Reducing the Need for Double Annotation

Dmitriy Dligach
Department of Computer Science
University of Colorado at Boulder
Dmitriy.Dligach@colorado.edu

Martha Palmer
Department of Linguistics
University of Colorado at Boulder
Martha.Palmer@colorado.edu

Abstract

The quality of annotated data is crucial for supervised learning. To eliminate errors in single annotated data, a second round of annotation is often used. However, is it absolutely necessary to double annotate every example? We show that it is possible to reduce the amount of the second round of annotation by more than half without sacrificing the performance.

1 Introduction

Supervised learning has become the dominant paradigm in NLP in recent years thus making the creation of high-quality annotated corpora a top priority in the field. A corpus where each instance is annotated by a single annotator unavoidably contains errors. To improve the quality of the data, one may choose to annotate each instance twice and adjudicate the disagreements thus producing the gold standard. For example, the OntoNotes (Hovy et al., 2006) project opted for this approach.

However, is it absolutely necessary to double annotate every example? In this paper, we demonstrate that it is possible to double annotate only a subset of the single annotated data and still achieve the same level of performance as with full double annotation. We accomplish this task by using the single annotated data to guide the selection of the instances to be double annotated.

We propose several algorithms that accept single annotated data as input. The algorithms select a subset of this data that they recommend for another round of annotation and adjudication. The single annotated data our algorithms work with can potentially come from any source. For example, it can be the single annotated output of active learning or the data that had been randomly sampled from some corpus and single annotated. Our approach is applicable whenever a second round of annotation is being considered to improve the quality of the data.

Our approach is similar in spirit to active learning but more practical in a double annotation multi-tagger environment. We evaluate this approach on OntoNotes word sense data. Our best algorithm detects 75% of the errors, while the random sampling baseline only detects less than a half of that amount. We also show that this algorithm can lead to a 54% reduction in the amount of annotation needed for the second round of annotation.

The rest of this paper is structured as follows: we discuss the relevant work in section 2, we explain our approach in section 3, we evaluate our approach in section 4, we discuss the results and draw a conclusion in section 5, and finally, we talk about our plans for future work in section 6.

2 Related Work

Active Learning (Settles, 2009; Olsson, 2009) has been the traditional avenue for reducing the amount of annotation. However, in practice, serial active learning is difficult in a multi-tagger environment (Settles, 2009) when many annotators are working in parallel (e.g. OntoNotes employs tens of taggers). At the same time, several papers recently appeared that used OntoNotes data for active learning experiments (Chen et al., 2006; Zhu, 2007; Zhong et al., 2008). These works all utilized OntoNotes gold standard labels, which were obtained via double annotation and adjudication. The implicit assumption, therefore, was that the same process of double annotation and adjudication could be reproduced in the process of active learning. However, this assumption is not very realistic and in practice, these approaches may not bring about the kind of annotation cost reduction that they report. For example, an instance would have to be annotated by two taggers (and each disagreement adjudicated) on each iteration before the system can be retrained and the next instance selected. Active learning tends to select ambiguous examples (especially at early stages), which are likely to cause an unusually high number of disagreements between taggers. The necessity of frequent manual adjudication would slow down the overall process. Thus, if the scenarios of (Chen et al., 2006; Zhu, 2007; Zhong et al., 2008) were used in practice, the taggers would have to wait on each other, on the adjudicator, and on the retraining, before the system can select the next example. The cost of annotator waiting time may undermine the savings in annotation cost.

The rationale for our work arises from these difficulties: because active learning is not practical in a double annotation scenario, the data is *single* annotated first (with the instances selected via active learning, random sampling or some other technique). After that, our algorithms can be applied to select a subset of the single annotated data for the second round of annotation and adjudication. Our algorithms select the data for repeated labeling in a single batch, which means the selection can be done off-line. This should greatly simplify the application of our approach in a real life annotation project.

Our work also borrows from the error detection literature. Researchers have explored error detection for manually tagged corpora in the context of pos-tagging (Eskin, 2000; Květoň and Oliva, 2002; Novák and Razímová, 2009), dependency parsing (Dickinson, 2009), and text-classification (Fukumoto and Suzuki, 2004). The approaches to error detection include anomaly detection (Eskin, 2000), finding inconsistent annotations (van Halteren, 2000; Květoň and Oliva, 2002; Novák and Razímová, 2009), and using the weights assigned by learning algorithms such as boosting (Abney et al., 1999; Luo et al., 2005) and SVM (Nakagawa and Matsumoto, 2002; Fukumoto and Suzuki, 2004) by exploiting the fact that errors tend to concentrate among the examples with large weights. Some of these works eliminate the errors (Luo et al., 2005). Others correct them automatically (Eskin, 2000; Květoň and Oliva, 2002; Fukumoto and Suzuki, 2004; Dickinson, 2009) or manually (Květoň and Oliva, 2002). Several authors also demonstrate ensuing performance improvements (Fukumoto and Suzuki, 2004; Luo et al., 2005; Dickinson, 2009). All of these researchers experimented with single annotated data such as Penn Treebank (Marcus et al., 1993) and they were often unable to hand-examine all the data their algorithms marked as errors because of the large size of their data sets. Instead, to demonstrate the effectiveness of their approaches, they examined a selected subset of the detected examples (e.g. (Abney et al., 1999; Eskin, 2000; Nakagawa and Matsumoto, 2002; Novák and Razímová, 2009)). In this paper, we experiment with fully double annotated and adjudicated data, which allows us to evaluate the effectiveness of our approach more precisely. A sizable body of work exists on using noisy labeling obtained from low-cost annotation services such as Amazon's Mechanical Turk (Snow et al., 2008; Sheng et al., 2008; Hsueh et al., 2009). Hsueh et al. (2009) identify several criteria for selecting high-quality annotations such as noise level, sentiment ambiguity, and lexical uncertainty. (Sheng et al., 2008) address the relationships between various repeated labeling strategies and the quality of the resulting models. They also propose a set of techniques for selective repeated labeling which are based on the principles of active learning and an estimate of uncertainty derived from each example's label multiset. These authors focus on the scenario where multiple (greater than two) labels can be obtained cheaply. This is not the case with the data we experiment with: OntoNotes data is double annotated by expensive human experts. Also, unfortunately, Sheng et al. simulate multiple labeling (the noise is introduced randomly). However, human annotators may have a non-random annotation bias resulting from misreading or misinterpreting the directions, or from genuine ambiguities. The data we use in our experiments is annotated by humans.

3 Algorithms

In the approach to double annotation we are proposing, the reduction in annotation effort is achieved by double annotating only the examples selected by our algorithms instead of double annotating the entire data set. If we can find most or all the errors made during the first round of labeling and show that double annotating only these instances does not sacrifice performance, we will consider the outcome of this study positive. We propose three algorithms for selecting a subset of the single annotated data for the second round of annotation.

Our **machine tagger** algorithm draws on error detection research. Single annotated data unavoidably contains errors. The main assumption this algorithm makes is that a machine learning classifier can form a theory about how the data should be labeled from a portion of the single annotated data. The classifier can be subsequently applied to the rest of the data to find the examples that contradict this theory. In other words, the algorithm is geared toward detecting inconsistent labeling within the single annotated data. The machine tagger algorithm can also be viewed as using a machine learning classifier to simulate the second human annotator. The machine tagger algorithm accepts single annotated data as input and returns the instances that it believes are labeled inconsistently.

Our **ambiguity detector** algorithm is inspired by uncertainty sampling (Lewis and Gale, 1994), a kind of active learning in which the model selects the instances for which its prediction is least certain. Some instances in the data are intrinsically ambiguous. The main assumption the ambiguity detector algorithm makes is that a machine learning classifier trained using a portion of the single annotated data can be used to detect ambiguous examples in the rest of the single annotated data. The algorithm is geared toward finding hard-to-classify instances that are likely to cause problems for the human annotator. The ambiguity detector algorithm accepts single annotated data as input and returns the instances that are potentially ambiguous and thus are likely to be controversial among different annotators.

It is important to notice that the machine tagger and ambiguity detector algorithms target two different types of errors in the data: the former detects inconsistent labeling that may be due to inconsistent views among taggers (in a case when the single annotated data is labeled by more than one person) or the same tagger tagging inconsistently. The latter finds the examples that are likely to result in disagreements when labeled multiple times due to their intrinsic ambiguity. Therefore, our goal is not to compare the performance of the machine tagger and ambiguity detector algorithms, but rather to provide a viable solution for reducing the amount of annotation on the second round by detecting as much noise in the data as possible. Toward that goal we also consider a **hybrid** approach, which is a combination of the first two.

Still, we expect some amount of overlap in the examples detected by the two approaches. For example, the ambiguous instances selected by the second algorithm may also turn out to be the ones that the first one will identify because they are harder to classify (both by human annotators and machine learning classifiers). The three algorithms we experiment with are therefore (1) the machine tagger, (2) the ambiguity detector, and (3) the hybrid of the two. We will now provide more details about how each of them is implemented.

3.1 General Framework

All three algorithms accept single annotated data as input. They output a subset of this data that they recommend for repeated labeling. All algorithms begin by splitting the single annotated data into N sets of equal size. They proceed by training a classifier on $N-1$ sets and applying it to the remaining set, which we will call the *pool*[1]. The cycle repeats N times in the style of N-fold cross-validation. Upon completion, each single annotated instance has been examined by the algorithm. A subset of the single annotated data is selected for the second round of annotation based on various criteria. These criteria are what sets the algorithms apart. Because of the time constraints, for the experiments we describe in this paper, we set N to 10. A larger value will increase the running time but may also result in an improved performance.

[1]Notice that the term *pool* in active learning research typically refers to the collection of *unlabeled* data from which the examples to be labeled are selected. In our case, this term applies to the data that is *already labeled* and the goal is to select data for *repeated* labeling.

3.2 Machine Tagger Algorithm

The main goal of the machine tagger algorithm is finding inconsistent labeling in the data. This algorithm operates by training a discriminative classifier and making a prediction for each instance in the *pool*. Whenever this prediction disagrees with the human-assigned label, the instance is selected for repeated labeling.

For classification we choose a support vector machine (SVM) classifier because we need a high-accuracy classifier. The state-of-the art system we use for our experiments is SVM-based (Dligach and Palmer, 2008). The specific classification software we utilize is LibSVM (Chang and Lin, 2001). We accept the default settings ($C = 1$ and linear kernel).

3.3 Ambiguity Detector Algorithm

The ambiguity detector algorithm trains a probabilistic classifier and makes a prediction for each instance in the *pool*. However, unlike the previous algorithm, the objective in this case is to find the instances that are potentially hard to annotate due to their ambiguity. The instances that lie close to the decision boundary are intrinsically ambiguous and therefore harder to annotate. We hypothesize that a human tagger is more likely to make a mistake when annotating these instances.

We can estimate the proximity to the class boundary using a classifier confidence metric such as the prediction margin, which is a simple metric often used in active learning (e.g. (Chen et al., 2006)). For an instance x, we compute the prediction margin as follows:

$$Margin(x) = |P(c_1|x) - P(c_2|x)| \qquad (1)$$

Where c_1 and c_2 are the two most probable classes of x according to the model. We rank the single annotated instances by their prediction margin and select *selectsize* instances with the smallest margin. The *selectsize* setting can be manipulated to increase the recall. We experiment with the settings of *selectsize* of 20% and larger.

While SVM classifiers can be adapted to produce a calibrated posterior probability (Platt and Platt, 1999), for simplicity, we use a maximum entropy classifier, which is an intrinsically probabilistic classifier and thus has the advantage of being able to output the probability distribution over the class labels right off-the-shelf. The specific classification software we utilize is the python maximum entropy modeling toolkit (Le, 2004) with the default options.

3.4 Hybrid Algorithm

We hypothesize that both the machine tagger and ambiguity detector algorithms we just described select the instances that are appropriate for the second round of human annotation. The hybrid algorithm simply unions the instances selected by these two algorithms. As a result, the amount of data selected by this algorithm is expected to be larger than the amount selected by each individual algorithm.

4 Evaluation

For evaluation we use the word sense data annotated by the OntoNotes project. The OntoNotes data was chosen because it is fully double-blind annotated by human annotators and the disagreements are adjudicated by a third (more experienced) annotator. This type of data allows us to: (1) Simulate single annotation by using the labels assigned by the first annotator, (2) Simulate the second round of annotation for selected examples by using the labels assigned by the second annotator, (3) Evaluate how well our algorithms capture the errors made by the first annotator, and (4) Measure the performance of the corrected data against the performance of the double annotated and adjudicated gold standard.

We randomly split the gold standard data into ten parts of equal size. Nine parts are used as a *pool* of data from which a subset is selected for repeated labeling. The rest is used as a test set. Before passing the *pool* to the algorithm, we "single annotate" it (i.e. relabel with the labels assigned by the first annotator). The test set always stays double annotated and adjudicated to make sure the performance is evaluated against the gold standard labels. The cycle is repeated ten times and the results are averaged.

Since our goal is finding errors in single annotated data, a brief explanation of what we count as an error is appropriate. In this evaluation, the errors are the disagreements between the first annotator and the gold standard. The fact that our data

Sense Definition	Sample Context
Accept as true without verification	I *assume* his train was late
Take on a feature, position, responsibility, right	When will the new President *assume* office?
Take someone's soul into heaven	This is the day when Mary was *assumed* into heaven

Table 1: Senses of *to assume*

is double annotated allows us to be reasonably sure that most of the errors made by the first annotator were caught (as disagreements with the second annotator) and resolved. Even though other errors may still exist in the data (e.g. when the two annotators made the same mistake), we assume that there are very few of them and we ignore them for the purpose of this study.

4.1 Task

The task we are using for evaluating our approach is word sense disambiguation (WSD). Resolution of lexical ambiguities has for a long time been viewed as an important problem in natural language processing that tests our ability to capture and represent semantic knowledge and and learn from linguistic data. More specifically, we experiment with verbs. There are fewer verbs in English than nouns but the verbs are more polysemous, which makes the task of disambiguating verbs harder. As an example, we list the senses of one of the participating verbs, *to assume*, in Table 1.

The goal of WSD is predicting the sense of an ambiguous word given its context. For example, given a sentence *When will the new President assume office?*, the task consists of determining that the verb *assume* in this sentence is used in the *Take on a feature, position, responsibility, right, etc.* sense.

4.2 Data

We selected the 215 most frequent verbs in the OntoNotes data and discarded the 15 most frequent ones to make the size of the dataset more manageable (the 15 most frequent verbs have roughly as many examples as the next 200 frequent verbs). We ended up with a dataset containing 58,728 instances of 200 frequent verbs. Table 2 shows various important characteristics of this dataset averaged across the 200 verbs.

Inter-annotator agreement	86%
Annotator1-gold standard agreement	93%
Share of the most frequent sense	71%
Number of classes (senses) per verb	4.44

Table 2: Evaluation data at a glance

Observe that even though the annotator1-gold standard agreement is high, it is not perfect: about 7% of the instances are the errors the first annotator made. These are the instances we are targeting. OntoNotes double annotated *all* the instances to eliminate the errors. Our goal is finding them automatically.

4.3 System

Our word sense disambiguation system (Dligach and Palmer, 2008) includes three groups of features. Lexical features include open class words from the target sentence and the two surrounding sentences; two words on both sides of the target verb and their POS tags. Syntactic features are based on constituency parses of the target sentence and include the information about whether the target verb has a subject/object, what their head words and POS tags are, whether the target verb has a subordinate clause, and whether the target verb has a PP adjunct. The semantic features include the information about the semantic class of the subject and the object of the target verb. The system uses Libsvm (Chang and Lin, 2001) software for classification. We train a single model per verb and average the results across all 200 verbs.

4.4 Performance Metrics

Our objective is finding errors in single annotated data. One way to quantify the success of error detection is by means of precision and recall. We compute **precision** as the ratio of the number of errors in the data that the algorithm selected and the total number of instances the algorithm selected. We compute **recall** as the ratio of the number of errors in the data that the algorithm selected to the total number of errors in the data. To compute baseline precision and recall for an algorithm, we count how many instances it selected and randomly draw the same number of instances from the single annotated data. We then compute precision and recall for the randomly selected data.

We also evaluate each algorithm in terms of classification accuracy. For each algorithm, we measure the accuracy on the test set when the model is trained on: (1) Single annotated data only, (2) Single annotated data with a random subset of it double annotated[2] (of the same size as the data selected by the algorithm), (3) Single annotated data with the instances selected by the algorithm double annotated, and (4) Single annotated data with all instances double annotated.

4.5 Error Detection Performance

In this experiment we evaluate how well the three algorithms detect the errors. We split the data for each word into 90% and 10% parts as described at the beginning of section 4. We relabel the 90% part with the labels assigned by the first tagger and use it as a pool in which we detect the errors. We pass the pool to each algorithm and compute the precision and recall of errors in the data the algorithm returns. We also measure the random baseline performance by drawing the same number of examples randomly and computing the precision and recall. The results are in the top portion of Table 3.

Consider the second column, which shows the performance of the machine tagger algorithm. The algorithm identified as errors 16.93% of the total number of examples that we passed to it. These selected examples contained 60.32% of the total number of errors found in the data. Of the selected examples, 23.81% were in fact errors. By drawing the same number of examples (16.93%) randomly we recall only 16.79% of the single annotation errors. The share of errors in the randomly drawn examples is 6.82%. Thus, the machine tagger outperforms the random baseline both with respect to precision and recall.

The ambiguity detector algorithm selected 20% of the examples with the highest value of the prediction margin and beat the random baseline both with respect to precision and recall. The hybrid algorithm also beat the random baselines. It recalled 75% of errors but at the expense of selecting a larger set of examples, 30.48%. This is the case because it selects both the data selected by the machine tagger and the ambiguity detector. The size selected, 30.48%, is smaller than the sum, 16.93% + 20.01%, because there is some overlap between the instances selected by the first two algorithms.

4.6 Model Performance

In this experiment we investigate whether double annotating and adjudicating selected instances improves the accuracy of the models. We use the same pool/test split (90%-10%) as was used in the previous experiment. The results are in the bottom portion of Table 3.

Let us first validate empirically an assumption this paper makes: we have been assuming that full double annotation is justified because it helps to correct the errors the first annotator made, which in turn leads to a better performance. If this assumption does not hold, our task is pointless. In general repeated labeling does not always lead to better performance (Sheng et al., 2008), but it does in our case. We train a model using only the single annotated data and test it. We then train a model using the double annotated and adjudicated version of the same data and evaluate its performance.

As expected, the models trained on fully double annotated data perform better. The performance of the fully double annotated data, 84.15%, is the ceiling performance we can expect to obtain if we detect all the errors made by the first annotator. The performance of the single annotated data, 82.84%, is the hard baseline. Thus, double annotating is beneficial, especially if one can avoid double annotating everything by identifying the single annotated instances where an error is suspected.

All three algorithms beat both the hard and the random baselines. For example, by double annotating the examples the hybrid algorithm selected we achieve an accuracy of 83.82%, which is close to the full double annotation accuracy, 84.15%. By double annotating the same number of randomly selected instances, we reach a lower accuracy, 83.36%. The differences are statistically significant for all three

[2]Random sampling is often used as a baseline in the active learning literature (Settles, 2009; Olsson, 2009).

Metric	Machine Tagger, %	Ambiguity Detector, %	Hybrid, %
Actual size selected	16.93	20.01	30.48
Error detection precision	23.81	10.61	14.70
Error detection recall	60.32	37.94	**75.14**
Baseline error detection precision	6.82	6.63	6.86
Baseline error detection recall	16.79	19.61	29.06
Single annotation only accuracy	82.84	82.84	82.84
Single + random double accuracy	83.23	83.09	83.36
Single + selected double accuracy	83.58	83.42	**83.82**
Full double annotation accuracy	84.15	84.15	84.15

Table 3: Results of performance evaluation. Error detection performance is shown at the top part of the table. Model performance is shown at the bottom.

algorithms ($p < 0.05$).

Even though the accuracy gains over the random baseline are modest in absolute terms, the reader should keep in mind that the maximum possible accuracy gain is 84.15% - 82.84% = 1.31% (when all the data is double annotated). The hybrid algorithm came closer to the target accuracy than the other two algorithms because of a higher recall of errors, 75.14%, but at the expense of selecting almost twice as much data as, for example, the machine tagger algorithm.

4.7 Reaching Double Annotation Accuracy

The hybrid algorithm performed better than the baselines but it still fell short of reaching the accuracy our system achieves when trained on fully double annotated data. However, we have a simple way of increasing the recall of error detection. One way to do it is by increasing the number of instances with the smallest prediction margin the ambiguity detector algorithm selects, which in turn will increase the recall of the hybrid algorithm. In this series of experiments we measure the performance of the hybrid algorithm at various settings of the selection size. The goal is to keep increasing the recall of errors until the performance is close to the double annotation accuracy.

Again, we split the data for each word into 90% and 10% parts. We relabel the 90% part with the labels assigned by the first tagger and pass it to the hybrid algorithm. We vary the selection size setting between 20% and 50%. At each setting, we compute the precision and recall of errors in the data the algorithm returns as well as in the random baseline. We also measure the performance of the models trained on on the single annotated data with its randomly and algorithm-selected subsets double annotated. The results are in Table 4.

As we see at the top portion of the Table 4, as we select more and more examples with a small prediction margin, the recall of errors grows. For example, at the 30% setting, the hybrid algorithm selects 37.91% of the total number of single annotated examples, which contain 80.42% of all errors in the single annotated data (more than twice as much as the random baseline).

As can be seen at the bottom portion of the Table 4, with increased recall of errors, the accuracy on the test set also grows and nears the double annotation accuracy. At the 40% setting, the algorithm selects 45.80% of the single annotated instances and the accuracy with these instances double annotated reaches 84.06% which is not statistically different ($p < 0.05$) from the double annotation accuracy.

5 Discussion and Conclusion

We proposed several simple algorithms for reducing the amount of the second round of annotation. The algorithms operate by detecting annotation errors along with hard-to-annotate and potentially error-prone instances in single annotated data. We evaluate the algorithms using OntoNotes word sense data. Because OntoNotes data is double annotated and adjudicated we were able to evaluate the error detection performance of the algorithms as well as their accuracy on the gold standard test set. All three al-

Metric	Selection Size			
	20%	30%	40%	50%
Actual size selected	30.46	37.91	45.80	54.12
Error detection precision	14.63	12.81	11.40	10.28
Error detection recall	75.65	80.42	83.95	87.37
Baseline error detection precision	6.80	6.71	6.78	6.77
Baseline error detection recall	29.86	36.23	45.63	53.30
Single annotation only accuracy	83.04	83.04	83.04	83.04
Single + random double accuracy	83.47	83.49	83.63	83.81
Single + selected double accuracy	83.95	83.99	84.06	84.10
Full double annotation accuracy	84.18	84.18	84.18	84.18

Table 4: Performance at various sizes of selected data.

gorithms outperformed the random sampling baseline both with respect to error recall and model performance.

By progressively increasing the recall of errors, we showed that the hybrid algorithm can be used to replace *full* double annotation. The hybrid algorithm reached accuracy that is not statistically different from the full double annotation accuracy with approximately 46% of data double annotated. Thus, it can potentially save 54% of the second pass of annotation effort without sacrificing performance.

While we evaluated the proposed algorithms only on word sense data, the evaluation was performed using 200 distinct word type datasets. These words each have contextual features that are essentially unique to that word type and consequently, 200 distinct classifiers, one per word type, are trained. Hence, these could loosely be considered 200 distinct annotation and classification tasks. Thus, it is likely that the proposed algorithms will be widely applicable whenever a second round of annotation is being contemplated to improve the quality of the data.

6 Future Work

Toward the same goal of reducing the cost of the second round of double annotation, we will explore several research directions. We will investigate the utility of more complex error detection algorithms such as the ones described in (Eskin, 2000) and (Nakagawa and Matsumoto, 2002). Currently our algorithms select the instances to be double annotated in one batch. However it is possible to frame the selection more like batch active learning, where the next batch is selected only after the previous one is annotated, which may result in further reductions in annotation costs.

Acknowledgements

We gratefully acknowledge the support of the National Science Foundation Grant NSF-0715078, Consistent Criteria for Word Sense Disambiguation, and the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022, a subcontract from the BBN-AGILE Team. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Steven Abney, Robert E. Schapire, and Yoram Singer. 1999. Boosting applied to tagging and pp attachment. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 38–45.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines.*

Jinying Chen, Andrew Schein, Lyle Ungar, and Martha Palmer. 2006. An empirical study of the behavior of active learning for word sense disambiguation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 120–127, Morristown, NJ, USA. Association for Computational Linguistics.

Markus Dickinson. 2009. Correcting dependency annotation errors. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 193–201, Morristown, NJ, USA. Association for Computational Linguistics.

Dmitriy Dligach and Martha Palmer. 2008. Novel semantic features for verb sense disambiguation. In *HLT '08: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 29–32, Morristown, NJ, USA. Association for Computational Linguistics.

Eleazar Eskin. 2000. Detecting errors within a corpus using anomaly detection. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 148–153, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fumiyo Fukumoto and Yoshimi Suzuki. 2004. Correcting category errors in text classification. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 868, Morristown, NJ, USA. Association for Computational Linguistics.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *NAACL '06: Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers on XX*, pages 57–60, Morristown, NJ, USA. Association for Computational Linguistics.

Pei-Yun Hsueh, Prem Melville, and Vikas Sindhwani. 2009. Data quality from crowdsourcing: a study of annotation selection criteria. In *HLT '09: Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*, pages 27–35, Morristown, NJ, USA. Association for Computational Linguistics.

Pavel Květoň and Karel Oliva. 2002. (semi-)automatic detection of errors in pos-tagged corpora. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Zhang Le, 2004. *Maximum Entropy Modeling Toolkit for Python and C++*.

David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12, New York, NY, USA. Springer-Verlag New York, Inc.

Dingsheng Luo, Xinhao Wang, Xihong Wu, and Huisheng Chi. 2005. Learning outliers to refine a corpus for chinese webpage categorization. In *ICNC (1)*, pages 167–178.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330.

Tetsuji Nakagawa and Yuji Matsumoto. 2002. Detecting errors in corpora using support vector machines. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Václav Novák and Magda Razímová. 2009. Unsupervised detection of annotation inconsistencies using apriori algorithm. In *ACL-IJCNLP '09: Proceedings of the Third Linguistic Annotation Workshop*, pages 138–141, Morristown, NJ, USA. Association for Computational Linguistics.

Fredrik Olsson. 2009. A literature survey of active machine learning in the context of natural language processing. In *Technical Report, Swedish Institute of Computer Science*.

John C. Platt and John C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press.

Burr Settles. 2009. Active learning literature survey. In *Computer Sciences Technical Report 1648 University of Wisconsin-Madison*.

Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 614–622, New York, NY, USA. ACM.

Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. 2008. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Morristown, NJ, USA. Association for Computational Linguistics.

Hans van Halteren. 2000. The detection of inconsistency in manually tagged text. In *Proceedings of LINC-00, Luxembourg*.

Z. Zhong, H.T. Ng, and Y.S. Chan. 2008. Word sense disambiguation using OntoNotes: An empirical study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1002–1010. Association for Computational Linguistics.

Jingbo Zhu. 2007. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *In Proceedings of ACL*, pages 783–790.