Revisiting Supertagging for Faster HPSG Parsing

Olga Zamaraeva
Carlos Gomez-Rodr'iguez Universidade da Coru na, CITIC Departamento de Ciencias de la Computaci'

Abstract

1 Introduction

We present new supertaggers for English and use them to improve parsing efficiency for Head-driven Phrase Structure Grammars (HPSG). Grammars have been gaining relevance in the natural language processing (NLP) landscape (Someya et al., 2024), since it is hard to interpret and evaluate the output of NLP systems without robust theories. Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994, HPSG) is a theory of syntax that has been applied in computational linguistic research (see Bender and Emerson 2021 §3–§4). At the core of such research are precision grammars which encode a strict notion of grammaticality — their purpose is to cover and generate only grammatical structures. They include a relatively small set of phrase-structure rules and a large lexicon where lexical entries contain information about the word's syntactic behavior. HPSG treebanks (and the grammars that produce them) encode not only constituency but also dependency and semantic relations and have proven useful in natural language processing, e.g. in grammar coaching (Flickinger and Yu, 2013; Morgado da Costa et al., 2016, 2020), natural language generation (Hajdik et al., 2019), and as training data for high precision semantic parsers (Lin et al., 2022; Chen et al., 2018; Buys and Blunsom, 2017). Assuming a good parse ranking model, a treebank is produced automatically by parsing text with the grammar, and any updates are encoded systematically in the grammar, with no need of manual treebank annotation.[1]

HPSG parsing, which is typically bottom-up chart parsing, is both relatively slow and RAM-hungry. Often, more than a second is required to parse a sentence (see Table 7), and sometimes the performance is prohibitively bad for long sentences, with a typical user machine requiring unreasonable amounts of RAM to finish parsing with a large parse chart (Marimon et al., 2014; Oepen and Carroll, 2002). It is important to emphasize that this is the state of the art in HPSG parsing, and its speed is one of the reasons why the true potential of HPSG parsing in NLP remains not fully realized despite the evidence that it helps create highly precise training data automatically.

Approaches to speed up HPSG parsing include local ambiguity packing (Tomita, 1985; Malouf et al., 2000; Oepen and Carroll, 2002), on the one hand, and forgoing exact search and reducing the parser search space, on the other (Dridan et al., 2008; Dridan, 2009, 2013). Here we contribute to the second line of research, aka supertagging, a technique to discard unlikely interpretations of tokens. Dridan et al. (2008) and Dridan (2009, 2013) used maximum entropy-based models trained on a combination of gold and automatically labeled data from English, requiring large-scale computation. They report an efficiency improvement of a factor of 3 for the parser they worked with (Callmeier,

[1]For a good parse ranking model, it is necessary to select "gold" parses from a potentially large parse forest at least once. This can be done semi-automatically (Packard, 2015).

2000) and accuracy improvements with respect to the ParsEval metric. We present new models for HPSG supertagging, an SVM-based one, a neural CRF-based one, and a fine-tuned-BERT one, and compare their tagging accuracy with a MaxEnt baseline. We now have more English gold training data thanks to the HPSG grammar engineering consortium's treebanking efforts (Flickinger, 2000; Oepen et al., 2004; Flickinger, 2011; Flickinger et al., 2012).[2](https://github.com/delph-in/docs/wiki/RedwoodsTop.) It makes sense to train modern models on this wealth of gold data. Then we use the supertags to filter the parse chart at the lexical analysis stage, so that the parser has fewer possibilities to consider. We report the results of parsing all of the test data associated with the English HPSG treebanks (Oepen and Carroll, 2002) in comparison with parsing the same data with the same parsing algorithm but with no tagging at all, as well as with the integrated MEMM-based tagger. If we use the tagger with some exceptions, our system is the most accurate one (using the partial dependency match metric). It is not faster than the MEMM-based tagger integrated into the parser for production mode, although it is of course much faster than parsing without tagging (by a factor of 3).

The paper is organized as follows. In §2, we give the background necessary for understanding the provenance of our training data. §3 presents the methodology, starting from previous work (§3.1). We then describe our training and evaluation data (§3.2), and finally how we trained the new supertag- [ILLEGIBLE]

2 Background

Below we explain HPSG lexical types (§2.1), which serve as the tags that we predict, and in §2.2, we give the background on the English treebanks which served as our training and evaluation data. §2.3 is a summary for HPSG parsing and the specific parser that we are using for the experiments.

2.1 Lexical types

Any HPSG grammar consists of a hierarchy of types, including phrasal and lexical types, and of a large lexicon which can be used to map surface tokens to lexical types. Each token in the text is recognized by the parser as belonging to one or more of the lexical entries in the lexicon (assuming such an orthographic form is present at all). Lexical entries, in turn, belong to lexical types (Figure 1). Lexical types are similar to POS tags but are more fine grained (e.g. a precision grammar may distinguish between multiple types of proper nouns or multiple types of wh-words, etc). Figure 1 shows the ancestry of two senses of the English word bark, a verb (to bark) and a noun (tree bark). The types differ from each other in features and their values. For example, the HEAD feature value is different for nouns and verbs; one of the characteristics of the main verb type is that it is not a question word; the noun subtype denotes divisible entities, etc. The token bark will be interpreted as either a verb or a noun during lexical analysis parsing stage. After the lexical analysis, the bottom-up parser runs a constraint unification-based algorithm (Carpenter, 1992) to return a (possibly empty) set of parses. To emphasize, a parser in this context is a separate program implementing a parsing algorithm. The grammar is the type hierarchy which the parser takes as input along with the sentence to parse.

[2]The data is available as part of the 2023 release of the English Resource Grammar (the ERG): [https://github.com/delph-in/docs/wiki/RedwoodsTop.

IMAGE NOT PROVIDED

Figure 1: Part of the HPSG type hierarchy (simplified; adapted from ERG). NB: This is not a derivation.

2.2 The ERG treebanks

The English Resource Grammar (ERG; Flickinger, 2000, 2011) is a broad-coverage precision grammar of English implemented in the HPSG formalism. The latest release is from 2023. Its intrinsic evaluation relies on a set of English text corpora. Each release of the ERG includes a treebank of those texts parsed by the current version. The parses are created automatically and the gold structure is verified manually. Treebanking in the ERG context is the process of choosing linguistically (semantically) correct structures from the multiple trees corresponding to one string that the grammar may produce. Fast treebanking is made possible by automatically comparing parse forests and by discriminant-based bulk elimination of unwanted trees (Oepen, 1999; Packard, 2015). The treebanks are stored as databases that can be processed with specialized software e.g. Pydelphin.

The 2023 ERG release comes with 30 treebanked corpora containing over 1.5 million tokens and 105,155 sentences. In principle, there are 43,505 different lexical types in the ERG (cf. 48 tags in the Penn Treebank POS tagset (PTB; Marcus et al., 1993)) however only 1299 of them are found in the training portion of the treebank. The genres include well-edited text (news, Wikipedia articles, fiction, travel brochures, and technical essays) as well as customer service emails and transcribed phone conversations. There are also constructed test suites illustrating linguistic phenomena such as raising and control. The ERG treebanks present more challenging test data compared to the conventional WSJ23 (which is also included). The ERG 2023's average accuracy (correct structure) over all the corpora is 93.77

2.3 HPSG parsing

Several parsers for different variations of the HPSG formalism exist. We work with the DELPH-IN formalism (Copestake, 2002) which is deliberately restricted for theoretical and performance considerations; it only encodes the unification operation natively (and not e.g. relational constraints). Still, the parsing algorithms' worst-case complexity is intractable (Oepen and Carroll, 2002). Carroll (1993, §3.2.3) (cited in Bender and Emerson 2021, p.1109) states that the worst-case parsing time for HPSG feature structures is proportional to $C^{2n\rho+1}$ where ρ is the maximum number of children in a phrase structure rule and C is the (potentially large) maximum number of feature structures. The unification operator takes two feature structures as input and outputs one feature structure which satisfies the constraints encoded in both inputs. Given the complex nature of such structures, implementing a fast unification parser is a hard problem. As it is, the existing parsers may take prohibitively long to parse a long sentence (see e.g. Marimon et al. 2014 as well as §4.2 of this paper).

3 Methodology

Supertagging (Bangalore and Joshi, 1999) reduces the parser search space by discarding the less likely interpretations of an orthography. For example, the word bark in English can be a verb or a noun, and in *The dog barks* it is a lot less likely to be a noun than a verb (see also Figure 1). In principle, there are at least two possible interpretations of the sentence *The dog barks*, as can be

IMAGE NOT PROVIDED

Figure 2: Two interpretations of the sentence The dog barks. The second one is an unlikely noun phrase fragment, which would be discarded with the supertagging technique. (Trees provided by the English Resource Grammar Delphin-viz online demo.)

seen in Figure 2. With supertagging, the pragmatically unlikely second interpretation would be discarded by discarding the noun lexical type (mass-count noun in Figure 1) possibility for the word *barks*. In HPSG, there are fine-grained lexical types within the POS class (e.g. subtypes of common nouns or wh-words), so the search space can be reduced further.

In precision grammars, supertagging comes at a cost to coverage and accuracy; selecting a wrong lexical type even for one word means the entire sentence will likely not be parsed correctly. Thus the accuracy of the tagger is crucial. Related to this is the matter of how many possibilities to consider for supertags: the more are considered, the slower the parsing, but the higher the accuracy. In this paper, we experiment with a single, highest-scored tag for each token. However, we combine this strategy (which prioritizes parsing speed) with a list of tokens exempt from supertagging (which increases accuracy).

3.1 Previous and related work

Bangalore and Joshi (1999) introduced the concept of supertagging. Clark and Curran (2003) showed mathematically that supertagging improves parsing efficiency for a lexicalized formalism (CCG). They used a maximum entropy model; Xu et al. (2015) introduced a neural supertagger for CCG. Vaswani et al. (2016) and Tian et al. (2020) further improved the accuracy of neural-based CCG supertagging achieving an accuracy of 96.25

Supertagging experiments with HPSG parsing speed using hand-engineered grammars are summarized in Table 1. In addition, there were experiments on the use of supertagging for parse ranking with statistically derived HPSG-like grammars (Ninomiya et al., 2007; Matsuzaki et al., 2007; Miyao and Tsujii, 2008; Zhang et al., 2009, 2010; Zhang and Krieger, 2011; Zhang et al., 2012). These statistically derived systems are principally different from the ERG as they do not represent HPSG theory as understood by syntacticians. In the context of the ERG, Dridan et al. 2008 represents our baseline SOTA for the tagger accuracy. Dridan 2013 is a related work on "ubertagging", which includes multi-word expressions. Specifically, an ubertagger considers various multi-word spans, whereas a supertagger relies on a standard tokenizer. We use the ubertagger that was implemented for the ACE parser for the parsing speed experiments, as the baseline (§4.2). Dridan's (2013) parsing accuracy results, however, are not comparable to ours; she used a different dataset, a different parser, and a different accuracy metric.

3.2 Data

We train and evaluate our taggers, both for the baseline (§4.1.1) and for the experiment (§3.3), on gold lexical types from the ERG 2023 release (§2.2). We use the train-dev-test split recommended in the release.[3] There are 84,894 sentences in the training data, 2,045 in dev, and 7,918 in test. WSJ section 23 is used as test data, as is traditional, but so are a number of other corpora, notably *The Cathedral and the Bazaar* (Raymond, 1999), a technical essay which serves as the out-of-domain test

[3]Download redwoods.xls from the ERG repository for details and see https://github.com/delph-in/docs/wiki/RedwoodsTop. This split is different than in Dridan 2009.

data. See Table 2 for the details about the test data. The column titled "training tokens" shows the number of tokens for the training dataset which is from the same domain as the test dataset in the row. For example, WSJ23 has 23K tokens and WSJ1-22 have 960K tokens in the ERG treebanks.

3.3 SVM, LSTM+CRF, and fine-tuned BERT

We train a liblinear SVM model with default parameters (L2 Squared Hinge loss, C=1, one-v-rest, up to 1,000 training iterations) using the scikit-learn library (Pedregosa et al., 2011). To train an LSTM sequence labeling model, we use the NCRF++ library (Yang and Zhang, 2018). We choose the model by training and validating 31 models up to 100 iterations with the starting learning rate of 0.009 and the batch size of 3 (the latter parameters are the largest that are feasible for the combination of our data and the library code). The best NCRF++ model is described in the Appendix in Table 10. To fine-tune BERT, we use the Huggingface transformers library (Wolf et al., 2019) and Pytorch (Paszke et al., 2017). We try both 'base-bert-cased' and 'base-bert-uncased' pretrained models which we fine-tune for up to 50 epochs (stopping once there is no improvement for 5 epochs) with weight decay=0.01. The 'cased' model with learning rate 2e-5 achieves the best dev accuracy (Table 14).

We construct feature vectors similarly to what is described in Dridan 2009 and ultimately in Ratnaparkhi et al. 1996. The training vector consists of the word orthography itself, the two previous and the two subsequent words, the word's POS tag, and, for autoregressive models, the two gold lexical type labels for the two previous words. Nonautoregressive models simply do not have the previous tag features. The test vector is the same except, for autoregressive models, instead of the gold labels for the two previous tokens, it has labels assigned to the two previous tokens by the model itself in the previous evaluation steps (an autoregressive model). The word orthographic forms come from the treebank derivation terminals obtained using the Pydelphin library.[4](https://pydelphin.readthedocs.io/) The PTB-style POS tags come from the treebanks and they were automatically assigned by an HMM-based tagger that is part of the ACE parser code. The POS tags provided by the parser are per token, not per terminal, so for terminals which consist of more than one token, we map the combination of more than one tag to a single PTB-style tag using a mapping constructed manually by the first author for the training data. Any combination of tags not in the training data are at test time mapped to the first tag based on that being the most frequently correct prediction in the training data.[5] We only saw 15 unknown combinations of tags in the entire dev and test data.

3.4 The ACE HPSG Parser

We work with ACE (Crysmann and Packard, 2012), which has seen regular releases since the publication date and remains the state-of-the-art HPSG parser. It is intended for settings which include individual use, including with limited RAM. This parser has default RAM settings[6] which can be modified, and also an in-built "ubertagger". While the ubertagger is based on Dridan 2013, it is not the same thing and its performance has never been published before. In particular, its tagging accuracy is unknown and we did not seek to evaluate it (evaluating a different MaxEnt model instead). The ubertagger was integrated into the ACE parser code with great care, optimizing for performance. We also do not seek to compete with such optimizations in our experiments. For our experiments, we provide ACE with the tags predicted by the best supertagger (the BERT-based

[4][https://pydelphin.readthedocs.io/

[5]The first tag is the correct tag in about 1/3 of the cases.

[6][ILLEGIBLE]

supertagger) along with the character spans corresponding to the token for which the tag was predicted.[7] We then prune all lexical chart edges which correspond to this token span but do not have the predicted lexical type. As such, we follow the general idea of using supertagging for reducing the lexical chart size but we do not use the same code that the integrated ubertagger uses for this procedure. We assume that our code could be further optimized for production.

3.5 Exceptions for supertagging

As already mentioned, mistakes in supertagging are very costly for precision grammar parsing; one wrongly predicted lexical type means the entire sentence will not be parsed correctly. After the maxent-based supertaggers were trained by Dridan 2009 and Dridan 2013, the developer of the English Resource Grammar Flickinger experimented with them and has come up with a list of lexical types which the supertagger tended to predict wrong. The list included fine-grained lexical types representing words such as *do*, *many*, *less*, *hard* (among many others).[8] Using such exception lists counteracts the effects of supertagging and slows down the parsing, while increasing accuracy. We include this exception list methodology into our experiments, but we compile our own list based on the top mistakes our supertaggers made on the dev data.

4 Results

4.1 Tagger accuracy and tagging speed

4.1.1 Tagging accuracy baseline

For our baseline, we use a MaxEnt model similar to Dridan 2009. While Dridan (2009) used off-the-shelf TnT (Brants, 2000) and CC (Clark and Curran, 2003) taggers, we use the off-the-shelf logistic regression library from scikit-learn (Pedregosa et al., 2011) which is a popular off-the-shelf tool for classic machine learning algorithms. The baseline tagger accuracy is included in Table 2. The details on how the best baseline model was chosen are in Appendix A. The results are presented in Table 2.

4.1.2 Tagger accuracy results

Table 2 shows that the baseline models achieve similar performance to Dridan 2009 (D2009 in Table 2) on in-domain data and are better on out-of-domain data. This may indicate that these models are close to their maximum performance on in-domain data on this task but adding more training data still helps for out-of-domain data. Dridan's (2009) models were trained on a subset of our data. Dridan (2009, p.84) reports getting 91.47

The SVM and the neural models are better than the baseline models on all test datasets, and fine-tuned BERT is the best overall. On the portion of WSJ23 for which we have gold data, fine-tuned BERT achieves 97.26

All models make roughly the same mistakes (Table 3), with prepositions, pronouns, and auxiliary verbs being the most misclassified tokens, and the proper noun being the least accurate tag.[9]

[7]The speed of the tagging itself is negligible because the tagger tags 346 sentences per second (0.003 sec/sen) while HPSG parsing is an order of magnitude slower.

[8]The full list can be found in the release of the ERG in the folder titled 'ut' (ubertagging).

[9]In Table 3, the "not closely related" column represents mistakes where the true label and the predicted label differ in their general subcategory; in this column, we did not count nouns mistaken for other types of nouns, etc. We use the ERG lexical type naming convention to filter the errors. The "n-c" type is a subtype of common noun; the "n-pn"

Figure 3: Pareto Frontier (Speed and F-score)

4.2 Results: Parsing Speed and Accuracy

We measure the effect of supertagging on parsing speed and accuracy using the ACE parser (§3.4). Recall that HPSG parsing is chart parsing, and for a large grammar, the charts can be huge. The goal of supertagging is to reduce the size of the lexical chart. This can make parsing faster, however if a good lexical analysis is thrown out by mistake (due to a wrong tag), the entire sentence is likely to be lost (not parsed or parsed in a meaningless way). The parser speed and the parser accuracy are therefore in tension: the more time we give the parser the more chances it will have to build the correct structure in a bigger chart. For accuracy, we report two metrics: exact match with the gold semantic structure (MRS) and partial match Elementary Dependency Match metric (EDM; Dridan and Oepen, 2011). The exact match is less important because it usually can only be achieved on short, easy sentences. The EDM (and similar) is the usual practice. The results are presented in Tables 4–9, which are also summarized in Figure 3.

4.2.1 Baseline

We compare our system with two systems: ACE with no tagging at all and ACE with the in-built "ubertagger". The system with no tagging at all is the baseline for parsing speed and, theoretically, the upper boundary for the parsing accuracy (as the parser could have access to the full lexical chart). However, in practice it is difficult to obtain this upper bound because it requires at least 54GB of RAM (see §A.5) and the parsing takes unreasonably long (up to several minutes per sentence). With realistic settings, the system with no tagging fails to parse some of the longer sentences because the lexical chart exceeds the RAM limit. It is precisely the problem that ubertagging/supertagging is supposed to solve: reduce the size of the lexical chart so that the parsing can be done with realistic RAM allocation and in reasonable time.

The ubertagger is a MEMM tagger based on Dridan 2013. It was trained on millions of sentences using large computational resources (the Titan system at University of Oslo) and as such is not easily reproducible. In contrast, our BERT-based model is fairly easy to fine-tune and reproduce on an individual machine. For the purposes of parsing accuracy and speed, rather than comparing our system to other experimental taggers presented in §4.1, we compare it to the ubertagger because the ubertagger is integrated into the ACE parser for production and as such is a more challenging baseline.

Below we present the results in two settings: (1) default settings, and (2) default RAM with tag exceptions. In Tables 4 and 7, the best result is bolded, and the experimental result is italicized in the cases where it is not the best but much closer to the ubertagger than to the no-tagging baseline.

4.2.2 Default parsing

Tables 4, 5, and 6 present the results for the ACE parser default RAM limit setting (1200MB). On the ubertagger and the supertagger side, we use all the predictions and do not exclude any tags from the pruning process.

and "n-pn-gen" types are subtypes of proper nouns; "v-np*" is a subtype of verbs that take clausal complements; "adj-i" is a subtype of intersective adjectives, "d-poss" is a possessive determiner.

The results show that while we can parse faster with tagging (the ubertagger being the fastest), both the ubertagger and the supertagger suffer from the high cost of each tagging mistake: while the new BERT-based supertagger is more accurate, its accuracy is still not 100

4.2.3 Parsing with exceptions lists

Tables 7–9 present the results for parsing with ubertagging and supertagging with exceptions. The no-tagging system's results are the same as before; we repeat them for convenience.

We have looked at the most common mistakes in the supertags in the training data and have compiled a list of 15 tags which BERT tends to predict wrong.[10] On the ubertagger side, there was already a list of exceptions. The ubertagger's exception list is a list of 1715 lexical entries (words, e.g. "my"), whereas ours is a list of 15 lexical types (tags, e.g. "d-poss-my", which is a supertype for "my" in the grammar). The ubertagger's list includes some of the most frequent function words. Using exception lists reduces the pruning effect of tagging and slows down parsing, but improves recall and overall accuracy.

5 Conclusion and future work

We used the advancements in HPSG treebanking to train more accurate supertaggers. The ERG is a major project in syntactic theory and an important resource for creating high quality semantic treebanks. It has the potential to contribute to NLP tasks that require high precision and/or interpretability including probing of the LLMs, and thus making HPSG parsing faster is strategic for NLP. We tested the new supertagging models with the state-of-the-art HPSG parser and saw improvements in parsing speed as well as accuracy. We consider the results on multiple domains, well beyond the WSJ Section 23. We show promising results but also confirm that domain remains important, and purely statistical systems are brittle and often require rule-based additions in real-life scenarios. We contribute the ERG datasets converted to huggingface transformers format intended for token classification, along with the code which can be adapted for other purposes.

6 Limitations

Our paper is concerned with training supertagging models on an English HPSG treebank. The limitations therefore are associated mainly with the training of the models including neural networks, and with the building of broad-coverage grammars such as the English Resource Grammar. Crucially, while our method does not require industry-scale computational resources, training a neural classifier such as ours still requires a certain amount of training data, and this means that our method assumes that a large HPSG treebank is available for training. The availability of such a treebank, in turn, depends directly on the availability of a broad-coverage grammar.

7 Acknowledgements

[ILLEGIBLE]

[10]The list includes: [ILLEGIBLE]

8 References

References

[1] Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

[2] Emily M. Bender and Guy Emerson. 2021. Computational linguistics and grammar engineering. *Annual Review of Linguistics*, 7:1–23.

[3] Stephen Blunsom. 2007. *Structured probabilistic models for natural language processing*. Ph.D. thesis, University of Cambridge.

[4] Thorsten Brants. 2000. TnT – A statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference*, pages 224–231.

[5] Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1215–1226.

[6] Bob Carpenter. 1992. *The logic of typed feature structures*. Cambridge University Press.

[7] John Carroll. 1993. *Practical unification-based parsing of natural language*. Ph.D. thesis, University of Cambridge.

[8] John Carroll and Stephan Oepen. 2002. Efficient parsing for unification-based grammars. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*. CSLI Press.

[9] Danqi Chen, Jason Bolton, and Christopher D. Manning. 2018. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 2358–2367.

[10] Stephen Clark and James R. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 126–133.

[11] Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications.

[12] Berthold Crysmann and Woodley Packard. 2012. Engineering efficient HPSG parsers. In *Proceedings of the 26th Pacific Asia Conference on Language, Information, and Computation*, pages 1–10.

[13] Rebecca Dridan. 2009. *Ubertags: A linguistic approach to parsing*. Ph.D. thesis, University of Melbourne.

[14] Rebecca Dridan. 2013. Accurate and efficient HPSG parsing. *Computational Linguistics*, 39(2):321–359.

[15] Rebecca Dridan, Stephan Oepen, and Emily M. Bender. 2008. Parsing with typed feature structures: Supertagging. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 191–200.

[16] Rebecca Dridan and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 225–230.

[17] Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28.

[18] Dan Flickinger. 2011. Accuracy vs. robustness in grammar engineering. In *Proceedings of the 2nd Workshop on Grammar Engineering Across Frameworks*.

[19] Dan Flickinger, Stephan Oepen, and Emily M. Bender. 2012. The LinGO Redwoods treebank: Motivations and applications. *Journal of Natural Language Engineering*, 18(3):1–37.

[20] Dan Flickinger and Jiye Yu. 2013. Toward more precision grammar applications. In *Proceedings of the 14th International Conference on Parsing Technologies*.

[21] Jiří Hajdik, Dan Flickinger, Stephan Oepen, and Francis Bond. 2019. Semantic transfer for multilingual generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3213–3223.

[22] Dirk Hovy and Anders Søgaard. 2015. Tagging performance correlates with author age. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 483–488.

[23] Yusuke Miyao and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics*, 34(1):35–80.

[24] Montserrat Marimon, Nuria Bel, and Lluís Padró. 2014. Automatic selection of HPSG-parsed sentences for treebank construction. *Computational Linguistics*, 40(3):523–531.

[25] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. Technical report, University of Pennsylvania.

[26] Takashi Ninomiya, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2007. A log-linear model with an n-gram reference distribution for accurate HPSG parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 60–68.

[27] Stephan Oepen. 1999. *[incr tsdb()] competence and performance laboratory*. User and reference manual.

[28] Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. 2004. LinGO Redwoods: A rich and dynamic treebank for HPSG. *Research on Language and Computation*, 2(4):575–596.

[29] Woodley Packard. 2015. Treebanking with discriminants. *Computational Linguistics*, 41(2):327–353.

[30] Fabian Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[31] Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

[32] Raymond. 1999. *The Cathedral and the Bazaar*. O'Reilly Media.

[33] Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

[34] Shigeki Someya, [ILLEGIBLE]. 2024. [ILLEGIBLE].

[35] Masaru Tomita. 1985. An efficient context-free parsing algorithm for natural languages. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 756–764.

[36] Ashish Vaswani et al. 2016. Supertagging with neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[37] Thomas Wolf et al. 2019. HuggingFace's transformers: State-of-the-art natural language processing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 4171–4186.

[38] Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 250–255.

[39] Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 74–79.

[40] Yue Zhang and Stephan Oepen Krieger. 2011. [ILLEGIBLE].

[41] Yue Zhang et al. 2009. [ILLEGIBLE].

[42] Yue Zhang et al. 2010. [ILLEGIBLE].

[43] Yue Zhang et al. 2012. [ILLEGIBLE].

A Appendix

A.1 Tuning ranges

The details on how the best baseline MaxEnt model was chosen are presented in this section. We tuned the regularization strength and feature selection parameters on the development set and selected the model with the highest tagging accuracy on dev. The tuning ranges and the final selected hyperparameters are summarized in Table 10.

A.2 Computational resources

We trained the neural models with NVIDIA GeForce RTX 2080 GPU, CUDA version 11.2. The SVM model and the MaxEnt baseline were trained using Intel Core i7-9700K 3.60Hz CPU. The parser was run on the same CPU. The default RAM settings of the ACE parser were used unless otherwise specified.

A.3 Development accuracies

This section reports the development set accuracies for all models that were trained and evaluated in this paper. The results for the MaxEnt baseline, the SVM model, the NCRF++ model, and the fine-tuned BERT model on the development data are summarized in Tables 11–13.

A.4 MaxEnt model

A.4.1 MaxEnt model selection

We experimented with multiple configurations of the MaxEnt model, varying the regularization parameters and the feature sets. The final model was selected based on its performance on the development data. The detailed results of these experiments are shown in Table 14.

A.4.2 MaxEnt classifiers

The MaxEnt classifiers were trained using the scikit-learn logistic regression implementation. Feature vectors followed the same general design as in Dridan (2009), including orthographic features, contextual word features, POS tags, and previous tag features for autoregressive variants. Further implementation details are provided in the tables above.