Efficient Syntactic Analysis with Active Learning

Bastiaan Zijlema

31st August 2007

Supervised by:
Dr. Gertjan van Noord
Dr. Gerard Renardel de Lavalette

Computer Science
RijksUniversiteit Groningen

Contents

Chapter 1

Introduction

1.1 Natural language processing

1.1.1 Overview

Natural language processing is a collective term for computational techniques that process written and spoken human language. Shortly said, it is an attempt to make a computer understand and process our own language.

Natural language processing (NLP) can have many practical applications. One can imagine operating a computer by just talking to it instead of entering text through a keyboard. There already exist some travel information systems which use natural language as an interface. Users can call the system by phone, and just say where they want to go and at what time. The system will retrieve the right information and give it to the user. The main advantage of systems using natural language as an interface, is that users can operate it in a very intuitive way and with a minimum of required training.

An example where NLP is used in a more sophisticated way is an automatic translation system. This is a computer system which translates text from one language into another. Because the structures of a language often differ strongly, it is impossible to make a one on one translation most of the time. So instead of only translating the words of a sentence, it is necessary to determine the meaning of a sentence. Subsequently, a syntactically correct sentence with the same meaning can be created in the destination language. It is obvious that a great amount of knowledge about the structures and meanings of a human language is required to build such a translation system.

Human language is very different from computer language. Computer language is defined by strict rules, and therefore it is very structured and unambigous. Human language however is often missing such strict rules. Because of that it is much more inconsistent and ambiguous, i.e. one word or sentence can have several different meanings. Therefore, it is not straightforward to make a computer understand natural language. To accomplish this goal, techniques and knowledge from various fields of science are used. The foundations of NLP lie in computer science, linguistics, mathematics, electrical engineering and psychology. The total process of NLP can be separated into different levels:

- **Speech recognition** - When processing spoken language, the first thing to do is to translate it to written text. The sound of the spoken language has to be converted into words and sentences. This text can then be processed further.
- **Words** - Every natural language consists of words, which are the atoms of a sentence. To understand a language the meaning of the words must be determined. Usually a dictionary is used to achieve this.
- **Syntax** - The structural relationship between words. By looking at the structure of a sentence the correlation between the words and phrases can be determined. For example, by looking at the syntax, it can be established what the subject or the object of a particular sentence is. Typically a grammar is used to determine the syntax.
- **Semantics** - The process of determining the meaning of a sentence. The meaning of the words together with the syntax define the semantics of a sentence.
- **Pragmatics** - The study of the meaning of a sentence in relation to its context. Two identical sentences can have a totally different meaning if the context in which they are used vary.

Most of the problems at all of these levels are caused by ambiguity. Every sound, every word, and every sentence can have multiple meanings, and most of the time they do. To understand, process or translate a text of a given language, it is important to choose the correct one of all possible meanings. Therefore, a large portion of natural language processing consists of resolving these ambiguities, as we will see in the next subsection.

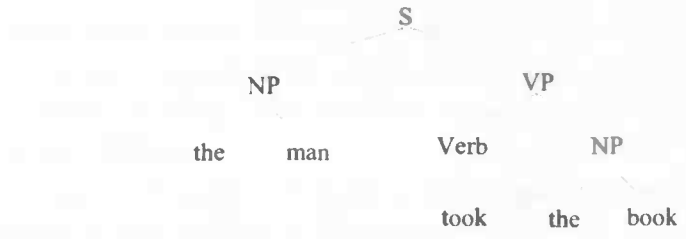

Figure 1.1: Parse tree of the sentence: "The man took the book"

1.1.2 Parsing

Retrieving the syntax of a given sentence is an important aspect of NLP. A parser is a piece of software that is used to determine the syntax, which is usually called the parse. Generally a parse tree or derivation tree is created to represent the syntax of a sentence graphically. An example can be seen in figure 1.1, which the parse tree of the sentence: "The man took the book." As we can see all words are grouped in so called phrases, which are then structured in a hierarchical way. The abbreviations NP and VP stand for "Noun Phrase" and "Verb Phrase" respectively. Usually a grammar is used to generate the parse of a sentence. A grammar consists of a set of production rules, which describe the way words can be combined to form phrases and sentences, as well as a lexicon, which contains the words of the language. The lexicon contains information about the words. Especially what kind of word it is, e.g. a verb or a noun. The rules of the grammar can then be used to group the words together in phrases, and phrases into sentences. The following three rules are an example of such a grammar:

(1) S $\rightarrow$ NP VP
(2) NP $\rightarrow$ det noun | noun
(3) VP $\rightarrow$ verb NP

The first rule prescribes that a sentence (S) is formed by a noun phrase (NP) followed by a verb phrase (VP). The second rule states that a noun phrase can be formed by a determiner (Det) followed by a noun, or by a noun only. The third rule defines that a verb phrase consists of a verb followed by a noun phrase. These three rules, together with a lexicon which contains the words used in the

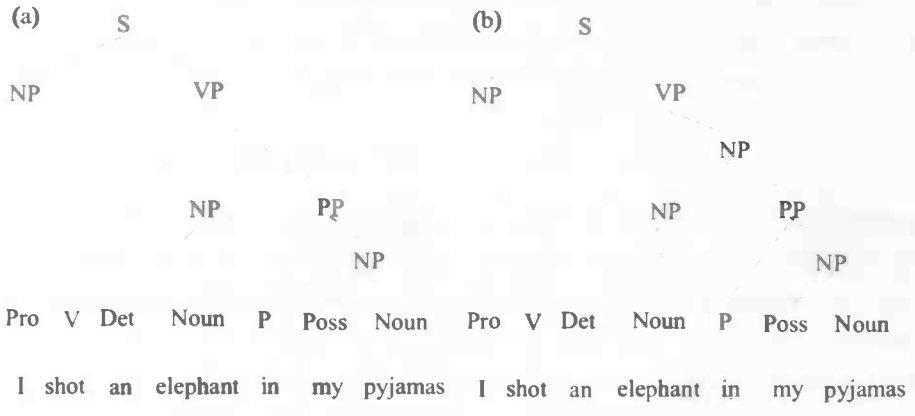

Figure 1.2: Two parse trees for an ambiguous sentence

example sentence, can be used to form the parse tree of figure 1.1. This example is of course a very simple version of a grammar. A grammar which can generate nearly all sentences of a particular language, usually needs a great amount of production rules and a large lexicon to cover all the words. As the amount and the complexity of the rules increase, the problem of ambiguity arises. One can imagine that the grammar can produce the same sentence in more than one way and with a different parse tree. An example of this can be seen in figure 2. Here we see two parse trees of the sentence "I shot an elephant in my pyjamas." The example was taken from Jurafsky [1] and the sentence was originally stated by Groucho Marx in Animal Crackers. Note that the two parse trees represent two different meanings of the sentence. The first parse represents the meaning that I shot an elephant while I was wearing my pyjamas. The second tree represents the meaning that the elephant was wearing my pyjamas when I shot it. The correct meaning, and therefore the correct parse tree, seems to be the first one, but gramatically both parse trees are correct. When sentences become longer and more complex, generally the number of parses increases drastically. For some sentences a grammar can even produce over 10.000 parses.

The goal of a parser is to retrieve the correct parse of a particular sentence. To achieve this, the grammar first generates all possible parses, and than attributes probabilities to each one of them. The parse with the highest probability is considered to be the correct one. To acquire the ability to choose the correct parse out of all possible parses, the parser can learn from other sentences. By analyzing a set of sample sentences and their correct parses, the

parser can determine relations between specific features of a sentence and the corresponding syntax. It can establish what structures occur more frequently than others in combination with particular words or phrases. Subsequently, the parser can deduct rules which can be used to retrieve the correct parse of arbitrary sentence.

A parser that operates in this way is actually called a statistical parser, because it uses statistics to determine the probability of a particular sentence. The more sentences the parser analyzes, the more information and parsing rules it can extract from it. So, for the parser to become reliable, a large number of sample sentences is needed. Such a collection of sentences is usually called a corpus. When the corpus also contains the correct parses of the sentences, it is said to be annotated. Annotation is the process of providing a given sentence with its correct parse.

1.2 Research scope

1.2.1 Corpus creation

For many tasks in the field of natural language processing it is necessary to have an annotated corpus. For example, when training a statistical parser, a sentence will be provided and then the parser will produce a corresponding parse tree. By comparing its own results with the correct parse, the parser can determine the errors made and adjust its weights accordingly. To train the parser well, a large amount of annotated sentences is required.

In many cases, a proper annotated corpus is not available. Some tasks might require a collection of sentences with particular properties. For example, to develop a question answering system, one might need a corpus which only consists of questions. Sometimes a larger corpus than the existing one is needed, and for several languages such corpora do not even exist. So in these cases, a corpus has to be created.

Creating an annotated corpus means selecting a number of sentences, and next determining the correct parse of each of the sentences. The process of annotation has to be accomplished by a human annotator, and therefore it is a very time-consuming job. Creating such an annotated corpus often turns out to be quite expensive because of the human labour it requires.

In this paper, we try to develop a method for reducing these annotation costs. Because man-labour is more expensive than computer-labour, it would

be desirable to let the computer do most of the work. Another possibility is to reduce the required amount of training material. If less sentences need to be annotated, the total annotation costs will automatically decrease.

1.2.2 Active learning

Active learning is a common technique in machine learning. Active learning means that the model which is learning, can select its own training material. In contrast to passive learning, where the training material is provided by the user, the data can be selected on certain criteria to accelerate the learning process. The main idea is that the model will choose the most informative samples from a large set of training data. This process is called sample selection.

In the case of natural language processing, this means that the system will select sentences of which it thinks it can learn the most. If a model performs well on a certain type of sentence, it is not very useful to provide the system with another sentence of this kind. The system will probably learn a lot more from one of a very different type, or one for which it performs poorly.

By choosing the most informative samples from a large set of training data, the total amount of training material can be reduced. By selecting the material in a clever way, i.e. avoiding low-informative sentences, the system can achieve the same performance level with less training data. Consequently, the total amount of data that has to be annotated is also reduced. The sentences in the corpus which are redundant are not selected by the model, so needless annotation can be avoided.

In the literature, we can see that active learning can be very effective in natural language processing. In her paper Rebecca Hwa [2] shows that even a reduction of 27% of the training material can be achieved while training a statistical parser, and even better results have been reported. So this method seems very promising in reducing the total annotation costs.

But this method has a large disadvantage for corpus creation. When the model is selecting training material, it chooses those sentences which are the most informative for itself at that moment. In this way, a very specific corpus is created which is particularly suited for this model. Using the same training set with another model might not lead to similar results. Baldridge and Osborne [3] show that reusing training material, which is achieved with active learning, with another model, can result in weak performance. It can even perfrom worse than training with a randomly chosen training set. So reusing training material

achieved with active learning may not lead to improvement and can even be harmful if a different model is used. This implies the training material selected by the first model is biased and should not be used on another model.

This means that active learning is a good thing if only one model is used. But in practice, this is not very likely. It is very common that a model evolves over time and changes drastically. In this case it is possible that the training material selected by the original model, may be inappropriate for the renewed one. And because annoting a corpus is very costly and time consuming, it is often reused for several activities. As a consequence it is much more useful to have an allround corpus which is capable of various tasks, instead of a specialized one which can only be used once. In this paper, we therefore concentrate on annotating a given corpus consisting of randomly chosen sentences, instead of a corpus where the sentences are selected with active learning.

1.2.3 Semi-automated labeling

When annotating a corpus each sentence is labeled with its correct parse. This parse can be created from scratch, but this is not very efficient. It can be very helpful to make use of an automated parser. This parser can rule out some of the labeling possibilities, or it can give the annotator a suggestion for the correct parse of the sentence. Then the annotator can determine the correct parse with the help of the information given by the parser. For example, the annotator can take the suggestion of the parser and, if necessary, can simply adjust the suggestion until the parse is correct. Annotating with the help of an automated parser is called semi-automated labeling. Using this technique the sentences can be annotated much more efficiently than annotating them from scratch.

If the parser is well trained, there is a large chance that the suggestion given by the parser is close to the correct parse. However, sometimes when a sentence is very difficult or the parser is not well trained the suggestion from the parser can be very inaccurate. If this is the case the annotator has to do a lot of adjustments, and annotation is more expensive. If a statistical parser is used for semi-automated labeling, the model can learn from annotated sentences and improve on performance. We assume that the parser is not trained at all in the beginning of the annotation process. During the creation of the corpus, more annotated material will be available continuously. If the model is trained wilth all sentences annotated so far, the performance of the parser can

improve constantly. So in the end of the annotating process, when the model is performing better than in the beginning, relatively less work has to be done by the annotator. In this way there is more work to be done at the beginning of the annotation than in the end. This is especially the case when the order of the sentences which are annotated is random.

This suggests that there might be a difference in total annotation costs if the order in which the sentences are being annotated varies when using semi-automated labeling. One can imagine that if the model is learning a lot from the sentences in the beginning of the annotation process, it will perform better on the remaining part of the corpus. In this way it can make better suggestions for the later sentences, and their annotation costs will decrease. So by providing the model with high-informative samples in the beginning, the annotation costs in the remaining part of the process will be lower. In doing so, it might be that the total annotation costs will substantially decrease compared to annotating the sentences in a random order.

On the other hand, if the model is learning a lot from the first sentences, it probably did not perform well on these sentences before they were learned. The most informative sentences are generally the ones which are the hardest for the model to annotate. It is very likely that the model had huge problems finding the correct parse for these sentences and the suggestion it provided to the annotator was very poor. Consequently, the annotation costs for these first sentences were probably fairly high. These extra costs might be neutralizing the benefits of the better performance of the model.

Therefore, a good strategy of reducing the total annotation costs could be the exact opposite of the previous one. Instead of providing the model first with the most informative (and likely harder) sentences, annotate the easy ones first. Due to this the model will learn much slower, but it will make fewer mistakes in the beginning of the process. The model will gradually increase in performance, while the sentences gradually become harder to annotate. In the beginning, when the model is not performing as good as it could be, the easier sentences are annotated. At the end when the hardest sentences have to be annotated, the model will perform at his best, and will probably make fewer mistakes than it would be if the same sentence had to be annotated in the beginning of the process. This strategy could turn out to be quite as effective as the previous one.

But how can we determine which are the easy sentences, and which the hard ones? One simple method is to sort the sentences by length. Parsers seem to

have little trouble with short sentences, which mostly do not contain complex grammatical structures. Long sentences on the other hand, do often contain complex structures and are much more ambiguous. Therefore, parsers tend to perform much worse on them. So if the shorter sentences are being annotated and learned first and then the longer ones, or in reversed order for that matter, we can possibly provide the model with the easiest or the hardest sentences first, and therefore reduce the total annotation costs.

Probably a more reliable method for determining the harder and easier sentences, is active learning. As described above, active learning is looking for those sentences on which the model is performing worst. But of course, it can also be used to determine the sentences for which the model will probably perform well. By evaluating all the available sentences, the model can determine which sentences are the easiest or the hardest for the model at that moment. If we use active learning to choose training data continuously during the annotation process, we are fairly sure that the hardest or easiest sentences are selected. So, by not using active learning in the traditional way, i.e. reducing the needed amount of training data, but let active learning determine the order in which the samples are presented to the model, we might have found a way to reduce the costs of creating an annotated corpus.

1.2.4 Problem statement

The goal of this paper is to reduce the annotation costs of a given corpus. Therefore, we try to find a good strategy to estimate an order in which the sentences should be annotated using semi-automated labeling, and we are especially interested to find out whether active learning can help us in this matter. So, the problem statement is: *Can active learning reduce the total annotation costs of a given corpus?*

Chapter 2

Active Learning

2.1 Introduction

Active Learning is a very promising and relatively new technique from the field of machine learning. The primary goal of active learning is to train a learning system more efficiently. The main idea of this technique is that the system takes the initiative in determining what data is used to train with. Usually the training data is chosen by a learner which tries to compose a balanced training set or just at random. With active learning the system can create his own training examples, state what kind of training data it requires, or choose its training data from a large collection of samples. The last technique is usually called sample selection. By letting the system select the samples in an intelligent way, the system can achieve the same performance level using less training material. The system will select the samples which are the most informative, i.e. the samples from which the system can learn the most. The less informative samples are ignored, so no time is wasted by training with samples which are a minor contribution to the performance of the system. Sample selection can determine the weak spots of the system and select the right samples to improve the system.

Sample selection is particularly useful if raw training data is widely available, but labeling it is an expensive or difficult task. This is often the case in natural language processing, where training data usually consists of annotated sentences or phrases. Sentences are easily obtained, but labeling them is a costly and time consuming job. By using sample selection not every sample has to be labeled

and the annotation cost van be reduced. Several researchers have succesfully applied sample selection on NLP, for example [2],[4] and [5]. We will take a closer look at some of them later.

The general procedure of sample selection can be represented by the following pseudo code:

U : unlabeled candidates
L : labeled training samples
S : current state of the system
Initialize
    $S \leftarrow Train(L)$
Repeat
    $N \leftarrow Select(n, U, S, f)$
    $U \leftarrow U - N$
    $L \leftarrow L \cup Label(N)$
    $S \leftarrow Train(L)$
Until (S is good enough) **or** ($U = \emptyset$)

First we need a small amount of labeled training material L and a large collection of potential training data U, which consists of unlabeled samples which might be more or less useful for the system to train with. The system will be initialized by training with the initial amount of available labeled training material. Then we use an evaluation function f to determine how informative each sample of U is for the current state of the system. This is done by assigning a score to each of the samples. The samples with the highest scores are the most informative ones. The n samples with the highest scores are selected and labeled. These samples are then removed from U and added to the collection of labeled training data L. Next, the system is trained with the new set of labeled data. After this, the state of the system will be changed because it is trained with more and new material. Then the whole process is repeated from the point where the unlabeled samples are evaluated and selected. The algorithm will stop when the system reaches a certain level of performance, or when there is no more unlabeled training data to select.

Because the state of the system changes in every cycle of the algorithm, it is likely for the evaluation function f to rate the samples with different scores than during the previous cycle. The function will select the n samples which

are the most informative at that particular state of the system. Therefore, the algorithm works best for low values of n, with $n = 1$ as the optimum. If n is too large there might be redundant information in the selected training material. If only few samples are added to the training data, the system can adapt more quickly to its own changes and select training data more efficiently. However, if there is only one sample selected in every cycle of the process a lot of computation is required. All samples of U have to be evaluated in every cycle, so more cycles means more evaluation. Therefore, a good value for n often depends on the situation but should be kept as small as possible.

But how can we determine how informative a sample is for the system? There are mainly two approaches for sample selection, certainty based [6] and committee based [7, 8].

Certainty based sample selection, often called uncertainty sampling, assumes that a sample for which the system is uncertain how to classify it, might be a good training example. That is, if the system probably cannot label a sample correctly, it is likely that the system can learn a lot from it. The system will examine the available training samples and estimates the probability that it will classify each sample correctly. The samples with the lowest probability will be selected for training.

Committee based sample selection uses a set of classifiers. The potential training data is presented to each classifier which tries to label the examples. The samples which cause the most disagreement among the classifiers are selected for training. If classifiers disagree on the label of a certain sample it is probably a very informative one. This method is working best if the used classifiers are complementary, i.e. each classifier possesses particular characteristics which sum up to a balanced and universal classification system. In the literature, good results have been reported for NLP using both committee based [9] and certainty based [2] sample selection methods. However, using committee based sample selection requires a lot of computation, and for our purpose this might be impracticle. Therefore, and because it is a good alternative, we will focus on uncertainty sampling only.

2.2 Uncertainty Sampling

In this paper we try to reduce the annotation costs of corpus creation using active learning. Therefore, we take a closer look at how sample selection is

implemented for natural language processing. Suppose we have a statistical parser. The parser takes a sentence as an input, and gives what he thinks is the most probable parse of that sentence as an output. The parser can be trained by providing it with annotated sentences. We also assume there is a large collection of unlabeled sentences available. We now like to select the most informative sentences for the current state of the statistical parser. In order to define how informative each sentence is, we need a good evaluation function. Because we focus on uncertainty sampling, this function must determine how uncertain the parser is about every sentence. In the literature we mainly find two methods to determine uncertainty: error-driven and probability distribution. In the next two subsections these methods are further explained and corresponding evaluation functions will be defined.

2.2.1 Error-driven function

With this method we want to estimate the chance that the parser will make an error in the classification of a sentence. To do this, we look at how certain the parser is about the most probable parse. As we have seen in chapter 1 a parser first generates all possible parses for a sentence and then determines which is the correct one. That is, the parser is assigning probabilities to all possible parses and the parse with the highest probability is what the parser considers to be the right one. The main idea is that if the probability of the most likely parse is very high, the model is reasonably certain about the classification of that sentence. If the probability is relatively low, the uncertainty is higher and it is more probable for the model to make a mistake. If the sentences are selected with the highest chance for the parser to make a mistake, we have probably retrieved the most informative ones.

Basically, the chance that the parser will make a mistake is approximated by 1 minus the relative probability of the most likely parse. To compute the error-driven evaluation function f_{err}, we first need to know the probability of the most likely parse of a sentence. Therefore, we define the collection of possible parses for a given sentence w as V. Every individual parse $v \in V$ is assigned a probability $P(v|w)$ by the parser, which represents the likelyhood that v is the correct parse for sentence w. The probability is the sum of the probabilities of all possible parses and equals 1. The probabilities of each parse are rated between 0 and 1:

$$P(w) = \sum_{v \in V} P(v|w) = 1,$$

where

$$P : V \rightarrow [0, 1].$$

The parse with the highest probability for w given by the parser can be written as:

$$v_{max} = Max_{v \in V} P(v|w).$$

When we know the probability for v_{max} the error-driven evaluation function can be easily obtained:

$$f_{err}(w) = 1 - P(v_{max}|w). \tag{2.1}$$

The error driven evaluation function is fairly easy to implement. Because the parser is assigning probabilities to every possible parse for sentence w anyway, a simple script can be used to compute formula (2.1).

2.2.2 Probability distribution

Probably the most widely used strategy for sample selection in NLP is to base the uncertainty of a sentence on the probability distribution of all parses. The main idea is that if the probabilities of all possible parses of a sentence are uniformly distributed, there are multiple candidates to be the correct parse. There is no obvious right parse for the sentence, so the system is uncertain which is the correct one. The more balanced the probabilities of the parses are distributed, the more uncertain the parser. For example, if there are four possible parses for a sentence and the parser rates each of them with a probability of 0.25, all parses are equally likely and the probabilities are distributed uniformly. But if the parser rates one parse with a probability of 0.2 and eighty others with a probability of 0.01 (so they also sum up to 1), the distribution is more scattered and the parser has a clear preference for one of the candidates. Because of that the evaluation function will rate the first sentence with a higher score than the second one.

Note that this approach is essentially different from the error-driven function. The error driven function selects on the lowest probability of the most

likely parse. In the previous example these probablities were 0.25 and 0.2 respectively, so the error-driven function would assign a higher score to the second sentence, while the probability distribution based function would rate the first sentence higher.

To quantify the extent of distribution of the probabilities, we can calculate the entropy of the distribution. In information theory, the entropy is defined by the following equation [10]:

$$H(V) = -\sum_{v \in V} P(v) log(P(v)). \tag{2.2}$$

Here V is the total collection of possible outcomes, and $P(v)$ defines the probability of each outcome $v \in V$. Now we want to calculate the entropy of a distribution of a sentence w. When we use equation (2.2) this means that V contains all possible parses for w and $P(v|w)$ computes the probability of one parse $v \in V$, where P has the same restrictions as in the previous subsection. We can now compute the entropy for sentence w using equation (2.2), so that our entropy based evaluation function f_{unc} will be:

$$f_{unc} = -\sum_{v \in V} P(v|w) log(P(v|w)).$$

The higher the entropy, the more homogeneous the probability distribution and the more uncertain the parser. However, if we base our evaluation function exclusively on entropy, it will probably tend to select sentences with many parses. The more possible parses a sentence contains, the higher the entropy of the sentence. Because longer sentences generally possess more parses, the function will probably prefer longer sentences over shorter ones. In the literature, we find two methods to prevent this. The first one is normalizing the entropy by the number of parses, proposed by Rebecca Hwa[4]. The entropy can be normalized by dividing the entropy by the log of the number of parses for that sentence. If we implement this, the final evaluation function f_{unc} will be:

$$f_{unc} = -\frac{\sum_{v \in V} P(v|w) log(P(v|w))}{log(||V||)}. \tag{2.3}$$

The second method for preventing the function to prefer long sentences is called word entropy [5]. The idea of this technique is to divide the entropy by

the length of the sentence l. This makes the evaluation function:

$$f_{unc} = -\frac{\sum_{v \in V} P(v|w) log(P(v|w))}{l}. \tag{2.4}$$

By dividing the entropy directly by the number of words in a sentence it calculates the entropy per word. In this way the effect of longer sentences having a higher entropy is compensated.

2.3 Results of Active Learning in the literature

In the literature, several examples can be found where Active Learning has been succesfully implemented in various research fields. In the field of Natural Language Processing some good results have been reported for sample selection and uncertainty sampling in particular. Therefore, we will look at two scientific articles on sample selection for statistical parsing where uncertainty sampling is used. The results of some of the experiments will be critically examined, and we will look at how these experiments are conducted and what we can learn from them. We are especially interested in the sample selection methods as described in section 2.2, but in the articles discussed, some other techniques are used in an attempt to refine the process of sample selection and maximize results. It is beyond the scope of this paper to explore the frontiers of active learning and try to improve sample selection. Therefore, we will focus on the techniques described earlier and ignore the others.

2.3.1 Article 1

The first article examined is Active Learning for Statistical Natural Language Parsing by Min Tang, Xiaoqiang Luo and Salim Roukos [4]. In this paper a statistical parser is created and trained for an air travel dialog system. They are using uncertainty sampling to see if the parser can be trained more efficiently. The technique used for sample selection is based on probability distribution, as described in subsection 2.2.2. First the entropy of the sentence as a whole is computed which they call sentence entropy. They also test the technique of word entropy by normalizing the sentence entropy by the number of words in the sentence. The statistical parser is first trained with an initial data set of 1000 sentences. There is a large pool of more than 20.000 labeled sentences from which the active learner can select its training samples. Next, in every cycle 100

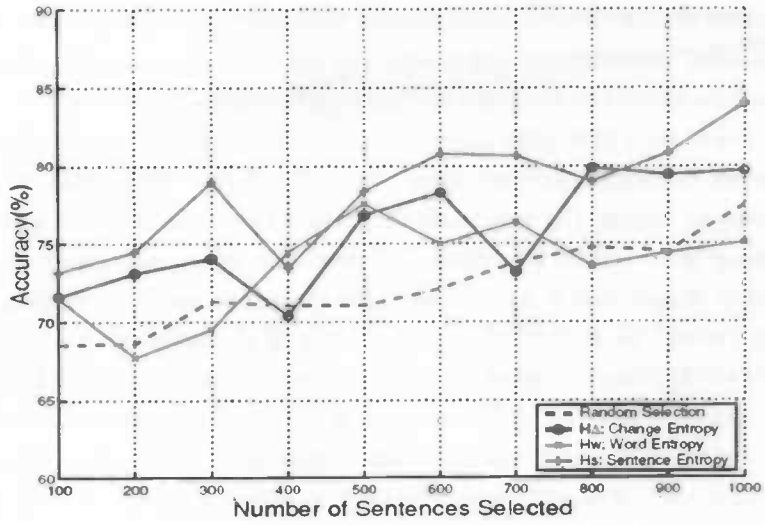

Figure 2.1: Results for active learning by Tang et al.

sentences are selected from this pool and the system is trained with them. After each cycle the accuracy of the parser is tested on an independent test set. The accuracy is the percentage of sentences for which the parser generated the correct parse tree. This experiment is executed for both sample selection techniques and the results are compared with random selection of the sentences. The outcome of the experiments are presented in figure 2.1. The graph also shows the results of a third technique called change of entropy, but we will ignore it for reasons described earlier.

We can see from the graph that sentence entropy leads to higher accuracy than random selection with the same amount of sentences used. But word entropy is giving much worse results and is sometimes even outperformed by random selection. The results for both sentence entropy and word entropy look very unstable, so this graph is not very convincing anyway. But more important is the unit which is used on the x-axis, i.e. the number of selected sentences. So in this graph the accuracy of the parser is compared to the number of sentences learned so far. This does not seem to be very sensible, because the length of sentences can differ dramatically. Therefore, the number of sentences do not necessarily indicate the amount of training data. And because longer sentences generally contain more information than shorter ones, the parser probably will learn more from five long sentences than from ten short ones. As we saw in the

previous section entropy based sample selection tends to select long sentences. Therefore, the better results of sentence entropy over random selection might as well be caused by the selection of longer sentences. Moreover, the authors of the article state that word entropy is not performing well in this graph because it tends to select short sentences. This should give them more than enough reasons to change the basic unit to present their results. The goal of active learning is to reduce the amount of needed training material, but when the number of sentences is used as the basic unit, this cannot be verified. A better choice would be to take the total number of words as the basic unit, which gives a better indication of the total amount of training material. The actual goal of active learning in this situation is to reduce the annotation costs, so the desirable unit would be the total annotation costs of the selected sentences so far. However, it is not straightforward to define what the actual annotation costs are as we will see in the next chapter.

Since the basic unit is not representative for the amount of training material, let alone the annotation costs, the results in this graph and even in the whole paper are nearly worthless. Because the length of the selected sentences is unknown, no legitimate conclusions can be derived by the presented data.

2.3.2 Article 2

The second article we are going to review is Sample Selection for Statistical Parsing by Rebecca Hwa [2]. In this paper she runs a number of experiments to test the effect of sample selection on statistical parsing. The experiments are performed on two different kinds of parsers, an Expectation-Maximization based parser and a history-based statistical parser. Because the history-based parser resembles the parser used in our experiments the most we will focus on the outcomes of this experiment. More information about the testing environment of our experiments will be provided in chapter 4.

The model used in this experiment is Collins Model 2 parser [11], which is a fully supervised history-based statistical parser. The corpus used is part of the Penn-treebank [12] which constists of labeled sentences taken from the Wall Street Journal. The model is initially trained with 500 labeled seed sentences and each cycle 100 sentences are selected and added to the training data. There is a large pool of around 39.000 unlabeled sentences from which new training data can be selected. The training samples are selected by four different algorithms. First, there is the baseline strategy which randomly selects training data

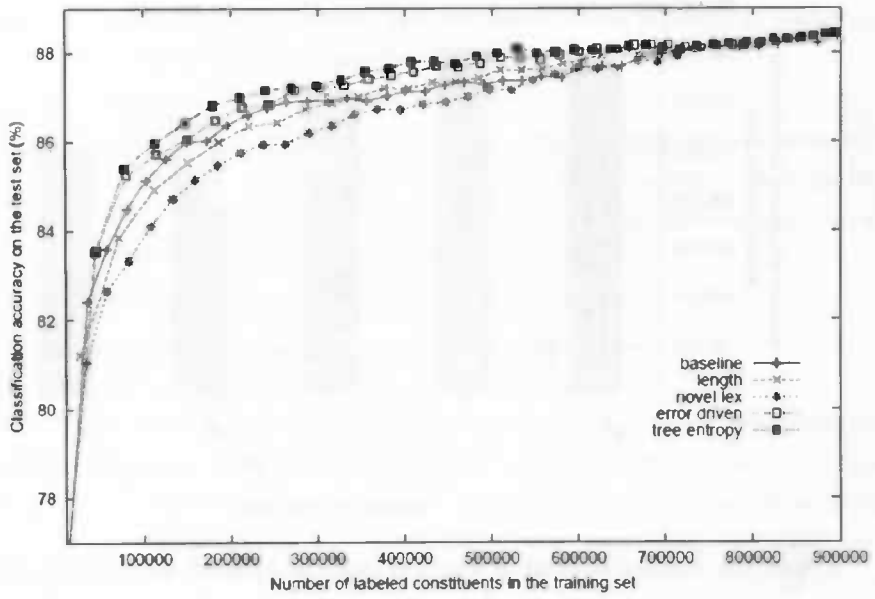

Figure 2.2: Results for active learning by Rebecca Hwa

from the pool of unlabeled sentences. The second algorithm sorts the sentences by length with the longest sentences first. At each cycle, the 100 longest sentences are selected and learned by the model. Longer sentences tend to possess more useful information, and therefore this strategy might leed to better results in performance, as we suggested earlier in the text. The remaining two strategies apply Active Learning and are based on the error-driven function and probability distribution, which are described in sections 2.2.1 and 2.2.2 respectively. To prevent the algorithm based on probability distribution from selecting longer sentences first the function is normalised by the log of the number of parses of each sentence. The performance of the model is calculated with the F-score, which is a balanced measure to express the accuracy.

The results of the experiment are shown in figure 2.2. The results of the algorithm using probability distribution are stated under the name tree entropy. The graph is showing one extra result called novel lex which we will ignore. One of the first things that stands out when looking at the graph is the unit on the x-axis, which is the number of labeled constituents in the training set. The constituents of a sentence can be seen as the nodes of the derivation tree of that sentence, so the numer of constituents the annotator has to label seem to

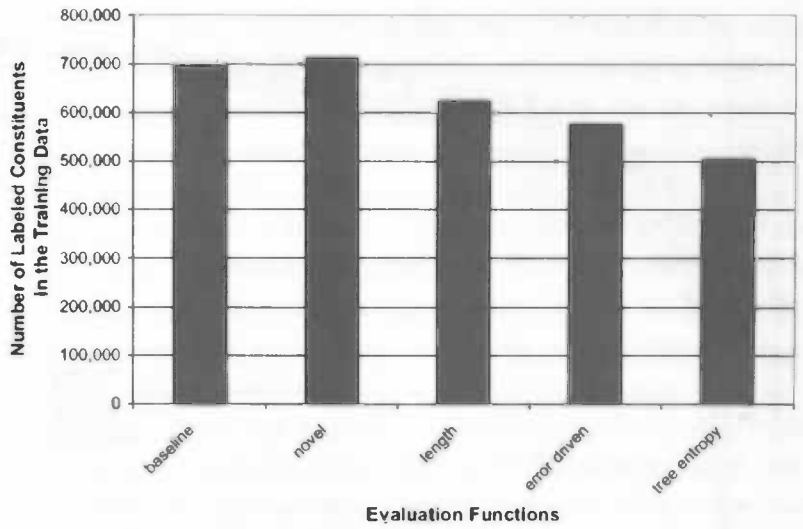

Figure 2.3: Needed amount of training material to reach an accuracy of 88%.

be a good measure for the annotation costs of the training data. Using labeled constituents as the base unit is definetely a better choice then to just count the number of sentences, as we saw in the previous subsection. This measure should give us a good indication of how much training material is used.

The graph is clearly showing that the Active Learning techniques perform better than the baseline strategy. When looking at the results of tree entropy, we can see that it produces a higher accuracy of the model compared to the baseline strategy when the same amount of training material is used. However, the gain in accuracy is relatively small, less than 1 percent most of the time. The real advantage of Active Learning can be seen when a certain accuracy must be obtained. The small gain Active Learning is providing in accuracy will cause the model to reach a given accuracy much quicker than using the baseline strategy. Because the model requires a lot of extra training material to improve on performance, the needed amount of training data can be dramatically reduced. In figure 2.3 the amount of used training material to reach an accuracy of 88% is shown for all strategies.

This figure is clearly showing that Active Learning is reducing the required training data by a large portion. Selecting the training material using probability distribution realizes a reduction of 27% in required labeled constituents. The graph also reveals that training with the longest sentences first results in a

reduction of needed training data, and the error-driven strategy is performing somewhat worse than probability distribution.

The most important conclusion that can be derived from this experiment is that Active Learning can produce a small gain in performance when looking at the accuracy level, but that this can result in a huge difference in needed training material when a certain level must be reached.

2.4 Reusing training material

We have seen that the amount of needed training material can be greatly reduced when using active learning. Only the most informative samples are selected and the less informative ones are ignored. This is done by testing all available training data on the current model and check which sentences are most useful. In this way a virtually ideal corpus is being created with only the absolutely required sentences. However, the available training data is only tested on the currently used model. When another model is used it will probably select a whole different set of sentences to train with. This implies that the selected sentences are particularly suited for the used model and the created corpus is biased. Therefore, this method is typically useful when only one model is used. But this is not very likely, as a model might evolve over time. Labeled sentences are needed to train the model, so the model can only be tested if we already created the training material. If the model then turns out to perform poorly, one could decide to use another model, or the feature set of the model could be changed to improve performance. Further research and new insights can cause the model to change dramatically. If this is the case, the training data is selected by the first model, but used for the improved model or even a completely different one.

Considering this, it seems sensible to investigate the effectiveness of training data which is selected by one model and then reused by another. If it turns out that the selected material is only suited for the model which created the corpus and cannot be effectively reused, it might be more useful to select the training material in another way. Baldridge and Osborne [3] have investigated the effect of reusing training material. They evaluate multiple scenarios to test the effect of reuse of a corpus created with active learning. For their experiments they used a hand-built broad-coverage HPSG grammar which is similar to the

grammar we use in our experiments, so the results of this research might be applicable to our situation. In their paper they use a variety of models, each with their own characteristics. One model is acting as the selector, which selects the training material, and another model is the reuser. The selector creates a corpus, wich the reuser will use to train with. Each model will act at least once as a selector for all other models. In the experiments there are two special cases, which will be considered as the baselines. The first baseline situation is when the selector and reuser are the same model, i.e. the model is selecting training material for itself, which is the case in the normal situation for active learning. If we compare the results of the other reuse scenarios with this one, we can determine if reuse can be just as effective as standard active learning. The other baseline is random sample selection. If the other scenarios turn out to perform worse than this baseline, it seems that reuse with active learning might be a bad idea and can be harmful in practice.

To determine whether a scenario is effective or not, we need a method to compare the different scenarios. Since we are trying to reduce the annotation costs, we look at how many annotation is needed to achieve a certain level of performance. If the same performance level is achieved with less annotation costs, the scenario is obviously better than the opposite situation. How to measure these annotation costs is a very delicate matter, and this will be discussed in the next chapter. Baldridge and Osborne have their own way of measuring annotation costs, but for now we just accept that these costs are given.

The models used in the experiments differ on two factors. The first factor is the learning algorithm which is used. Baldridge and Osborne use two algorithms: log-linear and perceptron based. The second factor is the feature set which is used. They distinguish four feature sets, each with their own characteristics. The names of the different models are composed by first stating the learning technique, LL (for log-linear) or P (for perceptron), and next the name of the feature set. For example LL-CONFIG is using the log-linear algorithm and the a feature set named CONFIG.

In figure 2.4 we see the results of one of the experiments. The model LL-CONFIG here is used as the reuser, and the training material is selected by two other models, LL-CONGLOM and P-MRS. Also the two baselines, selection by the model itself and random sample selection, are represented in the graph. In this figure we can see how many annotation cost it takes to achieve a certain level of accuracy. The first thing we see is that data selection by the model itself is clearly the most effective of all strategies. But the graph also shows

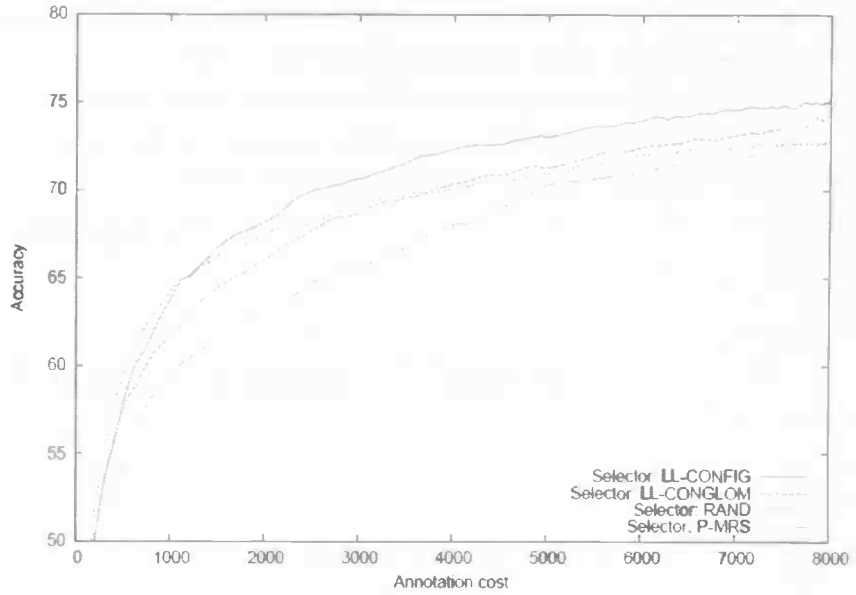

Figure 2.4: Results of Sample Selection with reuse

that random sample selection is actually more effective than LL-CONGLOM and P-MRS until 70% accuracy is reached.

The outcomes of the other experiments conducted in the paper show that all reuse strategies require significantly more annotation cost than self-reuse. Furthermore, they demonstrate that the random strategy is often a good selector compared to the other selectors. In some cases it even outperforms the selectors based on active learning, as can be seen in figure 2.4. This indicates that training data selected with active learning is indeed biased and is not necessarilly useful for another model. The effectiveness of a reused corpus seems to depend on the relatedness of the used models. If the selecting model is similar to the reusing model the training data will probably be well fit. But when the two models differ significantly, e.g. a whole different learning technique is used, the selected data will almost certain be unsuitable and the model might be better off with randomly selected training data.

The paper of Baldridge and Osborne shows that active learning can be brittle. If the selected data is reused by another model, its effectiveness can be dramatically reduced and can even be outperformed by random data selection. In practice it is very likely that a model will change over time or that the cor-

pus will be used for multiple purposes. In that case it is preferable to create an allround corpus which is not tuned for a particular model. For that reason it might be sensible to compose a corpus with randomly drawn sentences instead of using active learning to select the training material. In this paper, we will therefore focus on annotating a randomly generated corpus.

Chapter 3

Annotation Costs

3.1 Measuring annotation costs

In this paper our goal is to reduce the annotation costs of a given corpus. But before we can reduce anything we need to know what these costs really are and how we can measure them. In the literature, few things are written about annotation cost. Most papers on Active Learning are about reducing the needed amount of training material instead of reducing the total cost of obtaining the needed training material. Most of the time these two measures are correlated but not necessarily the same. In some research fields obtaining a training sample does require the same amount of effort every time, regardless of the nature of the sample. But in NLP the cost of obtaining training samples, i.e. sentences, can vary strongly, depending on the length and complexity of the sentence. It is even possible that annotating three easy sentences require less time and effort than annotating one complicated sentence. Therefore, it is often more useful to look at the total costs of obtaining the needed training material than at the amount of material by itself.

But how do we quantify the cost of annotating a sentence? Annotation is the process of determining the correct parse of a sentence. Usually this is done by humans to make sure the provided parse is trully the correct one. So in fact, the cost of annoting a sentence is the time the annotator needs to determine the correct parse. So by timing the annotation process we might obtain the exact annotation costs of a sentence. However, this is not very practical when running experiments and simulations. When using semi-automated labeling for

example, knowing how much time the annotator required at one time is not enough, because the next time the parser might provide a different suggestion and the time the annotator needs will differ. Therefore, the annotation process must be timed every single time. It is simply not feasible to annotate hundreds or thousands of sentences during each experiment. Besides, for the timing data to be consistent, all annotation must be accomplished by the same annotator under the same conditions. Otherwise, different annotation cost can be ascribed to different circumstances. And even if the annotator and conditions are identical, the timing data can vary due to distraction or fatigue.

It is clear that it is very difficult to obtain accurate timing information and it is often impractible to annotate every time the annotation costs need to be determined. Therefore, it might be sensible to find a way to estimate the annotation costs instead of measuring it. A good estimator can give us all the information we want without the need for annotation during every experiment. Obviously, it is not our goal to estimate the absolute annotation costs, but we are only interested in estimating the relative costs to see if the they have been reduced or increased. If we find a reliable estimator, we have a good unit we can use to represent the results of the experiments.

The costs of annotation strongly depend on the way the annotation process is conducted. When using semi-automated labeling for example, the annotation costs will be different from the time when the annotation is done from scratch. This implies that different annotation methods require different estimators. In the next subsections we will look at some of these methods and explore which estimators might be adequate for determining the annotation costs and are suitable to be used in our experiments.

3.2 Annotating from scratch

In the literature, most papers about active learning presume the parse has to be created from scratch. In practice this is often not the case, because semi-automated labeling is much more efficient. But in some situations an automated parser is not available or for another reason the parse has to be constructed totally by hand. If this is the case, there are several ways to make an estimation of the annotation costs of a sentence. We will now look at some of them which have been used in the literature and some ideas of our own.

- **Number of sentences**
  This is the most basic idea for measuring annotation costs, just look at the amount of training samples that are used. In some situations this might be a good way to quantify the cost of obtaining the training material, but as stated before, in the case of NLP this is not a good way to estimate the annotation costs. The main reason for this is that sentences can be of variable length and as a result the annotation cost can also vary. Despite this, in some scientific papers, e.g. [4] and [5], this measure is used to express the used training material nonetheless. As we saw in subsection 2.2.1 the results of experiments presented in this way are rather meaningless and no real conclusions can be derived from them. It is obvious that the number of sentences is a bad estimator for the annotation costs and is not suitable for our experiments.

- **Number of words**
  So when the number of sentences is incompetent as an estimator for annotation costs because the sentence length can vary, it may be logical to count the number of words in the sentences and use that as an estimator. The number of words is obviously a better measure for the total amount of training material than the number of sentences, and therefore it probably is a pretty good estimator for the annotation costs as well. However, it does not take into account that longer sentences tend to have relatively larger parse trees. Long sentences often posess more complicated grammatical structures, and therefore the parse trees will be larger and more complex. For this reason the annotation costs for longer sentences will be higher than for shorter ones most of the time. By only examining the number of words, this effect will be overlooked and the estimated annotation costs might be inaccurate.

- **Number of elements in parse tree**
  When annotating a sentence it is common to represent the grammatical structure in a parse tree. So, when estimating the annotation costs of a sentence, it might be sensible to look at the size of the parse tree, i.e. the number of elements in the tree. In her paper Rebecca Hwa [2] uses the total amount of labeled brackets (elements of the parse trees) to express the annotation costs in her experiments. Because it is closely related to the way the annotation is conducted it is probably a pretty good estimator for the effort it takes the annotator to create the parse.

- **Number of dependencies**
  The costs of annotation are obviously closely related to the way the annotation is conducted. In our testing environment the parses are represented by dependency structures instead of the parse trees we have seen so far. Dependency structures are slightly different than parse trees and they show the relation between words in the sentence. The dependency structure can also be expressed by a set of dependency relations, where each relation is representing an edge of the dependency structure. In the next chapter the testing environment will be described and dependency structures and relations will be treated in more detail over there. The point here is that in our tests we cannot count the number of elements in the parse tree and the number of dependencies can be a good substitute. Because the parses are represented by dependency relations, the number of depencies will probably be a good estimator for the annotation costs in this situation.

The estimators suggested in this section are all argued with common sense, not proved with hard data. Furthermore, the real annotation costs depend on some other factors which are hard to quantify, like the complexity of a sentence or the experience of the annotator with certain kinds of grammatical structures. Nonetheless the last two estimators are probably good enough to reliably measure the annotation costs during tests and simulations. One could imagine to invent a more detailed method to estimate the annotation costs, but the differences will probably be marginal or even negligible. Unfortunately, there are few researches about this subject and that is obviously beyond the scope of this paper, so we have to be satisfied with the estimators described in this section.

3.3 Semi-automated labeling

In the previous section we studied some estimators for the annotation costs when annotating from scratch. In practice however, the annotation is often accomplished using semi-automated labeling. If this is the case, an automated parser is used to assist with the annotation process and the syntactic structure can be created more efficiently. Obviously, using this method the annotation costs will be different compared to when the annotation is created from scratch. Furthermore, the annotation costs now also depend on the quality of the automated parser. The better the parser can help creating the correct parse, the lower the

annotation costs will be. In the next two subsections we will look at two cases where semi-automated labeling is used, and what the consequences are for the annotation costs and how they can be estimated. The first case is taken from literature and the second reflects the situation in our testing environment.

3.3.1 Selection with discriminants

Baldridge and Osborne [9] present in their paper a whole different method to annotate sentences. Instead of creating the correct parse, it is selected from the collection of all possible parse trees. First an automated parser creates all possible parse trees, the so called parse forest, and a human annotator will then determine which is the correct one. The search for the correct parse is done through discriminants, properties of the parse tree which split the parse forest in two parts (preferably of the same size). The annotator indicates if a discriminant will be part of the correct parse tree and consequently a large portion of the parse forest can be ruled out for being the correct parse. By evaluating discriminants the parse forest will be scaled down really fast (even exponentially most of the time) and the correct parse can be singled out. This method is probably very cost efficient compared to creating the parse from scratch.

When this method is used the annotation costs have nothing to do with the size of the parse tree or the number of words but they depend on the ambiguity of the sentence. If only one parse can be generated for a sentence the annotation will take no effort at all, but when the sentence is very ambiguous a lot of discriminants have to be evaluated before the correct parse is retrieved. Generally, the annotation costs will correspond to the number of discriminants which have to be evaluated before the correct parse is found. Baldridge and Osborne conducted some timing experiments to see if the number of discriminants would be a solid estimator for the annotation costs and the results suggested that this was truly the case.

One major drawback of this method is that it requires the correct parse to be in the parse forest. If it turns out that the correct parse cannot be selected the parse has to be created by hand anyway, so the annotation costs will be even higher. In some situations the correct parse will not be in the forest because the grammar which is used to create all parses is unable to construct the correct one. Another reason might be that there are so many possible parses, the number of parses in the forest will be capped at a certain amount. In the next chapter we

will see that in our testing environment this is often the case, and therefore this might not be a good annotation method in our situation.

3.3.2 Labeling with suggestion

So when we cannot choose from a parse forrest because the correct parse may not be in the forest, we have to find another way how an automated parser can help us annotate a sentence. A logical method is to let the parser select the parse he thinks is the correct one, and then let an annotator alter the syntactic structure (if necessary) until it reflects the correct parse. This is exactly the way syntactic structures are created in our testing environment. With this method the work that has to be done by the annotator strongly depends on the suggestion of the parser. Therefore, the annotation costs depend on the quality of the parser. The better the suggestion the parser provides, the less work has to be done by the annotator.

Note that if another annotation method was used like annotating with discriminants, the order of annotation would be of no influence on the total annotation costs. As a consequence, active learning would be of no help when annotating a given corpus and this paper would be obsolete. But with this method the parser can improve during the annotation process and the order of annotation will affect the total annotation costs.

With this method the annotation costs will depend on how well the suggestion from the parser corresponds to the correct parse. If the parser provides the correct parse immediately there will be no annotation costs at all. But most of the time the correct parse will differ from the suggestion and the annotator must edit the parse until it corresponds to the correct one. So what would be a good estimator for the annotation costs?

In the testing environment the parses are represented by a set of dependency relations. So when the parser gives a suggestion, the annotator can use all correct dependency relations from the suggestion and add the missing dependencies so that the correct parse is created. Therefore, the annotation costs depend on the correct dependencies in the suggestion, or actually on the missing dependencies. By counting how many dependencies have to be added by the annotator, we probably have a good estimator for the annotation costs. This amount can be easily computed by counting the corresponding dependencies from the suggestion and the correct parse, and then subtract this from the total amount of dependencies of the correct parse.

Intuitively the number of incorrect and missing dependency relations in the suggestion should give a good indication for the amount of work that has to be done by the annotator. Unfortunately, there is no timing data available for the annotation of the corpus in our testing environment, so there is no real validation for estimating the annotation costs in this way. However, there is no good alternative and this estimator closely relates to the actual procedure of annotation, so we must be satisfied with this way of estimating the annotation costs.

Chapter 4

Testing Environment

4.1 Alpino

To conduct our experiments we first need an automated parser to provide suggestions during the process of semi-automated labeling. For this task we use Alpino, a computational analyzer for Dutch which is designed for accurate parsing of unrestricted text. Alpino is developed at the university of Groningen (RUG) from 2000 until 2005. The main idea is that, with the use of a grammar, all possible parses will be constructed, and the most probable parse will then be selected. In this section we will take a closer look at some parts of the system and how they operate. A more detailed description of Alpino can be found in Van Noord [13].

4.1.1 Grammar

The Alpino grammar is a so called wide-coverage Head-Driven Phrase Structure (HPSG) grammar that takes a constructional approach. This means that it consists of a set of rules which define how a certain phrase can be build out of other phrases and/or words, just like the example grammar given in section 1.1.2. The grammar consists of about 600 detailed constructional rules that prescribe how Dutch sentences can be created.

To denominate all words of a sentence Alpino contains a large lexicon. The lexicon contains about 100,000 entries and is extended with about 200,000 named entities. The system also contains specific rules and methods to recognize dates, temporal expressions and other special named entities. When a

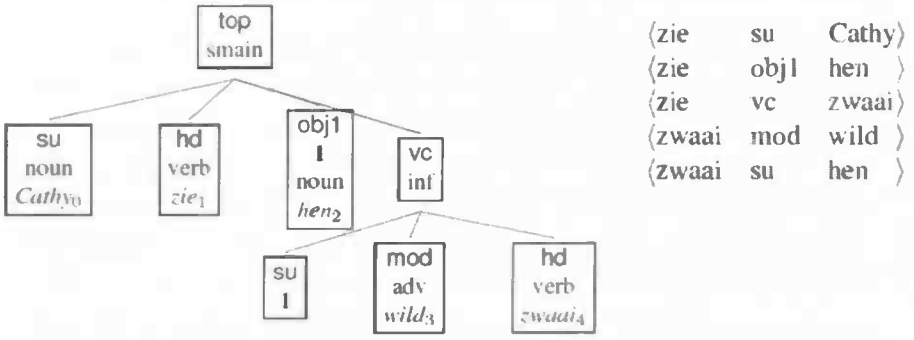

Figure 4.1: Dependency structure of the sentence "Cathy zag hen wild zwaaien" ("Cathy saw them wave wildly") and the corresponding set of dependency relations.

word is not recognized the system has several techniques to determine to which categorie a word belongs, like numbers or proper names.

Following the rules of the grammar together with the lexicon a parser can construct all possible parses of a given sentence and next a disambiguation component can retrieve the best one. This disambiguation component will be described in more detail in the next subsection. When no complete parse can be constructed for a sentence, the parser will construct a parse for each substring. The best set of non-overlapping parses will be selected as being the correct parse.

To represent the syntactic structure of a sentence, the grammar is designed to create dependency structures, which are slightly different from the parse trees as presented in the first chapter. An example of such a structure can be seen in figure 4.1. The "1" in the structure is an example of co-indexing, in this case used to indicate that "hen" is the object of "zie", but also the subject of "zwaai". The dependency structure can also be represented by a set of dependency relations, actually the edges of the dependency structure. An example of this can also be seen in figure 4.1. By representing a parse by the dependency relations the differences of parses can easily be calculated, as we will see later in this section.

4.1.2 Disambiguation Component

As stated before, a major problem in natural language processing is ambiguity. When parsing a sentence, a large number of possible parses are created using the grammar, and subsequently the correct one has to be selected. To determine which of the parses is the most probable, the Alpino system contains a disambiguation component. This system uses a large set of properties of parses to distinguish the good parses from the bad ones. These properties are called features and describe all kinds of detailed characteristics of parses. The Alpino system contains about 40,000 features. These features represent the application of a certain rule, combinations of certain words and their lexical category, dependencies between words and/or lexical categories and a lot more. The parses are stored as the set of features and the frequency in which each feature occurs in the parse.

All features will be assigned a weight, which can be both positive and negative. When the disambiguation component evaluates a parse it determines which features it contains and sum all features multiplied by their weights. The parse with the highest score calculated this way is considered to be the most likely. The weights of the features are estimated by training the model with annotated sentences. By analyzing the parses of a number of sample sentences the weigths are adjusted to reflect which features are representing good parses and which are representing bad parses. When a certain feature often occurs in a good parse, the weigth will be increased. But when the features is more prevalent in wrong parses, the weigth will be decreased. In this way, the good features will produce higher scores for parses and the bad ones lower scores. When the disambiguation component is untrained it will just select the first parse produced by the system.

When training the model, it might be a problem to evaluate all possible parses of a sample sentence. Long sentences tend to have a lot of possible parses, even exponentially by their length. Therefore, it is often not feasible to evaluate or even create all possible parses, simply because there are too many of them. To overcome this problem, a random selection of the possible parses is used to train the model. Since the weights of the features depend only on the expected value of the features in the possible parses, the results should be pretty similar if an unbiased subset of the parses is used. Some experiments show that this holds true if even only 250 randomly chosen parses per sentence are used instead of the complete set of parses. This solution offers a great increase in

efficiency when training the disambiguation component.

However, since not every parse can be evaluated by the disambiguation component, there is no guarantee that the correct dependency structure can be selected. When the correct parse is not in the set randomly selected parses, it is impossible for the disambiguation component to retrieve the correct dependency structure. Furthermore, when a sentence is very complicated, the grammar might be unable to construct the correct parse anyway. For these two reasons, ALPINO is unable to construct the correct dependency structure at all times, especially for complicated sentences with many parses.

4.1.3 Results

To measure the performance of a syntactic parser, the parses produced by the system are compared to the correct parse. In this way the accuracy of the parser can be determined. The more the produced parse correponds to the correct parse, the better the parser is performing. Since Alpino is producing dependency structures, we can easily evaluate how much dependency relations from the constructed parse match with the correct parse and how many are missing. A common measure for evaluating performance of parsers is the F-score. This is a weigthed measure of two accuracy metrics called precision and recall. The F-score is very useful when the size of the solution set is unknown in advance, but in parsing, the number of dependency relations of a sentence are practically given by the length of the sentence. Therefore, we can use another measure called concept accuracy (CA). Concept accuracy can be calculated for a sentence i according to the next equation:

$$CA^i = 1 - \frac{D_f^i}{max(D_g^i, D_p^i)} \tag{4.1}$$

D_p^i is the number of dependency relations produced by the parser for sentence i, D_g is the number of relations in the correct parse, and D_f is the number of incorrect and missing relations produced by the parser.

To calculate the performance of a parser on an entire corpus, we can compute the mean concept accuracy by dividing the total CA score by the number of sentences. In that case, shorter sentences are valued equally to longer sentences, while shorter sentences are usually much easier. To obtain a more informative

corpus	sents	lengths	F-sc	CA%
Alpino	7136	20	88.50	87.92
CLEF	1345	11	89.87	89.59
Trouw	1400	17	91.14	90.87

Table 4.1: Accuracy on development set and test sets for Alpino. The table lists the number of sentences, mean sentence length (in tokens), F-score and CA.

measure for the performance we can compute the total CA score:

$$CA = 1 - \frac{\sum_i D_f^i}{max(\sum_i D_g^i, \sum_i D_p^i)} \tag{4.2}$$

Using this measure the quality of the system can be reliably computed.

The accuracy of the fully trained Alpino system had been tested on three different corpora. The results of the tests are stated in table 4.1. The F-scores of the system are comparable to results of state of the art parsers for other languages, so Alpino seems to be a good lexical analyzer and certainly will be adequate for our experiments.

4.2 Corpus

Furthermore, we need a corpus for our experiments, and therefore we use the Alpino treebank [14]. The Alpino treebank consists of 7,100 annotated sentences (about 145,000 words) which are taken from the newspaper part of the Eindhoven corpus. For all sentences corresponding dependency structures as produced by Alpino are documented. Because long sentences can have a very large number of parses it is not feasible to store every dependency structure that Alpino constructed for every sentence. Therefore, the number of dependency structures is capped at 10,000. All sentences are also annotated by hand, so the correct dependency structure is available for every sentence.

For every dependency structure the CA score is also stored, so when training the model it can be estimated how useful a sentence is for the disambiguation component.

Chapter 5

Experiments

5.1 Introduction

The goal of our research is to reduce the annotation costs of a given corpus using Active Learning. Usually, Active Learning is used to reduce the needed amount of training material, but in our situation we are using Active Learning to determine how informative each sentence is for the current model. Because we want to annotate the entire corpus we do not let Active Learning decide which sentences are added to the training data, but instead let it prescribe the order in which the sentences are provided to the model. For our main experiment we are going to simulate the annotation of a corpus using an automated parsing model to help us with the annotation process. During the annotation process the model is trained with the sentences which are annotated so far. We assume the model is untrained at the start of the experiment. For our experiments we use Alpino and the Alpino treebank, described in chapter 4. To implement the experiments described in this chapter, we constructed a set of perl scripts, which make use of the scripts and tools of Alpino. The source code of these scripts can be found on the CD provided with this report.

As described in chapter 3 the annotation costs are estimated by the number of dependency relations from the correct parse which are missing in the suggestion provided by Alpino. This simulates the process of the annotator using the correct dependency relations from the suggestion and adding the rest of the relations by hand.

To conduct this experiment we have to implement Active Learining in the

first place. We discussed the theory in chapter 2, and we will use that to construct a good Active Learning algorithm. To test our algorithm we will do some experiments to see if it is working properly for the traditional goal of Active Learning, reducing the needed amount of training material by selecting the most informative sentences. If this is the case we can truly investigate whether Active Learning can reduce the annotation costs of a given corpus.

To identify whether and how much Active Learning will reduce the annotation costs we need we need some baseline data. Therefore, we conduct the same experiment with three baseline strategies and see how much annotation costs they generate. And to place the results of the experiments in perspective we are also trying to determine the boundaries of the annotation costs, i.e. the minimum and maximum costs. By estimating the minimum annotation costs we can establish the ultimate goal of reducing the costs and see how close this minimum can be approached. It is impossible to reduce them even further, so when these costs are close to the baseline results it might not be worth the trouble to use active learning.

5.2 Baseline strategies

To see how well Active Learning is able to reduce the annotation costs we need some baseline strategies. The first goal of a baseline strategy is to simulate the annotation process when the order in which the training material is presented is predetermined. While active learning will select the training material during the annotation process, our baselines will leave the order unchanged so the current state of the system will not be taken into account. This represents the situation in traditional passive learning, but is does not mean the order cannot be chosen in a smart way. So the general procedure for these experiments will be as follows. First all sentences in the corpus will be ordered by a certain criterion. Next, the first n sentences are selected and annotated by the Alpino system and the annotation costs for these sentences are estimated. Obviously, the annotation costs for each sentence are summed to represent the total costs of the entire corpus. Then Alpino will be trained with the sentences annotated so far and the next n sentences will be selected and annotated. This process is repeated until the entire corpus is annotated.

We now introduce two criteria to determine the order of the sentences.

5.2.1 Random order

The most basic baseline strategy is to annotate the sentences of the corpus in random order. In this way, no attempt is made to train the model in an intelligent manner, and this probably often reflects how it is done in practice. The strategy where the sentences are selected and evaluating by random we call RANDOMEVAL. Now we have to decide how much sentences are selected and annotated during each cycle of the annotation process. The lower the number of sentences during each cycle, the better the results, but it also increases the number of times it will be trained. Due to this, the total amount of needed time will increase severely and this might be impractible while annotating. We will let the number of sentences per cycle depend on the total number of sentences, and how many sentences are selected by the Active Learning experiment which we are comparing to.

5.2.2 Order by length

A more sophisticated but still very intuitive strategy is ordering the corpus by length. In this way the shorter sentences are presented first tot the system and then the longer ones, or in reversed order. Because longer sentences tend to be more complicated then shorter ones, and therefore are likely to be more informative for the system, we can train the model with the easy sentences first and then the more complicated ones, or in reversed order. By providing the system with the easy sentences first, the model will probably make very few mistakes in the beginning and is still able to learn and improve. When de easy sentences has to be annotated in the end the system is pretty well trained and it will probably make fewer mistakes on them compared to when they would be annotated in the beginning of the process. In this way the total annotation costs might reduce.

On the other hand, if the longer and more informative sentences are presented first, the system will probably learn very fast and that might result in lower annotation costs for the rest of the corpus. Therefore, this strategy might just be as effective as its counterpart.

Notice that the idea behind these strategies is the same as our Active Learning strategy, i.e. selecting the hardest (most informative) or easiest sentences. The main difference between these strategies and Active Learning is that they do not take into account how the current system is performing on certain training material. Furthermore, longer sentences tend to be more informative but

this is not necessarily the case. Therefore, Active Learning can search for the most informative (or easiest) more effectively. For these reasons we expect Active Learning to do a better job in determining the easiest or hardest sentences, and therefore result in a larger reduction of annotation costs.

The strategy where we order the sentences from short to long we will call LENGTEVAL and the inversed strategy where the sentences are sorted from long to short we will call INVLENGTHEVAL.

For practical reasons, during each cycle of the process all sentences with a certain length are annotated and evaluated. Because there is a different number of sentences of each length this means that there is a difference in the number of sentences evaluated in each cycle. This is a different situation from RANDOMEVAL, but we assume the differences to be marginal. Fewer sentences per cycle will result in lower annotation costs and more sentences in higher costs. In some cycles there will be more sentences, but the mean number is roughly the same as the number of sentences with RANDOMEVAL, so in the end it should even out.

5.3 Boundaries

5.3.1 Maximum costs

We are trying to set some boundaries of the annotation costs to place the results of the other experiments in perspective. By comparing the results with the maximum annotation costs we can see the relative reduction of the various strategies. To calculate the maximum annotation costs we will annotate the sentences with the help of Alpino, but this time the model is not trained during the annotation. This means that the entire corpus is evaluated with the uninitialized model, the so called zero-model, and therefore we call this strategy ZEROEVAL. Because the model will not be trained the order of the sentences is of no influence on the annotation costs.

The main thing we can learn from comparing the result of ZEROEVAL with other strategies is the effect of training the model during annotation. It seems obvious that training the model will reduce the annotation costs, but now we can see how much reduction it will offer and whether it is worth the effort.

5.3.2 Minimum costs

A more interesting exercise is to determine the minimum annotation costs. By examining the minimum costs we can see how much the system theoretically can improve and how close the various strategies approach this boundary. If the maximum gain obtained by the best possible strategy is not significant in comparison with a baseline method, one can argue whether it is useful to apply a complicated and time-consuming strategy like Active Learning. On the other hand, if a large reduction of annotation costs is possible, we might be more inclined to search for a good strategy. Furthermore, if the results of a strategy are close to the theoretical minimum, it has proved to be a very good strategy and it is unnessecary to look further.

Obviously, the minimum annotation costs are zero. This can be achieved if the system is always providing the correct parse as a suggestion and no corrections have to be made by the annotator. Unfortunately, it is impossible for the system to produce the correct parse at all times. For some complicated sentences the grammar is unable to construct the correct parse and for some sentences there are so many parses, it is practically impossible to construct all of them. In these cases the disambiguation component cannot select the correct parse and the best the system can do is selecting the parse with the least errors compared to the correct one. This situation corresponds to the case when the model is infinitely well trained, the system cannot perform better than this. So, the absolute upper limit of the system is when the disambiguation component chooses the best parse, but not necessarilly the correct one. We can simulate this situation by looking at the best parse Alpino has generated for each sentence and compute the annotation costs accordingly. We will call this strategy BESTEVAL, because it is the best the system can possibly operate.

With BESTEVAL, we assume that the model is infinitely well trained. This situation is preferable, but not very likely. A more realistic way of estimating the minumum annotation costs is to assume that the model is trained with the entire corpus, except for the sentence to be evaluated. The idea of training during the annotation process is that the system will improve by learning from the sentences annotated so far. So, the best level of performance will probably be reached when the model is trained with the entire corpus. Therefore, a good way to estimate the minimum annotation costs is to find out the annotation costs of a sentence when the model is trained with the entire corpus. If the model is also trained with the sentence it has to annotate, the system is biased because

it has seen the sentence before. Therefore, this sentence has to be left out of the training material. To do this for every sentence is very time consuming, so the final idea is to annotate n sentences, after training the system with the entire corpus minus these n sentences. This strategy, which we call GOODEVAL, will probably be a decent estimator for the minimum annotation costs and more realistic than BESTEVAL.

However, we assume that the model performs best on a sentence if it is trained with all other sentences in the corpus, but this might be untrue in some cases. If a sentence of certain kind has to be evaluated, and the model is only trained with sentences of the same type so far, it could perform better for this sentence than it would if it was also trained with sentences of other types. So theoretically, the total annotation costs could be lower when the model was only trained with a small part of the corpus, but this is not very likely.

5.4 Active learning

5.4.1 Implementing AL

Now we have to implement an Active Learning algorithm. In chapter 2 we discussed the theory behind Active Learning and presented two evaluation functions, an entropy (probability distribution) based, and an error-driven function. We tested both functions on a small part of the corpus to rate the performance. The result of this test can be seen in figure 5.1. The x-axis holds the number of dependency relations in the used training data, which we think is a good estimator for the amount of training material. On the y-axis the accuracy of the model is represented by a measure called error reduction. The error reduction is determined by calculating the actual performance as a percentage of the best possible performance of the model. This can be achieved by comparing the parse selected by the model with the best parse the model could have selected. So when the error reduction is 100%, the model always selects the correct parse, which represents the BESTEVAL strategy. By looking at the number of missing and incorrect dependencies in the selected parse compared to the best parse, the error reduction can easily be calculated. The maximum error reduction is obviously 100%, but to make the graph more clear the y-axis only goes to 70% here.

As we can see in the figure, the performance of both functions are almost exactly the same, so they both seem to be good evaluation functions. We

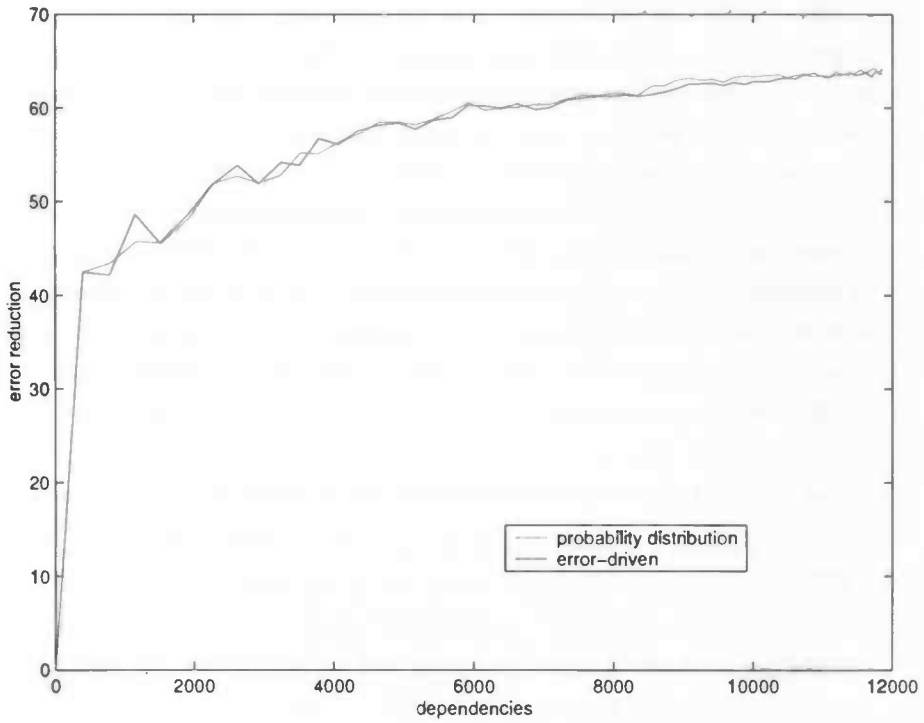

Figure 5.1: Results of active learning using entropy based and error-driven evaluation functions.

choose for the entropy based function, mainly because in the literature it is widely stated to be a reliable evaluation function. When we first tested the function it sticked out that it tends to select longer sentences. This is a known side-effect of the entropy based evaluation function, because longer sentences usually possess more parses, and therefore the entropy will be higher. If Active Learning is selecting the longer sentences first the strategy is too closely related to LENGTHSORT. In subsection 2.2.2 we introduced two methods to overcome this side-effect. The first is to normalize the entropy by the log of the number of parses (function 2.3). When we tried this the function still preferred longer sentences over shorter ones. The second solution is word entropy, i.e. normalizing the entropy by the number of words in the sentence (function 2.4). This technique proved to be much more effective and showed no clear preference for long or short sentences. So in our experiments, we use word entropy as our evaluation function for Active Learning.

To see if the evaluation function is working properly we are first testing

whether Active Learning is working in the classical way, i.e. reducing the needed training material when achieving a certain performance level. Therefore, we will use a small part of the corpus as a test set and the remaining corpus will be used for sample selection. our evaluation function will select n sentences from the remaining corpus, the system will be trained with these sentences. After each cycle the accuracy of the system will be tested using the test set. This is repeated until a certain accuracy level is obtained. We will then compare this to the situation where the training material is selected by random. If Active Learning will result in a reduction in required training material our evaluation function is operating in the way we want it and we can conduct some reliable experiments.

5.4.2 Reducing annotation costs with AL

Now that we have a good algorithm to implement Active Learning we can setup our main experiment, trying to reduce the annotation costs of a given corpus. As stated before, we like AL to dictate the annotation order of the corpus. At the beginning of the process the model will be untrained. During each cycle the entire corpus will be evaluated and the AL formula will select n sentences. These sentences will be annotated by Alpino and the annotation costs will be determined. Then Alpino will be trained with the sentences annotated so far. The next cycle all sentences except the ones that are already selected and annotated are evaluated by the AL algorithm. Again, n sentences are selected, annotated and added to the training material. When the model is trained again with the available data, the process is repeated until all sentences are annotated.

The main difference with the baseline strategies is that the order is determined during the annotation process, not beforehand. After each cycle the model will be different than the previous cycle because it is trained with more sentences. Since the active learning algorithm will be using the model to evaluate and select the next sentences to be annotated, it could rate a sentence differently through each cycle. So when the algorithm selects 10 sentences during a cycle, it is very well possible the 11th sentence is not selected in the next cycle. While the model rated that sentence very high in the first cycle, during the next cycle the system has learned and the same sentence might have become uninteresting. This is a major difference with for example LENGTHEVAL where the annotation order is independent from the used model.

We define two strategies where active learning is involved. The first strategy

is called ACTIVEEVAL. Active learning is designed to find the samples which are the most informative for the current state of the system. To achieve this, our algorithm finds the sentences which are the hardest for the model to annotate. At each cycle the hardest sentences are selected, annotated and added to the training material. By selecting the hardest, and therefore likely the most informative sentences the model will probably learn very fast. This will probably result in a higher accuracy level of the model in the beginning of the annotation process. Because the annotation costs heavily depend on the quality of the model the annotation costs of the remaining part of the corpus will probably decrease significantly. However, because the hardest sentences are selected the annotation costs will probably be very high in the beginning of the process. The model is hardly trained and is yet confronted with the most difficult sentences of the entire corpus. Therefore, it is expected to give very poor suggestions to the annotator in the beginning. So in order to decrease the total annotation costs the better trained model has to make up in the end for the extra annotation costs made in the beginning of the process. Whether the investment of making relatively more costs in the beginning will be paid off in the later part of the process, will be clarified by this experiment.

The second strategy is the inverse of the previous one and is called INVACTIVEEVAL. Instead of selecting the hardest sentences during each cycle, the easiest sentences are selected. The active learning algorithm rates all sentences by estimating how much trouble the model will have in finding the right parse. So by selecting the easiest sentences for the current model, it will probably make very few mistakes, and therefore generate low annotation costs. However, the easiest sentences are likely the least informative ones, so the model will be learning fairly slow. Therefore, it will probably make more mistakes on the easy sentences compared to when the system was well trained and the annotation costs of the easy sentences are increased. On the other hand, in the end of the annotation process when there are only hard sentences left to annotate, the system has been trained with a fairly large amount of data. It seems pretty logical to save the hardest sentences for last, when the system is better trained than in the beginning. A positive side effect of this strategy might be that the annotator has the opportunity to learn as well. In the beginning the annotator will be faced with easy sentences and can get used to the annotation process. The sentences will slightly become more difficult and the hardest sentences will come by when the annotator is most experienced. In the case of ACTIVEEVAL this will be quite reversed and the annotator will have to deal with the hardest

sentences right away. This effect is hard to measure and depends on the annotator, but it is something to keep in mind if the results of the strategies are pretty close.

Intuitively, these two strategies will exclude each other because they are so opposite. If ACTIVEEVAL is showing a significant decrease in annotation costs, it seems unlogical that INVACTIVEEVAL will produce the same results. However, there are no sound arguments why both strategies cannot perform better (or worse) than the baseline strategies. There is no obvious correlation between training order and annotation costs. These experiments will give an indication whether there is one and in which direction it can be found.

Chapter 6

Results

6.1 Baselines and Boundaries

The experiments to retrieve the baseline and boundary scores are conducted on the entire corpus of 7,100 sentences. With RANDOMEVAL and GOODEVAL we choose to annotate 100 sentences at each cycle, resulting in 71 cycles to annotate the entire corpus. With LENGTHEVAL and INVLENGTHEVAL the number of cycles depend on the various length of the sentences in the corpus. Because the lengths are varying between 1 and 74 words, the number of cycles is pretty much the same as with RANDOMEVAL. Only the number of sentences of each cycle is different, but as we explained in subsection 5.5.2, this should not lead to different results. Since there is no training involved with ZEROEVAL and BESTEVAL, no cycles are used with these strategies.

The results of the baseline and boundary strategies are shown in table 6.1. For all strategies the annotation costs are represented in the second column by

Strategy	Costs	Gain
ZEROEVAL	35545	97,27%
RANDOMEVAL	18018	0,00%
INVLENGTHEVAL	17916	-0,57%
LENGTHEVAL	17824	-1,08%
GOODEVAL	17021	-5,53%
BESTEVAL	9892	-45,10%

Table 6.1: Results of baselines and boundaries with 7,100 sentences

the total number of missing dependency relations. The last column shows the gain of annotation costs, i.e. the costs relative to the costs of RANDOMEVAL. By this measure we can see how much gain or reduction of annotation costs a certain strategy produces.

Looking at the results we can see that both LENGTHEVAL and INVLENGTHEVAL are performing a little bit better than RANDOMEVAL, but the cost reduction is very marginal. More alarming is the result of GOODEVAL. We consider this as a good estimator for the upper boundary, i.e. the minimum annotation costs, and this strategy is only producing a cost reduction of 5,5%. This implies that the maximum performance gain we can realize results in a cost reduction of a little over 5%. Obviously this is much less than we had hoped to achieve. The wishful upper boundary BESTEVAL is showing a reduction of 45%, which is more like we hoped. But again, this result is very unrealistic to achieve. The lower boundary estimator ZEROEVAL causes an increase in costs of almost 100%. This indicates that training the model during the annotation process can heavily reduce the annotation costs.

6.2 Active learning

When conducting the experiments concerning active learning, a problem arised. It turned out that the process of evaluating sentences on their informativeness is very time consuming. If we conducted the experiments on the entire corpus, they would literally take weeks to complete and in our situation this was simply not feasible. Therefore, we were forced to use a subset of the corpus for the active learning experiments.

The first experiment was aimed to see whether our active learning algorithm is working properly. Therefore, we created a testset consisting of 10% of the corpus (about 700 sentences). The rest of the corpus was left for sample selection. In each cycle 10 sentences were selected by the algorithm and added to the training material. Then the system was trained and the accuracy was tested on the testset. We ran this experiment as long as possible, and after 5 days 750 sentences were selected and learned. In the mean time the same experiment was running with random sample selection. The results of both experiments are presented in figure 6.1. The unit on the x-axis is the number of dependencies of the used training material and the y-axis represents the error reduction. The graph clearly shows that our word entropy based algorithm produces in a

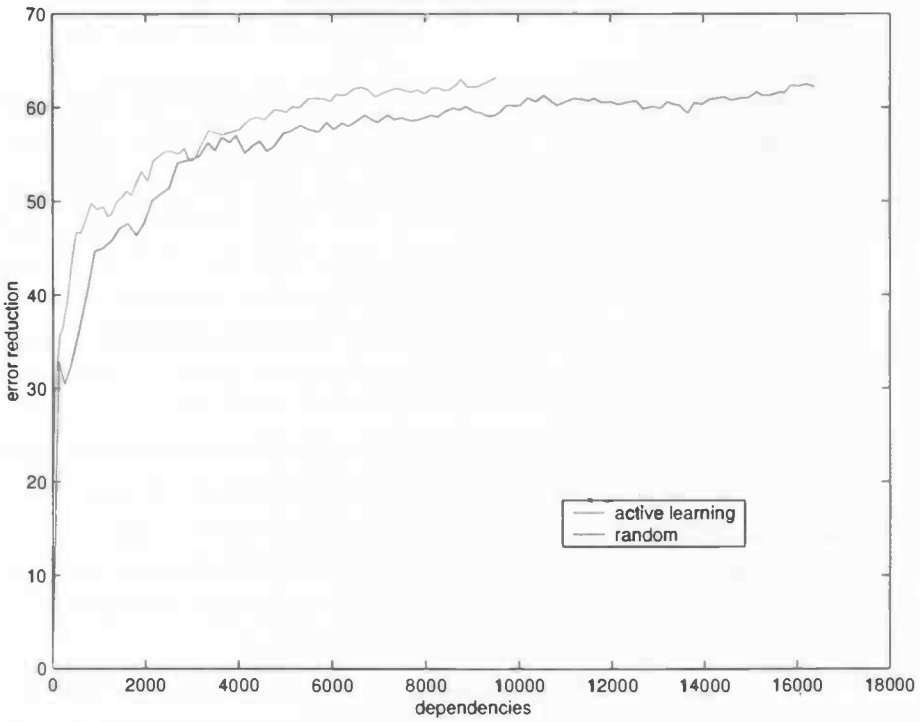

Figure 6.1: Results of Active Learning using word entropy.

Strategy	Costs	Gain
ZEROEVAL	3572	68,97%
INVACTIVEEVAL	2130	0,76%
RANDOMEVAL	2114	0,00%
INVLENGTHEVAL	2112	-0,09%
LENGTHEVAL	2109	-0,24%
ACTIVEEVAL	2095	-0,90%
GOODEVAL	1938	-8,33%
BESTEVAL	988	-53,26%

Table 6.2: Results of all strategies on 700 sentences

higher accuracy level with the same amount of training material. This results in achieving a certain accuracy level much faster, and that is what active learning is all about. When we look at the data, we see that random sample selection reaches a phi score of 62% after training with 15869 dependencies. The same accuracy is achieved with active learning after 8159 dependencies, meaning a reduction of almost 50%. These scores might be a little optimistic because only a small part of the training material is used, but it seems obvious that our algorithm is pretty good at choosing high informative training material, just as we wanted.

The experiments where we use active learning to reduce the annotation costs are conducted on a subset of 10% of the corpus, about 700 sentences. For comparison reasons, we also repeated the baseline and boundary experiments on this subset. Both RANDOMEVAL and the active learning strategies select 12 sentences per cycle to match the number of cycles used by LENGTHEVAL. The results of all experiments are stated in table 6.2.

The baseline and boundary experiments show rather similar results as with the entire corpus. The upper boundaries are a little better relative to RANDOMEVAL for the smaller dataset. This can be ascribed to the fact that the model is learning faster in the beginning of the training process than in the end, as can be clearly seen in figure 6.1. At first the learning curve is very steep but it becomes more gradual really fast. Both GOODEVAL and BESTEVAL represent a fully trained system and the difference with a partially trained system will be larger in the early stage of the training process. When less training data is used, this early stage forms a larger portion of the total training process and the relative performance gain will also be larger. Therefore, GOODEVAL and BESTEVAL will perform relatively better with smaller data sets.

The better performance of ZEROEVAL can also be explained by this theory. In the early stage of the process, the difference between ZEROEVAL and a learning strategy is much smaller than at a later stage. At the start the performance is even the same because in both strategies the system is still untrained. Because the early stage forms a larger part of the process, the mean difference in performance level is smaller than with larger data sets.

The baseline strategies LENGTHEVAL and INVLENGTHEVAL producing an even smaller cost reduction than with the entire corpus. This can be caused by the smaller dataset, but the differences with RANDOMEVAL are so small that this might well be just variance.

The results from the active learning strategies is a bit more explicit but also rather marginal. ACTIVEEVAL is producing a cost reduction of almost 1% and INVACTIVEEVAL is even performing worse dan the baselines. Again, these differences are rather marginal and do not suggest a possible significant cost reduction.

6.3 Conclusions

The results of the active learning experiments show no significant reduction in annotation costs compared to the baseline strategies. Also the smarter baseline strategies based on sentence length cannot impress. But more importantly, the highest possible cost reduction appears to be pretty small when we consider the results of GOODEVAL. We cannot expect the model to perform better during the training process than at the end when more training material has been used. GOODEVAL is simulating the situation where all sentences are evaluated in the end of the training process and is only able to reduce the annotation costs by 5,5% on the entire corpus. So, even if we were able to find the perfect strategy, it would probably still be resulting in a small reduction of annotation costs, while requiring a lot of computational effort implementing the strategy. We have seen how much computational time it took the active learning algorithm to select the sentences to annotate. If during the annotation process the annotator has to wait a considerable amount of time for the system to select the sentences to annotate, the possible saving of time using the strategy would probably be counterfeited.

In the experiments the active learning strategies were unable to come close to the estimated upper bound of 5,5%. Unfortunately it was impossible for us

to test the strategies on the complete data set. However, the results on the smaller dataset were no reason to believe the results would have been better on the whole corpus. Although INVLENGTHEVAL was performing better on the entire corpus, and the idea behind ACTIVEEVAL and INVLENGTHEVAL is te same, this is no guarantee for ACTIVEEVAL to perform better on the whole data set. The idea behind INVACTIVEEVAL and LENGTHEVAL is also the same, but the results are quite different. While LENGTHEVAL is causing a small reduction of annotation costs, INVACTIVEEVAL is even outperformed by RANDOMEVAL on the small dataset. This and the small margins between the strategies suggest that the differences might as well be ascribed to variance.

Probably the only way to come close to the results of GOODEVAL is for the model to become accurate very fast. Our idea was to accomplish this through active learning, but it turns out that this is not what active learning does. As can be seen in figure 6.1 the only thing active learning can realize is to let the model learn a little bit faster, especially in the beginning of the training process. This will result in a small increase of performance at virtually any point of the training process. Because the performance of the model is increasing rather slow, except in the beginning, the small increase of performance caused by active learning can result in reaching a certain accuracy level much sooner. So active learning can lead to a large reduction of required training material, but it cannot force a major increase in performance, which is needed to resemble the GOODEVAL strategy. Also the idea of INVACTIVEEVAL, first annotating the easy sentences, seem to have no significant effect on the annotation costs. Herefore we must conclude that active learning cannot help us to reduce the annotation costs of a given corpus.

We tried to use active learning to find the optimal annotation order when using semi-automated labeling. To determine the cost reduction of the optimal order, all different annotation orders have to be tested. This is of course impossible due to the vast amount of possible orders. However, the results of the optimal order are probably inferior to the results of GOODEVAL, because it is unlikely the model is performing better during the training process than at the end. Since the cost reduction of GOODEVAL is pretty small already, the gain of the optimal training order will almost certainly be very marginal. We tried annotating the easy sentences first and annotating the hard sentences first but there was no significant cost reduction observed. Probably high annotation costs in the beginning are compensated in the end and vice versa. So it seems that the order of annotation will not affect the total annotation costs in

a significant way. Therefore, the best order is probably the one which suits the annotator best and takes no large effort to calculate, for instance annotating the shorter sentences first to accomodate to the annotation process. But it is obvious that active learning will be no help when annotating a given corpus.

References

[1] Daniel Jurafsky, James H. Martin, "Speech and Language Processing", Prentice Hall, 2000

[2] Rebecca Hwa, "Sample Selection for Statistical Parsing", Computational Linguistics, 2004

[3] Jason Baldridge and Miles Osborne. "Active Learning and the Total Cost of Annotation". EMNLP, Barcelona, 2004.

[4] Min Tang, Xiaoqiang Luo, Salim Roukos, "Active Learning for Statistical Natural Language Parsing", Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, 2002

[5] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney, "Active Learning for Natural Language Parsing and Information Extraction", Proceedings of the Sixteenth International Machine Learning Conference, 1999

[6] David D. Lewis, Jason Catlett, "Heterogeneous Uncertainty Sampling for Supervised Learning", Proceedings of ICML-94, 11th International Conference on Machine Learning, 1994

[7] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan, "Active Learning with Statistical Models", Advances in Neural Information Processing Systems, volume7, 1995

[8] H. S. Seung, Manfred Opper, Haim Sompolinsky, "Query by Committee", Computational Learning Theory, 1992

[9] Miles Osborne , Jason Baldridge, "Ensemble-based Active Learning for Parse Selection", NAACL, Boston, 2004.

[10] Claude Shannon, "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. 27, 1948

[11] Michael Collins, "Three Generative, Lexicalised Models for Statistical Parsing", ACL, 1997

[12] The Penn Treebank Project website, "http://www.cis.upenn.edu/~treebank/"

[13] Gertjan van Noord, "At Last Parsing Is Now Operational", TALN06. Verbum Ex Machina. Actes de la 13e conference sur le traitement automatique des langues naturelles, 2006

[14] Leonoor van der Beek, Gosse Bouma, Robert Malouf, Gertjan van Noord, "The Alpino Dependency Treebank", Computational Linguistics in the Netherlands, 2002