AUTOMATIC MORPHOLOGICAL ANALYSIS FOR RUSSIAN: A COMPARATIVE STUDY

Dereza O. V. (oksana.dereza@gmail.com),
Kayutenko D. A. (kayutenko@mail.ru),
Fenogenova A. S. (alenush93@gmail.com)
National Research University Higher School of Economics, Moscow, Russia

Abstract

In this paper we present a comparison of ten systems for automatic morphological analysis: TreeTagger, TnT, HunPos, Lapos, Citar, Morfette, Mystem, Pymorhy, Stanford POS tagger and SVMTool. Different training and tagging approaches are discussed together with the strengths and weaknesses of each system. Probabilistic taggers were trained and tested on the Russian National Disambiguated Corpus and achieved accuracy scores as high as 96,94% on POS tags and 92,56% on the whole tagset. However, most of the existing taggers cannot resolve various cases of morphological ambiguity and show a better performance for morphologically rich languages. We believe that the detailed examination of errors caused by homonymy can help to solve the disambiguation problem and to improve tagging results.

Keywords: automatic morphological analysis, POS tagging, disambiguation, taggers

АВТОМАТИЧЕСКИЙ МОРФОЛОГИЧЕСКИЙ АНАЛИЗ ДЛЯ РУССКОГО ЯЗЫКА: СРАВНИТЕЛЬНЫЙ АНАЛИЗ СИСТЕМ

Дереза О. В. (oksana.dereza@gmail.com),
Каютенко Д. А. (kayutenko@mail.ru),
Феногенова А.С. (alenush93@gmail.com)
НИУ ВШЭ, Москва, Россия

Ключевые слова: автоматический морфологический анализ, частеречная разметка, дизамбигуация, таггеры

1. Introduction

The number of systems for automatic morphological analysis with a free license, both statistical and dictionary-based, has reached a point where a thorough comparative analysis of their performance on the same data is absolutely necessary. In this article, we discuss several taggers and various approaches to POS tagging with their strengths and weaknesses. We trained and tested probabilistic taggers on the Russian National Disambiguated Corpus, a fully annotated corpus for the Russian language; as for dictionary-based analyzers mentioned in this article, we used their versions inherently developed for Russian. In Section 2 we discuss different taggers and algorithms they use, in Section 3 we describe the tests we run and the results we received, and in Section 4 we outline the major types of mistakes made by TreeTagger, which is the winner of our competition.

2. Taggers

For our research, we tested ten part-of-speech taggers, which are all free licensed and for the most part open source: TreeTagger, TnT, HunPos, Lapos, Citar, Morfette, Mystem, Pymorhy, Stanford POS tagger and SVMTool.

2.1. TreeTagger

TreeTagger (Schmid 1994) is an HMM-based tagger, which differs from the traditional probabilistic taggers in the way it estimates the transition probabilities. In contrast to the N-gram taggers that typically use smoothing to deal with the sparsity of the training data, TreeTagger uses a binary decision tree, built recursively from a training set of trigrams, to obtain estimates of transition probabilities. The decision tree automatically determines the appropriate size of the context which is used to estimate the probabilities. Possible contexts are not only N-grams, but also other kinds of contexts such as e.g. (tag_{-1} = ADJ and tag_{-2} != ADJ and tag_{-2} != DET). The tagger is also capable of assigning tags to unknown words using a suffix lexicon organized in a tree for this purpose. At the tagging stage user can provide the tagger with lists of tokens to be added to the initial lexicon and choose the values to assign to tokens with zero frequencies. At the training stage user can provide the tagger with additional lexicon lists.

2.2. TnT

Trigrams'n'Tags (TnT) is a statistical part-of-speech tagger based on second order Markov models. It uses linear interpolation of unigrams, bigrams and trigrams as a smoothing technique for contextual probabilities, where probability distribution coefficients λ_1, λ_2 and λ_3 are estimated by deleted interpolation. The algorithm consists in calculating maximum likelihood estimations for a sequence of words $w_i \ldots w_T$ of length T, where $t_1 \ldots t_T$ are elements of the tagset, and t_{-1}, t_0 and t_{T+1} are beginning- and end-of-sequence markers (Brants 2000: 224).

$$\underset{t_1 \ldots t_T}{\operatorname{argmax}} \left[\prod_{i=1}^{T} P(t_i|t_{i-1}, t_{i-2}) P(w_i|t_i) \right] P(t_{T+1}|t_T)$$

The core of TnT tagging algorithm is Viterbi beam search. Unknown words are handled with the help of suffix analysis proposed in (Samuelson, 1993); the list of suffixes is derived from a lexicon of rare words with a frequency less than a set value at the training stage. In addition to that, the tagger treats uppercase and lowercase words differently at both stages. TnT has a wide range of adjustable parameters: user can tune smoothing and suffix analysis techniques, choose how to handle unknown words and mark them in the output, get all the possible tags for a token with a probability higher than a given value etc.

2.3. Hunpos

Hunpos is an OCaml re-implementation of TnT functionality that shows competitive results both in accuracy and speed. It is also built on HMMs that estimate N-gram probabilities for a given sequence of words, but in contrast to TnT, available only as an executable, Hunpos has an open source code. Unlike traditional HMM-based taggers, Hunpos computes emission probabilities using both current and previous tags, which makes the error rate 10% lower (Halácsy, Kornai, Oravecz 2004: 210). For unseen words, Hunpos generates all possible labels and then assigns weights to them by the suffix guessing algorithm based on rare word distribution. If the tagger is provided with a full morphological lexicon as proposed in (Banko, Moore 2004), its performance on unseen words significantly improves. Hunpos training options include the order of tag transition probability, the order of emission probability, the maximum frequency of a word to be included in the lexicon and the maximum suffix length.

2.4. Stanford POS tagger

The Stanford Tagger (Toutanova 2003) is a MaxEnt POS tagger, which uses a bidirectional approach to building probabilistic models, i.e. it makes explicit use of both preceding and following tag contexts by means of a bidirectional dependency network representation. Additionally, it uses broad lexical features by conditioning on multiple consecutive words, i.e. the word itself, the preceding and the following words, which allows the model to learn facts about the frequent

idiomatic word sequences. Quadratic regularization (Gaussian prior smoothing) is used to deal with overtraining. It also introduces a fine-grained modeling of unknown word features.

2.5. SVMTool

SVMTool (Giménez, Marquez 2004) is a highly configurable and easy to use language independent tagger based on Support Vector Machines. It implements a one-vs-all strategy where separate SVM classifiers are trained for each tag and the most confident tag from all the binary classifiers is selected while tagging. SVMTool can learn both from supervised and unsupervised data. For supervised learning it is possible to provide the tagger with additional information apart from the token and its tag. The learning process is controlled by means of a configuration file, which allows the user to specify such parameters as the size of the context window, the set of features used, feature filtering and SVM model compression, dictionary repairing, etc. It is also possible to provide lists of ambiguous tags and open classes of words and a backup lexicon containing words not present in the training corpus. User can choose between four available models for training. Tagging process is also highly configurable and makes it possible to set the direction, scheme and strategy of tagging, the number of passes, to provide the backup and lemma lexicons etc.

2.6. Lapos

Lapos is a C++ implementation of the perceptron-based POS tagging algorithm described in (Yoshimasa 2011), which uses the lookahead process with a proof of convergence. The training algorithm is an adaptation of margin perceptrons model (Krauth, Mezard 1987) with the difference that Lapos makes use of the states and their scores obtained from lookahead searches. Such an algorithm allows to tune the perceptron weight in such a way that the tagger can correctly choose the right action for the current state at each decision point given the information from the lookahead process. The lookahead mechanism considers possible sequences of future actions and the states realized by those sequences. The performance of Lapos in POS tagging, chunking and NER on English data is claimed to be competitive with state-of-the-art approaches.

2.7. Citar

Citar, a C++ library that provides POS tagging[1], partly implements the algorithm used in TnT. It is also based on a trigram HMM, but with the linear interpolation smoothing. To produce probabilities, the tagger compiles lexicon and N-grams from the training data. The probabilities from the trigram model are smoothed by the interpolation function that tries to find the smooth parametrization of available data and to estimate the results at the intermediate point, which makes Citar both fast and accurate.

2.8. Morfette

Morfette is a data-driven probabilistic system for joint POS tagging and lemmatization developed specially for inflectional and agglutinative languages with rich morphology. It consists of three modules: two Maximum Entropy classifiers that predict morphological tags and lemmas and a decoder that searches for the best sequence of tag-lemma pairs in a given sequence. As the system does not use any finite lexicon, lemma classes are derived automatically and correspond to the shortest edit script between reversed word forms and lemmas. For a focus word w_i in context $c \in C$ for each possible tag $m \in M$ the model returns $p(m|c)$, and for each possible lemma class $l \in L$ the model returns $p(l|c,m)$. The beam search keeps n-best sequences of $(m, l) \in M \times L$ pairs up to the current position in the input sequence. The list of tag probabilities $(m_0, p_0)... (m_j , p_j)$ is sorted in a decreasing order, and then the tags that do not satisfy a certain condition with a threshold value, are filtered out (Chrupała, Dinu, Van Genabith 2008: 2-3). User can add an optional dictionary and

[1] There is also a Java version called Jitar.

select the number of iterations for training the models, and choose the beam size and the number of n-best sequences to keep for tagging. Morfette also has a built-in evaluation module.

2.9. Mystem and Pymorphy2

Mystem (Segalovich 2003) and Pymorphy2 (Korobov 2015) are dictionary based morphological analysis and disambiguation systems widely used in various NLP projects. Mystem, originally developed by Ilya Segalovich for Yandex search engine, is built on Zaliznyak's Russian Grammar Dictionary (in its base version for Russian). It uses a dictionary represented as a set of tries and can guess morphology of unknown words by looking at the closest words in the dictionary. Pymorphy2 is a morphological analyzer and form generator developed by Mikhail Korobov, which uses OpenCorpora dictionaries with a set of linguistically motivated rules developed to enable morphological analysis and generation of out-of-vocabulary words observed in real-world documents. It is worth noting that due to their dictionary-based design these taggers cannot be retrained on a different corpus or dictionary. Both systems have their own tagsets with specific lists of grammemes. Since some of the grammemes used in the RNC are either simplified or absent in these tagsets (e.g. Mystem does not use second dative or second accusative), it is impossible to fully convert them into the RNC format. That is why comparing the performance of these two taggers and the ones we trained on the RNC disambiguated corpus would have been inconsistent. Although Mystem and Pymorphy2 could not take part in the final experiment, we still wanted to present some evaluation of their performance. For this purpose, we compiled a reduced tagset valid for both these taggers and RNC (see conversion table in Appendix I) that helped us to make rough estimations of their accuracy and understand where they stand compared to the other taggers. Pymorphy achieved an accuracy of 90,65% and Mystem – an accuracy of 96,43% on POS tags.

3. Evaluation

3.1. Memory

Some of the systems require such an amount of RAM to train a model that an end user running them on an average machine simply cannot do it. We managed to carry out cross-validation for Stanford POS tagger, SVMTool, Lapos and Morfette only on the 1/12 of our data (500,000 tokens) annotated only with POS tags (except for SVMTool), which is why we could not include these taggers in the final evaluation. However, their performance on the smaller test set was generally good, and the results can be found in Table 1.

Accuracy	TreeTagger	TnT	HunPos	Citar	SVMTool	Stanford	Morfette	Lapos
POS	96,94%	96,19%	96,41%	94,76%	93,43%	95,82%	93,03%	20,07%
All tags	92,56%	89,24%	89,29%	86,10%	86,24%	–	–	–

Table 1

3.2. Time

Train and test time can play the crucial role when one has to deal with large corpora, especially if morphological analysis is just one of the modules in a bigger system. Though taggers based on MaxEnt and SVM yield comparable or even slightly better results, than HMM-based taggers, their train/test cycle is orders of magnitude longer. Another major factor that affects the tagger's speed is the implementation language: obviously, C and C++ taggers are much faster than others. TreeTagger appears to be the best system in this respect: it requires about 13 seconds to train a POS-only model

on the ⅘ of 6 million corpus and about 9 seconds to tag the remaining ⅕ on an average machine. Although other HMM-based taggers outperform it in train time on the whole tagset, it is still the fastest in tagging. The average train and test time of other systems compared to TreeTagger is given in Table 2.

	TreeTagger	TnT	HunPos	Citar	SVMTool	Stanford	Morfette	Lapos
Approach	HMM, decision tree	HMM	HMM	HMM	SVM	MaxEnt	MaxEnt, average perceptron	Margin perceptron, look ahead
Language	C++, Perl	ANSI C	OCaml	C++	C++, Perl	Java	Haskell	C++
Train (POS)	~ 12,78 sec	× 1,5	× 5,5	× 0,8	× 1150,0	× 800,0	× 1550,0	× 1120,0
Tag (POS)	~ 8,62 sec	× 2,0	× 3,0	× 1,5	× 8,0	× 15,0	× 560,0	× 2000,0
Train (All)	~ 601,59 sec	× 0,05	× 0,3	× 0,05	× 25,0	–	–	–
Tag (All)	~ 32,33 sec	× 1,5	× 2,5	× 5,5	× 20,0	–	–	–

Table 2

3.3. Comparative results

In the final experiment, four taggers – TreeTagger, TnT, HunPos and Citar – were trained and tested on the Russian National Disambiguated Corpus of 6 million tokens, and the results were evaluated by 5-fold cross-validation. Average accuracy scores for POS tags and the whole set of tags are given in Table 3.

	TreeTagger	TnT	HunPos	Citar
Accuracy (POS)	96,94%	96,19%	96,41%	94,76%
Accuracy (All tags)	92,56%	89,24%	89,29%	86,10%

Table 3

Although TreeTagger shows the best results on POS tags, the performance of other analyzers is less than a percent lower. When it comes to the complete morphological annotation, the gap between the taggers drops down to 0,2%. Such results are not surprising due to the similar base of algorithms used by all four taggers.

4. Error analysis

As the analysis of tagger's errors can help a lot in improving its performance, we chose the tagger that achieved the highest score – TreeTagger – and examined its results, focusing on mistakes connected with homonymy. Such mistakes were grouped into three categories: caused by paradigmatic ambiguity (tokens with similar lemmas and word forms), caused by inter-word ambiguity (tokens with different lemmas but the same word forms) and caused by joint paradigmatic and inter-word ambiguity. The distribution of mistakes for POS tags for inter-word ambiguity, paradigmatic ambiguity and joint ambiguity classes are shown in Figure 1, Figure 2 and Figure 3 respectively (see Appendix II). The most common mistakes (correct tag: tagger's output) are displayed along the horizontal axis, while the vertical axis shows their frequencies. Figures 4, 5 and 6 represent the same for the whole tagset. The diagrams show that a very large number of mistakes

is connected with homonymy, which turns out to be mostly paradigmatic homonymy for inflected words. A more detailed analysis of these mistakes, as done in (Sharoff 2015) for Mystem and TnT-Russian, is a subject of future research, aimed at developing a tagger with a powerful disambiguation module.

5. Conclusion

In a series of tests on a corpus of 6 million tokens the highest accuracy of 96,94% on POS tags and of 92,56% on the whole tagset was achieved by TreeTagger. However, it has problems with various cases of morphological ambiguity as well as the other systems. We believe that the examination of errors caused by homonymy that we made will help us to develop a disambiguation algorithm, which will improve tagging quality for Russian.

References

1. Banko M., Moore R. C. (2004), Part of speech tagging in context, Proceedings of the 20th international conference on Computational Linguistics. – Association for Computational Linguistics, pp. 556.
2. Brants T. (2000), TnT: a statistical part-of-speech tagger, Proceedings of the sixth conference on Applied natural language processing. – Association for Computational Linguistics, pp. 224-231.
3. Chrupała G., Dinu G., Van Genabith J. (2008), Learning morphology with morfette.
4. Giménez J., Marquez L. (2004), Fast and accurate part-of-speech tagging: The SVM approach revisited, Recent Advances in Natural Language Processing III, pp. 153-162.
5. Halácsy P., Kornai A., Oravecz C. (2007), HunPos: an open source trigram tagger, Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions. – Association for Computational Linguistics, pp. 209-212.
6. Krauth W., Mézard M. (1987), Learning algorithms with optimal stability in neural networks, Journal of Physics A: Mathematical and General. Vol. 20. – №. 11., pp. L745.
7. Ratnaparkhi A. et al. (1996), A maximum entropy model for part-of-speech tagging, Proceedings of the conference on empirical methods in natural language processing. Vol. 1. – pp. 133-142.
8. Samuelsson C. (1993), Morphological tagging based entirely on Bayesian inference, 9th Nordic conference on computational linguistics.
9. Schmid H. (1994), Probabilistic part-of-speech tagging using decision trees, Proceedings of the international conference on new methods in language processing. Vol. 12. – pp. 44-49.
10. Seddah D. et al. (2010), Lemmatization and lexicalized statistical parsing of morphologically rich languages: the case of French, Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages. – Association for Computational Linguistics, pp. 85-93.
11. Segalovich I., 2003, A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine, MLMTA, pp. 273-280.
12. Toutanova K., Klein D., Manning C., and Singer Y. (2003), Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. Proceedings of HLT-NAACL 2003, pp. 252-259.
13. Tsuruoka Y., Miyao Y., Kazama J. (2011), Learning with lookahead: can history-based models rival globally optimized models? Proceedings of the Fifteenth Conference on Computational Natural Language Learning. – Association for Computational Linguistics, pp. 238-246.
14. Korobov M. (2015), Morphological Analyzer and Generator for Russian and Ukrainian Languages, Analysis of Images, Social Networks and Texts, pp. 320-332.
15. Шаров С.А., Беликов В.И., Копылов Н.Ю., Сорокин А.А., Шаврина Т.О. (2015), Корпус с автоматически снятой морфологической неоднозначностью: к методике лингвистических исследований.

Appendix I

Converted Mystem Tags	
Mystem tag	**Resulting tag**
ADVPRO	ADV-PRO
ANUM	A-NUM
APRO	A-PRO
SPRO	S-PRO
Converted RNC tags	
RNC tag	**Resulting tag**
PRAEDIC-PRO	PRAEDIC

Converted Pymorphy2 Tags	
Pymorphy tag	**Resulting tag**
NOUN	S
ADJF	A
ADJS	A
COMP	A
VERB	V
INFN	V
PRTF	V
PRTS	V
GRND	V
NUMR	NUM
ADVB	ADV
NPRO	S-PRO
LATN	NONLEX
NUMB, intg	NUM

UNKN	NONLEX
Converted RNC Tags	
RNC tag	**Resulting tag**
A-PRO	A
ADV-PRO	ADV

Appendix II

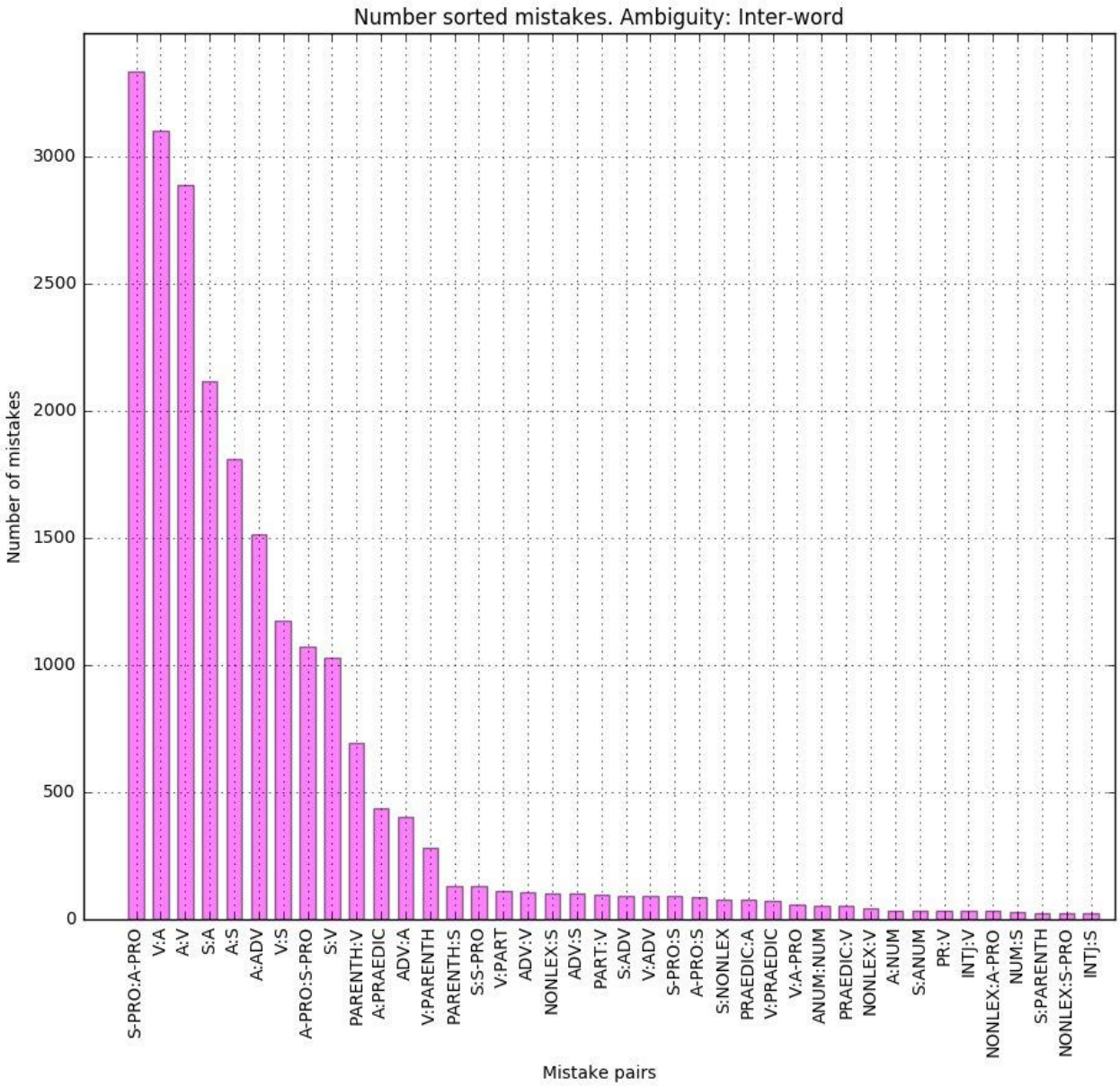

Figure 1

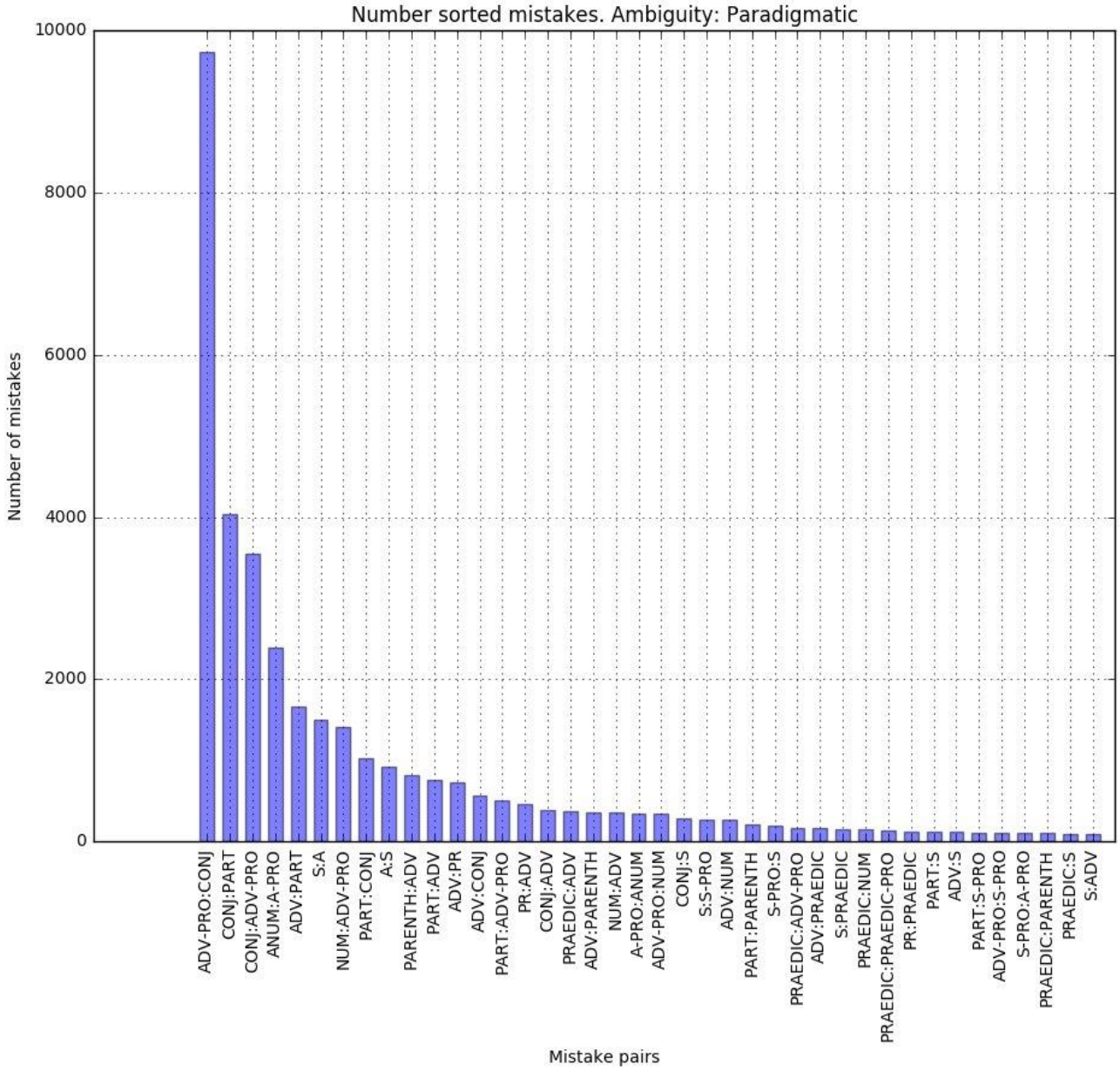
Number sorted mistakes. Ambiguity: Paradigmatic
Number of mistakes
10000
8000
6000
4000
2000
0
ADV-PRO:CONJ
CONJ:PART
CONJ:ADV-PRO
ANUM:A-PRO
ADV:PART
S:A
NUM:ADV-PRO
PART:CONJ
A:S
PARENTH:ADV
PART:ADV
ADV:PR
ADV:CONJ
PART:ADV-PRO
PR:ADV
CONJ:ADV
PRAEDIC:ADV
ADV:PARENTH
NUM:ADV
A-PRO:ANUM
ADV-PRO:NUM
CONJ:S
S:S-PRO
ADV:NUM
PART:PARENTH
S-PRO:S
PRAEDIC:ADV-PRO
ADV:PRAEDIC
S:PRAEDIC
PRAEDIC:NUM
PRAEDIC:PRAEDIC-PRO
PR:PRAEDIC
PART:S
ADV:S
PART:S-PRO
ADV-PRO:S-PRO
S-PRO:A-PRO
PRAEDIC:PARENTH
PRAEDIC:S
S:ADV
Mistake pairs

Figure 2

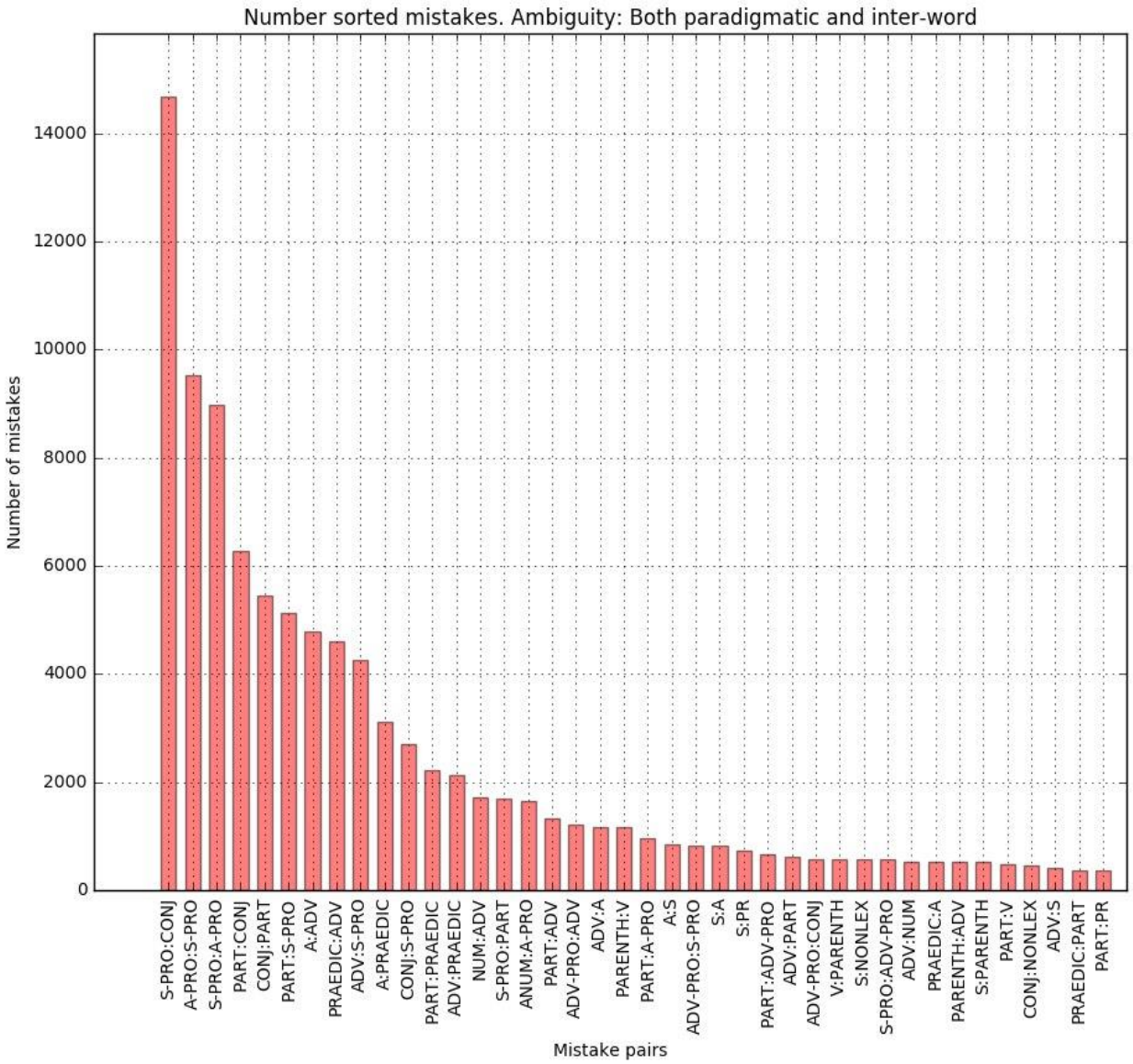
Number sorted mistakes. Ambiguity: Both paradigmatic and inter-word
Number of mistakes
0
2000
4000
6000
8000
10000
12000
14000
S-PRO:CONJ
A-PRO:S-PRO
S-PRO:A-PRO
PART:CONJ
CONJ:PART
PART:S-PRO
A:ADV
PRAEDIC:ADV
ADV:S-PRO
A:PRAEDIC
CONJ:S-PRO
PART:PRAEDIC
ADV:PRAEDIC
NUM:ADV
S-PRO:PART
ANUM:A-PRO
PART:ADV
ADV-PRO:ADV
ADV:A
PARENTH:V
PART:A-PRO
A:S
ADV-PRO:S-PRO
S:A
S:PR
PART:ADV-PRO
ADV:PART
ADV-PRO:CONJ
V:PARENTH
S:NONLEX
S-PRO:ADV-PRO
ADV:NUM
PRAEDIC:A
PARENTH:ADV
S:PARENTH
PART:V
CONJ:NONLEX
ADV:S
PRAEDIC:PART
PART:PR
Mistake pairs

Figure 3

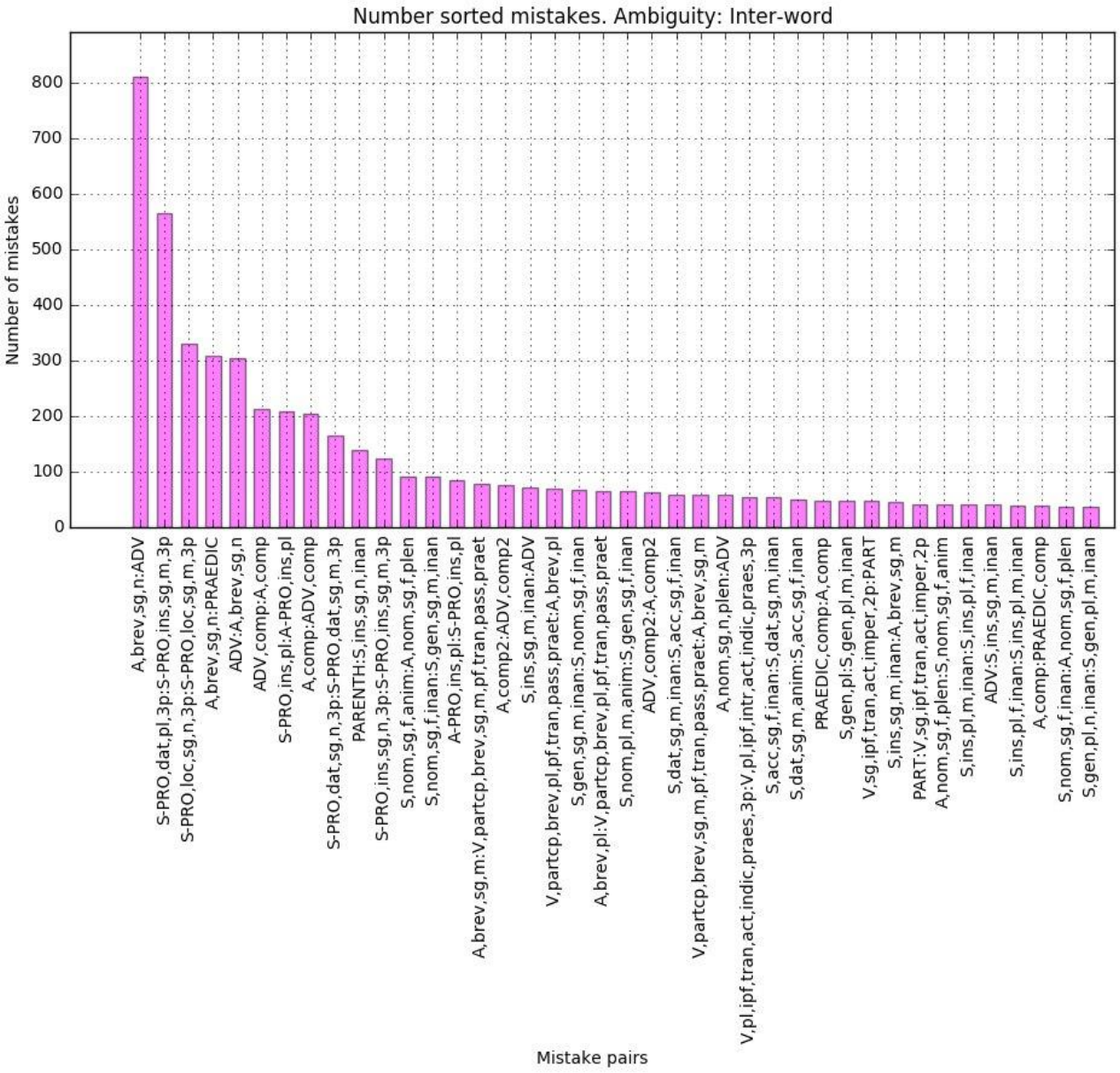
Number sorted mistakes. Ambiguity: Inter-word
Number of mistakes
0
100
200
300
400
500
600
700
800
A,brev,sg,n:ADV
S-PRO,dat,pl,3p:S-PRO,ins,sg,m,3p
S-PRO,loc,sg,n,3p:S-PRO,loc,sg,m,3p
A,brev,sg,n:PRAEDIC
ADV:A,brev,sg,n
ADV,comp:A,comp
S-PRO,ins,pl:A-PRO,ins,pl
A,comp:ADV,comp
S-PRO,dat,sg,n,3p:S-PRO,dat,sg,m,3p
PARENTH:S,ins,sg,n,inan
S-PRO,ins,sg,n,3p:S-PRO,ins,sg,m,3p
S,nom,sg,f,anim:A,nom,sg,f,plen
S,nom,sg,f,inan:S,gen,sg,m,inan
A-PRO,ins,pl:S-PRO,ins,pl
A,brev,sg,m:V,partcp,brev,sg,m,pf,tran,pass,praet
A,comp2:ADV,comp2
S,ins,sg,m,inan:ADV
V,partcp,brev,pl,pf,tran,pass,praet:A,brev,pl
S,gen,sg,m,inan:S,nom,sg,f,inan
A,brev,pl:V,partcp,brev,pl,pf,tran,pass,praet
S,nom,pl,m,anim:S,gen,sg,f,inan
ADV,comp2:A,comp2
S,dat,sg,m,inan:S,acc,sg,f,inan
V,partcp,brev,sg,m,pf,tran,pass,praet:A,brev,sg,m
A,nom,sg,n,plen:ADV
V,pl,ipf,tran,act,indic,praes,3p:V,pl,ipf,intr,act,indic,praes,3p
S,acc,sg,f,inan:S,dat,sg,m,inan
S,dat,sg,m,anim:S,acc,sg,f,inan
PRAEDIC,comp:A,comp
S,gen,pl:S,gen,pl,m,inan
V,sg,ipf,tran,act,imper,2p:PART
S,ins,sg,m,inan:A,brev,sg,m
PART:V,sg,ipf,tran,act,imper,2p
A,nom,sg,f,plen:S,nom,sg,f,anim
S,ins,pl,m,inan:S,ins,pl,f,inan
ADV:S,ins,sg,m,inan
S,ins,pl,f,inan:S,ins,pl,m,inan
A,comp:PRAEDIC,comp
S,nom,sg,f,inan:A,nom,sg,f,plen
S,gen,pl,n,inan:S,gen,pl,m,inan
Mistake pairs

Figure 4

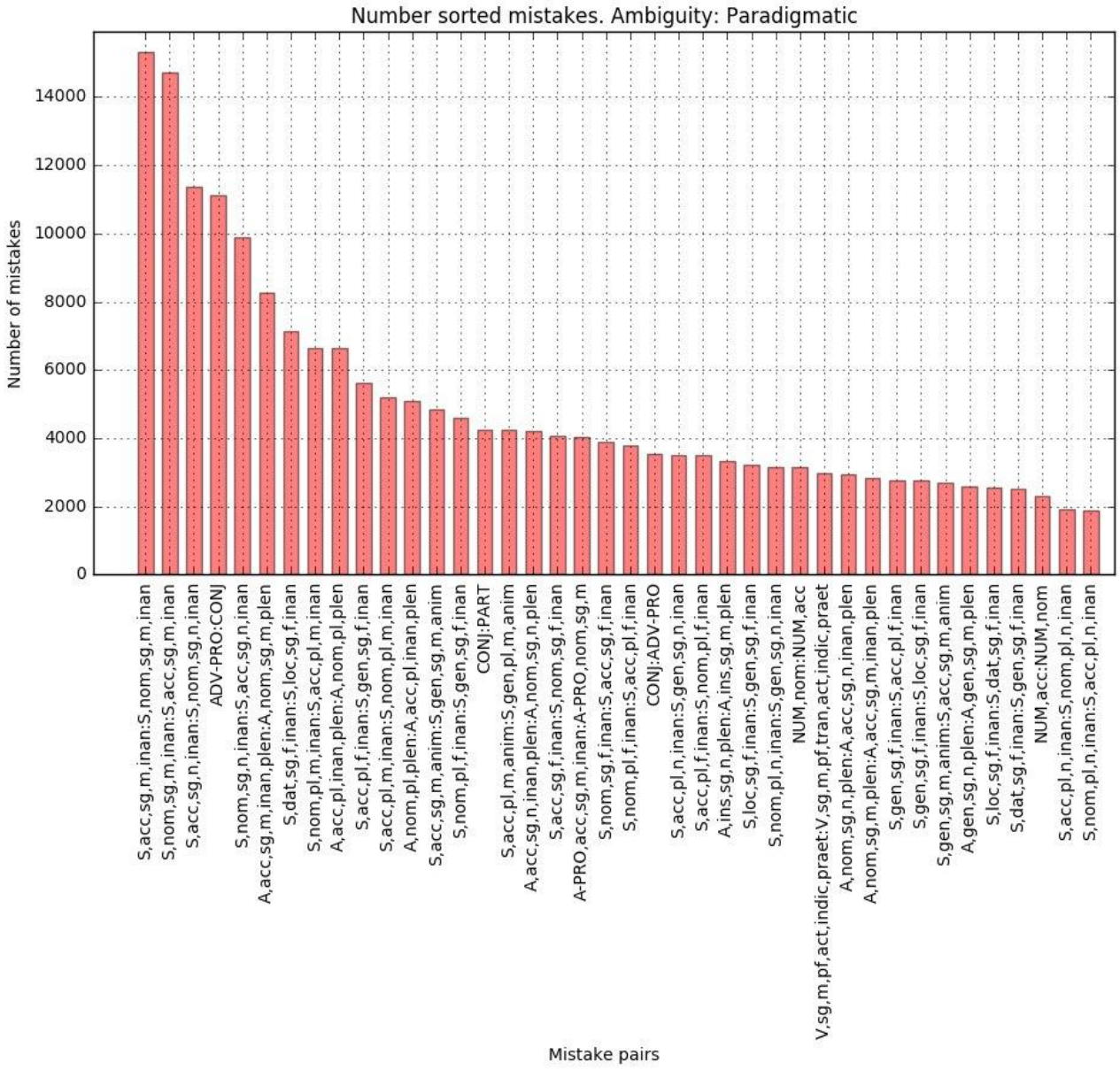
Number sorted mistakes. Ambiguity: Paradigmatic
Number of mistakes
0
2000
4000
6000
8000
10000
12000
14000
S,acc,sg,m,inan:S,nom,sg,m,inan
S,nom,sg,m,inan:S,acc,sg,m,inan
S,acc,sg,n,inan:S,nom,sg,n,inan
ADV-PRO:CONJ
S,nom,sg,n,inan:S,acc,sg,n,inan
A,acc,sg,m,inan,plen:A,nom,sg,m,plen
S,dat,sg,f,inan:S,loc,sg,f,inan
S,nom,pl,m,inan:S,acc,pl,m,inan
A,acc,pl,inan,plen:A,nom,pl,plen
S,acc,pl,f,inan:S,gen,sg,f,inan
S,acc,pl,m,inan:S,nom,pl,m,inan
A,nom,pl,plen:A,acc,pl,inan,plen
S,acc,sg,m,anim:S,gen,sg,m,anim
S,nom,pl,f,inan:S,gen,sg,f,inan
CONJ:PART
S,acc,pl,m,anim:S,gen,pl,m,anim
A,acc,sg,n,inan,plen:A,nom,sg,n,plen
S,acc,sg,f,inan:S,nom,sg,f,inan
A-PRO,acc,sg,m,inan:A-PRO,nom,sg,m
S,nom,sg,f,inan:S,acc,sg,f,inan
S,nom,pl,f,inan:S,acc,pl,f,inan
CONJ:ADV-PRO
S,acc,pl,n,inan:S,gen,sg,n,inan
S,acc,pl,f,inan:S,nom,pl,f,inan
A,ins,sg,n,plen:A,ins,sg,m,plen
S,loc,sg,f,inan:S,gen,sg,f,inan
S,nom,pl,n,inan:S,gen,sg,n,inan
NUM,nom:NUM,acc
V,sg,m,pf,act,indic,praet:V,sg,m,pf,tran,act,indic,praet
A,nom,sg,n,plen:A,acc,sg,n,inan,plen
A,nom,sg,m,plen:A,acc,sg,m,inan,plen
S,gen,sg,f,inan:S,acc,pl,f,inan
S,gen,sg,f,inan:S,loc,sg,f,inan
S,gen,sg,m,anim:S,acc,sg,m,anim
A,gen,sg,n,plen:A,gen,sg,m,plen
S,loc,sg,f,inan:S,dat,sg,f,inan
S,dat,sg,f,inan:S,gen,sg,f,inan
NUM,acc:NUM,nom
S,acc,pl,n,inan:S,nom,pl,n,inan
S,nom,pl,n,inan:S,acc,pl,n,inan
Mistake pairs

Figure 5

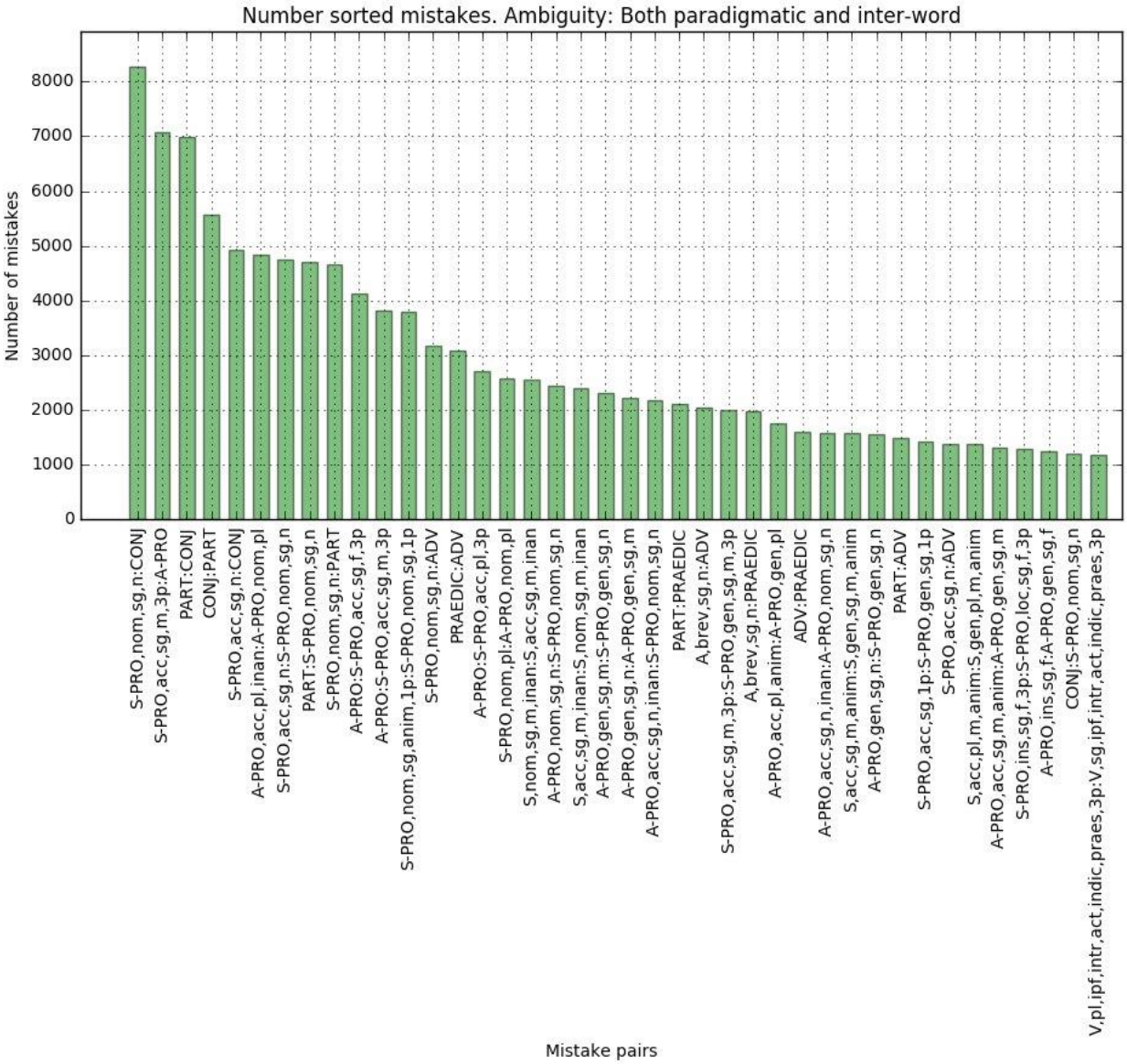
Number sorted mistakes. Ambiguity: Both paradigmatic and inter-word
Number of mistakes
0
1000
2000
3000
4000
5000
6000
7000
8000
S-PRO,nom,sg,n:CONJ
S-PRO,acc,sg,m,3p:A-PRO
PART:CONJ
CONJ:PART
S-PRO,acc,sg,n:CONJ
A-PRO,acc,pl,inan:A-PRO,nom,pl
S-PRO,acc,sg,n:S-PRO,nom,sg,n
PART:S-PRO,nom,sg,n
S-PRO,nom,sg,n:PART
A-PRO:S-PRO,acc,sg,f,3p
A-PRO:S-PRO,acc,sg,m,3p
S-PRO,nom,sg,anim,1p:S-PRO,nom,sg,1p
S-PRO,nom,sg,n:ADV
PRAEDIC:ADV
A-PRO:S-PRO,acc,pl,3p
S-PRO,nom,pl:A-PRO,nom,pl
S,nom,sg,m,inan:S,acc,sg,m,inan
A-PRO,nom,sg,n:S-PRO,nom,sg,n
S,acc,sg,m,inan:S,nom,sg,m,inan
A-PRO,gen,sg,m:S-PRO,gen,sg,n
A-PRO,gen,sg,n:A-PRO,gen,sg,m
A-PRO,acc,sg,n,inan:S-PRO,nom,sg,n
PART:PRAEDIC
A,brev,sg,n:ADV
S-PRO,acc,sg,m,3p:S-PRO,gen,sg,m,3p
A,brev,sg,n:PRAEDIC
A-PRO,acc,pl,anim:A-PRO,gen,pl
ADV:PRAEDIC
A-PRO,acc,sg,n,inan:A-PRO,nom,sg,n
S,acc,sg,m,anim:S,gen,sg,m,anim
A-PRO,gen,sg,n:S-PRO,gen,sg,n
PART:ADV
S-PRO,acc,sg,1p:S-PRO,gen,sg,1p
S-PRO,acc,sg,n:ADV
S,acc,pl,m,anim:S,gen,pl,m,anim
A-PRO,acc,sg,m,anim:A-PRO,gen,sg,m
S-PRO,ins,sg,f,3p:S-PRO,loc,sg,f,3p
A-PRO,ins,sg,f:A-PRO,gen,sg,f
CONJ:S-PRO,nom,sg,n
V,pl,ipf,intr,act,indic,praes,3p:V,sg,ipf,intr,act,indic,praes,3p
Mistake pairs

Figure 6