Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS

Jan Václ

A Suspected Annotation Detection

Institute of Formal and Applied Linguistics

Supervisor of the thesis:	Mgr. Barbora Vidová Hladká, Ph.D.
Study programme:	Informatika
Specialization:	Obecná informatika

Prague 2011

I would like to thank Mgr. Barbora Vidová Hladká for her great help and assistance with this thesis, and my parents for their admirable support and patience.

Název práce: Detekce podezřelých anotací

Autor: Jan Václ

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt:

Tato práce popisuje způsob kontroly morfologické anotace pomocí strojového učení a představuje implementaci tohoto přístupu – aplikaci MissTagger. Procedura kontroly zahrnuje jak detekci chyb, tak jejich opravu. Tento přístup je založen na zjednodušeném algoritmu strojového učení, který si jednotlivé trénovací případy (instance) ukládá přímo do paměti bez zobecňování. Za tyto instance jsou považovány morfologické značky jednotlivých slov a jako rysy těchto instancí je brán jejich větný kontext pevné délky. Konkrétní slova, jejichž morfologické značky tvoří tento kontext, se vybírají buď přímo podle lineární struktury věty, nebo na základě závislostního stromu její syntaktické analýzy. Do experimentů k vyhodnocení tohoto přístupu jsou zapojeny dva jazyky – čeština a angličtina.

Klíčová slova: anotace, detekce chyb, strojové učení

Title: A suspected annotation detection

Author: Jan Václ

Supervisor: Mgr. Barbora Vidová Hladká, Ph.D., Institute of Formal and Applied Linguistics

Abstract:

This work describes a machine learning approach for checking the part-of-speech annotation, and presents its implementation – a system called MissTagger. The checking procedure covers both error detection and error correction. MissTagger employs a simplified instance-based learning algorithm where the words in the text are recognized as instances. Part-of-speech tags of context of static length are selected as features, no lexical information is included. The words whose tags comprises this context are chosen based either on a linear or on a dependency-tree structure of the sentence. Two languages are examined in the experiments for evaluation, Czech and English.

Keywords: annotation, error detection, machine learning

Contents

Introduction

0.1 Motivation

Annotated linguistic corpora serve as a valuable resource for various linguistic purposes. Their large sizes are especially helpful as a training material for a number of statistical methods often leading to practical applications. However, the performance of these automatic approaches is closely related to the correctness of the linguistic data and particularly its annotation, as summarized in (Dickinson, 2005, pp. 5–15). With the sizes of the corpora, manual error checking is very expensive in the terms of money, time and people (qualified linguists are often needed). Thus the research in this area is focused on the options of an *automatic* error correction, or at least error detection.

0.2 Corpus Annotation

Linguistic corpus is a term for a large structured set of texts of a given language (or multiple languages) with an additional information called *annotation*. This annotation is of linguistic nature and is often designed to serve both manual examination and automatic processing of the data. While its format may be quite diverse, the linguistic areas that the information targets allows us to distinguish two most common types of annotation:

Morphology Morphological annotation is word-oriented, i.e. to each word, there is a corresponding piece of annotation called morphological *tag*. The tag describes one or more features of the word, for example its part of speech, gender, number etc. These features are often dependent on the context, some of them can be determined directly from the form of the word, but often with some degree of ambiguity. The term *part-of-speech tag* is also used in this area for denoting a tag which indicates only the part of speech of this word. The process of assigning morphological tag to each word in a text is called (morphological) *tagging* and can be performed manually (by a linguist) or automatically (by an application called *tagger*). The automatic tagging usually has two phases; the first one is called *morphological analysis* and its output is a set of possible tags for a given word regardless of its context (dictionaries and morphological rules are usually used to achieve this task). The second phase, called *disambiguation* (or simply *tagging* again), then chooses the appropriate tag from this set. This problem can be used by linguistic rule-based approaches, machine learning methods or a combination of both.

Syntax Syntactic annotation tries to depict the structure of sentence. It often consists of word-oriented pieces of information describing the functions of the words in the sentence, and a structure capturing relations between the words. In modern linguistics, there are many theories about language *grammars* and particular syntactic annotation form varies according to the theory it is based on. One of the most common structure used in corpora is the *phrase structure*, which is characteristic by gradually "grouping" the words into phrases forming the sentence constituents. In this work, the *dependency syntax* is used, since it is the most common approach for capturing the structure of the Czech language sentences. As its name suggests, it represents rather dependency relations between the words and is somehow more robust to pitfalls of the free word-order languages. Creating the syntactic structure of a sentence is called *parsing*, and similarly to tagging, it can be achieved either manually or automatically. Similarly to the case of morphology, an automatic *parser* can be based on linguistic rules based on the corresponding grammar theory, on a machine learning method or on a combination of both these approaches.

0.3 Goals and Contents

This work describes an experimental implementation of a simple supervised machine learning idea oriented on detection and correction of errors in the morphological annotation of a linguistic corpus. The system called MissTagger is based on a simplified instance-based learning algorithm and works only with the morphological tags instead of words. It takes the context of the examined word as a feature vector, matches it with a learned instance, and in the case of error suspicion gives a suggestion of an alternative tag. However, the experimental part is particularly the feature of choosing the context with the help of a syntactic dependency tree instead of the simple linear sentence structure. The context size can be set parametrically, but remains fixed throughout both the training and the classification phase.

Description of the testing experiments are also presented in this work, as well as their results. The MissTagger system is tested on Czech data (from the Prague Dependency Treebank 2.0[1] – PDT) with both linear and dependency context,

[1]`http://ufal.mff.cuni.cz/pdt2.0`

and on English data (Wall Street Journal from the Penn Treebank[2] – WSJ). The evaluation is presented using the standard performance measures.

The MissTagger application itself works with morphological (and syntactic) annotation in the PML format[3]. However, several scripts enclosed as a part of this work (see 4.4) provide conversion to allow working with the Penn Treebank format. More format conversion scripts are also available as parts of the tools for PDT and Czech Academic Corpus mentioned further in this work.

The first part of Chapter 1 shows some research and publications related to the problem of annotation error detection and correction, the second one compares generally these approaches with our solution. Chapter 2 depicts a basic theoretical background of our approach and describes the focal part of the experiment – choosing the type of classification features. The transition from the theory to implementation is analysed in Chapter 3. Chapter 4 then shows the results of the experiments, as well as the description of the testing process and the evaluation measures used.

[2]`http://www.cis.upenn.edu/~treebank/`

[3]PML is the main format for PDT and Czech Academic Corpus 2.0.

1. Research

1.1 Related Work

Several automatic methods have already been examined in the area of corpora annotation error detection, namely concerning the part-of-speech annotation. The approaches used so far are based either on linguistic rules as in (Květoň and Oliva, 2002) or on statistical models, as described for example in (Dickinson and Meurers, 2003), (Dickinson, 2005), and in (Boyd et al., 2007).

The authors Květoň and Oliva (2002) use automatically induced rules for tagging error detection with the focus on negative (i.e. impossible) unigrams and bigrams of tags and manual adjustments between multiple passings through the corpus. The rules are applied gradually to unigrams, bigrams, and n-grams. However, these n-grams are in fact also bigrams, but somehow stretched — certain classes of words are allowed to come between this pair of words (for example punctuation).

Dickinson and Meurers (2003) suggest focusing on so-called *variations*, i.e. strings of the same words annotated differently in different contexts. This idea is structurally quite close to our approach, although they work with n-grams of a variable length. They employ two heuristics to distinguish whether a variation is an error or just a genuine ambiguity. The first one distrusts the long variation n-grams, i.e. the longer the identical context is, the more likely the variation is an error. The second heuristict considers the position of the examined word within the variation — those located near the fringe of the variation n-gram are more likely to be ambiguities than errors.

The idea of focusing not on the actual word strings, but working with their part-of-speech classes, is proposed in (Dickinson, 2005). This approach implies a significant increase of the error detection recall, since the more coarse division of the compared units brings more statistical information about them. However, more sophisticated heuristics are needed to maintain some reasonable level of precision. An example of application of these ideas is shown in (Boyd et al., 2007). The heuristics employed there make use of the phrase structure trees to identify more reliable context for deciding between the error and the ambiguity.

It is also notable that almost none of the systems concerned with the annotation checking detects and corrects errors produced by a tagger, but only the manual annotation. While an exception can be found for example in (Elworthy, 1994), the application mentioned there is quite specific — it is specialized on one type of tagger only and can be seen as just an extension of the tagger.

1.2 Our Approach

Our approach has two key ideas. The first one is the same as one of the key ideas in (Dickinson, 2005), that is, somehow generalizing from the actual words to their morphological tags. Actually, we go a little bit further than any of the methods mentioned earlier — we do not concern with the word forms at all. Our method checks each word (or to be more precise, its morphological tag) in a way that it fetches its context's tags and compares them with the learned instances, as well as the word's tag itself.

The second key idea of our approach is an alternative view of the context. In the second part of our experiments, the fixed-size context is not based on the linear structure of the sentence, but rather on the syntactic dependencies as recognized by a parser. It should be kept in mind, however, that since we are operating on possibly erroneous data, the errors in the morphological annotation can be negatively reflected in the local efficiency of the parser, thus making this dependency information less reliable.

2. Theory

2.1 Supervised machine learning

Supervised learning is a type of machine learning where the target function is inferred with the help of *labeled training data*. This data consists of a set of training examples paired with the according target function outputs. The learning algorithm can then make use of the training dataset to construct an appropriate hypothesis as an approximation of the target function (Kotsiantis, 2007).

The crucial part of the supervised learning approach is handling the previously unseen instances, i.e. how to classify the instances that were not members of the training dataset. The most common method is to generalize in some way. This generalization often takes form of constructing a data structure, based on the training data, that infers the information from this dataset to classify the new instances in a similar manner. In the real-world applications, this is of course done at the cost of losing the classification precision on data similar to (or same as) the training set — the more generalization applied, the less precise this result could be. Thus some balance between the ability to classify unknown instances (generalization) and maintaining the classification precision has to be established. The process of changing parameters of the machine learning algorithm to find this balance is called *tuning* the algorithm (or its parameters).

One of the weak spots of relying on the training data blindly is their questionable correctness. In the real applications, no training data (of some reasonable amount) can be said to be absolutely correct, there is always some *noise*, a certain amount of wrongly labeled examples. This is yet another reason to generalize in some way over the training data.

The actual supervised learning algorithms can be divided into several categories according to the employed data structure used for the generalization:

- *Logic based*, for example decision tree learning,
- *Perceptron-based*, also known as *neural networks*, can be a single- or multi-layered, or *RBF* (Radial Basis Function network – a special case of a three-layered perceptron),
- *Statistical – Naive Bayes* or *Bayesian networks*
- *Instance-based*, also called lazy, for example *k-Nearest Neighbor*

- *Support vector machines*

A nice table comparing these methods from a number of various aspects can be found in (Kotsiantis, 2007).

2.2 Instance-based learning

One of the most straightforward concepts among the machine learning approaches is the instance-based learning. It is based on the idea of storing all the training examples in memory and postponing any generalization to the moment of classification. Then, the new instances are compared to the stored examples and the classification is computed directly from the training data labels. In case of an unknown instance, however, some generalization has to be done. The solution is to find the most *similar* instance among the trained examples and let its output label determine the classification of the new one.

The key part of this approach is obviously in defining what *similar* means. A common solution is to establish some metric in the space of the instances and according to this metric, the closest instance is considered to be the most similar.

The choice of the distance metric is often the crucial part of the algorithm design. The most common is the Euclidean metric – the distance between two instances x_i and x_j is defined:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - (a_r(x_j))^2}$$

where $a_r(x)$ denotes the value of the r-th attribute of instance x.

The *k-Nearest Neighbor* algorithm (also known just as *k-NN*) is a simple generic method of the instance-based learning application (Mitchell, 1997). The letter k stands for the number of nearest (most similar) training examples which are taken into account when deciding how the system should classify a new instance. Then a majority vote (or the mean value, in case of a real-valued target function) of these neighbors is usually used for determining the final classification of the instance. Formally, the discrete-valued classifier function (i.e. the approximation of the target function) $\hat{f}$ can be defined as follows:

$$\hat{f}(x) \equiv argmax_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where V is finite set of possible output classifications $\{v_1, ..., v_s\}$ and $\delta(a, b) = 1$ if $a = b$ and 0 otherwise.

Another refinement of this method employs weighting of the votes according to the neighbors' distance to the new instance, so that the nearest neighbor have more influence to the final classification than the most distant one. This weighting will be reflected in $\hat{f}$ by multiplying each neighbor by its weight:

$$\hat{f}(x) \equiv argmax_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where the weights w_i are computed as follows:

$$w_i \equiv \frac{1}{d(x, x_i)^2}$$

2.3 Classification Features

The learning algorithm used in this work belongs to the class of supervised learning algorithms, namely instance-based learning algorithms (Aha et al., 1991). The instances are assumed to be represented each by the same set of m features. We recognize a *feature* to be a part-of-speech tag in a specific position either in a linear list or in a tree structure, and a feature vector of length m is considered as a *context*.

2.3.1 Linear Features

We assume a text $Text = w_1, w_2, ..., w_N$ to be a sequence of words in the sentences. If the words are manually or automatically tagged, we can write $Text_t = w_1/t_1, w_2/t_2, ..., w_N/t_N$ where t_i is the part-of-speech tag corresponding to the word w_i. Since we disregard any lexical information, we are interested in a sequence of tags $T = t_1, t_2, ..., t_N$ only. A linear list T represents the data structure from which we select the *linear contexts*. For the instance w_i, we assume X preceding and Y following tags as a feature list, $X + Y = m$. The resulting feature vector $(t_{i-X}, ..., t_{i-1}, t_{i+1}, ..., t_{i+Y})$ of length m is considered as the linear context $m_{X.Y}$. See Table 2.1 for the lengths of linear contexts we employ in this work.

For illustration, we list Czech sentence *"Měsíční produkce uzenin a bouraného* ***masa*** *se pohybuje okolo 120 tun."* (En: *"The monthly production of the smoked and cut meat fluctuates around 120 tons."*) in Table 2.2 with morphological tags of the words.

context	description
$m_{1.1}$	one tag before and one after
$m_{2.0}$	two tags before and none after
$m_{2.1}$	two tags before and one after
$m_{2.2}$	two tags before and two after

Table 2.1: Linear contexts

word	in English	morphological tag
Měsíční	(The) monthly	`AAFS1----1A----`
produkce	production	`NNFS1-----A----`
uzenin	(of the) smoked (meat)	`NNFP2-----A----`
a	and	`J^-------------`
bouraného	cut	`AANS2----1A----`
masa	meat	`NNNS2-----A----`
se	(itself)	`P7-X4----------`
pohybuje	revolves	`VB-S---3P-AA---`
okolo	around	`RR--2----------`
120	120	`C=-------------`
tun	tons	`NNFP2-----A----`
.	.	`Z:-------------`

Table 2.2: Linear context sentence example

Working with the linear context $m_{2.1}$, the resulting feature vector for the instance *masa* is

(`J^-------------`, `AANS2----1A----`, `P7-X4----------`)

i.e. tags of the underlined words.[1]

In a sample English sentence *"The company had no comment on whether a* ***replacement*** *would be named."*, using the linear context $m_{2.1}$, the feature vector of the word *replacement* is (`IN`, `DT`, `MD`).

2.3.2 Dependency-tree Features

We assume a text to be a sequence of sentences $S = w_1, w_2, ..., w_n$. If the sentences are manually or automatically parsed and the words are manually or automatically tagged, we can write $Tree_S(V, E)$. Applying dependency-based approach to parsing, a sentence is represented as a rooted tree with labeled nodes V and edges E that capture dependency relation between two nodes, i.e. between the depen-

[1]For details on the tag set used see Section 4.1.3.

dent (*child*) and its governor (*parent*). The nodes are labeled by the words and their part-of-speech tags. A dependency tree $Tree$ represents a data structure from which we select *dependency-tree contexts*. Let node N stands for the word w_i in Figure 2.1; we assume the (grand)parent nodes $P_1, P_2, ..., P_X$ on the path from N to the root node R and children C_1, C_2, ..., C_Y of N as the feature list, $X+Y=m$. Then the resulting feature vector $(P_1, ..., P_X, C_1, ..., C_Y)$ of length m is considered as the dependency-tree context $s_{X.Y}$ of the word w_i. See Table 2.3 for the dependency-tree context types we employ in the experiments.

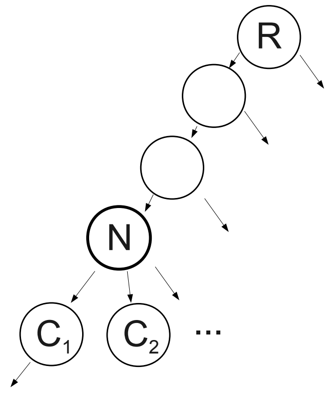

Figure 2.1: Children and parents in the dependency-tree contexts

context	description
$s_{1.1}$	parent's tag and the first child's tag
$s_{0.2}$	tags of the first two children
$s_{1.2}$	parent's tag and tags of the first two children
$s_{2.2}$	parent's tag, grandparent's tag, and the first two children's tag

Table 2.3: Dependency-tree contexts

Returning to the sentence listed above, its dependency tree is shown in Figure 2.2. Working with the dependency-tree context $s_{1.2}$, the resulting feature vector for the instance *masa* is

(`J^------------=`, `AANS2----1A----=`, `0--------------=`[2])

i.e. tags in the circled nodes.

Using dependency-tree contexts is motivated by an idea that these contexts are more relevant and thus more informative and reliable than the linear contexts, which are of rather local character from the word order perspective. This concerns

[2]A special tag for "null child".

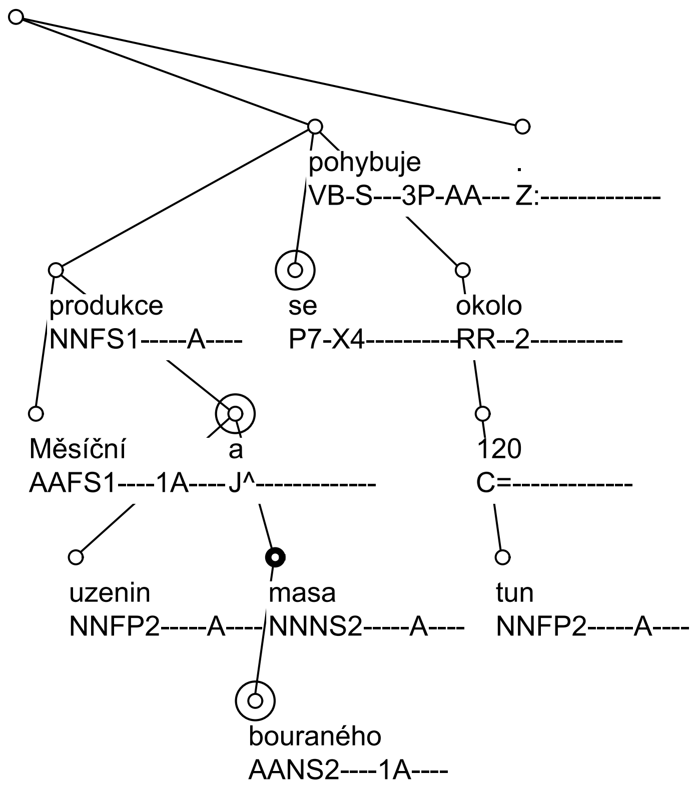

Figure 2.2: An example of a dependency tree

mainly the free word order languages, for which the dependencies captured in trees bring particularly valuable information.

2.3.3 Positional and Non-positional Tags

Our basic typology of the morphological (or part-of-speech)[3] tag notation is based on the division summarized in (Feldman and Hana, 2010, pp. 60–63). They distinguish two basic categories of tagsets, *atomic* and *structured*, dividing the second one into two further subcategories – *compact* and *positional.* Since the tagsets employed in this work are only the atomic and positional ones, we will leave the compact subcategory aside. Also, for the sake of contrast, we will often use a denomination *non-positional* for the atomic tags in the scope of this work.

The *positional tags* are applied mostly to the morphological tags (in the literal sense) since they allow the description of more word features in a more compact way. In this system, the tag of a word is a string of fixed length where each position has its own meaning, specified by the author of the system. Using the

[3]The terms *morphological* and *part-of-speech* tag are often used synonymously in this context, meaning the broader sense of these labels. However, literally taken, part-of-speech tags capture less information than the morphological tags, which can describe many other features of the word, such as case or number.

example from Table 2.2, the word **masa** has a positional tag `NNNS2-----A----` (using the Czech Positional Tag System[4]) with the meaning shown in Table 2.4.

pos	meaning of the position	value	meaning of the value
1	Part of Speech	N	Noun
2	Detailed Part of Speech	N	Noun (general)
3	Gender	N	Neuter
4	Number	S	Singular
5	Case	2	Genitive
6	Possessor's Gender	-	Not applicable
7	Possessor's Number	-	Not applicable
8	Person	-	Not applicable
9	Tense	-	Not applicable
10	Degree of comparison	-	Not applicable
11	Negation	A	Affirmative (not negated)
12	Voice	-	Not applicable
13	Unused	-	Not applicable
14	Unused	-	Not applicable
15	Variant, Style, Register, Special Usage	-	Not applicable

Table 2.4: Example – positional tag

The *atomic* tag system does not use position-specific information and often denotes only one feature of a word, typically part of speech. On the other hand, this notation is often more readable as the value of the tag may be an (more or less) expressive abbreviation of its meaning, since its length is not theoretically limited. For example, the word **replacement** from the English example from Section 2.1 has tag `NN`, meaning that it is a "Noun, singular or mass" (using the Penn Treebank tagset as described in (Marcus et al., 1993)). This example also demonstrates that non-positionality is not necessarily constrained to one feature only, however its characteric property is a limited set of tags given explicitly by an enumeration.

[4]`http://ufal.mff.cuni.cz/czech-tagging/`

3. From Theory to Implementation

During the reading of this chapter, it is important to have in mind the main targets we pursued while designing the implementation. These were the runtime speed of the final application and simplicity of the ideas and constructions used (to enable a possible consecutive development). These targets were chosen because of the experimental character of this work and influenced some of the decisions mentioned further.

The implementational aspects are described in this chapter in rather general manner. For more details on any of the actual programming solutions, the reader is encouraged to read the Programming Documentation, which can be found on the CD-ROM as a part of this work.

3.1 Instance-based Idea

After thorough examining the problem of experimental annotation detection and correction, the instance-based machine learning approach was the foremost candidate for the solution. The first and main reason is that its quite straightforward idea is simple to implement and the computational demands are not high even when working with large amounts of the linguistic data.

The decision not to work with the lexical information of the words as the instance features and consider their morphological tags only was motivated by the idea to “help” the statistics. Using the wordforms (or lemmata) instead, after training we would end up with an enormous amount of instances in memory, while almost each of it had been seen only once. Then during the classification, we would again encounter a large number of unknown words and moreover, we would have no reasonable way to establish any similarity metric between the instances. By generalizing to the morphological tags, which represent certain “bins” for the words, these problems can be largely solved.

3.1.1 Metric and Similarity

The question of how to establish the similarity metric between two tags was maybe the most problematic one in the whole implementation design. Working with positional morphological tags, the first simple solution that comes in mind is to define the distance of given two tags as the number of different values on the corresponding positions.[1] The definition of the distance of two strings (tags) s_1 and s_2 of the same lenght l would be defined in this sense as follows:

$$d(s_1, s_2) \equiv \sum_{i=1}^{l} \delta(s_1[i], s_2[i])$$

where $\delta(a, b) = 1$ when $a = b$ and 0 otherwise.[2] Moreover, there is also the possibility of weighting the individual positions to emphasize the morphological categories according to them. Having the weight w_i for the position i, we can slightly modify the previous definition:

$$d(s_1, s_2) \equiv \sum_{i=1}^{l} w_i \cdot \delta(s_1[i], s_2[i])$$

However, it proved to be that using this method, the mechanism of searching for the most similar instances among the learned examples in the memory would be either really slow (for each word to classify, it would have to go through all the stored instances[3]), or very difficult to implement – using some sophisticated heuristics. Thus the final decision was not to focus on the particular positions within the tag and define the distance between two strings s_1, s_2 simply by their comparison:

$$d(s_1, s_2) \equiv \delta(s_1, s_2)$$

(having defined δ similarly for whole strings). In other words, the generic term *similarity* is instantiated in this implementation as *identity*.

3.2 Learning and Memory Representation

During the learning phase, the necessary annotation data are read from the individual annotation files and saved into the memory. The data input of the application is one or more PML files with the morphological layer of annotation (plus the referenced analytical-layer files, if the dependency-tree context approach is chosen). A simple XML parser extracts the tag contexts of each token and this context is stored in the memory along with the given token's own tag. When encountering the sentence boundary while fetching the context, a special tag is added to the context (one for the beginning of the sentence, another for its end

[1]This is also known as the *Hamming distance* as defined in (Hamming, 1950).

[2]And using the common computer science notation of indexing the string positions with the square brackets.

[3]Number of the stored contexts varied from tens to hundreds of thousands in the experiments.

– i.e. *after* the final punctuation). Similarly, when fetching a dependency-tree context, another special tag is added for the "null child" to capture the context of a token which has not enough dependency children. (A special tag for a "null parent" is not needed in this case, because the PDT implementation adds a *technical root* to each dependency tree, not corresponding to any of the words in the sentence.)

In fact, the string of text representing the context serves as a key for further searching — each unique context is stored only once and indexes a set of "target" tags encountered in this context with their absolute frequency count among the training data. The memory representation of an entry corresponding to one context (of type $m_{2.1}$) can be seen as a lookup table entry shown in the Table 3.1.

context	tag	count
(`IN`, `DT`, `MD`)	`NN`	104
	`,`	43
	`VB`	37

Table 3.1: Example – memory representation of a context entry

3.3 Searching and Classification

When the learning is finished, the application opens the file with the morphological annotation to correct (or more files, but they are actually processed one at a time) and begins the classification phase with the use of the learned information. Again, the application proceeds token by token, and the procedure of fetching the context is analogous to the one during the learning phase. When the context of a token is collected, the learned table is searched for the corresponding entry. If this context is not found, the application does not dare to make any decision and leaves the token as it is. But if the context exists in the table, then its set of "target" tags is searched for the given token's tag. If the tag exists in this set (regardless of its frequency count), everything seems to be okay[4] and the token is again left as it is. But if the token's tag is not found in the learned context's tag set, it seems suspicious and the application will alter the tag. For this substitution, the tag with the maximum frequency count is chosen from the context's learned set of "target" tags.

For illustration, going back to the example shown in Table 3.1; let us con-

[4]This case can be seen as if the program were saying "I have seen this combination of context and tag somewhere, so it is alright."

sider having the learned table consisting only of this one context.[5] Then the Table 3.2 shows the summary of the possible classification outputs depending on the circumstances.

examined token's context	examined token's tag	action	output tag
(IN, XX, MD)	YY	no change	YY
(IN, DT, MD)	,	no change	,
(IN, DT, MD)	YY	change to NN	NN

Table 3.2: Example – classification outputs

One remark has yet to be made: When the tag of a token is changed, the newly assigned tag is used when fetching the following tokens' contexts.[6]

3.4 Time and Memory Complexity

Due to the simplification of the *similarity* concept (see 3.1.1 for details), all the procedures handling the learned table entries could be implemented quite fast. Therefore in practice, more time-consuming are the operations of file opening and reading than the actual computation. However, we will describe the theoretical ammortized complexity of the individual procedures. For sake of the following description, let n be the number of tokens in the given training data set and m the number of tokens in the test data set.

Time

- The learning procedure for each training token consists of searching the learning table for the given context and searching its set of "target" tags for the tag of the token. Thanks to the effective tree implementation of these two sets, each of the operations takes up to $O(log(n))$ time units. Adding an entry to these sets costs each up to an additional $O(log(n))$. Having considered this process for up to n tokens, the time complexity of the whole learning phase is $O(n \cdot log^2(n))$.

- Searching the learned table during the classification is similar to the learning case, but when changing a suspicious tag, the process is a little more

[5]Of course, this would be practically impossible, as the context tags would have to be learned as well along with *their own* contexts.

[6]This setting can be switched off in a specific combination of options described explicitly in the parameter file.

complex. It requires going through all the "target" tags in the context's set looking for the one with the greatest frequency count, which costs $O(n)$ in the worst case. However, the worst cases of the two stages are in fact somehow inverted (the more contexts, the smaller tag set has each of them, and vice versa), therefore the combination of these two operations is again somewhere around $O(log^2(n))$. Hence for m testing data tokens, the overall time complexity of the classification phase is $O(m \cdot log^2(n))$.

Memory The main and by far the most memory demanding data structure is the learning table, consisting of a large amount of context and some number of tags corresponding to them. This combination varies by the input language and types of input texts used, but the theoretical upper bound of this number is $O(n)$. Or to put in another way, it is at most $(t + 1) \cdot n$, where t is the context length (i.e. each entry of the learned table is represented mainly by the context and one "target" tag). In this case, the size of a string representing the morphological tag is used as a unit.

3.5 User Interface

As an experimental application intended for use mainly by computational linguists, `MissTagger` is designed as a command-line application. The paths to the relevant files are passed to it through the command-line arguments. Most of the parameters related to the learning algorithm and runtime of the application (such as context type setting, details of the output format etc.) are specified in a special parameter file.

For details on the usage of `MissTagger` and overview of its actual features and settings, the reader is encouraged to read the User Documentation (and examine the sample parameter file), which can be found along with the Programming Documentation on the CD-ROM enclosed to this work.

3.6 Programming Language

The `MissTagger` application is programmed in `C++` with use of the STL library containers and algorithms. This choice was made early in the beginning of the work and was motivated by the speed of the resulting code, which proved to be a valid expectation. However, if this decision had to be re-thought again now when the work is done, it might have ended up differently. The most serious candidate language as an alternative would be `Perl`. The speed difference between these two languages (most likely only slight on this task) would probably be only a small price for the comfort of the code writing and backgrounds of the linguistic libraries available especially from our university researchers.

4. Experiments and Evaluation

4.1 Data

Although the `MissTagger` application was originally intended to work with Czech data only, we decided to conduct some of the experiments also on English data for comparison of the results from the language-level perspective. This process involved some adaptation of the data formats and still the English data was restricted to participate only on some parts of the experiments.

4.1.1 Data Sources

Czech data The *Prague Dependency Treebank* (PDT) presents the largest annotated corpus of Czech language.[1] The texts are syntactically analyzed using the dependency approach with the main role of the verb. The annotations go from the morphological layer through to the intermediate syntactic-analytical layer to the tectogrammatical layer (the layer of an underlying syntactic structure). The process of annotation was performed in the same direction, i.e. from the simplest layer to the most complex. This fact corresponds with the amount of data annotated on each level — 2 million words have been annotated on the lowest morphological layer, 1.5 million words on both the morphological and the syntactic layer, and 0.8 million words on all three layers.

English data *The Penn Treebank*[2] was the first syntactically annotated English corpus and remains quite popular for various academic purposes. It was created at the University of Pennsylvania and contains about one million words, both morphologically and syntactically annotated. The main part of the data consists of the Wall Street Journal articles gathered during years 1988–89.

4.1.2 Training and Test Datasets

Both Czech training and test data were selected from PDT 2.0. The statistics and description of data selection from the LDC edition of PDT 2.0 [3] (Hajič et al., 2006) is summarized in Table 4.1, Table 4.2 and Table 4.3. Both English training and test data were selected from the Wall Street Journal section of the Penn

[1] `http://ufal.mff.cuni.cz/pdt2.0`

[2] `http://www.cis.upenn.edu/~treebank/`

[3] `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T01`

Treebank, release 2.[4] The statistics and description of the English data selection is presented along with the Czech in Tables 4.1, 4.2 and 4.3.

notation	source corpus	# of tokens	# of sentences
d_1^{CZ}	PDT 2.0	364,640	22,333
d_2^{CZ}	PDT 2.0	518,647	29,768
d_3^{CZ}	PDT 2.0	883,287	52,101
d_4^{CZ}	PDT 2.0	1,171,191	68,495
d_5^{CZ}	PDT 2.0	1,535,831	90,828
d_1^{EN}	PennTreebank	392,764	17,045
d_2^{EN}	PennTreebank	875,105	38,241
d_3^{EN}	PennTreebank	1,234,804	53,981

Table 4.1: Training data sets

notation	source corpus	# of tokens	# of sentences
d_0^{CZ}	PDT 2.0	25,255	1,678
d_0^{EN}	PennTreebank	24,995	1,061

Table 4.2: Testing data sets

notation	source folder
d_1^{CZ}	full/mw/train-[1-8]
d_2^{CZ}	full/amw/train-[1-8]
d_3^{CZ}	full/{mw,amw}/train-[1-8]
d_4^{CZ}	full/{amw,tamw}/train-[1-8]
d_5^{CZ}	full/{mw,amw,tamw}/train-[1-8]
d_0^{CZ}	full/mw/dtest[5]
d_1^{EN}	TAGGED/WSJ/[00-07]
d_2^{EN}	TAGGED/WSJ/[00-16]
d_3^{EN}	TAGGED/WSJ/[00-23]
d_0^{EN}	TAGGED/WSJ/24/WSJ_24[50-99].POS

Table 4.3: Training and test data source folders

The original intention was to include also some data from the Czech Academic Corpus 2.0[6] (Hladká et al., 2008) in the test data, as its morphological annotation is possibly erroneous. However, since there is no "correct" counterpart for this annotation, all the testing data would have to be corrected and evaluated

[4] `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95T7`

[5] For technical reasons, only 139 (out of 179) files from this folder were used.

[6] `http://ufal.mff.cuni.cz/rest/CAC/cac_20.html`

manually in order to measure the performance. Unfortunately, performing this task on such amount of data (i.e. of a size comparable to the other two testing datasets) would require time and resources far beyond the scope of this work. For those reasons, testing on the CAC 2.0 was relinquished eventually.

4.1.3 Tag Sets

Two primary tag sets were used in the experiments. The first one is the Czech morphological positional tag system,[7] the second one is the non-positional part-of-speech tag set of the Penn Treebank – for illustration see Table 4.4.

When working with a positional tag set, the application allows the users to focus only on some of the categories, i.e. tag positions. Therefore, we have conducted two subsections of the experiment with the Czech tag set. The first one used all of the 15 positions (t_{15}^{CZ}) of the morphological tag, the second one focused only on the positions for part-of-speech, gender, number, case and person of the given word (t_5^{CZ}). These five categories were chosen for their supposed contribution to the desired context information, because in the Slavic languages, the last four are the categories involving in the agreement (Corbett, 1994) (and the part-of-speech is added for its undisputable importance for the context reliability). The settings of the position choosing was common for both context tags and the tag of the examined word, and, of course, remained the same through both training and application phase.

description	notation	example
Czech positional tag set	t_{15}^{CZ}	`AANS2----1A----=`, `P7-X4----------=`
Czech restricted positional tag set (* stands for an ignored position)	t_5^{CZ}	`A*NS2**-********`, `P*-X4**-********`
Penn Treebank tag set	t^{EN}	`NPS`, `JJS`

Table 4.4: Tag sets

4.2 Tools

4.2.1 Taggers and Parsers

The following linguistic tools were used for the morphological and syntactic analysis of the Czech and English data:

[7]http://ufal.mff.cuni.cz/czech-tagging/

- **Czech**
  - perceptron-based tagger `Morče`;[8]
  - perceptron-based parser (McDonald et al., 2005).
- **English**
  - Stanford Log-linear Part-Of-Speech Tagger (Toutanova et al., 2003).[9]

The analyzers listed above are referred to in this work as *AnyTagger* and *AnyParser*, respectively (for the corresponding language).

The accuracy of Czech tagger on the test data (see Table 4.2) is 88.5% and the accuracy of English tagger on the test data is 96.7%. Such high accuracy of the English tagger influenced our decision not to perform the experiments with dependency-tree features for English.

4.2.2 Other Tools

For more convenient tagging, parsing and other data manipulation, several other tools were used:

- `tool_chain`[10] – UNIX shell script for automatic text processing, integrating processes of tokenization, morphological analysis, tagging and parsing, as well as some basic format conversions (using `btred` – see further). It was created for the Czech Academic Corpus project and is part of its distribution.
- `btred`[11] – Perl-based interface for macro scripting specialized on processing the PDT data. Created as a tool for PDT 2.0, and used in this work especially for various data format conversions.
- various Perl and shell scripts – for tasks when the tools listed above were insufficient, some simple scripts were written for the purposes of this work. These tasks included especially the `MissTagger` output evaluation, but also format conversions and adaptations of the English data. All these scripts are enclosed to this work on the CD-ROM (see 4.4).

[8] `http://ufal.mff.cuni.cz/morce`
[9] `http://nlp.stanford.edu/software/tagger.shtml`
[10] `http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/eng/html/ch3.html#nastroje`
[11] `http://ufal.mff.cuni.cz/~pajas/tred/btred.html`

- `Tree Editor TrEd`[12] – a viewer and editor of the PDT annotation files, part of the PDT 2.0 distribution.
- `Lexical Annotation Workbench (LAW)`[13] – an integrated environment for morphological annotation created for Czech Academic Corpus project. It is intended to be the primary viewer and editor for the `MissTagger` output.

4.3 Performance measures

We use standard performance measures to evaluate MissTagger. Before specifying them, we introduce a set of variables that stands for the frequency of five different situations that can appear while MissTagger checks the output of AnyTagger; the variables are listed in Table 4.5.

variable	# of words that
ok_no	AnyTagger tags correctly and MissTagger confirms its output
ok_ko	AnyTagger tags correctly and MissTagger re-tags them incorrectly
ko_no	AnyTagger mistags and MissTagger confirms its output
ko_ok	AnyTagger mistags and MissTagger re-tags them correctly
ko_ko	AnyTagger mistags and MissTagger re-tags them incorrectly

Table 4.5: Variables to evaluate experiments

We evaluate the subtasks of error detection and error correction using measures Accuracy, Recall, Precision, and F-measure separately:

- $A_{det} = \frac{ok\_no+ko\_ok+ko\_ko}{ok\_no+ok\_ko+ko\_no+ko\_ok+ko\_ko}$
- $R_{det} = \frac{ko\_ok+ko\_ko}{ko\_no+ko\_ok+ko\_ko}$
- $P_{det} = \frac{ko\_ok+ko\_ko}{ok\_ko+ko\_ok+ko\_ko}$
- $F_{det} = \frac{2*P_{det}*R_{det}}{P_{det}+R_{det}}$

- $A_{cor} = \frac{ok\_no+ko\_ok}{ok\_no+ok\_ko+ko\_no+ko\_ok+ko\_ko}$
- $R_{cor} = \frac{ko\_ok}{ko\_no+ko\_ok+ko\_ko}$
- $P_{cor} = \frac{ko\_ok}{ok\_ko+ko\_ok+ko\_ko}$
- $F_{cor} = \frac{2*P_{cor}*R_{cor}}{P_{cor}+R_{cor}}$

[12]`http://ufal.mff.cuni.cz/~pajas/tred/index.html`
[13]`http://ufal.mff.cuni.cz/~hana/law.html`

4.4 Results

We have performed 36 experiments with "any-tagged" Czech data so that $5{\cdot}4{+}4 = 24$ experiments with the linear contexts (five different training sets times number of different linear contexts plus four experiments with the chosen tag positions) and $2 \cdot 4 + 4 = 12$ experiments with dependency-tree contexts was conducted. Only $3 \cdot 4 = 12$ experiments with the English data were performed. The smaller number of experiments with the English data was caused by the fact that there was only three different training data sets and we considered linear contexts only (also because of the non-positional nature of the English tag set there was no meaning of experimenting with chosen tag positions). Tables 4.6–4.9 provides a concise summary of all the experiments. Each experiment is uniquely specified by the training data set and by the context, the performance is listed separately for the error detection and the error correction.

The experiments with the chosen tag positions ("t_5^{CZ}", see Section 4.1.3 for details) were performed only on the largest training datasets of the Czech data (i.e. d_5^{CZ} for linear contexts and d_4^{CZ} for dependency-tree contexts). Due to their relatively non-convincing results we decided not to employ it in the further experiments. The results obtained in these eight experiments are summarized in Tables 4.8 and 4.9. All the other not explicitely marked results with the Czech data concerns performing with all the tag positions ("$t_1 5^{CZ}$").

Some of the results are highlighted graphically. Namely, Figure 4.1 visualizes the F-measure of error detection with the contexts $m_{1.1}$ and $s_{1.1}$ and with the largest Czech and English training data sets, Figure 4.2 shows the best results across the contexts and training datasets for individual "tasks" with both Czech and English. One of the interesting things in this last comparison is that not everywhere the largest training data sets are the most successful. This is due to the nature of the learning algorithm, described in more detail in Section 3.

Detection When examining more deeply the relative success of the particular context types within each language, there is a nice exhibit of a different character of these two languages. While the Czech benefits mainly from the tags of the preceding words (in the linear context), in the English also the following tags are important. Looking at the results of the dependency-tree experiments with the Czech sentences, it appears to be that going deep or broad into the tree does not pay off, when the $s_{1.1}$ context type gives by far the best results.

Correction Probably the most important column of the correction results from the practical point of view is the A_{cor} measure. It shows the accuracy of a fully automated tagging system where AnyTagger is followed by the MissTagger correction. Comparing it with the separate AnyTagger accuracy (88.5% for the Czech tagger and 96.7% for the English tagger), the numbers actually show certain degradation. Although the performance results in this field are not quite convincing, some smaller tendencies can be seen here as well. The results appear to be quite unrelated to the size of the training data (although there is a slight tendency of the precision growth with the growth of the data volume), but some of the context types are systematicely more successful than the others. Interestingly, these are different ones than for the detection task – $m_{2.2}$ for Czech and $m_{1.1}$ for English, this time. Although the dependency-tree results are not as explicit as the linear ones, there can also be a surmise of need of a "bigger" context for the Czech language.

Yet another view on the performance of MissTagger is presented by the Table 4.10. Its first numeric column displays results of experiments when no AnyTagger was employed and MissTagger was applied on correct ("clean") data. The second column shows the numbers of the previously described experiments and the difference between these two columns is computed in the third one (in percentage points). For easy reference and comparison, the numbers expressing the error rates of the Czech and English tagger are repeated in the last column, because these two last values are somehow comparable. The difference between the number of changes MissTagger has made on "clean" and any-tagged data should ideally be adequate to the AnyTagger error rate — this relation shows how the MissTagger reacts to the actual number of errors in the annotation checked. However, it is obvious from this table that also this performance aspect of MissTagger has quite poor results. Speaking of the English experiments, it could be almost said that MissTagger is just guessing almost blindly, regardless of the actual errors.

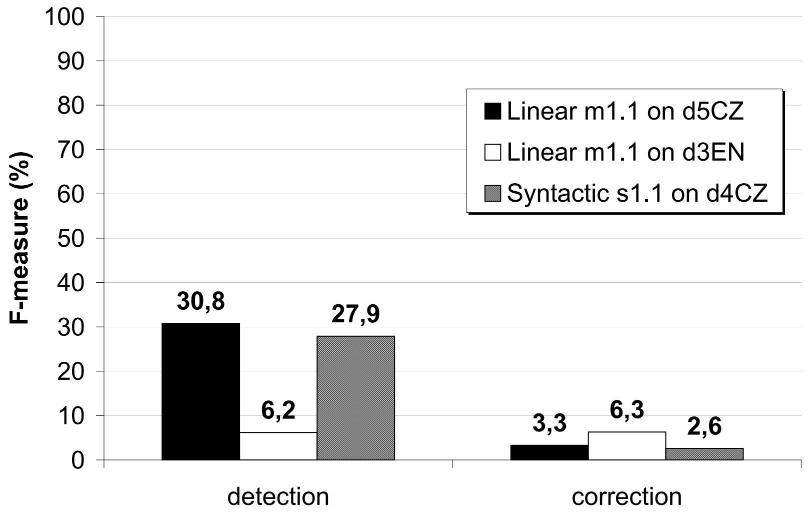

Figure 4.1: F-measure, contexts $m_{1.1}$, $s_{1.1}$, the largest training data sets

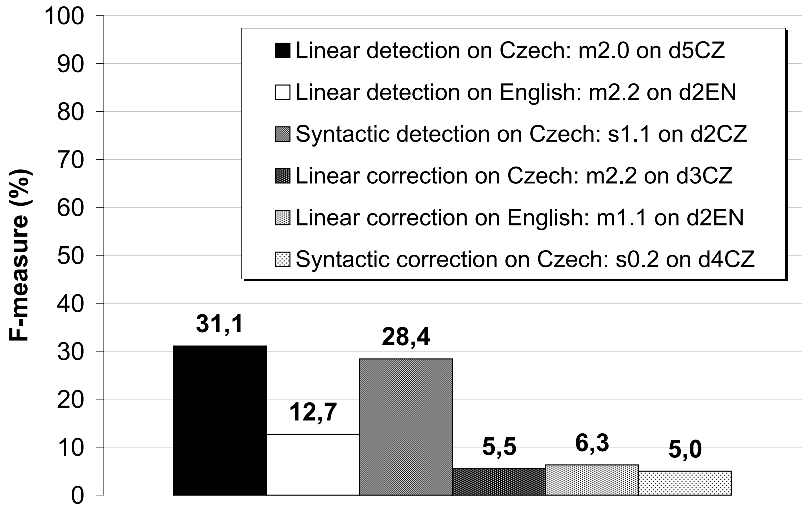

Figure 4.2: F-measure, best results across contexts

data set	context	detection				correction			
		A_{det} (%)	P_{det} (%)	R_{det} (%)	F_{det} (%)	A_{cor} (%)	P_{cor} (%)	R_{cor} (%)	F_{cor} (%)
d_1^{CZ}	$m_{1.1}$	81.7	25.2	31.3	28.0	78.5	2.8	2.2	2.5
d_1^{CZ}	$m_{2.0}$	77.8	23.5	42.5	30.2	73.4	2.9	2.3	2.5
d_1^{CZ}	$m_{2.1}$	74.7	18.1	34.9	23.8	71.5	3.8	3.3	3.5
d_1^{CZ}	$m_{2.2}$	82.0	25.0	29.5	27.1	79.3	6.0	4.8	5.3
d_2^{CZ}	$m_{1.1}$	81.8	25.7	32.1	28.6	78.4	2.5	1.9	2.1
d_2^{CZ}	$m_{2.0}$	78.6	23.0	37.9	28.7	74.6	1.9	1.5	1.7
d_2^{CZ}	$m_{2.1}$	74.6	18.2	35.6	24.1	71.1	2.8	2.3	2.5
d_2^{CZ}	$m_{2.2}$	81.1	25.5	34.9	29.5	77.7	4.9	3.8	4.3
d_3^{CZ}	$m_{1.1}$	84.1	30.1	30.3	30.2	81.0	3.5	2.5	2.9
d_3^{CZ}	$m_{2.0}$	82.1	27.0	34.1	30.2	78.5	2.8	2.1	2.4
d_3^{CZ}	$m_{2.1}$	76.5	20.0	35.7	25.6	73.1	3.4	2.8	3.0
d_3^{CZ}	$m_{2.2}$	80.9	24.6	33.1	28.2	77.9	6.2	5.0	5.5
d_4^{CZ}	$m_{1.1}$	84.1	30.1	30.1	30.1	81.0	3.2	2.3	2.7
d_4^{CZ}	$m_{2.0}$	82.6	28.2	34.4	31.0	79.1	3.3	2.5	2.9
d_4^{CZ}	$m_{2.1}$	76.4	19.4	34.4	24.8	73.1	3.3	2.8	3.0
d_4^{CZ}	$m_{2.2}$	79.9	24.7	37.9	29.9	76.2	4.8	3.8	4.2
d_5^{CZ}	$m_{1.1}$	85.3	33.1	28.7	30.8	82.4	4.0	2.8	3.3
d_5^{CZ}	$m_{2.0}$	84.2	30.7	31.4	31.1	81.0	3.8	2.7	3.2
d_5^{CZ}	$m_{2.1}$	77.8	20.6	33.7	25.6	74.5	3.7	3.0	3.3
d_5^{CZ}	$m_{2.2}$	80.4	24.5	35.1	28.8	77.1	5.6	4.5	5.0
d_1^{EN}	$m_{1.1}$	96.0	18.9	6.1	9.2	95.9	6.1	5.3	5.7
d_1^{EN}	$m_{2.0}$	95.7	8.4	3.0	4.5	95.6	2.8	2.7	2.8
d_1^{EN}	$m_{2.1}$	92.6	10.4	15.9	12.6	92.2	2.3	2.1	2.2
d_1^{EN}	$m_{2.2}$	87.6	8.0	25.9	12.2	86.9	1.7	1.6	1.7
d_2^{EN}	$m_{1.1}$	96.4	24.8	4.3	7.3	96.3	7.0	5.7	6.3
d_2^{EN}	$m_{2.0}$	96.2	10.3	1.9	3.3	96.1	3.5	3.2	3.3
d_2^{EN}	$m_{2.1}$	94.1	11.6	11.8	11.7	93.7	2.0	1.8	1.9
d_2^{EN}	$m_{2.2}$	89.9	8.9	22.0	12.7	89.3	1.5	1.4	1.5
d_3^{EN}	$m_{1.1}$	96.5	27.1	3.5	6.2	96.4	7.1	5.6	6.3
d_3^{EN}	$m_{2.0}$	96.3	10.7	1.6	2.8	96.2	2.7	2.5	2.6
d_3^{EN}	$m_{2.1}$	94.5	11.8	9.9	10.7	94.3	2.6	2.3	2.4
d_3^{EN}	$m_{2.2}$	90.6	8.7	19.2	12.0	90.0	1.5	1.4	1.4

Table 4.6: Error detection and correction: linear contexts

data set	context	detection				correction			
		A_{det} (%)	P_{det} (%)	R_{det} (%)	F_{det} (%)	A_{cor} (%)	P_{cor} (%)	R_{cor} (%)	F_{cor} (%)
d_2^{CZ}	$s_{1.1}$	84.4	29.7	27.3	28.4	83.0	3.3	2.7	3.0
d_2^{CZ}	$s_{0.2}$	84.5	21.7	14.3	17.2	81.7	4.4	3.2	3.7
d_2^{CZ}	$s_{1.2}$	82.4	19.3	17.3	18.3	80.8	3.3	2.8	3.0
d_2^{CZ}	$s_{2.2}$	83.3	21.2	17.3	19.0	81.8	5.4	4.5	5.0
d_4^{CZ}	$s_{1.1}$	85.8	32.9	24.2	27.9	83.6	3.0	2.4	2.6
d_4^{CZ}	$s_{0.2}$	85.0	22.3	13.1	16.5	83.4	6.0	4.3	5.0
d_4^{CZ}	$s_{1.2}$	83.4	20.3	16.0	17.9	81.9	4.0	3.3	3.6
d_4^{CZ}	$s_{2.2}$	83.3	20.3	16.1	18.0	81.9	5.4	4.5	4.9

Table 4.7: Error detection and correction: dependency-tree contexts

data set	context	detection				correction			
		A_{det} (%)	P_{det} (%)	R_{det} (%)	F_{det} (%)	A_{cor} (%)	P_{cor} (%)	R_{cor} (%)	F_{cor} (%)
d_5^{CZ}	$m_{1.1}$	86.3	24.5	10.0	14.2	85.2	1.2	0.9	1.1
d_5^{CZ}	$m_{2.0}$	86.3	27.3	12.6	17.3	84.9	2.0	1.5	1.7
d_5^{CZ}	$m_{2.1}$	79.7	16.3	19.3	17.7	77.6	1.4	1.2	1.3
d_5^{CZ}	$m_{2.2}$	78.5	16.4	22.0	18.8	76.2	2.0	1.7	1.9

Table 4.8: Error detection and correction: t_5^{CZ} with linear contexts

data set	context	detection				correction			
		A_{det} (%)	P_{det} (%)	R_{det} (%)	F_{det} (%)	A_{cor} (%)	P_{cor} (%)	R_{cor} (%)	F_{cor} (%)
d_4^{CZ}	$s_{1.1}$	86.5	26.8	11.1	15.7	85.3	3.4	2.6	3.0
d_4^{CZ}	$s_{0.2}$	86.4	16.6	5.1	7.8	85.8	1.7	1.5	1.6
d_4^{CZ}	$s_{1.2}$	83.9	15.7	9.6	11.9	82.9	1.6	1.4	1.5
d_4^{CZ}	$s_{2.2}$	82.5	15.0	11.6	13.1	81.3	1.4	1.2	1.3

Table 4.9: Error detection and correction: t_5^{CZ} with dependency-tree contexts

data	context	clean changed (%)	any-tagged changed (%)	difference (pp)	AnyTagger error rate (%)
d_1^{CZ}	$m_{1.1}$	11.6	14.1	+2.5	11.5
d_1^{CZ}	$m_{2.2}$	11.4	13.4	+2.0	11.5
d_5^{CZ}	$m_{1.1}$	6.6	9.8	+3.2	11.5
d_5^{CZ}	$m_{2.2}$	14.0	16.2	+2.2	11.5
d_1^{EN}	$m_{1.1}$	1.0	1.1	+0.1	3.3
d_1^{EN}	$m_{2.2}$	10.7	10.8	+0.1	3.3
d_3^{EN}	$m_{1.1}$	0.4	0.4	+0.0	3.3
d_3^{EN}	$m_{2.2}$	7.3	7.4	+0.1	3.3
d_2^{CZ}	$s_{1.1}$	5.7	10.4	+4.7	11.5
d_2^{CZ}	$s_{2.2}$	7.1	9.2	+2.1	11.5
d_4^{CZ}	$s_{1.1}$	3.9	8.4	+4.5	11.5
d_4^{CZ}	$s_{2.2}$	6.6	9.0	+2.4	11.5

Table 4.10: Cross-section comparison of results with clean and any-tagged data

Conclusion

In this work we have implemented an instance-based learning system called `MissTagger` and evaluated it on the output of an automatic tagger. This application checks the morphological layer of a corpus annotation for possible annotation errors. The main focus of this work was on involving also the syntactic layer of annotation and to compare it to the morphological-only approach. In the experiments, either linear or dependency-tree feature vectors were trained on the manually tagged and parsed data. Then the test data were tagged and parsed automatically by the supervised learning procedures that show accuracy below 90% for Czech (accuracy 88.5% of the Czech tagger and 84.5% accuracy of the Czech parser) and 96.5% accuracy of English tagger. Looking at the evaluation results, the F-measures of English experiments are significantly smaller than F-measures of Czech experiments mainly because of the significantly high difference between the performance of the two taggers.

Contrary to the initial anticipation, the dependency-tree contexts does not show significantly better results to the linear contexts. It is possible to guess that this is caused by the relatively low accuracy of Czech parser. Having Czech parser with higher accuracy will probably bring some improvement over the linear contexts with respect to a free word order character of Czech. However, excellent results cannot be expected since the parser itself operates on the output of a tagger which is again automatic, therefore has also only a limited accuracy.

The error detection is definitely more successful than the error correction. Improving the error correction is closely related to the size of training data and the tag sequences distribution in it. This tendency can be seen also on comparison of the Czech and English correction results.

Experimenting with choosing the tag positions in the Czech tag set did not show any performance improvement. Although the corresponding categories play important role in the agreement and context comparison, focusing only on them did not help significantly the results of the method employed. One of the reasons for this could be the quite complex relations between all the tag positions, so it is difficult to just isolate some of them.

Unfortunately, the results have shown that the application in this form is not employable in large scale in practice even for annotation error detection. With this task in mind, the key measure is recall, and having the best numbers of this value only around 35% (i.e. only one third of the actual errors are covered, plus the corresponding precision also not quite convincing), the practical use of this method does not seem realistic. However, the approach idea is still promising and it seems worth further attempts to some modification and improvement. One of the direction of the adjustment could be for example loosening the *similarity* definition, thus increasing the detection recall. Another idea yet to be examined is some sort of more tight cooperation between the morphological and syntactic layer, for example starting with the more confident parts of the automatic tagger and parser outputs and building on them gradually further.

Bibliography

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-Based Learning Algorithms. In *Machine Learning*, vol. 6(1):pp. 37–66.

Boyd, A., Dickinson, M., and Meurers, D. (2007). Increasing the Recall of Corpus Annotation Error Detection. In *Proceedings of the Sixth Workshop on Treebanks and Linguistic Theories*. Bergen, Norway.

Corbett, G. G. (1994). *Agreement*, pp. 54–60. Pergamon Press, Oxford, UK.

Dickinson, M. (2005). *Error Detection and Correction in Annotated Corpora*. Ph.D. thesis, The Ohio State University, USA.

Dickinson, M. and Meurers, W. D. (2003). Detecting Errors in Part-of-Speech Annotation. In *Proceedings of EACL-03*, pp. 107–114. Budapest, Hungary.

Elworthy, D. (1994). Automatic Error Detection in Part of Speech Tagging. In *Proceedings of the International Conference on New Methods in Language Processing*.

Feldman, A. and Hana, J. (2010). *A resource-light approach to morpho-syntactic tagging*. Rodopi, Amsterdam/New York, NY.

Hajič, J., Panevová, J., Hajičová, E. J., Sgall, P., Pajas, P., Štěpánek, J., Havelka, J., and Mikulová, M. (2006). *Prague Dependency Treebank 2.0*.

Hamming, R. W. (1950). Error Detecting and Error Correcting Codes. In *The Bell System Technical Journal*, vol. XXIX(2).

Hladká, B. V., Hajič, J., Hana, J., Hlaváčová, J., Mírovský, J., and Raab, J. (2008). *Czech Academic Corpus 2.0*.

Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. In *Informatica*, vol. 31:pp. 249–268.

Květoň, P. and Oliva, K. (2002). (Semi-)Automatic Detection of Errors in PoS-Tagged Corpora. In *Proceedings of the 19th Conference on Computational Linguistics*. Taipei, Taiwan.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. In *Computational Linguistics*, vol. 19(2):pp. 313–330.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *HLT '05 Proceedings of the conference on Human Language, Technology and Empirical Methods in Natural Language Processing*, pp. 523–530. Vancouver, Canada.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. ISBN 0-07-042807-7.

Toutanova, K., Klein, D., Manning, C., and Singer, Y. (2003). Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of HLT-NAACL*, pp. 252–259.

List of Figures

List of Tables

Appendix - CD-ROM Contents

- `data` folder – Contains sample data from three annotated copora: the Czech Academic Corpus 2.0 (`cac20_sample`), the Prague Dependency Treebank 2.0 (`pdt20_sample`) and the Wall Street Journal part of the Penn Treebank (Release 2, `penntb-wsj_sample`). All the files with the `.m` (or `.a`) extension in this folder are in the PML format and ready to be used by MissTagger. Sample filelists with a basic division of the files to training and testing sets are also located in the respective folders.
- `misstagger` folder – Source, binary and documentation of the MissTagger application. Contains all the original source files along with the other modules used by MissTagger. The file `bin/MissTagger-win32.exe` is an executable binary file compiled for the 32-bit Windows operating system. Linux users are recommended to use the `Makefile` in the `source` folder to compile the binary themselves. The `install-readme.txt` file provides the details about the installation, the two `.pdf` documentation files contains various information about the application usage and implementation.
- `scripts-[linux|perl|win]` folder – Auxiliary scripts for data preparation and evaluation of the MissTagger procedure.
- `vacl-bach_thesis.pdf` file – This work in pdf format.
- `readme.txt` file – General information about the contents of the CD-ROM.