The WSD Development Environment

Rafał Młodzki[1] and Adam Przepiórkowski[1,2]

[1] Institute of Computer Science PAS, ul. Ordona 21, 01-237 Warszawa, Poland
adamp@ipipan.waw.pl

[2] University of Warsaw, Krakowskie Przedmieście 26/28, 00-927 Warszawa, Poland
rmlodzki@gmail.com

Abstract. In this paper we present the Word Sense Disambiguation Development Environment (WSDDE), a platform for testing various Word Sense Disambiguation (WSD) technologies, as well as the results of first experiments in applying the platform to WSD in Polish. The current development version of the environment facilitates the construction and evaluation of WSD methods in the supervised Machine Learning (ML) paradigm using various knowledge sources. Experiments were conducted on a small manually sense-tagged corpus of 13 Polish words. The usual groups of features were implemented including bag-of-words, parts-of-speech, words with their positions, etc. (with different settings), in connection with popular ML algorithms (including Naive Bayes, Decision Trees and Support Vector Machines). The aim was to test to what extent standard approaches to the English WSD task may be adopted to free word order and rich inflection languages such as Polish. In accordance with earlier results in the literature, the initial experiments suggest that these standard approaches are relatively well-suited for Polish. On the other hand, contrary to earlier findings, the experiments also show that adding of some features beyond bag-of-words increases the average accuracy of the results.

Keywords: word sense disambiguation, machine learning, feature selection, Polish

1 Introduction

The Word Sense Disambiguation (WSD) task consists of choosing the most appropriate sense of a word from all its senses in a given context. For example in the context:

> **(K1)** *Chodzi mi o to, aby* ***język*** *giętki*
> *Powiedział wszystko, co pomyśli głowa.*
> Juliusz Słowacki[3]

[3] Roughly:
Let my nimble tongue (język)
represent the Mind.
Juliusz Słowacki

the word "język" is used in the meaning "tongue", and in the context:

> **(K2)** *W* ***języku*** *słowackim da się powiedzieć mniej więcej tyle, co w języku polskim.*[4]

the most appropriate sense for *język* is "language".

Accurate WSD could be of great importance for numerous tasks in Natural Language Processing (NLP), such as text categorization, information retrieval or machine translation. The adjective "accurate" is essential ([2]). So far the best accuracy has been obtained (English) within the supervised Machine Learning (ML) paradigm.

Despite the above and the fact that WSD in English dates back to the 1950s, hardly any work on this subject has been reported for Polish (one exception is the preliminary analysis in [3]). Moreover, there is currently no publicly available sense-tagged corpus of Polish of a decent size. Both these issues are addressed in the National Corpus of Polish project (Pol. *Narodowy Korpus Języka Polskiego*; NKJP; `http://nkjp.pl/`; [10, 11]), where the necessary resources and tools are created. The remainder of this paper presents the first step in this direction: the WSD Development Environment (WSDDE) and initial experiments in applying this platform to WSD in Polish, for the time being using a small sense-tagged corpus from [3].

2 WSD Development Environment

The environment is a bundle of a number of resources, applications, procedures and scripts which facilitate the development and evaluation of WSD methods. The overall scheme of the environment is depicted in Figure 1. There are several possible scenarios of how to use the environment.

In the simplest scenario the user creates a single WSD method by specifying various settings to be described in detail below (e.g., the source and number of training examples, the size of the bag-of-words window, the particular Machine Learning algorithm, feature filtering methods, etc.) and runs the application. On the basis of these settings, an appropriate WSD method will be prepared, trained and evaluated. The results can be saved in the database. The settings are divided into parameters of feature generators (e.g., the size of the bag-of-words window) and those that are not directly connected with the generation of features (e.g., an ML algorithm).

Another scenario is to create many WSD methods at once. One way to achieve this is to underspecify some settings in the configuration file. For example, the user can set {1, 5, 10, 20, 40} as the size of the bag-of-words window. The application will generate all the possible methods combined from all the settings (i.e., one method where this size is 1, another method where the size is 5, etc.). If the number of possible methods is too large, then a random sample is created. Another way to do this is to choose some WSD methods (e.g., the best 5% among

[4] *In the Slovak language* (język) *one can express as much as, more or less, in Polish.*

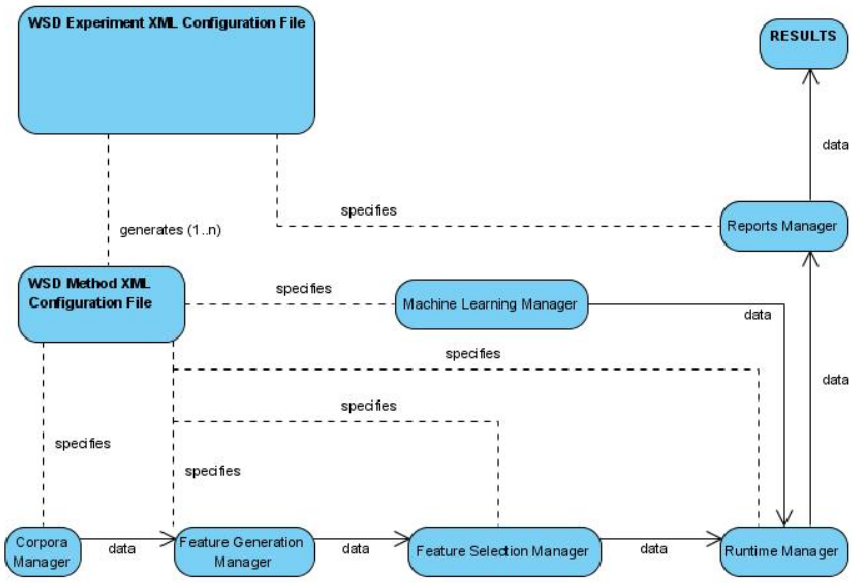

Fig. 1. The scheme of the WSDDE

the ones evaluated so far and stored in the database) and generate other WSD methods that differ from those in one or two parameters. All the results saved in the database can be analysed from different points of view with prepared SQL queries (part of WSDDE). The results reported in § 3 have been obtained with these queries.

2.1 Corpora

This group of settings informs the corespondent WSD manager which corpus is to be used to train and test the WSD method. The WSDDE contains a tool for importing sense-tagged corpora in text format. Since there are not many such corpora, it also contains a tool for creating sense-tagged-like corpora from the IPI PAN Corpus (`http://korpus.pl/`; [7]) using the technique of pseudowords ([4, 14]).[5] The user can also specify the number of examples within the training and test sets. There is a possibility to choose the cross-validation method instead of evaluating WSD methods on a test set.

[5] The idea of this technique is as follows. Two monosemous words A and B are fused into one artificial bisemous (psuedo)word AB with senses A and B, and all occurrences of words A and B in the corpus are replaced with the pseudoword AB. By knowing which original word occurred in which AB position, we know the senses of all occurrences of AB.

2.2 Feature Generation

The feature generator uses user settings and contexts from the training set to generate a group of features specific to it and then is used to compute feature values for all the contexts. Due to the open architecture of WSDDE, it is easy to implement one's own feature generators or override the existing ones (by changing the number of parameters or their semantics). In the current version of the system, the user can choose from the following built-in feature generators: thematic feature generator, structural feature generator I/II and keyword generator.

Thematic Feature Generator (TFG) This generator is the source of features which could characterise the domain or general topic of a given context. This is achieved by checking whether certain words are present in the wide context (e.g., up to 50 positions to the left or the right from the disambiguated word). It seems that the occurrence of a word in such a distant position cannot indicate anything but the general topic of a context, hence the name of the generator. The behaviour of the generator is currently controlled by three basic parameters:

1. **The size of the bag-of-words window**: this parameter determines the size of the text window the words are taken from. Its value is an integer in the range of 0–100, where values within the range 20–50 are most common.
2. **Lemmatisation** indicates whether it is the orthographic word form (0) or the base form (lemma; 1) that is considered. The lemmatisation is provided by the TaKIPI tagger ([6])[6].
3. **Binary** indicates whether feature values are binary (presence or absence; 0) or continuous (frequencies; 1).

Depending on the settings of these parameters, the generated features indicate the presence or the frequency of particular word forms or lemmata in the text window of a given size.

For example, if the training set consists only of two contexts K1 and K2 given in § 1, and the only generator is the thematic feature generator with parameters set respectively to (10,1,1) (i.e., window size 10, lemma frequencies as features), then a fragment of the feature vector for *język* could be depicted as in Table 1.

Table 1. Some features and feature vectors for the topical feature generator and training set {K1, K2}

	giętki	**głowa**	**słowacki**	**powiedzieć**	**polski**
K1	1	1	1	1	0
K2	0	0	1	1	1

[6] See also [5, 1].

This generator, like any other in WSDDE, can be easily modified; for example, lemmatisation could be replaced by stemming or asymmetrical text windows could considered for bag-of-words.

Structural Feature Generator I (SFG1) This generator produces features which can describe some structural properties of the disambiguated word, namely, the presence of particular words on a particular position in the close proximity to the disambiguated word. A small context (window up to the size of 5) is considered. The behaviour of the generator is controlled by three parameters:

1. **The size of the window**: this parameter determines the size of the text window considered and, hence, the size of the feature vector. Its value is an integer in the range of 0–5, where values 0–3 are most common.
2. **Lemmatisation** indicates whether orthographic or base forms are considered.
3. **Binary** indicates whether feature values are binary or continuous.

For example, if the training set consists only of two contexts K1 and K2, and the only generator is the structural feature generator I with parameters set to (2,1,1), then the part of the feature vector could be depicted as in Table 2.

Table 2. Some features and feature vectors for the structural feature generator I and training set {K1, K2}

	w-1	*słowackim*+1	*da*+2	,-2	*aby*-1	*giętki*+1
K1	0	0	0	1	1	1
K2	1	1	1	0	0	0

Structural Feature Generator II (SFG2) This generator also generates features describing structural properties, but it works at the part of speech (POS) level: it checks which parts of speech occur in which position in the close proximity of the disambiguated word. The behaviour of the generator is controlled by three parameters:

1. **The size of the window**. This parameter determines the size of the window the parts of speech are taken from. Its value is an integer in the range of 0–5 where values around 0–3 are most common.
2. **Tagset** indicates which tagset is used. One can choose between the original fine-grained IPI PAN Tagset ([12, 13])[7] and a coarse-grained tagset consisting of 5 main parts of speech.
3. **Binary** indicates whether feature values are binary or continuous.

[7] See also [9].

For example, if the training set consists only of two contexts K1 and K2 and the only generator is the structural feature generator II with parameters set to (2,basic,1), then the feature vector could be depicted as in Table 3.

Table 3. Some features and feature vectors for structural feature generator II and training set {K1, K2}

	prep-1	adj+1	verb+2	interp-2
K1	0	1	1	1
K2	1	1	1	0

Keyword Feature Generator (KFG) The keyword feature generator creates features related to the disambiguated word itself. In the current version, it generates the following features: the particular orthographic form of the word, its part of speech, and whether it starts with a capital letter.

Table 4. Features and feature vectors for the keyword feature generator and training set {K1, K2}

	język	*języku*	loc	nom	sg	capitalized
K1	1	0	0	1	1	0
K2	0	1	1	0	1	0

Other Feature Generators Apart from the above basic feature generators, some more experimental generators are available, including thematic feature generator II based on the WordNet hierarchy and structural feature generator III based on a naive attempt to recognise some grammatical relations (e.g., the closest noun is treated as the subject, etc.).

2.3 Feature Selection

These settings determine how to select features. One can use all the feature selection algorithms from the WEKA package (or one's own, if they are compatible with WEKA interfaces). Generally, feature selection based on feature ranking works much faster than feature selection based on subsets. It is possible to use the second filter after the first one, e.g., at first one makes feature selection with feature ranking, e.g., 200 features remain, and then one makes feature selection on these remaining 200 features based on feature subsets.

2.4 Machine Learning Algorithms

This setting indicates which ML algorithms should be used. All classifiers from the WEKA package ([15]) are available. By default they are run with default parameters. It is also possible to add additional classifiers.

2.5 Runtime

This setting informs the controller about the maximum time and memory size for learning and evaluation of the given method.

2.6 Reports

These settings concern the whole experiment (not a single WSD method) and indicate what kind of reports — based on prepared SQL queries — should be produced.

3 Experiments

In order to compare the results with the literature, we based the experiments on the toy corpus used in [3] — the only previous article about Polish WSD known to us. As a starting point we used the settings of the best WSD method from the cited article: NaiveBayes as the ML algorithm, the size of a bag-of-word window equal to 20, lemmatisation turned on both for bag-of-word and word-with-its-position (WWIP) features. We also used — as in [3] — leave-one-out cross-validation as the evaluation method. On the other hand, feature selection was always based only on the current training set (without the left-one-out example). This more standard evaluation procedure resulted in a smaller accuracy score: on the average about 10% (sic!). Also the algorithm of feature selection was different — first we took 200 features[8] with the highest information gain (InfoGain in WEKA), and then we used subset feature selection (CfsSubsetEval in WEKA) on this set in order to filter out features which were mutually correlated. Feature selection was carried out for all the features (not only for bag-of-words features, as in the cited article).

The WSD method described above (NaiveBayes, bag-of-words 20, no POS-features, no WWIP-features) achieved a 74% accuracy on that corpus (unweighted average over all WSD tasks). Next we carried out an experiment which aim was to evaluate WSD methods with settings differing slightly from what was mentioned above (the baseline). They differ in the use of structural feature generator I and II (POSs and WWIPs), the keyword feature generator and in the parameters of the thematic feature generator (the size of the window).

We obtained the following results (see Table 5): by adding extra features (generated by SFG1(2,1,1), SFG2(2,basic,1) and KFG()) accuracy was improved

[8] This number is a compromise between quality and efficiency, established during previous experiments.

by about 2% (up to 76%). By resizing the bag-of-words window up to 30 (using only features from the thematic feature generator) it was possible to obtain a 77% accuracy (+3% compared to the baseline). Adding extra features (SFG1(2,1,1), SFG2(2,basic,1), KFG()) to that resized window resulted in more than 80% accuracy, which is about 6% more than the baseline. A further resizing of the bag-of-words window yielded 78% for bag-of-words features and 82% for bag-of-words features with additional features. A bigger size of bag-of-words does not lead to significant improvements. ML algorithms other than NaiveBayes had a much lower accuracy (DecisionTrees) or worked much much slower (SVM).

Table 5. The results of the experiment

bag-of-words size	extra features	accuracy
20		74
20	SFG2(2,basic,1), SFG1(2,1,1), KFG()	76
30		77
30	SFG2(2,basic,1), SFG1(2,1,1), KFG()	80
40		78
40	SFG2(2,basic,1), SFG1(2,1,1), KFG()	82
50		78
50	SFG2(2,basic,1), SFG1(2,1,1), KFG()	81

The share of features generated by particular generators in the feature vector is shown in Table 6.

Table 6. Constitution of feature vector in relation to feature generators (average over all cases when all build-in feature generators were used)

Feature Generator	percent
Thematic	81
Structural I	7
Structural II	6
Keyword	6

4 Conclusions

The first obvious conclusion is that the bigger a bag-of-word window is used, the more considerable level of accuracy is achieved.

As far as the extra features and their impact on the results are concerned, it is possible to explain the difference (a few percent) in comparison to the cited article by differences in feature selection (described above) and the use of the keyword feature generator. In the present article we use feature selection for all the features, whereas in the cited article it was only used for the bag-of-words features and all the other features were simply added without filtration, which could have been a source of noise. The use of the keyword feature generator resulted in the improvement in the WSD task for POWÓD ('reason' or 'plaintiff') from 88% to 96%: the genitive of POWÓD 'reason' is *powodu*, whereas the genitive of POWÓD as 'plaintiff' is *powoda*.[9] Also plural forms of plaintiff (e.g., *powodzi*) are rather uncommon. Features generated by the keyword feature generator are rather useless in WSD of English, but for Polish with its rich inflection it may have some importance.

Table 7. The size of training sets, accuracy and time of computations for corpus of 2-pseudowords generated in WSDDE

number of examples	accuracy	time
25	54,3	148,21
50	61,29	142,74
100	64,41	208,14
200	76,76	303,85
300	80,39	441,95
400	81,33	682
500	82,45	996,81
600	83,23	1171,64
700	83,83	1600,9
800	84,36	1885,36
900	84,42	2614,07
1000	84,92	2919,66
1100	84,53	3066,28
1200	84,99	4124,69
1300	85,37	4429,55
1400	85,07	5549,63
1500	84,54	6230,39
1600	84,91	7008,87
1700	85,77	7927,07
1800	85,56	9229,4
1900	85,92	9986,21
2000	85,8	10293,14

[9] Note that *genetivus* seems to be the most frequent case in Polish ([8]).

The above confirms that the standard approach which was used for WSD in English also works quite well for Polish. Accuracy at the level of about 80% is relatively high, given such a basic approach and considering the small number of examples in training sets (less than 100 per sense); previous experiments on pseudowords (Table 7) have shown that accuracy grows significantly with the growth of the number of examples up to 500 examples per sense.

Of course, results obtained on the basis of such a small corpus are not very reliable. We deliberately do not present more detailed results, nor do we use more advanced features, in order to avoid presenting results which might be dubious. We wait until a large high-quality sense-tagged corpus for Polish is available within the framework of NKJP.

References

1. Acedański, S., Przepiórkowski, A.: Towards the adequate evaluation of morphosyntactic taggers. In: Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010); Poster Session. pp. 1–8. Beijing (2010)
2. Agirre, E., Edmonds, P. (eds.): Word Sense Disambiguation: Algorithms and Applications, Text, Speech and Language Technology, vol. 33. Springer, Dordrecht (2006)
3. Baś, D., Broda, B., Piasecki, M.: Towards Word Sense Disambiguation of Polish. In: Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT 2008): Computational Linguistics – Applications (CLA'08). pp. 73–78. PTI, Wisła, Poland (2008)
4. Gale, W.A., Church, K.W., Yarowsky, D.: Work on statistical methods for word sense disambiguation. In: AAAI Fall Symposium on Probabilistic Approaches to Natural Language. pp. 54–60. Cambridge (1992)
5. Karwańska, D., Przepiórkowski, A.: On the evaluation of two Polish taggers. In: Goźdź-Roszkowski, S. (ed.) The proceedings of Practical Applications in Language and Computers PALC 2009. Peter Lang, Frankfurt am Main (2009)
6. Piasecki, M.: Polish tagger TaKIPI: Rule based construction and optimisation. Task Quarterly 11(1–2), 151–167 (2007)
7. Przepiórkowski, A.: The IPI PAN Corpus: Preliminary version. Institute of Computer Science, Polish Academy of Sciences, Warsaw (2004)
8. Przepiórkowski, A.: The IPI PAN Corpus in numbers. In: Vetulani, Z. (ed.) Proceedings of the 2nd Language & Technology Conference. pp. 27–31. Poznań, Poland (2005)
9. Przepiórkowski, A.: A comparison of two morphosyntactic tagsets of Polish. In: Koseska-Toszewa, V., Dimitrova, L., Roszko, R. (eds.) Representing Semantics in Digital Lexicography: Proceedings of MONDILEX Fourth Open Workshop. pp. 138–144. Warsaw (2009)
10. Przepiórkowski, A., Górski, R.L., Lewandowska-Tomaszczyk, B., Łaziński, M.: Towards the National Corpus of Polish. In: Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC 2008. ELRA, Marrakech (2008)
11. Przepiórkowski, A., Górski, R.L., Łaziński, M., Pęzik, P.: Recent developments in the National Corpus of Polish. In: Levická, J., Garabík, R. (eds.) NLP, Corpus Linguistics, Corpus Based Grammar Research: Proceedings of the Fifth International

Conference, Smolenice, Slovakia, 25–27 November 2009. pp. 302–309. Tribun, Brno (2009)
12. Przepiórkowski, A., Woliński, M.: A flexemic tagset for Polish. In: Proceedings of Morphological Processing of Slavic Languages, EACL 2003. pp. 33–40. Budapest (2003)
13. Przepiórkowski, A., Woliński, M.: The unbearable lightness of tagging: A case study in morphosyntactic tagging of Polish. In: Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03), EACL 2003. pp. 109–116 (2003)
14. Schütze, H.: Context space. In: AAAI Fall Symposium on Probabilistic Approaches to Natural Language. pp. 113–120. Cambridge (1992)
15. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco, 2nd edn. (2005), `http://www.cs.waikato.ac.nz/ml/weka/`