ambiguity or, in case of Spanish, if there are similar words which show similar issues. This also involves testing other taggers with our training and testing sets, to check if they get the same errors over *que* and *a* as we did.

[7] `http://www.brasiliana.usp.br/bbd`

[8] `http://cogroo.sf.net/`.

[9] *History of Portuguese spoken in São Paulo (caipira Project).*

References

Rachel Virgínia Xavier Aires. 2000. Implementação, adaptação, combinação e avaliação de etiquetadores para o português do brasil. mathesis, Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo - Campus São Carlos, Oct.

Thorsten Brants. 2000. Tnt – a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, WA.

Eric Brill. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*.

Peter Bühlmann and Abraham J. Wyner. 1999. Variable length markov chains. *Annals of Statistics*, 27(2):480–513.

Michael Collins. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.

Archias Alves de Almeida Filho. 2002. Maximização de entropia em lingüística computacional para a língua portuguesa, 12.

Maria Clara Paixão de Sousa, Fábio Natanael Kepler, and Pablo Picasso Feliciano de Faria. 2009. E-Dictor: Novas perspectivas na codificação e edição de corpora de

textos históricos. In *Linguística de Corpus: Sínteses e Avanços. Anais do VIII Encontro de Linguística de Corpus*, UERJ, Rio de Janeiro, RJ, Brasil, 11. Shepherd, T. and Berber Sardinha, T. and Veirano Pinto, M. To be published.

Cícero Nogueira dos Santos, Ruy L. Milidiú, and Raúl P. Rentería. 2008. Portuguese part-of-speech tagging using entropy guided transformation learning. In *PROPOR - 8th Workshop on Computational Processing of Written and Spoken Portuguese*, volume 5190 of *Lecture Notes in Artificial Intelligence*, pages 143–152, Vitória, ES, Brazil. Springer-Verlag Berlin Heidelberg.

Linguateca.pt, 2008. *The Floresta Sintá(c)tica project*.

ICMC-USP, 2010. *NILC's Corpora*. ICMC-USP.

IEL-UNICAMP and IME-USP, 2010. *Córpus Histórico do Português Anotado Tycho Brahe*. IEL-UNICAMP and IME-USP.

Fábio Natanael Kepler and Marcelo Finger. 2006. Comparing two markov methods for part-of-speech tagging of portuguese. In Jaime Simão Sichman, Helder Coelho, and Solange Oliveira Rezende, editors, *IBERAMIA-SBIA*, volume 4140 of *Lecture Notes in Artificial Intelligence*, pages 482–491, Ribeirão Preto, Brazil, 10. Springer Berlin / Heidelberg.

Dan Klein. 2005. *The Unsupervised Learning of Natural Language Structure*. phdthesis, Stanford University.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations Of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*, University of Pennsylvania, 5.

Jorma Rissanen. 1983. A universal data compression system. *IEEE Trans. Inform. Theory*, IT-29:656 – 664.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic, 6. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 173–180, Morristown, NJ, USA. Association for Computational Linguistics.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 467–474, Morristown, NJ, USA. Association for Computational Linguistics.

Andrew James Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimal deconding algorithm. *IEEE Transactions on Information Theory*, pages 260 – 269, 4.