Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Kamil Kos

Adaptace nových metrik strojového překladu pro češtinu

Adaptation of New Machine Translation Metrics for Czech

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Ondřej Bojar
Studijní program: Informatika, obor Obecná informatika

2008

I would like to express my gratitude to my supervisor, RNDr. Ondřej Bojar, for his suggestions and editing of the Bachelor Thesis. My thanks also belong to my friend and fellow student Jan Tattermusch who implemented the Aho-Corasick search algorithm that was used in this thesis.

I hereby declare that I have prepared this paper on my own, using only the materials cited. Ideas taken from other sources are identified as such. I give consent to publish this thesis.

August 4, 2008 Kamil Kos

Contents

Title: Adaptation of New Machine Translation Metrics for Czech
Author: Kamil Kos
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Ondřej Bojar
Supervisor's e-mail address: Ondrej.Bojar@mff.cuni.cz

Abstract: In the present work we study semi-automatic evaluation techniques of machine translation (MT) systems based on comparison of the MT system's output to human translations of the same text. Various metrics were proposed in the past years, ranging from metrics using only unigram comparison to metrics that try to take advantage of additional syntactic or semantic information. The main goal of this thesis is to compare these metrics with respect to their correlation with human judgments and to propose the most suitable ones for evaluation of MT systems with Czech as target language. An implementation of a tool that computes the MT metrics is part of this work.

Keywords: machine translation, evaluation, metric, Czech, natural language processing

Název práce: Adaptace nových metrik strojového překladu pro češtinu
Autor: Kamil Kos
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: RNDr. Ondřej Bojar
e-mail vedoucího: Ondrej.Bojar@mff.cuni.cz

Abstrakt: V předložené práci studujeme poloautomatické způsoby hodnocení systémů strojového překladu, které jsou založeny na porovnávání výstupu systému s lidskými překlady zdrojového textu. V minulých letech byly navrženy rozličné metriky. Některé používají pouze porovnávání unigramů, jiné se snaží využít i syntaktické nebo sémantické informace. Cílem této práce je porovnat korelaci vybraných metrik s lidskými hodnoceními a navrhnout nejvhodnější metriky pro hodnocení systémů strojového překladu, které překládají z cizího jazyka do češtiny. Součástí práce je implementace nástroje, který dané metriky počítá.

Klíčová slova: strojový překlad, hodnocení, metrika, čeština, zpracování přirozeného jazyka

Chapter 1

Introduction

This work compares the quality of metrics that are being used in natural language processing (NLP) to evaluate the quality of machine translation (MT) systems. The goal is to find the most suitable metrics for evaluation of MT systems with Czech as the target language.

The quality of metrics which are examined in this work has already been compared many times to each other. However, it was usually done for English or some other widespread language, but not for Czech. Because languages have different sentence structure, especially if you compare languages with fixed sentence order, e.g. English, to languages with relatively free sentence order like Czech, some metrics can be more suitable for the one or the other language. Hence, we would like to see if there are any differences in performance of these metrics for Czech as the target language.

1.1 Evaluation of MT Systems

Machine translation belongs to one of the most demanding tasks in NLP. Even though there are various approaches to machine translation, ranging from rule based systems to statistical MT systems, there is only one thing that matters. Namely the quality of the translation they deliver. But how to compare the quality of output of two MT systems? Which translation is better?

The easiest way to evaluate the quality of a MT system output (*candidate translation*) is to give the source text and the output to a translator who can check whether the translation is correct. Even if the exact understanding of a text can differ among people, an experienced translator should be able to assess the quality of a translation. However, this is a very expensive and time consuming approach since every single translated sentence must read by a human and evaluated. Besides, it is difficult to define quality criteria so that every translator would evaluate the same translation in the same way.

Much better approach is to have a set of evaluation sentences with their correct translation(s) - *reference translation(s)*. These translations can be prepared in advance by human translators and shared for evaluation of various MT systems. Then, it is possible to define some similarity metric that compares the candidate translation (MT system output) to the given reference translation(s). The more reference translations we have, the better because there are often several ways how to translate the source sentences. Each reference translation is considered as one possible way how to translate the source text.

In order to simplify the evaluation process, it is suitable to evaluate the candidate translation sentence by sentence. The amount of information contained in one sentence is not excessively large and there are not too many ways how to convey the information. Hence, the probability that the information in a correct candidate translation is expressed in similar way as in some of the reference translations is relatively high. Therefore, a reasonable approach is to compare the candidate translation and its structure to the reference translation on the sentence level.

1.2 Previous Work

Early approaches to scoring a candidate translation with respect to a reference translation were based on the idea that the similarity score should be proportional to the number of matching words (e.g. [11]).

Another idea is that matching words in the right order should result in higher scores than matching words out of order (e.g. [15]). Perhaps the simplest version of the same idea is that a candidate text should be rewarded for containing longer contiguous subsequences of matching words. One of the first metrics following this principle was *BLEU* [13] introduced in 2001, which established itself as the standard metric in MT evaluation. In 2002, another version of this idea was proposed, now commonly known as the *NIST* score [5]. However, most of the metrics were using only the lexical level and syntactic or semantic features were ignored.

In 2005, Meteor metric [2] was introduced which not only used the morphological level by working with stems of words but also incorporated word synonymy. In the recent years new metrics emerged that try to take advantage of more syntactic or semantic information that can be obtained by analyzing the translations, e.g. a set of metrics based on linguistic features [7].

1.3 Outline of the Thesis

The following chapters are divided as follows: in Chapter 2, MT metrics are described whose quality has been evaluated in this work. Chapter 3 contains description of experiments that have been conducted. Chapter 4 contains a description of the MT evaluation tool *MTrics* that has been implemented as part of this work and used for computing the metric scores. The last chapter summarizes this work and provides suggestions for future work.

Appendix A contains user documentation of *MTrics* and Appendix B contains the DTD specification of the *mtveal* file format that can be used for input files by *MTrics*.

Chapter 2

Metrics

In this chapter, we describe the most common metrics in the MT system evaluation. We use following notation: an MT evaluation metric scores a sequence of MT system output segments (sentences in our case) $S = s_1, s_2, ..., s_I$ with respect to a set of references R. References R are correct translations of the respective segments. Since it is possible to have multiple reference translations of a candidate translation, we consider $R = R_1, R_2, ..., R_I$, where R_i is a set of reference translations of s_i.

2.1 N-gram Metrics

N-gram metrics can be considered as the most common metrics that are used for MT system evaluation. They are easy and fast to compute since they just compare normalized counts of n-grams in candidate and reference translation.

2.1.1 BLEU

The BLEU metric [13] is based on the geometric mean of n-gram precision. The score is given by:

$$BLEU = BP * \exp\left[\sum_{n=1}^{N} w_n * log(p_n)\right]$$

where N is the maximum n-gram size and w_n has the uniform weight $1/N$ for all $n = 1 \dots N$. The n-gram precision p_n is given by:

$$p_n = \frac{\sum_{i \in I} |\{ngram \mid ngram \in ngr_n(s_i) \cap maxngr_n(R_i)\}|}{\sum_{i \in I} |\{ngram' \mid ngram' \in ngr_n(s_i)\}|}$$

where $ngr_n(s_i)$ is a bag of n-grams of the length n found in candidate translation s_i and $maxngr_n(R_i)$ is a bag of n-grams of the length n which contains for each unique $ngram \in \bigcup_{i=1 \ldots |R_i|} ngr_n(r_i)$ the maximum number of occurrences of $ngram$ found in some $r_i \in R_i$. The n-gram precision p_n is accumulated over all sentences that are being evaluated.

The brevity penalty BP penalizes MT output for being shorter than the corresponding references and is given by:

$$BP = \begin{cases} 1 & \text{if c >r} \\ \text{exp(1 - r/c)} & \text{if c} \leq \text{r} \end{cases}$$

where c is the accumulated number of words in the candidate sentences and r is the accumulated number of words in the corresponding reference sentences. In case of multiple reference translations, the reference whose length is the closest to the candidate on the sentence level is taken.[1] The BLEU brevity penalty is a single value computed over the whole set of sets.

2.1.2 NIST

The NIST metric [5] also uses n-gram precision, differing from BLEU in that an arithmetic mean is used, weights are used to emphasize informative word sequences and the formula for brevity penalty is different.

$$NIST = BP * \sum_{n=1}^{N} \frac{\sum_{i \in I, ngram \in ngr_n(s_i) \cap maxngr_n(R_i)} info(ngram)}{\sum_{i \in I} |\{ngram' \mid ngram' \in ngr_n(s_i)\}|}$$

where $ngr_n(s_i)$ is a bag of n-grams of the length n found in candidate translation s_i and $maxngr_n(R_i)$ is a bag of n-grams of the length n which contains for each unique $ngram \in \bigcup_{i=1 \ldots |R_i|} ngr_n(r_i)$ the maximum number of occurrences of $ngram$ found in some $r_i \in R_i$.

The function $info(ngram)$ is computed over the set of reference translations, according to the following equation:

$$info(ngram = w_1 w_2 \ldots w_n) = log_2 \left(\frac{count(w_1 w_2 \ldots w_{n-1})}{count(w_1 w_2 \ldots w_n)} \right)$$

where $count(ngram)$ is the number of reference translations, in which $ngram$ can be found. Note that the weight of an n-gram occurring in many references is

[1]The original paper does not specify which length to take if two references, one longer and one shorter than the candidate sentence, have the same distance to the candidate. We have decided to take the shorter one.

considered to be lower than the weight of a phrase occurring only in one reference.

For NIST, the brevity penalty is computed as

$$BP = exp\left(\beta * log^2\; min(\frac{c}{r}, 1)\right)$$

where c is the length of the MT system output, r is the average number of words in the reference translation and β is chosen to make BP = 0.5 when $c = 2/3$.

2.2 Unigram Metrics

Unigram metrics compare counts of single tokens (words) between the candidate and the reference translation. Like n-gram metrics, they are easy to compute. However, they suffer from deficiencies in the evaluation of the fluency of a candidate translation. Hence, some of the metrics try to compensate this drawback by additional features, such as *runs* in GTM metric.

2.2.1 F-measure

F-measure metric is defined as the harmonic mean of *precision (prec)* and *recall (rec)*: $\frac{prec+rec}{2*prec*rec}$. In case of multiple references, precision $prec_i$ and recall rec_i are defined as:

$$prec_i = \frac{max_{r_i \in R_i}[clip(s_i, r_i)]}{|nrg_1(s_i)|} \qquad rec_i = \frac{max_{r_i \in R_i}[clip(s_i, r_i)]}{|ngr_1(bestClipRef(s_i, R_i))|}$$

where $clip(s_i, r_i)$ is the number of tokens observed both in s_i and r_i, i.e. $clip(s_i, r_i) = |ngr_1(s_i) \cap ngr_1(r_i)|$. $BestClipRef(s_i, R_i)$ returns the reference $r_i \in R_i$ such that $clip(s_i, r_i)$ is the biggest. In case that for two references r_{i1} and r_{i2} $clip(s_i, r_{ij})$ has the same size, $bestClipRef(s_i, R_i)$ returns the shorter reference.

Intuitively, in case of one reference, precision is the number of words that co-occur in candidate and reference sentence divided by the size of the candidate sentence, and recall is the number of words that co-occur in candidate and reference sentence divided by the size of reference sentence.

If a complete text is to be evaluated, the co-occurring words are summed up for all sentences at first and then divided by the total candidate or reference length.

$$prec = \frac{\sum_{i \in I} max_{r_i \in R_i}[clip(s_i, r_i)]}{\sum_{i \in I} |ngr_1(s_i)|} \qquad rec = \frac{\sum_{i \in I} max_{r_i \in R_i}[clip(s_i, r_i)]}{\sum_{i \in I} |ngr_1(bestClipRef(s_i, R_i))|}$$

Sometimes, it is useful to assign weights P and R to precision and recall. The score is then computed as $\frac{P*prec+R*rec}{(P+R)*prec*rec}$.

2.2.2 Position-independent Word Error Rate

PER [19] is similar to WER (see Section 2.3.1) except that word order is not taken into account. Both sentences are treated as bags of words:

$$PER(s_i, R_i) = \frac{min_{r_i \in R_i} dist(s_i, r_i)}{|ngr_1(bestDistRef(s_i, R_i))|}$$

where $dist(s_i, r_i)$ is the distance between s_i and r_i defined as

$$dist(s_i, r_i) = max\left[|ngr_1(s_i) \backslash ngr_1(r_i)|, |ngr_1(r_i) \backslash ngr_1(s_i)|\right]$$

and $bestDistRef(s_i, R_i)$ returns the reference $r_i \in R_i$ such that $dist(s_i, r_i)$ is the smallest.

To evaluate the complete set of sentences S, the distance between s_i and r_i is accumulated for all $i \in I$ and then divided by the reference length:

$$PER(S, R) = \frac{\sum_{i \in I} min_{r_i \in R_i} dist(s_i, r_i)}{\sum_{i \in I} |ngr_1(bestDistRef(s_i, R_i))|}.$$

2.2.3 General Text Matcher

GTM metric [20] is inspired by the plain F-measure trying to eliminate (one of) its major drawbacks. Since F-measure is based only on unigram matching, two sentences containing the same words always get the same F-measure rating regardless of the correct order of the words in the sentence.

GTM uses the notion of *matching*. A *matching* is a one-to-one assignment of words from candidate sentence to words of reference sentence which are the same.[2] It is possible that some words stay unmatched if they do not have a counterpart in the reference sentence. The *match size* of a matching is the number of words that have a counterpart in the other sentence. A *maximum matching* is a matching of maximum possible *match size* for a particular pair of candidate and reference translations.

A contiguous sequence of words such that their counterparts are also a contiguous sequence (in the correct order) is called a *run*. Now, we can describe how GTM is computed. GTM value is defined as the harmonic mean of precision $prec_i$ and recall rec_i where

$$prec_i = \frac{size(M_i)}{|ngr_1(s_i)|} \qquad rec_i = \frac{size(M_i)}{|ngr_1(r_i)|}$$

[2]The word *dog* can be matched only to *dog* but not to *dogs*.

and M_i is a maximum matching. The size of maximum matching is bounded by the average reference length in case of multiple reference translations. In order to prefer longer contiguous sequences, the *size of maximum matching* M[3] is defined as follows:

$$size(M) = \sqrt[e]{\sum_{r \in M} length(r)^e}$$

where r is a run in matching M. The exponent e is used to prefer longer sequences of words. If $e = 1$, the function $size(M)$ only returns the number of words of the matching M, i.e. the match size of M.

To score the whole S, accumulated precision *prec* and recall *rec* are used:

$$prec = \frac{\sum_{i \in I} size(M_i)}{\sum_{i \in I} |ngr_1(s_i)|} \qquad rec = \frac{size(M_i)}{\sum_{i \in I} |ngr_1(r_i)|}.$$

2.3 Edit Distance Metrics

Edit distance metrics define the quality measure of a candidate translation on the basis of distance of the candidate translation to the reference translation. This distance is defined by the number of edit operations that are needed to transform one sentence into another.

2.3.1 Word Error Rate

WER metric [17] is defined as the minimum number of edit operations required to transform one sentence into another normalized by the length of the reference translation

$$WER(s_i, r_i) = \frac{min\,(I(s_i, r_i) + D(s_i, r_i) + S(s_i, r_i))}{|r_i|}$$

where $I(s_i, r_i)$, $D(s_i, r_i)$ and $S(s_i, r_i)$ are the number of insertions, deletions and substitutions respectively. The numerator of the equation above is also known as the Levenshtein distance. In case of multiple references, we take the reference for which the Levenshtein distance to candidate sentence is the smallest. The cumulative score for S is

$$WER(S, R) = \frac{\sum_{i \in I} min\,(I(s_i, r_i) + D(s_i, r_i) + S(s_i, r_i))}{\sum_{i \in I} |ngr_1(r_i)|}.$$

[3]Note, that *size of maximum matching* M is something different from *match size* of M

2.3.2 Translation Error Rate

TER metric [16] is also based on the number of operations needed to transform the candidate sentence into the reference sentence. However, it allows one additional operation: the *block shift*. Hence, possible operations include insertion, deletion, and substitution of single words as well as shifts of word sequences. A *shift* moves a contiguous sequence of words within the candidate sentence to another location within the candidate sentence. All edits, including shifts of any number of words, by any distance, have equal cost

$$TER(S,R) = \frac{\sum_{i \in I} min\left(I(s_i,r_i) + D(s_i,r_i) + S(s_i,r_i) + \mathit{Shift}(s_i,r_i)\right)}{\sum_{i \in I} |ngr_1(r_i)|}$$

where $I(s_i,r_i)$, $D(s_i,r_i)$, $S(s_i,r_i)$ and $\mathit{Shift}(s_i,r_i)$ are the number of insertions, deletions, substitutions and block shifts respectively.

In case of multiple references, we take the reference $r_i \in R_i$ for which the number of edit operations is the lowest.

2.4 Semantic Metrics

2.4.1 Meteor

Meteor metric [2] was one of the first metrics that tried to incorporate some semantic information in the evaluation process. Originally, it was designed for English as the target language and here we propose some minor modifications to adapt it for Czech.

Given a candidate translation s_i and a reference translation $r_i \in R_i$, Meteor creates an alignment between the two translations. An alignment is defined as a mapping between unigrams, such that every unigram in each translation maps to zero or one unigram in the other translation, and to no unigrams in the same translation. Thus in a given alignment, a single unigram in one translation cannot map to more than one unigram in the other translation. This alignment is incrementally produced according to Meteor modules that are being used.

Each module defines a set of word-to-word mappings (one word being from s_i and the other one from r_i) such that mapped words can be matched. The mappings contain only unigrams that have not been mapped to any unigram in any of the preceding stages. In the original implementation, the modules are *exact*, *porter stem* and *WN synonymy*. *Exact* module matches two words if they have the same surface representation (e.g. *dog* matches *dog* but not *dogs*). *Porter stem* module matches two words if they have the same stem according to Porter stemmer [14] (e.g. *dogs* matches *dog*) and *WN synonymy* module matches two words if they are synonyms. Our modification of the metric replaces the *porter stem* module with *lemma* module which matches two words, if they have the same lemma.

For each module, the largest subset of these unigram mappings is selected such that the resulting set constitutes an alignment as defined above (that is, each unigram must map to at most one unigram in the other string). If more than one subset constitutes an alignment, and also has the same cardinality as the largest set, Meteor selects that set that has the least number of *unigram mapping crosses*. Intuitively, if the two strings are typed out on two rows one above the other, and lines are drawn connecting unigrams that are mapped to each other, each line crossing is counted as a *unigram mapping cross*.

After all modules have been run, precision $prec_i$ and recall rec_i are computed for the obtained alignment A_i (the maximal alignment with minimal unigram mapping crosses):

$$prec_i = \frac{size(A)}{|ngr_1(s_i)|} \qquad\qquad rec_i = \frac{size(A)}{|ngr_1(r_i)|}$$

where $size(A)$ is the number of mapped pairs in A. Then, $Fmean$ is computed:

$$Fmean_i = \frac{10 * prec_i * rec_i}{9 * prec_i + rec_i}.$$

To take into account longer matches, Meteor introduces the notion of *chunks* and uses them to compute score penalty. A *chunk* is a contiguous sequence of unigrams such that their counterparts in the other translation are also a contiguous sequence (in the correct order).[4] The penalty is then computed as follows:

$$Penalty = 0.5 * \left(\frac{chunks(A)}{size(A)}\right)^3$$

where $chunks(A)$ is the number of chunks in A. Finally, the METEOR Score for the given alignment is computed as follows:

$$Meteor = Fmean * (1 - Penalty)$$

For multiple reference translations, Meteor computes the above score for each reference translation, and uses the best score as the score for the sentence translation. The score of the whole candidate translation is calculated based on aggregate

[4] *Chunks* are equivalent to *runs* by GTM metric.

statistics accumulated over the entire test set, similarly to the way this is done in BLEU. We calculate aggregate precision, aggregate recall, an aggregate penalty, and then combine them using the same formula used for scoring individual segments.

In 2007, the authors modified the parameters that are used for the computation of Meteor. They put more weight on recall than before and use different parameters in the penalty formula:

$$Fmean_i = \frac{5 * prec_i * rec_i}{4 * prec_i + rec_i} \qquad Penalty = 0.28 * \left(\frac{chunks(A)}{size(A)}\right)^{0.83}$$

2.4.2 Semantic POS Overlapping

Semantic Part of Speech Overlapping metric is inspired by [7] which introduced a set of metrics using various linguistic features on syntactic and semantic level. One of the best performing metrics was *semantic role overlapping*. Since we did not find a tool that would assign semantic roles as defined in [7] to words in a Czech sentence, we decided to use a slightly different metric. The *TectoMT* framework [1] can assign semantic part of speech (semantic POS) to words. The set of semantic POS tags in *TectoMT* is depicted in Table 2.1.

The semantic POS overlapping for a given semantic POS type t is defined for a candidate sentence s_i and a set of reference sentences R_i as

$$overlapping(t) = max_{r_i \in R_i} \left[\frac{clip(s_i, r_i, t)}{count(r_i, t)}\right]$$

where t is a semantic POS type (as defined in Table 2.1), $clip(s_i, r_i, t)$ is the number of matched unigrams of type t between s_i and r_i (one-to-one mapping) and $count(r_i, t)$ the number of unigrams of type t in r_i.

The semantic POS overlapping score is averaged over all semantic POS types:

$$overlapping = \frac{1}{|T|} * \sum_{t \in T} overlapping(t)$$

where T is the set of semantic POS types.

To evaluate the whole MT system output, we compute aggregate counts at first and then compute overlapping for each type.

Semantic POS tag	**Description**
n.denot	denominating semantic noun
n.denot.neg	denominating semantic noun with which the negation is represented separately
n.pron.def.demon	definite pronominal semantic noun: demonstrative pronoun
n.pron.def.pers	definite pronominal semantic noun: personal pronoun
n.pron.indef	indefinite pronominal semantic noun
n.quant.def	definite quantificational semantic noun
adj.denot	denominating semantic adjective
adj.pron.def.demon	definite pronominal semantic adjective: demonstrative pronoun
adj.pron.indef	indefinite pronominal semantic adjective
adj.quant.def	definite quantificational semantic adjective
adj.quant.indef	indefinite quantificational semantic adjective
adj.quant.grad	gradable quantificational semantic adjective
adv.denot.ngrad.nneg	non-gradable denominating semantic adverb, impossible to negate
adv.denot.ngrad.neg	non-gradable denominating semantic adverb, possible to negate
adv.denot.grad.nneg	gradable denominating semantic adverb, impossible to negate
adv.denot.grad.neg	gradable denominating semantic adverb, possible to negate
adv.pron.def	definite pronominal semantic adverb
adv.pron.indef	indefinite pronominal semantic adverb
v	semantic verb

Table 2.1: Semantic POS types

Chapter 3

Quality of Metrics

All metrics described in the previous chapter were examined with respect to their correlation with human judgments. We wanted to find metrics that correlate best.

3.1 Test Data

The test data and human judgments were taken from the data used at the Third Workshop on Statistical Machine Translation [4]. We have chosen only systems and human judgments which had Czech as the target language.

The output of the following systems was considered:

- BOJAR - Charles University, Bojar [3]
- TMT - Charles University, TectoMT [1]
- UEDIN - University of Edinburgh [9]
- PCT - PC Translator (a commercial MT provider from the Czech Republic)

The test data consisted of two test sets. The first one contained a total of 90 articles which were selected from a variety of Czech, English, French, German, Hungarian and Spanish news sites. The other test set was drawn from Czech-English news editorials. The articles test set contained 2050 sentences and the editorials test set contained 2028 sentences. The reference translations contained only one human translation for each sentence.

The human judgments contained 243 system scores of 156 unique sentences[1] for the editorials test set and 267 system scores of 165 unique sentences for the

[1]The difference is due to the fact that some sentences were scored several times.

Human score	**Metric score**	**Human rank**	**Metric rank**
1	0.62	1.5	1
3	0.54	3	3
1	0.54	1.5	3
5	0.54	4	3

Table 3.1: Conversion of scores to rankings

articles test set. We considered human scores of the same sentence as independent of each other and included all of them in the ratings.

3.2 Correlation with Human Judgments

To measure the correlation of the metric ratings with the human judgments we used the Pearson correlation coefficient. This coefficient captures the extent to which two different rankings correlate with each other.

The human judgments contained scores of the translation quality on the scale 1 to 5, one being the best. It was possible that several translations obtained the same score. We transformed these human scores to rankings for each sentence. If several systems obtained the same score, we used the average position for each of them. In the case that all systems had the same score, we did not use the human judgment.

For automatic metrics, we computed the system scores on the sentence level and converted the scores to rankings in the same manner as for human judgments. Table 3.1 shows how we created the rankings.

In order to measure the correlation, we compared the metric rankings to the human rankings and calculated the Pearson correlation coefficient ρ using the equation:

$$\rho = \frac{n\left(\sum x_i y_i\right) - \left(\sum x_i\right)\left(\sum y_i\right)}{\sqrt{n\left(\sum x_i^2\right) - \left(\sum x_i\right)^2}\sqrt{n\left(\sum y_i^2\right) - \left(\sum y_i\right)^2}}$$

where n is the number of evaluated systems and x_i, y_i the position of the i^{th} system in the human and metric rank. The possible values of ρ range between 1 (all systems are ranked in the same order) and -1 (systems are ranked in the reverse order). Thus, an evaluation metric with a higher value for ρ reflects the human judgments better than a metric with a lower ρ.

	Editorials	**Articles**	**Average**
NIST	**0.258**±0.623	0.22±0.596	0.239
F-measure	0.23±0.626	**0.24**±0.582	0.235
GTM	0.23±0.626	**0.24**±0.582	0.235
Meteor	0.236±0.617	0.228±0.570	0.232
Meteor(orig)	0.228±0.622	0.233±0.566	0.231
PER	0.243±0.627	0.218±0.599	0.231
GTM(e=2)	0.219±0.629	0.237±0.582	0.228
WER	0.233±0.62	0.206±0.601	0.22
TER	0.233±0.62	0.206±0.601	0.22
SemPOS	0.192±0.608	0.215±0.571	0.204
BLEU	0.023±0.617	0.032±0.625	0.028

Table 3.2: Average sentence-level correlations for the metrics including standard deviation show that BLEU is not an accurate metric on the sentence level

Editorials	**Articles**
0.516±0.548	0.515±0.492

Table 3.3: Average inter-human correlations (standard deviation in brackets)

3.3 Results

The results of the experiment are depicted in Table 3.2. They show that the correlation of the metric evaluation with human judgments is not very high. The best metric on the news editorial test set was NIST. On the articles test set F-measure and GMT obtained the best correlation coefficient. The correlation between the BLEU metric and the human judgments suggests that there is no relation of the two rankings on the sentence level.

The correlation of automatic metric ratings with human judgments on the sentence level is not very high. This can be either because the metrics are not suitable to evaluate the quality of system outputs with Czech as the target language or the human judgments are themselves inaccurate. Therefore, we computed the correlation of human judgments with each other.

We took the human scores for sentences for which there were at least two human judgments and computed the Pearson's correlation coefficient for them. If there were more than two ratings of the same sentence, we considered all possible

	Editorials	**Articles**	**Average**
SemPOS	0.8	0.8	0.8
Meteor	0.8	0.4	0.6
Meteor(orig)	0.8	0.4	0.6
BLEU	0.4	0.4	0.4
NIST	0.4	0.4	0.4
F-measure	0.4	0.4	0.4
GTM	0.4	0.4	0.4
GTM(e=2)	0.4	0.4	0.4
PER	0.4	-0.2	0.1
WER	0.4	-0.4	0
TER	0.4	-0.4	0

Table 3.4: System-level correlations for the metrics

(distinct) pairs and used them to compute the coefficient. For the editorials test set, we obtained 63 pairs of human judgments and for the news articles test set 112 pairs. The results are shown in Table 3.3.

According to the results from Table 3.2, the best metric to evaluate MT systems on the sentence level is NIST, which has the highest correlation coefficient on the editorials test set. Other metrics that showed high correlation with human judgments are F-measure and GTM. Both of them are based on combination of precision and recall. The best metric from error rate metrics was PER which ranked on the shared 5^{th} place with the (original) Meteor metric. However, the difference in the correlation coefficients for the metrics (except for SemPOS or BLEU) is very low and does not provide any clear evidence which metric is better to distinguish the quality of the metrics on the sentence level.

We were surprised by the extremely low correlation of BLEU on the sentence level. Therefore, we measured the correlation of metrics with human judgments on the system level for both test sets. We computed the automatic metric system-level scores and converted them into rankings. For human judgments, we created the system rankings based on the *percent of time that a sentence (produced by the system) was better than or equal to the translations of any other system.* Then we computed the Pearson correlation coefficient ρ. Since there were no ties in the scores, the Pearson correlation coefficient is equivalent to the Spearman's rank correlation coefficient defined as:

$$\rho_{sp} = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where d_i is the difference between the ranks for $system_i$ and n is the number of systems.

Table 3.4 shows the Pearson correlation coefficients for both test sets on the system level. We can see that the Semantic POS Overlapping metric has the biggest correlation, followed by the Meteor metric. Moreover, the correlation coefficients on the system level are significantly higher for all metrics except for the distance metrics PER, WER and TER than on the sentence level.

However, the average correlation coefficients are based only on two test sets. Hence, we would need more test sets or use bootstrapping in order to get more accurate results.

If we compare our results with the correlation coefficients on the system-level that were published in [4], we can see that the results for Czech and English as the target language are very similar. Meteor and SemPOS (which is similar to Semantic Roles Overlapping (SR) metric from [4]) correlate the best with human judgments, while TER (mTER in [4]) has one of the lowest correlation coefficients. Even if the close relation of the results for Czech and English is obvious, we cannot make any exact conclusions because the values we obtained are based only on two test sets. Therefore, more experiments should be done to obtain conclusive results.

Chapter 4

MTrics Tool

4.1 Description

MTrics is a command-line tool which evaluates the quality of machine translation (MT) systems. The evaluation is based on comparison of MT system output to reference translation(s).

MTrics supports the most common metrics which are used in machine translation evaluation and provides some additional features, such as confidence intervals. Implemented metrics are:

- *BLEU* [13],
- *NIST* [5],
- *F-measure*,
- *Position-independent WER (PER)* [19],
- *General Text Matcher (GTM)* metric [20],
- *Word Error Rate (WER)* [17],
- *Translation Error Rate (TER)* [16],
- *Meteor* [2],
- *Semantic POS Overlapping.*

In some NLP literature, it is possible to encounter metrics like *mPER* or *mWER* where the letter *m* stresses that *PER* or *WER* consider multiple references. In *MTrics*, this is handled automatically. Depending on how many reference translations are provided for evaluation, *MTrics* computes either *PER* (single reference) or *mPER* (multiple references). The same applies to *WER*.

Mtrics is written in C++. It does not use any platform dependent libraries so it is possible to compile it under Unix/Linux as well as under Windows. However, Meteor and Semantic POS Overlapping metrics require the *TectoMT* framework [1] for computation of these metrics. This framework is available only under Unix/Linux and the authors do not intend to port in under Windows in the near future. Nevertheless, other metrics can be computed under Windows without any problems.

Figure 4.1 on page 33 shows the names of source files and classes that are included in these files. Classes in italics are abstract classes which define interface for derived classes.

4.2 Features

4.2.1 Confidence Intervals

Confidence intervals are one of the useful features *MTrics* supports.

Confidence intervals are used for interval estimation of a parameter. They indicate the reliability of an estimate. Confidence intervals, compared to point estimates, can express more information, because they do not only convey the information about the current estimate, but also an indication of the accuracy with which the parameter is computed.

Confidence intervals are constructed on the basis of a sample from a population. Given a sample $(X_1, \ldots, X_n)$, the observed outcome can be treated as a random variable X characterized by the unobservable parameter θ. The confidence interval is specified by two functions $u(X)$ and $v(X)$, such that

$$\mathrm{Pr}_{X;\theta}(u(X) < \theta < v(X)) = 1 - \alpha$$

The number $1 - \alpha, \alpha \in (0, 1)$ is called the *confidence level* or *confidence coefficient* and the interval estimate $(u(X), v(X))$ is denoted as $(1 - \alpha)$ confidence interval for θ.

In case of machine translation evaluation, the population consists of all possible translations by a machine translation system. We want to approximate the quality of the system based on the sample we were provided for evaluation. However, we have only one sample from the population of all possible translations. Therefore, some method must be used that creates additional samples. We use a method called *bootstrapping*.

Bootstrapping

Bootstrapping [6] is a statistical technique used to study the distribution of a random variable based on an existing set of values. This is done by randomly resampling with replacement (i.e. allowing repetition of the values) from the existing sample and computing the desired parameters of the distribution of the samples. It is required that the original sample is from an independent and identically distributed population.

The bootstrapping algorithm can be summarized as follows:

1. Given a sample $X = (X_1, X_2, \ldots, X_n)$ from a population P, generate N random samples of size n by drawing n values from the sample with replacement (each value having the probability of $1/N$).
2. The resulting population $\bar{P}$, denoted as $\bar{X} = (\bar{X}_1, \ldots, \bar{X}_N)$, with $\bar{X}_i = (\bar{X}_{i_1}, \bar{X}_{i_2}, \ldots, \bar{X}_{i_n})$, $i = 1..N$, constitute the N bootstrapped samples.
3. If the original estimator of a given population parameter was $\theta(X)$, with the bootstrapped samples we can calculate the same estimator as $\theta(\bar{X})$.

An important parameter of bootstrapping is N, the number of bootstrapped samples, or the number of times the process is repeated. This number should be large enough to build a representative number of samples, however not too large, which would negatively influence the computation time. Hence, a compromise has to be found. Values of $N = 200$ were shown to provide slightly biased estimations [6] so a larger N is preferred, for example $N = 1,000$ [6, 8].

A source of errors in inference statistics is that we use a particular sample to represent the whole (unknown) population. Therefore, the quality of confidence intervals is also influenced by the size of the sample which is being evaluated. The larger the original sample is, the better the confidence interval estimation.

4.3 Implementation Details

4.3.1 Flow Control

The main class that steers the flow of the whole program is the class *MTrics*. This class represents a framework into which specialized classes for input parsing or computing the metric scores can be plugged. The class *MTrics* provides functions for parsing command line parameters and functions that control the computation of metrics. The flow control can be divided into the following phases:

1. loading of candidate and reference translations,
2. processing of translations segment by segment and computing relevant data for each metric,
3. computing the final score for each metric,
4. output of the results.

In each phase, several classes are dominant and responsible for the computation. They are designed in a modular way so that they can be replaced or enhanced easily.

Translation Loading

Classes responsible for loading of the translations are derived from abstract classes *InputParser* and *SentenceTokenizer*. For non-semantic metrics, the class *InputParser* gets an input stream and extracts the translations from it. It uses class *SentenceTokenizer* which obtains a sequence of characters from *InputParser* and returns it divided into words. The words are stored in an instance of class *Translation*, whose only purpose is to store the translations in the first phase of computation, i.e. the translation loading phase. Semantic metrics require a special input file format (see User Documentation in Appendix A). The input loading is done in class *TMTInputParser* which calls a Perl script *tmt_to_mtrics.pl* in order to extract the necessary information from input files via the API of *TectoMT*.

The rules for the extraction of the individual words from a sequence of characters are implemented in classes derived from *SentenceTokenizer*: *WhiteSpaceTokenizer* and *MTevalTokenizer*. *WhiteSpaceTokenizer* assumes that all words are separated by white space,[1] e.g. blank space or tabulator. However, this behavior is not suitable in some cases. Punctuation marks should be regarded as separate words even if they directly follow a word, like the full stop at the end of a sentence. These rules are respected in *MTevalTokenizer*. The tokenization of words follows the rules which are used in the MT evaluation script *mteval* [12]. *MTevalTokenizer* implements the following rules:

- punctuation marks are tokenized (they are considered as separate tokens), except for hyphen (-), apostrophe ('), full stop (.) and comma (,)

[1]A sequence of white space is considered as one white space, thus no empty words can occur.

- if full stop (.) or comma (,) are not followed by a digit, they are also tokenized
- hyphen (-) preceded by a digit is tokenized
- sequences of digits separated by blank spaces are joined together

If other tokenization rules should be used, it is possible to implement a descendant of the class *SentenceTokenizer*. The only method to reimplement is `SentenceBlock * parseTranslation( string_T & str)`, which gets a string and returns a pointer to the class *SentenceBlock*.[2] While implementing a new child of *SentenceTokenizer*, implementation details of *MTrics* should be taken into account: If a new word is recognized, it is checked whether the same word has not been already loaded in the preceding text. If so, the old instance is returned. Otherwise, a new word is created. This is done in the class *WordStorage*. More about this optimization technique can be found in Section 4.3.2. However, if *TMTInputParser* is used to load the input files, no tokenization is performed since all words are already tokenized by the *TectoMT* framework.

After all input files are processed, the sentences are restructured and stored in `Metric::candidates` and `Metric::references`. The restructuring takes place in order to store translations of each source sentence[3] in one vector for easier manipulation during the computation of metrics' ratings. If Meteor of Semantic POS Overlapping metric are computed, the translations are also stored in `Meteor::meteorCandidates`, `Meteor::meteorReferences` for Meteor or in `LEMetric::leMetricCandidates`, `LEMetric::leMetricReferences` for Semantic POS Overlapping.

Metric Representation

The evaluation of the candidate translations is done in the second phase segment by segment. The central role in this phase is played by the descendants of abstract classes *Metric* and *MetricInfo*. Metrics are divided into five major classes: *NgramMetric*, *UnigramMetric*, *DistanceMetric*, *Meteor* and *LEMetric*. The purpose of this division is to cluster similar metrics and reuse computed values, if possible. Hence, it is possible to save computation time if more metrics are to be computed on the same input.

The class hierarchy of metrics is depicted in Figure 4.2 on page 34. The abstract class *Metric* defines an interface so that all metrics can be processed in a uniform way. The classes *NgramMetric*, *UnigramMetric*, *DistanceMetric*, *Meteor*

[2]This data structure stores the words of a sentence.

[3]This is the case if multiple candidate or reference translations are present.

and *LEMetric* are also abstract and they refine the interface in order to fit it to the needs of their descendants.

Metric ratings are not stored in descendants of class *Metric* directly. The children serve only as proxies to descendants of class *MetricInfo*, which reflect the hierarchy of *Metric*. The reason for this implementation is that *Metric* should serve only as enclosing box for lightweight storage classes that collect the necessary data for the metric computation. This is useful especially if confidence intervals are computed. A lot of instances of the storage class are needed[4] for each metric. Another reason is to have only one class that represents each of the metrics, if more MT systems are to be evaluated. This improves the readability of the code and gives the possibility to modify the code easier, if necessary.

Sentence Segment Evaluation

The most important part of the computation is performed in the function

```
void MTrics::updateMetrics(...).
```

In this function, ratings for each sentence segment are computed and updated for descendants of *NgramMetric*, *UnigramMetric* and *DistanceMetric*. *Meteor* and *LEMetric* are not updated in this function because they work with their own representation of sentences that cannot be shared with other metrics. Each of the three abstract classes *NgramMetric*, *UnigramMetric* and *DistanceMetric* has its own block of code, so that computed values can be reused for all metrics descending from the same abstract class. Nevertheless, *NgramMetric* and *UnigramMetric* share part of the code: The number of occurrences of n-grams is computed only once and than relevant information for descendants of *NgramMetric* and *UnigramMetric* is extracted.

The n-gram count is computed in the class *aho_corasick*,[5] an implementation of the Aho-Corasick search algorithm. The implementation uses TRIE to store the search automaton. Nevertheless, the edges in the TRIE are not stored in a vector with constant access to elements, but in the C++ STL class *Map*. This STL class has logarithmic complexity in the number of stored elements to find an item. Therefore, the complexity of parsing one sentence segment is $\mathcal{O}(log(n) * n)$, where n is the total number of words (see Section 4.3.2 for details on representation of words) in candidate and reference sentence. The pattern search automaton of Aho-Corasick algorithm is initialized with all unique n-grams from candidate

[4]One thousand instances is the default value.

[5]The implementation of the algorithm was provided by Jan Tattermusch [18].

sentence. Then, the candidate sentence is searched for the patterns in order to get the count of individual patterns. The same search is done for the reference sentence.

For each sentence segment, metrics' internal data are updated in function

`void updateSentenceLevelCount(...).`

This function has different parameters for *NgramMetric*, *UnigramMetric*, *DistanceMetric*, *Meteor* and *LEMetric*. It computes the necessary data for each metric and passes it on to

`void updateSentenceLevelMetricInfo(...).`

This function is also specific for each metric. It stores the data in a descendant of the class *MetricInfo*. If confidence intervals are to be computed, this is the place where data are distributed among the instances of the class *MetricInfo*.

After all sentence segments are processed, the total rating is computed for all required metrics. Data collected during the sentence segment processing are used to compute the final score.

4.3.2 Optimization

Computational efficiency is the main focus of *MTrics* so that large amounts of data can be evaluated quickly. *MTrics* is designed as a tool which supports various metrics and takes advantage of the simultaneous computation of them.

***Unigram-* and *NgramMetric*s**

Since the metrics belonging to these classes are based on comparison of n-grams found in candidate and reference translation(s), it is possible to reuse the computed values if several metrics are processed on the same input. The n-gram counts for each metric are extracted from the global n-gram counts computed by the Aho-Corasick algorithm which is computed only once.

***DistanceMetric*s**

The *WER* metric uses dynamic programming to compute the Levenshtein distance between two sentences. Algorithm 1 shows the pseudocode. Hence, the complexity for a sentence pair (s_i, r_i) is $\mathcal{O}(|s_i| * |r_i|)$.

In the case of *TER* metric, an optimal sequence of edits (with shifts) is very expensive to find. Therefore, a greedy search is used to select the set of shifts.

Algorithm 1 Calculate the Levenshtein distance

```
Input: Candidate sentence $s$, reference sentence $r$
Output: Levenshtein distance $d$
  $m \leftarrow |s_i|$
  $n \leftarrow |r_i|$
  $D$ is a table with m+1 rows and n+1 columns
  for $i = 0$ to $m$ do
    $D[i, 0] \leftarrow i$
  end for
  for $j = 0$ to $n$ do
    $D[0, j] \leftarrow j$
  end for
  for $i = 1$ to $m$ do
    for $j = 1$ to $n$ do
      if $s[i] = r[i]$ then
        $cost \leftarrow 0$
      else
        $cost \leftarrow 1$
      end if
      $D[i, j] \leftarrow minimum(D[i-1, j] + 1, D[i, j-1] + 1, D[i-1, j-1] + cost)$
    end for
  end for
  return $D[m, n]$
```

Moreover, several other constraints are used in order to further reduce the space of possible shifts.

- The shifted words must match the reference words in the destination position exactly.
- The word sequence of the candidate sentence s_i in the original position and the corresponding reference words must not exactly match.
- The word sequence of the reference that corresponds to the destination position must be misaligned before the shift.

Algorithm 2 shows the pseudo-code for calculating the number of *TER* edits for a candidate sentence s_i and reference sentence r_i. The minimum-edit-distance algorithm is $\mathcal{O}(n^2)$ in the number of words. We use beam search, which reduces the computation to $\mathcal{O}(n)$. Because the loop in Algorithm 2 can be iterated at most $|s_i|$ times, the complexity of the finding the minimal number of TER edits is $\mathcal{O}(n^2)$.

Algorithm 2 Calculate number of TER edits

```
Input: Candidate $s_i$, reference $r_i$
Output: Number of edits $E$
  $E \leftarrow 0$
  $s'_i \leftarrow s_i$
  repeat
    Find shift $S$ that reduces $MinEditDistance(s'_i, r_i)$ the most
    if $S$ reduces edit distance then
      $s'_i \leftarrow$ apply $S$ to $s'_i$
      $e \leftarrow e + 1$
    end if
  until No shifts that reduce edit distance remain
  $e \leftarrow e + MinEditDistance(s'_i, r_i)$
  return $E$
```

The implementation of *GTM* metric (see Algorithm 3) also takes advantage of some optimizations. To compute the maximum matching and the corresponding runs, greedy search is applied. The maximum matching is built iteratively by adding the largest non-conflicting aligned blocks to already computed matching. Experiments in [20] have shown that this approximation finds the true maximum matching 99% of the time. In the rare remaining cases, the size of the output matching is at least 80% of the maximum.

Meteor

Algorithm 4 depicts how an alignment between a candidate and a reference sentence is computed. The creation of the set P has the complexity of $\mathcal{O}(|s_i|*|r_i|)$. The subset P' is computed by a depth-first search. The depth-first search is bounded by the maximum number of steps which is set to 10000. Hence, it is possible that only a suboptimal solution is found for longer sentences.

Semantic POS Overlappling

The complexity of computing an overlapping for a candidate sentence s_i and a reference sentence r_i is $\mathcal{O}(|s_i| * |r_i|)$.

Word Representation

Source sentences in MT are mostly taken from a newspaper or a book and often speak about one topic. Many words occur more than once in the translations. So,

Algorithm 3 Calculate the maximal GTM matching

```
Input: Candidate $s_i$, reference $r_i$, maximum number of matched words $m$, exponent $e$
Output: Size of the matching $M$
  $M = \emptyset$; $size \leftarrow 0$
  Compute the set of runs $R$ for the pair $(s_i, r_i)$
  for $r \in R$ do
    if $|M| > m$ then
      break
    end if
    if $M \cap r$ is empty then
      add $r$ to $M$
      $size \leftarrow length(r)^e$
    end if
  end for
  return $\sqrt[e]{size}$
```

it is sensible to represent same words only with one object. The longer the translations are, the more storage space can be saved. Class *WordStorage* is responsible for keeping only one instance of each word. When translations are parsed, each word is checked, whether it has already occurred. If so, pointer to the old instance is returned. Otherwise, an instance of the new word is created and stored in class *WordStorage*. Another advantage of this approach is that words can be compared only on the level of pointers, which also speeds up the computation.

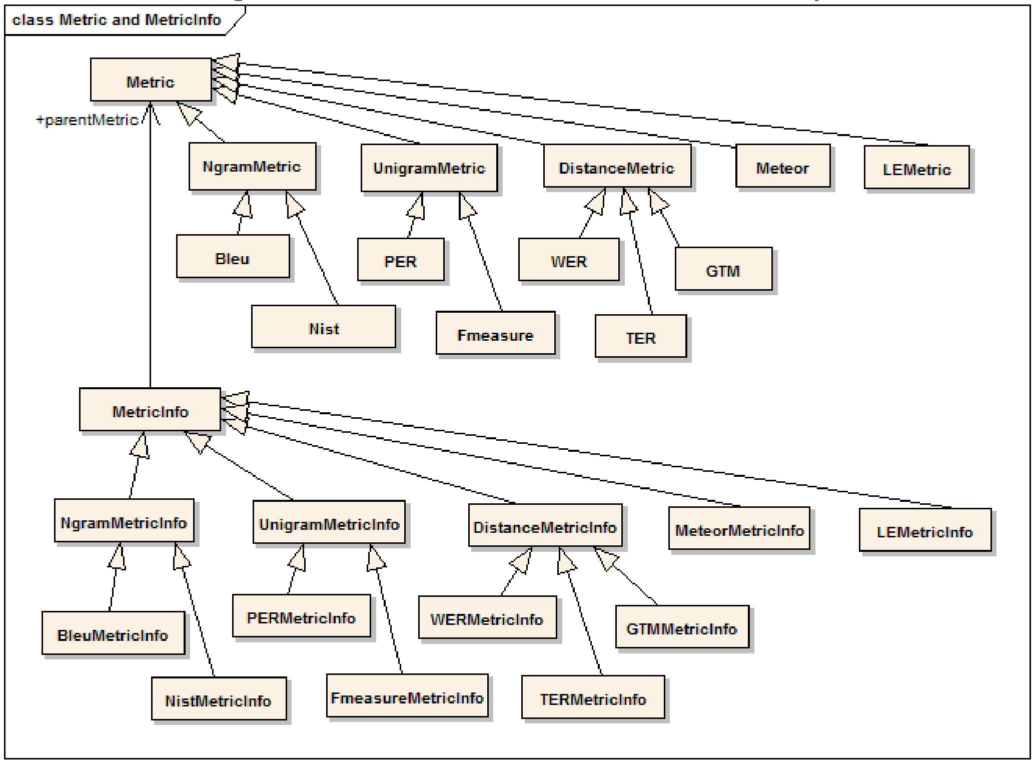
Algorithm 4 Calculate the maximal Meteor alignment

```
Input: Candidate $s_i$, reference $r_i$, vector of Meteor modules $M$
Output: Alignment $A$
  for $k = 1$ to $size(M)$ do
    compute the set $P$ of possible word pairs $(s_i[m], r_i[n])$ such that $s_i[m], r_i[n]$
      are not yet aligned and can be aligned according to module $m_k$
    select some $P' \subset P$ such that $P'$ is an alignment, $size(P') \geq size(P'')$,
      $P'' \subset P$ an alignment and $P'$ contains minimum unigram mapping crosses
    $A = A \cup P'$
  end for
  return $A$
```

File name	**Classes included**	**Description**
mtrics.h, mtrics.cpp	MTrics	Class containing the entry point, validation of parameters
metric.h, metric.cpp	*Metric*, *NgramMetric*, *UnigramMetric*, *DistanceMetric*, Nist, Bleu, PER, Fmeasure, WER, TER, GTM, MeteorWord, LEMetricWord, Meteor, LEMetric	Implementation of the MT metrics
metricInfo.h, metricInfo.cpp	*MetricInfo*, *NgramMetricInfo*, *UnigramMetricInfo*, *DistanceMetricInfo*, NistMetricInfo, BleuMetricInfo, FmeasureMetricInfo, PERMetricInfo, WERMetricInfo, TERMetricInfo, GTMMetricInfo, LEMetricInfo, MeteorMetricInfo	Classes storing metric data
inputParser.h, inputParser.cpp	*InputParser*, NormalInputParser, MTevalInputParser, TMTInputParser	Extraction of sentences from input files
sentenceTokenizer.h, sentenceTokenizer.cpp	*SentenceTokenizer*, WhiteSpaceTokenizer, MTevalTokenizer	Tokenization of sentences into separate words
translation.h, translation.cpp	Translation, SentenceBlock	Storage classes for translations
wordStorage.h, wordStorage.cpp	WordStorage	Storage class for words (flyweight design pattern)
ac.h	aho_corasick	Implementation of Aho-Corasick search algorithm
support.h, support.cpp	–	Support functions
typedef.h	–	Type definitions

Figure 4.1: Source files

Figure 4.2: Metric and MetricInfo class hierarchy



Chapter 5

Conclusion

5.1 Summary

This work has examined the most common MT system evaluation metrics that are currently used. The experiments have shown that the most suitable metrics for the system-level evaluation of MT systems with Czech as target language are Semantic POS Overlapping and Meteor, followed by BLEU and NIST. These results are consistent with data that were published for systems with English as target language.

The evaluation of metrics on the sentence level proved as unsuitable because of a relatively low correlation with human judgments and because it was not possible to distinguish the quality of metrics very well. We only found out that BLEU does not correlate with human judgments on the sentence level. However, the results were influenced by the quality of human judgments which had only moderate intra-human correlation.

We implemented a command line tool *MTrics* that was used to compute all metric ratings in this work. The tool supports computation of multiple metrics on the same input. For metrics requiring additional syntactic or semantic information, we used the *TectoMT* framework.

5.2 Future Work

More accurate results about the quality of MT metrics for Czech as the target language can be obtained if the experiments we have done on the system level would be repeated on more data or bootstrapped samples.

Moreover, other metrics that emerged recently can be implemented and evaluated. This concerns especially metrics that were published in [7]. Several of them show high correlation with human judgments for English. The TectoMT framework can provide most of the required features to compute these metrics for Czech sentences.

Appendix A

User Documentation

A.1 Synopsis

```
mtrics -c | --candidate candidateFile [candidateFile2 ... candidateFileN]
       -r | --reference referenceFile [referenceFile2 ... referenceFileN]
       [-o | --output outputFile]
       [-b | --bleu [ngramList]]
       [-n | --nist [ngramList]]
       [-p | --per]
       [-f | --fmeasure [P,R]]
       [-g | --gtm [exponent]]
       [-w | --wer]
       [-t | --ter]
       [-m | --meteor [orig]]
       [-sempos | --semantic-pos-overlap]
       [-T | --tokenize]
       [-M | --mteval-file-format]
       [-MT | --mteval-input]
       [-TMT | --tectomt-input [layer]]
       [-conf | --confidence-intervals [num]]
       [-cl | --confidence-level percent]
       [-s | --case-sensitive]
       [-v | --verbose]
       [-h | --help]
```

At least one of the metric parameters *-b*, *-n*, *-p*, *-f*, *-g*, *-w*, *-t*, *-m*, *-sempos* has to be specified. However, *-sempos* cannot be used at the same time with other metric parameters.

A.2 Options

Input and Output

- *-c | - -candidate candidateFile [candidateFile2 ... candidateFileN]*
  Parameter *-c* is mandatory. It is followed by a sequence of filenames containing candidate translation(s). Candidate translations must be translations of one source document, e.g. generated by various MT systems. It is not possible to score translations of different source documents at the same time. For more details on input format, see Section A.3.

- *-r | - -reference referenceFile [referenceFile2 ... referenceFileN]*
  Parameter *-r* is mandatory and is followed by a sequence of filenames containing reference translation(s). The order of reference files can be arbitrary. If several references are specified, all of them will be used in metric scoring process. The reference translations must have the same source document as the candidate translations.

- *-o | - -output outputFile*
  Write results to file *outputFile*.

Metrics

- *-n | - -nist [ngramList]*
  Parameter *-n* stands for *NIST* metric [5]. This metric uses n-grams for the computation of ratings. By default, unigrams up to 5-grams are used for the evaluation. Nevertheless, it is possible to choose only some n-grams. This can be done by appending the list of n-grams to parameter *-n*. The list consists of numbers or intervals separated by commas. Example 1 shows several examples how to use parameter *-n*.

  Example 1.
  ```
  mtrics ...-n 1-4 ...
  mtrics ...-n 3,1,5 ...
  mtrics ...-n 6,1-3,7 ...
  ```

  Note that it is possible to omit some n-grams like in the third example, in which 5-grams are not evaluated. The order of numbers or intervals can be arbitrary.

  The *NIST* score is in interval $(0, \infty)$. The higher the score, the better the machine translation.

- *-b | - -bleu [ngramList]*
  Parameter *-b* stands for the *BLEU* metric [13]. If no n-gram specification

follows the *-b* parameter, the metric uses unigrams to 4-grams by default. Otherwise, it is possible to choose n-grams in a similar way as for the parameter *-n*. For example, one can compute *BLEU* score of a translation using only unigrams and bigrams by specifying `-b 1-2` or `-b 1,2`.

The *BLEU* score lies in the interval $(0, 1)$. Higher score means better quality of the translation.

- *-p | - -per*
  The parameter *-p* specifies the *Position-independent Word Error Rate* (PER) metric [19]. This metric is a distance metric and ignores ordering of words within the sentence. It is defined as the minimum number of deletions, insertions and substitutions to transform the bag of words of the candidate sentence into the reference sentence words, normalized by the reference length.

  The *PER* score ranges between zero and one. Because *PER* is an error metric, the best score is zero.

- *-f | - -fmeasure [P,R]*
  F-measure is computed by *-f* parameter. The standard implementation combines *precision (prec)* and *recall (rec)* into one value according to the following expression: $\frac{prec+rec}{2*prec*rec}$, also known as the harmonic mean. *MTrics* allows to assign weights P and R to precision and recall. The rating is then computed by $\frac{P*prec+R*rec}{(P+R)*prec*rec}$. Both weights are specified after *-f*, separated by comma. If precision should have weight 9 and recall weight 1, the parameter would look like *-f 9,1*.

  The *F-measure* score ranges from zero to one. Higher score means better quality of the translation.

- *-g | - -gtm [exponent]*
  General Text Matcher (GTM) [20] is a metric based on precision and recall. Compared to plain *F-measure*, it takes word order into account by scoring contiguous sequences of words better. The optional parameter *exponent* determines the rate of preferring longer sequences of words. The default value is one.

  The score ranges from 0 to 1, one being the best score.

- *-w | - -wer*
  Word Error Rate (WER) [17] is another distance metric. It is defined as the Levenshtein distance [10] between the candidate sentence and the reference sentence, divided by the reference length for normalization. The Levenshtein distance is the minimum number of deletions, insertions and substitutions of single words to transform one sequence of tokens (words) into another.

Similar to *PER*, the score ranges between zero and one, zero being the best score.

- *-t* | *- -ter*
  Translation Error Rate (TER) metric [16]. This is a distance metric, similar to WER. Beside three basic edit operations (insertion, deletion and substution), it also allows movements of contiguous word sequences.

  The score is from the interval $\langle 0, 1\rangle$, zero being the best score.

- *-m* | *- -meteor [*orig*]*
  Meteor metric [2]. This metric uses *exact*, *lemma* and *WordNet synonymy* modules and can be used to compute score only for Czech sentences. If you use the optional string parameter *orig*, the score is computed according to the original paper [2], in which Meteor was introduced. Otherwise, different parameters (higher significance of precision and modified penalty) are used. These parameters are taken from the latest version of the evaluation script *meteor.pl*[1] provided by the authors of Meteor.

  This metric must be used with the *-TMT* parameter. The translations can be extracted either from the analytical or the morphological layer, which can be specified after the *-TMT* parameter. By default, the *TCzechA* layer is used. The selected layer must contain the *lemma* attribute for each word. This metric can be computed with other metrics, except for the Semantic POS Overlapping.

  The score is from the interval $\langle 0, 1\rangle$, one being the best score.

- *-sempos* | *- -semantic-pos-overlap*
  Semantic Part-of-Speech Overlapping metric inspired by [7]. This metric computes the ratio of matched words to the reference length with respect to their semantic POS type.[2] The score is the average of the ratios of all semantic POS types.

  This metric must be used with the *-TMT* parameter which specifies the exact name of the tectogrammatical layer from which the translations are to be extracted. The tectogrammatical layer must contain the *sempos* attribute for each word. It is not possible to compute this metric with any other metrics for the same input.

  The score ranges from 0 to 1, one being the best score.

[1]Version 0.6 from May 3, 2007 - `http://www.cs.cmu.edu/~alavie/METEOR/`

[2]Semantic POS types are taken from PDT 2.0 -
`http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/t-layer/html/ch05s03s01.html`

Input Parsing

- *-s* | *- -case-sensitive*
  Use case sensitivity. Default value is insensitive.

- *-T* | *- -tokenize*
  Use advanced tokenization of words. The tokenization rules are as follows:

  - punctuation marks are tokenized (they are considered as separate tokens), except for hyphen (-), apostrophe ('), full stop (.) and comma (,)
  - if full stop (.) or comma (,) are not followed by a digit, they are also tokenized
  - hyphen (-) preceded by a digit is tokenized
  - sequences of digits separated by blank spaces are joined together

  By default, words are separated by white-space characters.

- *-M* | *- -mteval-file-format*
  Use input files in *mteval* file format. For more details, see Section A.3.

- *-MT* | *- -mteval-input*
  This parameter can be used instead of specifying *-M* and *-T*.

- *-TMT* | *- -tectomt-input [layer]*
  Use input files in *TectoMT* file format. The optional parameter *layer* is a string that specifies the TectoMT layer from which the translations are extracted, e.g. *TCzechA* or *TCzechM*. The default layer is *TCzechA*. For more details, see A.3.

Confidence Intervals

- *-conf* | *- -confidence-intervals [num]*
  Confidence intervals (95% confidence level by default) are computed by parameter *-conf*. We use bootstrapping [8] to generate the necessary amount of samples from the candidate sentences. The default number of samples is 1000. The user can specify a different number of samples in order to reduce the computational costs or to increase the statistical significance of confidence intervals by writing the required number after *-conf*, like *-conf 5000*.

- *-cl* | *- -confidence-level percent*
  Set confidence level to *percent*. *Percent* must be a number from (0,1), e.g. 0.9 meaning the 90% confidence level.

Output Details

- *-v* | *- -verbose*
  Show detailed information about metric scores for all sentence segments.

Help

- *-h* | *- -help*
  Print out help on `STDOUT`.

A.3 Input format

MTrics requires all input files in UTF-8 encoding.

Plain Text

The default input file format is plain text. Every sentence must be on a separate line and all files must have exactly the same number of lines: on the ith line of each file must be the translation of the ith source sentence.

If a line is left blank, it is supposed that the translation of the line is missing, i.e. the line translation contains zero words. After the input files are processed, *MTrics* checks whether each translation contains the same number of sentences.

Names of the candidate files are used to identify the MT systems in output summaries.

mteval Input Format

In order to be compatible with other MT scoring programs, *MTrics* supports the input data format specified by the *National Institute of Standards and Technology* for the evaluation script *mteval* [12]. The translations are stored in SGML files with some additional information, like source and target language or MT system name. The script *mteval* requires three input files: source, candidate and reference documents. A simplified structure of the files is shown below. For detailed description of the file structure see the DTD specification in Appendix B.

- Source documents:

```
<srcset setid="[tst-set-name]" srclang="[src-lang]">
<DOC docid="[doc-name]">
```

```
<seg> ... </seg >
      .
      .
</DOC>
</srcset>
```

where

- *setid* is a unique identifier for the test set
- *docid* is a unique identifier for each document of the test set.

• Candidate documents (MT system output):

```
<tstset setid="[tst-set-name]" srclang="[src-lang]" trglang="[tgt-lang]">
<DOC docid="[doc-name]" sysid="[system-name]">
<seg> ... </seg >
      .
      .
</DOC>
</tstset>
```

where

- *setid* is the *same* unique identifier for the test set, as given in the source documents
- *docid* is the same unique identifier for each document of the test set
- *sysid* is the MT system identifier. This name will be used to identify the system in the scoring reports.

• Reference documents:

```
<refset setid="[ref-set-name]" srclang="[src-lang]" trglang="[tgt-lang]">
<DOC docid="[doc-name]" sysid="[system-name]">
<seg> ...  </seg >
      .
      .
</DOC>
</refset>
```

where

- *setid* is the reference set name
- *docid* is the same unique identifier for each document of the test set

– *sysid* identifies reference translations. Each reference translation should have its own identifier.

The data segments in a document are tagged with a segment tag <seg>. For example:

```
<seg> This is a sample language data segment. </seg>
```

With this scheme, each experiment is defined by the unique combination of a ref-set-name and a tst-set-name.

For the purposes of *MTrics*, the file with source sentences is not used, since it only serves to check whether an experiment is well-formed. It is supposed that all files comply with the file format specification. Their correctness is not explicitly checked. Nevertheless, if the tags are mismatched, an error occurs.

Here is an example of the structure of files containing data for evaluation:

Example 2 (Example of source, candidate and reference translation files).

```
<srcset setid="PCEDT_WSJ" srclang="Czech">
<DOC docid="docxxx" sysid="cz">
     <seg id="1"> Konsorcium soukromých investorů fungující jako
     LJH Funding Co. sdělilo, že dalo nabídku za 409 milionů dolarů
     v hotovosti na vetšinu holdingů v oblasti realit a nákupních
     center firmy L.J.Hooker Corp.</seg>
     <seg id="2"> Tato 409milionová nabídka zahrnuje také
     odhadovaných 300 milionů dolarů v zaručených závazcích na tyto
     nemovitosti, jak uvádí nabízející strana.</seg>
</DOC>
</srcset>

<tstset setid="PCEDT_WSJ" srclang="Czech" trglang="English">
<DOC docid="docxxx" sysid="exp_ugly_pako_union/f-dev.std.pbt.hyp">
     <seg id="1"> The private investors working as UNKNOWN_LJH
     Funding Co. said that could offer for 409 million dollars
     in cash for most UNKNOWN_holdingů in the area real and shopping
     centers firm L.J. Hooker Corp.</seg>
     <seg id="2"> The 409 million offer includes also an estimated
     $ 300 million in the guaranteed commitment to these real
     estate , according to the bidding party .</seg>
</DOC>
</tstset>

<refset setid="PCEDT_WSJ" srclang="Czech" trglang="English">
<DOC docid="docxxx" sysid="retran1">
```

```
    <seg id="1"> A group of private investors operating under the
    name LJH Funding Co. has announced that they have submitted a
    bid of $409 million in cash for the majority of L.J. Hooker
    Corp. holdings in the field of real-estate and shopping
    centers.</seg>
    <seg id="2"> This offer of $409 million also includes a
    estimated $300 million in secured bonds of this real estate,
    claimed the bidder.</seg>
</DOC>
</refset>
```

TectoMT Input Format

Meteor and *Semantic POS Overlapping* metrics require additional information about the translations like lemmata of the words or their semantic POS. This information can be obtained for a translation by the TectoMT framework [1]. This framework uses different layers in which the information about a translation is stored. The basic layers are

- word layer
- morphological layer
- analytical layer
- tectogrammatical layer.

For the purpose of *MTrics*, only the last three layers are supported. Morphological and analytical layer can be used for computation of all metrics except for *Semantic POS Overlapping. Semantic POS Overlapping* metric cannot be computed with other metrics for the same input and can use only the tectogrammatical layer.

The translations are extracted from the TectoMT files with ending *.tmt* by a perl script *tmt_to_mtrics.pl* which uses the TectoMT API. By default, the layer *TCzechA* is used for extraction but it is possible to use a different layer by specifying its name after the *-TMT* parameter. The layers are identified by the last letter of their name, e.g. *TCzechM* stands for the morphological level. The layer name must exactly match the layer in which the translation is stored in the .tmt file.

A.4 Output Format

The evaluation results are written to the standard output (`STDOUT`), by default. Another possibility is to output the results to a file specified after the `-o` parameter. The format of the output is plain text. Example 3 shows a sample output of *MTrics*.

Example 3 (Example of MTrics output).

```
MT evaluation run on Thu May 15 20:08:45 2008

Computation finished in 2 seconds

Parameters: -c cand.xml -r ref4.xml -MT -g -w

Total score:

f-dev.std.pbt.hyp.keepunk:      GTM:      0.7322
f-dev.std.pbt.hyp:              GTM:      0.6963
f-dev.std.pbt.hyp.dropunk:      GTM:      0.7409
f-dev.std.pbt.hyp.keepunk:      WER:      0.5373
f-dev.std.pbt.hyp:              WER:      0.611
f-dev.std.pbt.hyp.dropunk:      WER:      0.5352
```

Appendix B

DTD for *mteval*

```
<!DOCTYPE MTEVAL [

    <!ELEMENT MTEVAL -- (srcset|refset|tstset|DOC+)>
    <!ELEMENT (srcset|refset|tstset) -- (DOC+)>
    <!ATTLIST (srcset|refset|tstset) setid CDATA                #REQUIRED
                                     srclang (Chinese|Arabic)   #REQUIRED
                                     trglang (English)          #IMPLIED
    >

    <!--
        Files of type "srcset" contain source documents to be translated.
        Files of type "refset" contain reference translations to be used
        in evaluation.
        Files of type "tstset" contain output translations to be evaluated.
    -->

    <!ELEMENT DOC -- ((hl|p|poster|seg)*)>
    <!ATTLIST DOC docid CDATA #REQUIRED
                  sysid CDATA #IMPLIED
                  genre CDATA #REQUIRED
    >

    <!ELEMENT hl -- (seg*)>
    <!ELEMENT p -- (seg*)>
    <!ELEMENT poster -- (seg*)>
    <!ELEMENT seg -- (#PCDATA)*>
    <!ATTLIST seg id CDATA #IMPLIED>
]>
```

Bibliography

[1] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. TectoMT: Highly modular MT system with tectogrammatics used as transfer layer. In *In Proceedings of the Third Workshop on Statistical Machine Translation*, pages 167–170, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[2] S. Banerjee and A. Lavie. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization at the 43th Annual Meeting of the Association of Computational Linguistics (ACL-2005)*, pages 65–72, Ann Arbor, Michigan, June 2005.

[3] Ondřej Bojar and Jan Hajič. Phrase-based and Deep Syntactic English-to-Czech Statistical Machine Translation. In *In Proceedings of the Third Workshop on Statistical Machine Translation*, pages 143—146, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[4] Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. Further Meta-Evaluation of Machine Translation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 70–106, Columbus, Ohio, 2008. Association for Computational Linguistics.

[5] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, pages 138–145, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[6] B. Efron and G. Gong. A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. In *The American Statistician*, volume 37, pages 36–48. American Statistical Association, February 1983.

[7] Jesús Giménez and Lluís Màrquez. Linguistic Features for Automatic Evaluation of Heterogenous MT Systems. In *Proceedings of the Second Workshop*

on Statistical Machine Translation, pages 256–264, Prague, June 2007. Association for Computational Linguistics.

[8] P. Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*, pages 388–395, Spain, July 2004.

[9] Philipp Koehn, Abhishek Arun, and Hieu Hoang. Towards Better Machine Translation Quality for the German-English Language Pairs. In *In Proceedings of the Third Workshop on Statistical Machine Translation*, pages 139—142, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[10] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Technical Report 8, 1966.

[11] I. Dan Melamed. Automatic Evaluation and Uniform Filter Cascades for Inducing N-Best Translation Lexicons. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 184–198, Somerset, New Jersey, 1995. Association for Computational Linguistics.

[12] National Institute of Standards and Technology. MT evaluation software. http://www.nist.gov/speech/tests/mt/2008/scoring.html. Last visited on July 18, 2008.

[13] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, July 2002.

[14] Martin Porter. The Porter Stemming Algorithm. http://www.tartarus.org/martin/PorterStemmer/index.html. Last visited on July 16, 2008.

[15] M. Rajman and T. Hartley. Automatically predicting MT systems rankings compatible with Fluency, Adequacy or Informativeness scores. In *In Proceedings of the Workshop on Machine Translation Evaluation: "Who Did What To Whom"*, pages 29–34, Santiago de Compostela, Spain, 2001.

[16] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas*, pages 223–231, Morristown, NJ, USA, August 2006. The Association for Machine Translation in the Americas.

[17] K. Su and J. Wu. A New Quantitative Quality Measure for Machine Translation Systems. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 433–439, Nantes, France, July 1992.

[18] Jan Tattermusch. Implementation of Aho-Corasick Algorithm. www.atrac.cz. Last visited on September 18, 2007.

[19] Christoph Tillmann, Stefan Vogel, Hermann Ney, A. Zubiaga, and H. Sawaf. Accelerated DP Based Search for Statistical Translation. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 2667–2670, Rhodes, Greece, September 1997.

[20] Joseph P. Turian, Luke Shen, and I. Dan Melamed. Evaluation of Machine Translation and its Evaluation. In *Machine Translation Summit IX*, pages 386–393. International Association for Machine Translation, September 2003.