Confidence Estimation for Machine Translation

John Blatz, Erin Fitzgerald, George Foster, Simona Gandrabur,
Cyril Goutte, Alex Kulesza, Alberto Sanchis, Nicola Ueffing

March 29, 2004

Contents

List of Figures

List of Tables

Chapter 1

Introduction

All current NLP technologies make mistakes. Applications based on these technologies can tolerate mistakes if they have some reliable idea of when they are likely to be made. For instance, in a speech recognition dialog system, low confidence in the analysis of a user's utterance can lead the system to prompt for a repetition. This strategy has the potential to significantly improve the usability of the system, but it will be effective only to the extent that accurate estimates of correctness are available.

The binary classification problem of assessing the correctness of an NLP system's output is known as confidence estimation (CE). It has been extensively studied for speech recognition, which is arguably the most mature NLP technology, but is virtually unknown in other natural language applications. The motivation for our workshop was to apply CE techniques to another non-trivial area of NLP, measure their performance, and attempt to draw conclusions that would be applicable to the study of CE for NLP as a distinct topic of research.

Due to a strong interest in machine translation on the part of the workshop sponsors, as well as the existence of a parallel workshop on the topic to provide data, MT was a natural choice for our base NLP task. In retrospect, however, it was probably not an ideal choice, at least not given the aims stated in the previous paragraph. The infamous MT evaluation problem and the poor performance of our state-of-the-art base system created difficulties for the application of standard CE techniques, as described below, and led us into areas that were very specific to MT. Despite this, we were able to draw some conclusions about CE in general from our experiments. Our work also represents a concrete first step in the difficult but very pertinent problem of CE *for MT*.

In the rest of this chapter we first provide some background on confidence estimation, then describe the workshop activities, and finally give an outline of the report that follows.

1.1 Confidence Estimation

The goal of CE is to characterize the behaviour of a base NLP system that produces an output y given an input x. One way of doing so, which we will call *weak CE*, is to build a classifier that takes x and y as input and returns a score intended to be monotonic with probability of correctness. Decisions such as whether or not to prompt a user to repeat an utterance can then be based on thresholding this score, and the threshold can be adjusted so as to optimize performance in different situations.

A related approach, which we will call *strong CE*, is to return a direct estimate of the probability of correctness. This is slightly more flexible than weak CE, since in principle it eliminates the need to re-tune thresholds for different conditions in which the final system will be used. Provided the costs for different courses of action are known (and provided probability estimates are accurate), the optimal action for a given input pair can be determined dynamically from the probability of correctness using standard decision theory. Note that it is straightforward to convert a weak CE score into a probability estimate (see [15] for an example of this).

About half of the CE work reported in the speech recognition literature is devoted to each of these approaches. Both are typically evaluated according to how well they separate correct and incorrect examples, and strong CE is additionally evaluated according to the accuracy of its probability estimates. Section 2.5 gives details about the standard evaluation techniques, and further reading can be found in [40] and [22].

Another axis along which CE techniques differ is whether or not there is a separate "CE layer" distinct from the base NLP system. Many speech recognition approaches, eg [43], simply derive confidence scores directly from quantities in the base system. When the latter is probabilistic, these scores can be the probability estimates required for strong CE. Typically these are not the same scores used as search criteria to establish the output word sequence in the first place, but rather posterior probabilities $p(y|x)$ calculated from an nbest list or from a word lattice as in [20, 43].[1] A major advan-

[1]Note that $p(y|x)$ is not necessarily the same as $p(\text{correct}|x, y)$. For problems like speech recognition, in which there is only one right answer, the two expressions are equivalent. However, for problems like MT with potentially many right answers, $p(\text{correct}|x, y)$ is a

tage of this technique, apart from simplicity, is that it is unsupervised (with respect to confidence estimation).

Approaches in which the CE portion is separate from the base system predominate in the literature. This is a typical binary classification problem where a vector of features is extracted (the most useful features come from the base system), and a classifier is trained on a corpus of examples x, y labeled for correctness. The advantages of using a separate CE layer are mainly due to modularity. They include: separating the problem of determining the best output from that of assigning correctness, for which different feature sets and learning techniques may be optimal; allowing quicker and easier (partial) adaptation to new domains by re-training only a small CE layer rather than an unwieldy base system; and the potential for developing a CE infrastructure common to more than one base system or application. Systems using this approach have employed a wide range of machine learning algorithms, including Adaboost [6], naive Bayes [39], Bayesian nets [25], neural networks [13, 19, 42, 45], boosting [25], support vector machines [21, 45], linear models [12, 19, 25], and decision trees [19, 25, 26, 45].

Although by far the bulk of work on CE concerns speech recognition, there are a few recent efforts in other fields, such as information retrieval [23]. For MT, the only previous work is by members of our workshop. Ueffing et al [41] describe several methods, including posterior probabilities, for estimating the correctness of individual words in MT output. Gandrabur and Foster [11] describe the use of a neural-net CE layer to sharpen probability estimates for text predictions in an interactive translators' tool.

The primary purpose of confidence estimation is to enable broader and more effective deployment of imperfect NLP technologies, but the basic techniques have several other potential applications, especially in their strong (probabilistic) form. When a collection of different systems is available for some task, CE probabilities provide a principled and convenient way of combining their outputs, one that does not require the base systems to be statistical themselves. CE can also be used for active learning approaches, in which human annotation effort is targeted to those examples from which the model stands to benefit most [27]. Finally, confidence estimates can be used to calibrate the scores of partial hypotheses during search, leading to more accurate algorithms [26].

preferable formulation for CE, because it allows high probabilities of correctness to be assigned to many different hypotheses y simultaneously.

1.2 Overview of the Workshop

For MT output, there is a natural distinction between confidence estimation at the sentence level and at the subsentence level. Different techniques are appropriate to each case, and different applications can be envisaged for the resulting classifiers. Our workshop was organized along these lines. Because sentence-level CE seemed a more straightforward problem, we attacked it first and spent most of our time on it. Subsentence MT was not tackled until about halfway though the workshop, and was mostly the responsibility of one very hard working and competent team member. (In retrospect, our priorities should have perhaps have been reversed, since subsentence CE emerged as a more fruitful area for investigation.) The following sections give an overview of our approaches to both problems.

In both cases, our data was supplied by the *Syntax for MT* group, and was generated by the MT system described in [31]. For some experiments, we also obtained data from an MT system developed at CMU. Details of both corpora are given in section 2.1.

1.2.1 Sentence-Level Confidence Estimation

In sentence-level CE, the task is to determine whether MT output is correct for a given source sentence. This has many applications, including filtering translations for human post-editing or information gathering, combining output from different MT systems, and, possibly, reordering nbest lists output by the base system.

Since we wanted to investigate learning techniques, the first step for sentence-level CE was to assemble a corpus of source sentences and their machine-generated translations, then label each translation as correct or not. Two problems emerge here:

1. Performance is poor: the proportion of correct translations at the sentence level is very low. This makes learning and evaluation more difficult.

2. There is no completely satisfactory automatic method for determining whether MT output is correct or not at the sentence level, even if reference translations are available (as they were in our case). This makes it hard to assemble a reliable corpus of labeled examples.

Our solution to the first problem was essentially to move the goalposts. Instead of insisting that translations be completely correct, we required only that they be better than a given threshold according to an automatic metric. This let us control the number of positive examples by varying the threshold. The motivation behind this is that different thresholds will correspond to different levels of translation quality that might be useful in different applications. For instance, a very high threshold might correspond to human-quality text, a medium threshold might correspond to text that is sufficient for gisting purposes, and a low threshold might correspond to text that is sufficient for applications like cross-language information retrieval, which can make do with only a rough bag of words translation.

Changing the definition of correctness does not of course solve the problem that the automatic measures of MT quality we threshold against are unreliable. The methods that are currently popular are based on relatively crude word and ngram matches between generated and reference translations. These are reasonably stable for entire texts, but exhibit high variance at the sentence level. To quantify the variance, and to be able to justify our choice of a metric to threshold against, we ran our own MT evaluation exercise. This produced a corpus of approximately 600 sentence pairs with human-assigned correctness ratings. Although we did not solve the fundamental problem of automatic MT evaluation, this study gave us insight into it, and let us bound the error in our gold-standard correctness assignments.

1.2.2 Subsentence-Level Confidence Estimation

The aim in subsentence-level CE is to tag individual words or ngrams in MT output. This has potential application in an interactive postediting environment, where a translator selectively replaces portions of a machine-generated translation. Another intriguing possibility is to use confidence scores as a basis for recombining parts of alternate sentence hypotheses in an nbest list or output from different MT systems.

Subsentence CE has neither of the problems described above for sentence-level CE. Although entire sentences are only very rarely correct, parts of sentences are much more likely to be correct, and hence the learning problem is more meaningful. Also, whereas an exact match to a reference translation is a far too stringent condition at the sentence level, it is quite reasonable for words or ngrams. This eliminates the noise inherent in the sentence-level correctness tagging. Subsentence CE is not without problems, however, because it is not completely clear what it means for parts of a translation to be correct. For instance, if only one bigram in a machine-generated translation matches reference translation 1, but three separate unigrams match reference translation 2, should we tag the bigram or the unigrams as correct? Should this decision depend on whether the unigrams occur in the same order as in reference 2, or in similar positions? Standard MT evaluation metrics embody answers to these questions, and we explore various alternatives in chapter 4.

1.3 Outline of the Report

The rest of the report is structured as follows:

- Chapter 2 describes the experimental setting, giving details of the corpora we used, our basic method(s) for assigning confidence scores and probabilities, some practical problems we encountered, and the techniques we used to evaluate results.

- Chapter 3 concerns our experiments on sentence-level CE, including the features used for machine learning, evaluations of classification performance, and the results of some experiments to test potential applications.

- Chapter 4 deals with our experiments on subsentence-level CE, including the features used, the different methods for tagging words and ngrams as correct or not, and the evaluation of the resulting classifiers. Due to the way the workshop was organized, this chapter can to a certain extent be read independently from the rest of the report.

- Chapter 5 describes the MT evaluation exercise we carried out.

- Chapter 6 summarizes our results and contains some pointers for future work.

Chapter 2

Experimental setting

In this chapter, we introduce the experimental setting of our work on confidence estimation for machine translation.

In section 2.1, we give details about the different corpora we used, and describe how we derived the data that we needed for our task from these corpora.

Section 2.2 sets out our approach to confidence estimation, and in particular explains how we cast it as a machine learning problem. We describe the automatic evaluation measures used to assign correctness to individual translation hypotheses, and address the scaling problem that arises when these measures are calculated from differing numbers of reference translations.

We then briefly present in section 2.3 the two machine learning methods used in our experiments: naive Bayes and the multi-layer perceptron. Section 2.4 gives details on how we solved a number of practical problems related to handling very large datsets with potentially tens of millions of examples.

Finally, we close the chapter with a description in section 2.5 of the metrics used to evaluate performance on the confidence estimation task, in both its strong and weak variants, and a discussion of some of their properties. We also discuss the bootstrap technique used to derive error bars on these statistics.

2.1 Corpora

In order to learn confidence estimation for machine translation, we gathered data from two MT systems from ISI and CMU that participated in the

Chinese-to-English track of the 2003 NIST evaluation. These systems rely on different technology and are described in the proceedings of the evaluation [30]. Both systems are trained on a very large training set. For our purpose, we considered the output of these systems on various collections of Chinese source sentences:

1. 993 sentences from the evaluation set of the 2001 NIST competition, each with 4 reference translations.

2. 4107 sentences from an additional multi-reference LDC corpus, each with 1 to 4 reference translations.

3. 565 sentences from the same multi-reference LDC corpus, each with 4 reference translations.

4. 878 sentences from the evaluation set of the 2002 NIST competition, each with 4 references.

In addition, the evaluation set of the 2003 NIST competition was available. It was meant to be used as a final "blind" test set, but was actually never used during the workshop.

The two candidate systems produced a list of most probable translations for each sentence, the so-called *N-best list*. Each N-best list contains between 101 and 16384 hypothesis translations for the same source sentence. For the ISI system we obtained N-best lists for all sentences, while for the CMU systems, we obtained N-best lists for only the 993 sentences from the 2001 NIST evaluation and the 878 sentences for the 2002 NIST evaluation. In addition, the size of the lists was limited to 1000 hypotheses for the CMU system.

The available data was split into three datasets used to carry out our machine learning experiments:

- The *training set* is used to estimate the parameters of our models. For the ISI system, the training set consisted of the N-best lists from the first two collections, amounting to 993+4107 = 5200 N-best lists. For the CMU system, the training set consisted of the first 700 N-best lists from the first collection.

- The *validation set* is used to tune some hyper-parameters of the learning process, such as the model structure, number of iterations of the training procedure, etc. For the ISI system, we used the 565 N-best lists from the third collection as validation set. For the CMU system, we used the last 293 sentences from the first collection.

- The *test set* is used to evaluate the final performance of the model, after training and optimisation. For both systems, the test set consisted of the 878 N-best lists from the fourth collection. Note that after tuning the hyper-parameters of the learning process on the validation set, it is possible to re-train the model on the combined training+validation sets.

In the context of our work, we would like to estimate the confidence in each proposed translation (at the sentence level). Using the N-best list, each source sentence therefore potentially generates between 101 and 16384 examples. In general, we considered these to be *independent* examples, but took some steps to try to compensate for this rather strong assumption, as described in section 2.2.3.

2.2 Automatically Identifying Correct Translations

As explained in the introduction, confidence estimation in MT is concerned with determining whether a given machine-generated translation is correct or not. In general, we make this judgment by analyzing the source text, the target hypothesis, and possibly extra information related to the translation task.

To formalize the CE task, we introduce a feature vector $\mathbf{x}$ representing an hypothesis, and a binary variable c indicating whether the translation is correct or not. The features in $\mathbf{x}$ may capture different aspects of the translation and will be detailed later, in the chapters related to the actual experimental results. In particular, they differ for sentence-level and subsentence-level confidence estimation. In general, for a given example (s, h) consisting of a hypothesis translation h for source sentence s, features in $\mathbf{x}$ may depend on s and/or h, but may not depend, for example, on the reference translation for s (which would be unavailable at testing time).

Our basic technique is to define a parameterized function $f(\mathbf{x}; \theta)$ (where θ is a parameter vector) intended to be correlated with the true probability of correctness $P(c = 1|\mathbf{x})$. This is very general: once we have such a function, we can make correctness decisions by comparing $f(\mathbf{x}; \theta)$ to a threshold, which can be set appropriately for any particular application. We experimented with three different ways of establishing f:

1. Using a single component of the input vector $\mathbf{x}$ (θ is empty in this approach).

2. Doing regression against an MT evaluation metric.

3. Directly estimating the probability of correctness, so that:
$$f(\mathbf{x};\theta) = \widehat{P}(c = 1|\mathbf{x};\theta), \tag{2.1}$$
where $\widehat{P}(c = 1|\mathbf{x};\theta)$ is an estimate of the true probability of correctness.

The first approach is very straightforward, but it is widely and successfully used in speech recognition—note that the components of $\mathbf{x}$ can themselves be sophisticated quantities such as posterior probabilities extracted from the base system, as in [41]. The second and third methods are standard supervised machine learning problems, requiring a labelled data set for training as described in the next paragraph. The first two methods are applicable only to weak CE; the final method is applicable to both the weak and strong CE scenarios.

In order to learn the parameters θ for probability estimation, we need a dataset of input-output examples $\mathcal{D} = \{(\mathbf{x}^{(i)}, c^{(i)})\}_{i=1\ldots n}$, where $\mathbf{x}^{(i)}$ is an input example (vector of features) and $c^{(i)}$ is the corresponding label (ie correctness). Of course, the actual correctness c of MT output is usually unknown for large datasets. A very restrictive way of defining it would be to decide that a sentence is correct only if it matches one of the reference translations. However this is overly strict: too few sentences would pass this test, and potentially many correct sentences would not (in particular, reference sentences would usually not pass when tested against other references). Our solution is to rely on automatic MT evaluation scores, generated using a set of reference translations, to provide the necessary labels $c^{(i)}$:
$$c^{(i)} = \begin{cases} 1, & E(\mathbf{x}^{(i)}, R^{(i)}) \geq \tau \\ 0, & \text{else,} \end{cases}$$
where $E(\mathbf{x}, R)$ is an automatic evaluation score for $\mathbf{x}$, based on the correspondence between the translation hypothesis it contains and a set of reference translations R. The correctness threshold τ is set as described below. To do regression, we learn parameters so as to make $f(\mathbf{x};\theta)$ a direct approximation of $E(\mathbf{x}, R)$ (independent of any particular threshold on E for deciding correctness).

2.2.1 MT Evaluation Measures

In our experiments on sentence-level confidence estimation, we estimate the correctness by thresholding one of the following measures:[1]

[1]Chapter 4 describes the methods for estimating correctness for *sub*sentence-level confidence estimation.

WERg: Word Error Rate, normalised by the length of the Levenshtein alignment.

NIST: Sentence-level NIST score, ie weighted average of n-gram precisions.

These two measures were chosen because they were best correlated to human judgements in the evaluation exercise that we carried out during the workshop. More details on this evaluation exercise and on automatic MT evaluation measures will be provided in chapter 5, together with some analysis of their correlation with human judgement.

WERg provides, for each hypothesis translation h, a score between 0 and 1. 0 is perfection, in that it indicates that the hypothesis h is identical to one of the reference translations. 1 means that the hypothesis h has basically no word aligned with either of the reference translations. For WERg, we threshold such that hypotheses with a WERg *below* the threshold are considered correct ($c^{(i)} = 1$) and hypotheses with a WERg *above* the threshold are considered incorrect ($c^{(i)} = 0$).

NIST provides, for each hypothesis translation h, a positive score. A NIST score of 0 means that the hypothesis h and the reference translations have essentially no n-gram in common (the worst score), while higher positive scores suggest better translations. For NIST, we threshold by assigning $c^{(i)} = 1$ to hypotheses h with NIST scores *above* the threshold, and $c^{(i)} = 0$ to hypotheses h with NIST scores below the threshold.

By varying the threshold, we can impose more or less stringent conditions on correctness, and vary the proportion of examples labelled as positive. For both measures, we considered several thresholds. In particular, we experimented with labels obtained by thresholding such that 30% and 5% of the examples (in the training corpus) are labelled as correct.

2.2.2 Handling Multiple References

With the thresholding strategy described above, the values of the scores are treated on the same scale for all sentences. If the threshold of the NIST score is 6, all sentences that have a NIST score above 6 will be considered correct, regardless of the source sentence, number of references, etc.

However, the way automatic evaluation scores are calculated for multiple references introduces a systematic bias with the number of reference translations. The WERg requires a Levenshtein alignment of the translation with the reference(s). When there are more references, there is better chance of finding correct alignments. The NIST score is a weighted average of n-gram

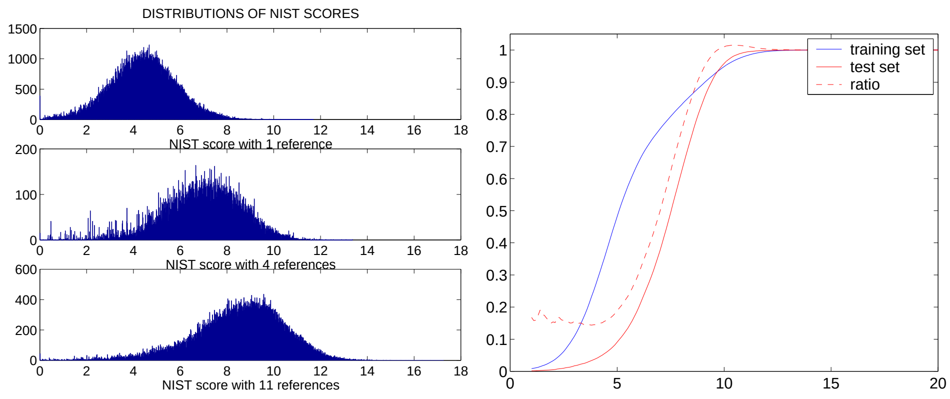


Figure 2.1: Left: distribution of NIST scores for sentences with 1 (top), 4 (middle) and 11 (bottom) reference translations. Right: Proportion of examples tagged correct in the training and test set, as a function of the threshold.

precisions. When there are more references, there is a wider choice of n-grams and therefore a better chance of finding a match between n-grams from the proposed translation and those from the reference translations.

This is illustrated in figure 2.1 where we present the distribution of the NIST score for sentences with various number of reference translations. Part of the training data (the 2001 NIST evaluation data) actually has up to 11 reference translations, although we only retained 4 of them in our work. In this comparison, however, we consider all these references. We therefore have many sentences with only 1 reference, around 1000 with 4 references, and around the same number with 11 references. Figure 2.1 (left panel) shows that the distribution of the NIST score varies greatly with the number of reference translations:

- For sentences with 1 reference (top), the NIST score varies roughly from 0 to 8, with an average about 4.5;
- For sentences with 4 references (middle), the NIST score varies roughly from 2 to 10, with an average about 7;
- For sentences with 11 references (bottom), the NIST score varies roughly from 4 to 12, with an average about 9.

With a threshold of 6, the result will be that almost all sentences with 1 reference are labelled incorrect, while most of the sentences with 4 references and almost all the sentences with 11 references will be considered correct. We would therefore introduce in the labelling of the examples a systematic bias depending not on the quality of the translation, but simply on the number of references translations. One side effect is an inconsistent labelling between the training set, where the number of reference translation is variable, and the test set, where there are always 4 reference translations. This is exemplified on the right panel of figure 2.1, where we plot the percentage of incorrect examples in the training (blue) and test (red) sets. Because most of the training sentences have only one reference translation, thresholding at a NIST score of 6 yields around 70% incorrect examples (30% correct), while on the test set (red curve), there are less than 15% incorrect examples, and more than 85% correct. The test and training sets are therefore very different, and this will typically lead to biased results.

In order to compensate for this, we adopt the following strategy:

- For sentences with 11 reference translations, we used only the first 4.
- For sentences with only 1 reference translation, we rescale the scores with a two-parameter affine transformation such that the distribution of the rescaled scores is similar to the distribution of the scores of the sentences with 4 reference translations.

This is done for both NIST score and WERg (separately), and thresholding is done after rescaling. As a consequence, the proportions of correct and incorrect examples on the training and test sets are similar, and we therefore expect to limit the bias introduced by the varying number of references.

2.2.3 Feature Normalization

The standard supervised machine learning framework assumes a set of independent labelled examples. Our examples potentially violate this assumption because they are grouped into N-best lists, one per source sentence. Typically, features that depend on the source sentence will be related or even identical. Even hypothesis-specific features may be related, eg the length of the target hypotheses in the same N-best list will be related. In order to compensate for this, we tried two kinds of feature normalisations. Features that are believed to have no dependency on the specific source sentence are normalised globally in order to have 0 mean and unit variance. Features that are believed to be very dependent on an N-best list, and to typically have larger between-sentence variance than within-sentence variance, are normalised on a source-sentence basis: for each N-best list, the feature is normalised to have 0 mean and unit variance. Note that this implies that the global mean is also 0.

2.3 Machine Learning Techniques

In this section, we describe the machine learning techniques used in our experiments. We used naive Bayes models for probability estimation, and multi-layer perceptrons for both probability estimation and regression.

2.3.1 Naive Bayes

To derive the naive Bayes model, we denote the class variable by c as before ($c = 1$ for a correct example, and $c = 0$ for an incorrect example). Each example is represented by a D-dimensional vector of discrete features $\mathbf{x}$. The class posteriors can be calculated via the Bayes rule as

$$P(c|\mathbf{x}) = \frac{P(c)\,P(\mathbf{x}|c)}{\sum_{c'} P(c')\,P(\mathbf{x}|c')}. \tag{2.2}$$

The key assumption in this model is that the features are statistically independent, so that:

$$P(\mathbf{x}|c) = \prod_{d=1}^{D} P(x_d|c).$$

Given a sample of N training examples $\{(\mathbf{x}^{(i)}, c^{(i)})\}_{i=1\ldots N}$, the maximum likelihood estimate of the unknown probabilities are the empirical frequencies

$$\begin{aligned} P(c) &= \frac{N(c)}{N} \\ P(x_d|c) &= \frac{N(x_d, c)}{N(c)} \qquad\qquad d = 1, \ldots, D \end{aligned} \tag{2.3}$$

where the $N(\cdot)$ are suitably defined event counts. $N(c)$ is the number of training examples in class c, and $N(x_d, c)$ is the number of examples with feature value x_d in class c.

The maximum likelihood estimates give a value of 0 to the conditional probability of features that do not appear in the training set for class c. This can prove harmful as new document that have this feature will automatically be given a probability $P(\mathbf{x}|c) = 0$, regardless of the other features they may contain.

To prevent null estimates, we have considered an *absolute discounting* smoothing model imported from statistical language modelling. The idea is to discount a small constant $b \in (0, 1)$ to every positive count and then distribute the gained probability mass (with a uniform backoff) among the null counts (unseen events). Thus, for each class, if $N(x_d, c) = 0$ for one or more possible values of x_d, let us denote by N_+ the number of features x_d with $N(x_d, c) > 0$ and N_0 the number of features x_d with $N(x_d, c) = 0$. The probability estimate (2.3) becomes

$$P(x_d|c) = \begin{cases} \dfrac{N(x_d, c) - b}{N(c)} & \text{if } N(x_d, c) > 0 \\ \dfrac{b}{N(c)} \dfrac{N_+}{N_0} & \text{if } N(x_d, c) = 0 \end{cases}$$

Once the model has been trained this way, conditional probabilities of correctness are obtained from equation (2.2).

In practice, some features may have continuous rather than discrete domains. In that case, the domain is discretised into a fixed number of bins (usually around 20). The discretisation is performed in a semi-automatic manner by setting a minimum and maximum value for each feature. The range is split in a number of evenly-spaced bins of fixed size. The minimum, maximum and bin size are set by visual inspection of the histograms of the features of the examples from the correct and incorrect classes. Given this information, our naive Bayes implementation includes a function that maps the continuous feature value x_i to the corresponding discrete bin number. Then the probability estimation procedure is used as explained above on the discretised features.

2.3.2 Multi-Layer Perceptron

The multi-layer perceptron (MLP, cf. [4]) is a generalisation of linear models obtained by stacking layers of perceptrons. In our work we use only one hidden layer, giving a discriminant function for an input $\mathbf{x}$ of the form:

$$f(\mathbf{x}; \theta) = s\left(\mathbf{v}.h(\mathbf{W}.\mathbf{x})\right) \tag{2.4}$$

where $\theta = \{\mathbf{W}, \mathbf{v}\}$, $\mathbf{W}$ is a matrix of *input layer weights*, and $\mathbf{v}$ is a vector of *output layer weights*. $h(\cdot)$ is a transfer function which non-linearly transforms the linear combination of inputs $\mathbf{W}.\mathbf{x}$. The same transfer function is used for all "hidden units" j. $s(\cdot)$ is an activation function: for regression just the identity function; and for probability estimation the so-called "softmax" function, similar to logistic regression [18], which transforms the unconstrained MLP output into the probability estimate $\widehat{P}(c = 1|\mathbf{x}; \theta)$.

Parameter estimation, aka *training*, is done by minimising the empirical loss using a stochastic gradient descent. The empirical loss is:

$$C(\theta) = \sum_i \ell(\mathbf{x}^{(i)}, E(\mathbf{x}^{(i)}, R^{(i)}))$$

(recall that E is the MT evaluation score used to define the correctness of $\mathbf{x}$). For regression, we use the standard squared-error loss $\ell(\mathbf{x}, E(\mathbf{x}, R)) = (f(\mathbf{x}; \theta) - E(\mathbf{x}, R))^2$. For probability estimation, we use negative log-likelihood: $\ell(\mathbf{x}, c) = -\log(cP(c = 1|\mathbf{x}) + (1 - c)(1 - P(c = 1|\mathbf{x})))$, where c is derived by thresholding E as described in section 2.2.

In stochastic gradient descent, examples are presented one at a time, the output is calculated, and the gradient of the loss for this example is calculated using *back-propagation*, which is essentially a convenient way to derive the gradient of the loss wrt the parameters using the structure of the model. The parameters are then updated incrementally by moving them slowly in the direction of the gradient:

$$\widehat{\theta} \leftarrow \widehat{\theta} - \eta \nabla_\theta \, \ell(\mathbf{x}^{(i)}, E(\mathbf{x}^{(i)}, R^{(i)}))$$

where η is a positive *learning rate*.

In order to ensure good theoretical as well as practical convergence of the stochastic gradient descent, it is extremely important that the examples should be presented in random order. This issue will be addressed below. It should be noted that stochastic gradient descent offers no guarantee against local minima, but in practice it may actually converge quite fast towards a minimum, especially when the data is very redundant. Considering that in our confidence estimation problem, many features are very redundant, especially for hypotheses relating to the same source sentence, stochastic gradient descent seems an attractive learning strategy.

To implement MLP training, we used the neural network on-line algorithm provided by the free machine learning library Torch [7], with some modifications detailed below.

2.3.3 From Maximum Entropy to Multi-Layer Perceptrons

In this section we note some similarities between the popular Maximum Entropy technique [3, 36], the Multi-Layer Perceptron and a particular case of MLP, the *Single-Layer Perceptron* (SLP), when used for probability estimation.

All these models involve a log-linear combination of feature functions. In Maximum Entropy, the posterior class probability is given by:

$$P(c|\mathbf{x}) \propto \exp\left(\sum_m \lambda_m f_m(c,\mathbf{x})\right) \tag{2.5}$$

where $f_m(c,\mathbf{x})$ are the feature functions, which may depend on the classes.

A Single-Layer Perceptron equipped with a "softmax" layer essentially implements logistic regression. For each M-dimensional input vector $\mathbf{x}$, the posterior class probability is given by:

$$P(c|\mathbf{x}) \propto \exp\left(\sum_{i=1}^{M} w_i^c.x_i\right) \tag{2.6}$$

Where $\mathbf{w}^c$ is the M-dimensional weight vector for class c. There are some interesting relationships between MaxEnt and SLP. MaxEnt uses a single parameter vector $\boldsymbol{\lambda}$, but potentially richer features. When both models are trained by maximum likelihood, MaxEnt can actually simulate a SLP by choosing a specific set of features. With C classes $c = 1 \ldots C$, let us define the $C \times M$ dimensional vector:

$$\mathbf{f}(c,\mathbf{x}) = [\,\underbrace{0,\ldots 0}_{(c-1)\times M}, \mathbf{x},\ \underbrace{0,\ldots 0}_{(C-c)\times M}\,]$$

Then the maximum likelihood parameters $\boldsymbol{\lambda}$ are

$$\boldsymbol{\lambda} = [\mathbf{w}^1, \mathbf{w}^2, \ldots \mathbf{w}^C]$$

So MaxEnt may be used to train and implement a SLP. The reverse is also true to some extent. Notice that in Maximum Entropy, the feature functions are arbitrary, but fixed during the learning process. Equation 2.6 uses directly $\mathbf{x}$ as input, but in fact $\mathbf{x}$ usually contains features calculated from the data. Using any fixed feature expansion $\boldsymbol{\Phi}(\mathbf{x})$ instead of $\mathbf{x}$ is therefore equivalent. In that case, $P(c|\mathbf{x}) \propto \exp(\mathbf{w}^c.\boldsymbol{\Phi}(\mathbf{x}))$, where the weight vector $\mathbf{w}^c$ (for class c) now has a dimension that matches the feature space. In order to transform a MaxEnt model (2.5) with F feature functions $\mathbf{f}(c,\mathbf{x}) = [f_1(c,\mathbf{x}), f_2(c,\mathbf{x}), \ldots f_F(c,\mathbf{x})]$ into a SLP (eq. 2.6), let us define the following feature vector:

$$\boldsymbol{\Phi}(\mathbf{x}) = [\mathbf{f}(1,\mathbf{x}), \mathbf{f}(2,\mathbf{x}), \ldots \mathbf{f}(C,\mathbf{x})]$$

A SLP with parameter vectors

$$\mathbf{w}^c = [\,\underbrace{0,\dots 0}_{(c-1)\times F},\ \boldsymbol{\lambda},\ \underbrace{0,\dots 0}_{(C-c)\times F}\,] \tag{2.7}$$

will then exactly correspond to the MaxEnt model, as $\mathbf{w}^c.\boldsymbol{\Phi}(\mathbf{x}) = \boldsymbol{\lambda}.\mathbf{f}(c,\mathbf{x})$. Note however that simply training a SLP on $\boldsymbol{\Phi}(\mathbf{x})$ will *not* result in a model equivalent to MaxEnt, as the weight vectors $\mathbf{w}^c$ will be independently fit to the data. In order to ensure an exactly equivalent model, it is necessary to enforce the constraints of equation 2.7, namely that most parameters are fixed to 0, and the non-zero part of the vector are identical accross all different $\mathbf{w}^c$.

This exemplifies one key difference between Maximum Entropy and the Single-Layer Perceptron. As Maximum Entropy may use different features for different classes, and may use other features in common between several classes, it offers a compact but powerful representation of the data.

A Multi-Layer Perceptron with a "softmax" layer (eq. 2.4) models the posterior class probability as:

$$P(c|\mathbf{x}) \propto \exp\left(\sum_j v_j^c.h\left(\mathbf{W}_{j\cdot}.\mathbf{x}\right)\right)$$

If the input weights $\mathbf{W}$ are held fixed, $f_j = h\left(\mathbf{W}_{j\cdot}.\mathbf{x}\right)$ may be used as a feature function and this reduces again to a special case of Maximum Entropy. Usually, however, both input and output weights ($\mathbf{W}$ and $\mathbf{w}^c$) are estimated from the data. Both Maximum Entropy and MLP therefore implement some kind of log-linear combination, but with two main differences:

- Maximum Entropy naturally allows to share feature functions between classes or tailor specifically some features to a particular class.
- On the other hand, MLPs can "learn" the feature functions $f_j = h\left(\mathbf{W}_{j\cdot}.\mathbf{x}\right)$ from the data by adapting the input weights $\mathbf{W}$.

Although the use of a single transfer function $h(\cdot)$ in MLP may at first seem overly restrictive, there exist numerous proofs that, under mild conditions on h, MLPs are universal approximators, ie they can approximate any (continuous) function arbitrarily closely (see, eg [17]).

2.4 Learning From A Huge Data Set

It is a general principle in machine learning that more data is always preferable. Large data sets allow for stable training procedures, increased generalization power, and potentially reduce the need for developing complex, high-level features. Many theoretical results demonstrate improved guarantees on performance for larger training sets. For such reasons, we are lucky to have at our disposal a very large set of data on which to train and test our models.

The responsibility of dealing with such quantities of data, however, has proved to be non-trivial, even on modern, well-equipped systems. Our data requirement commonly exceeds the available 4GB of memory, and the sheer quantity of information in its raw form approaches 100GB. As a result we have invested significant time into developing and discovering methods that, though of a practical nature, have been critical to the success of our machine learning approach.

2.4.1 Compression

A primary consideration is the ability to store the entirety of the data set on disk; our resource contraints required the use of compression even for this purpose. In order to optimize our access to the compressed data, we developed a standardized interface for reading and writing compressed files, and made a systematic comparison of the popular compression methods gzip and bzip2. Our results are summarized in figure 2.2. We observe a performance hit of approximately 50% moving from a flat file to the gzip standard; however, this drop is accompanied by a compressed size reduction of 87%. Moving to the more recent bzip2 format, we observe an additional 33% drop in compressed size, but the read times increase dramatically by approximately 700%. Comparisons of compressed write speed produced qualitatively similar results, though in all cases write speed is significantly slower than read speed.

Our original choice of compression, bzip2, had been made with respect to its superior compression ratios. However, while our tests do confirm this advantage, the speedup of 5 to 10 times observed using gzip easily outweighs the 50% larger on-disk size, especially considering that a single pass through the data set requires on the order of 6 hours if it has been compressed using bzip2. Conversion of all large data files to the gzip format allowed us to generate features and conduct experiments requiring a full sweep of the data in roughly one hour.

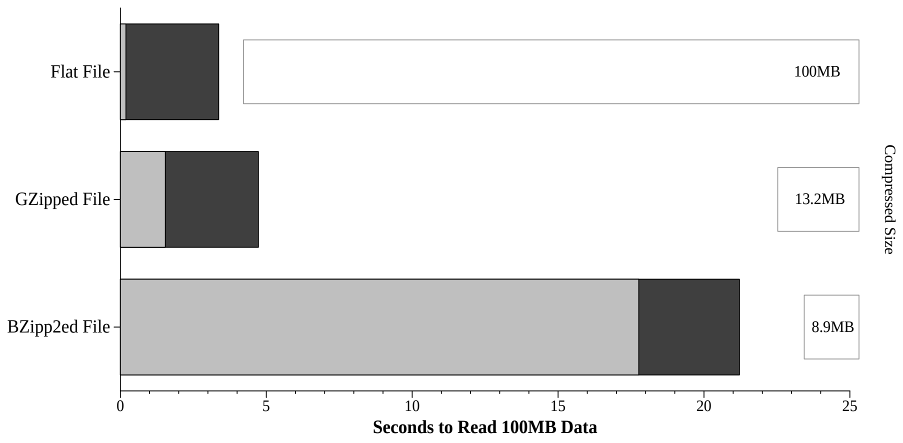


Figure 2.2: Performance of gzip and bzip2 in comparison to a flat file. Dark bars indicate the time taken reading one character at a time, and grey bars indicate the time taken reading one line at a time. Boxes indicate compressed size. The file is a 100MB sample of our data set.

2.4.2 Data Caching

For the purposes of training our models via gradient descent, we require stochastic presentation of the example set. In particular, if the example stream is not sufficiently randomized, we expect to see dramatically increased overfitting as the model becomes locally adapted to an organized data set. It is crucial, then, that we present examples in a random way.

Randomizing our data, however, is non-trivial because its size typically exceeds our available memory by a factor of 10 or more. Furthermore, random access on disk is extremely expensive, thus we would like a method of randomization that reads sequentially from the data set. To satisfy these conflicting goals, we have incorporated a caching system into the Torch machine learning library. The basic principle of the example cache is that it is loaded sequentially but unloaded randomly; in this way we provide a compromise between true randomization and minimal I/O overhead. The caching process is summarized in figure 2.3.

If the data set consists of n examples $\{x_1, x_2, \ldots, x_n\}$, and examples are requested at times $t = 1, 2, \ldots$ from a cache of size C, then example x_i will enter the cache at time $\max(0, i - C)$. Since any example in the cache is selected with probability $\frac{1}{C}$ at each request, the number of requests r for

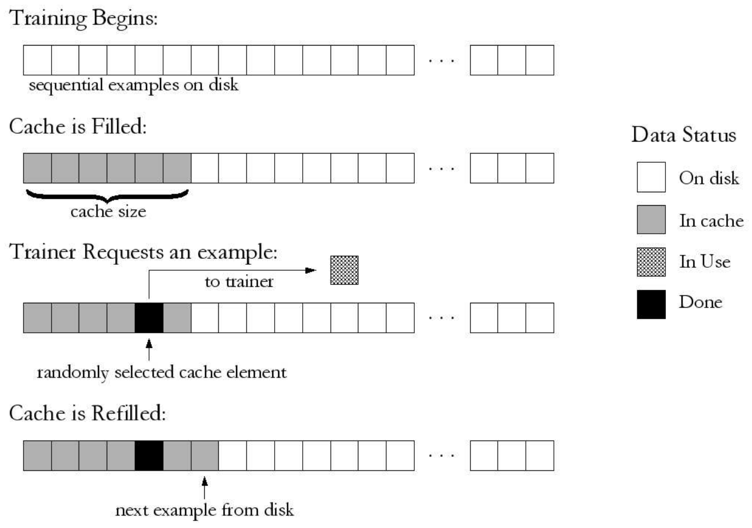


Figure 2.3: Example caching procedure. When an example is requested, a random element in the cache is selected and removed. The hole is filled by the next example from disk.

which an example will remain in the cache is a geometric random variable with expectation C. Thus, the expected time of presentation for example x_i is $max(C, i)$, and in general, ignoring edge effects, the cache does not alter the expected presentation time of any example. However, the variance of r is approximately $C(C-1)$, thus for large C the standard deviation of r is about C. This implies that the necessary cache size should vary roughly linearly with the degree of data organization, measured by the linear movement distance required to escape effects of locality.

Since our data are organized into N-best lists of size approximately 16,000, themselves deriving from source sentences in small groups of 1-5, we estimate that a cache size of approximately 100,000 should be adequate to achieve sufficient randomness. Empirically, we observe the results in figure 2.4. We observe large drops in error rate as the cache size grows beyond successive multiples of the N-best list size, and a cache size of even 50,000 appears to produce results indistinguishable from the fully random condition on our data. Furthermore, with a cache of at least 100,000 examples, the variation in performance is extremely low. Our experiments thus confirm the intuition above, and, as we have enough memory available for a cache of several million examples, it seems safe to say that the example cache will

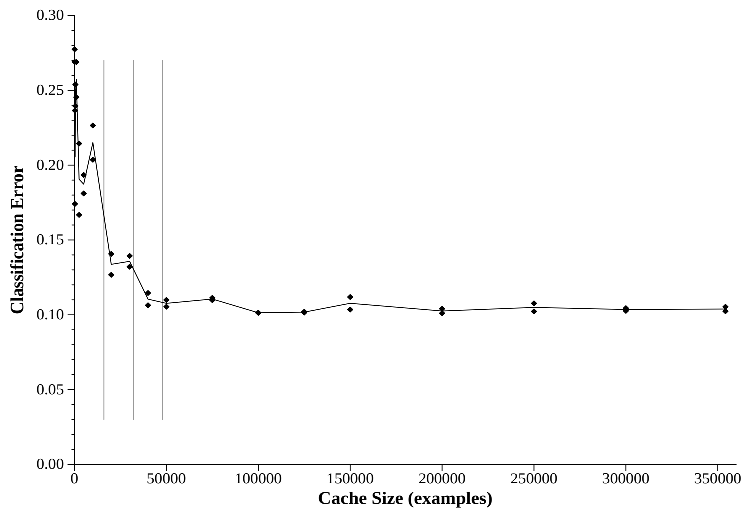


Figure 2.4: Cache performance as measured by classification error of the resulting model. The vertical lines represent cache sizes equal to one, two, and three N-best lists. Small dots represent individual trials; the solid line connects average results for each tested cache size. The largest cache is equal to the size of the sample dataset; thus it provides completely random example presentation.

have no perceptible effect on the performance of our models. On the other hand, since the implementation of the cache allows us to read the data set from disk (in its compressed form) only once per iteration and in a squential manner, the time savings are significant over a disk-based randomization method.

2.4.3 Parallel Training

As a final optimization, we have streamlined the learning process by allowing large banks of arbitrary MLPs to be trained simulataneously. Though our data compression and example cache reduce the cost of the training process significantly, a full run of many iterations is still expensive, requiring on the order of 10 hours, due to the extensive disk use. It thus remains impractical to train large numbers of models or to search for optimal parameters in a sequential way.

To compensate, we have expanded the Torch machine learning toolkit to train large numbers of arbitrary MLP models at once, sharing the same example stream for efficiency. This framework allows us to tune model parameters in batches, in addition to providing convenient tools for performace comparison and model selection. We estimate that the time saved by this method approaches 80-90% as the number of models trained simultanouesly becomes large (in other words, when training one model, approximately 80-90% of the time is normally spent on disk operations).

2.5 Evaluating Confidence Estimation Performance

As described in the introduction, we are interested in assessing the performance of confidence estimation techniques in two slightly different settings: a strong version requiring accurate probabilities of correctness, and a weak version requiring only binary classification decisions. The metrics we use for each task are described in each of the following two sections. In both cases, they are calculated over a labelled test corpus $\mathcal{D} = \{(\mathbf{x}^{(i)}, c^{(i)})\}_{i=1 \ldots n}$, where $c^{(i)}$ is 1 if $x^{(i)}$ is correct, otherwise 0.

2.5.1 Metrics for Probability Estimates

For evaluating probability estimates, a natural metric is the negative log-likelihood assigned to the test corpus by the model, normalized by the number of examples in the corpus:

$$\begin{aligned} \text{NLL} &= -\sum_i \log p(c^{(i)}|\mathbf{x}^{(i)})/n \\ &= -\sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \sum_c \tilde{p}(c|\mathbf{x}) \log p(c|\mathbf{x}), \end{aligned}$$

where $\tilde{p}$ denotes an empirical (relative frequency) distribution over $\mathcal{D}$. Negative log-likelihood is the loss function we used to train our MLP classifiers, and it is related to the loss function used for naive Bayes models (the latter is $-\sum_i \log p(\mathbf{x}^{(i)}, c^{(i)})$). As the second line in the above equation shows, it is also the expected cross entropy between the empirical distribution and the model distribution, across all contexts $\mathbf{x}$. Thus it can be seen as a kind of estimated distance from the true distribution to the model distribution.

Log-likelihood has the problem that it is sensitive to the prior probability of getting correct output from the base system—log-likelihood scores will tend to be better if the base system is either very bad or very good. Direct comparisons of log-likelihood scores are therefore not very meaningful if they pertain to base systems whose performance differs significantly. To address this problem, we use *normalized cross entropy* (nce) scores [29] instead of plain log-likelihoods. Normalized cross-entropy measures the relative drop in log-likelihood compared to a baseline (NLL_b) that depends on the prior probability of correctness:

$$\mathrm{NCE} = (\mathrm{NLL}_b - \mathrm{NLL})/\mathrm{NLL}_b.$$

Thus NCE normally ranges from 0 for baseline performance to 1 for perfect performance. Negative scores are also possible, and indicate performance worse than the baseline.

To establish the baseline score NLL_b, we assume a model that assigns some fixed probability of correctness, p_1, to all examples. Writing NLL as a function of p_1 gives:

$$\mathrm{NLL}(p_1) = -(n_0 \log p_0 + n_1 \log p_1)/n$$

where n_0 and n_1 are the numbers of correct and incorrect examples in $\mathcal{D}$, and $p_0 = 1 - p_1$. It is easy to show using calculus that this is minimized when p_0 and p_1 are set to the corresponding priors, giving:

$$\mathrm{NLL}_b = -[n_0 \log(n_0/n) + n_1 \log(n_1/n)]/n.$$

2.5.2 Metrics for Discriminability

For weak CE, we need to measure only how well our model discriminates between correct and incorrect examples, rather than the accuracy of its probability estimates. As described in section 2.2, in this setting we focus on a discriminant function $f(\mathbf{x};\theta)$ with the intention of setting a threshold that will be optimal for a given application. Evaluating f in the abstract, for all possible applications, requires techniques that capture classification performance across the range of all possible thresholds. In this section, we describe three such techniques: minimal classification error rate, ROC curves, and IROC.

Classification Error Rate

The most intuitive measure is simply the proportion of errors made by the "best" classifier over the test corpus. Given an optimum threshold $\hat{\tau}$, and a decision procedure characterized by:

$$g(\mathbf{x}) = \begin{cases} 1, & f(\mathbf{x};\theta) \geq \hat{\tau} \\ 0, & \text{else,} \end{cases}$$

the classification error rate is defined as:

$$\mathrm{CER} = \sum_i (1 - \delta(g(x^{(i)}), c^{(i)}))/n,$$

where $\delta()$ is 1 if its arguments are equal, otherwise 0. To set $\hat{\tau}$, we optimize either on a separate validation corpus or directly on $\mathcal{D}$. The former method gives an unbiased estimate of CER; the latter gives an overoptimistic estimate, but one that is convenient to calculate and useful for comparing the relative performance of different classifiers on the test set. In both cases, we use as a baseline the error rate $\min(n_1, n_0)/n$ of a classifier that assigns all examples to the most frequent class.

ROC Curves

There are four basic statistics that describe the performance of any binary classifier, shown in table 2.1. Since three numbers are required for a complete characterization, no single scalar metric is entirely satisfactory, and different research communities have tended to develop their own preferred measures, eg accuracy in machine learning; precision, recall, and F-measure in information retrieval, etc. The tradition in confidence estimation is to use correct acceptance and correct rejection rates, defined with respect to the intersection labels in table 2.1 as CA $= A/(A + B)$ and $CR = D/(C + D)$, respectively.

The plot of CR against CA for all values of the classification threshold τ is known as a Receiver Operating Characteristic (ROC) curve.[2] ROC curves lie on the unit square and connect the points $(0, 1)$ and $(1, 0)$: random separation of the classes gives a curve that runs along the diagonal, and perfect separation gives one that coincides with the top and right edges of the square (or the bottom and left edges in the case of perfect inverted separation). ROC curves thus provide a normalized picture of classifier performance that makes it easy to compare classifiers across the range of (relative) threshold values. Classifier A dominates classifier B if A's curve is always above B's whenever the two curves differ. It is possible for neither classifier to dominate the other (see section 4.4 for example of such a curve). Other properties of ROC curves can be found in [8].

To plot an ROC curve, one typically sorts the examples in the test corpus in ascending order by assigned score $f(\mathbf{x}; \theta)$. Then each rank $1, \ldots, n$ in the resulting corpus $\mathcal{D}$' corresponds to a distinct pair of (CA,CR) values that

[2]Minor variants are sometimes used instead, such as $1-$CA versus $1-$CR or CA versus $1-$CR.

	1	0
1	A	B
0	C	D

Table 2.1: Binary contingency table. Vertical (bold) labels indicate true class, and horizontal labels indicate assigned class. Letters indicate the proportion of examples in the intersections of these categories, eg A is the proportion of correct examples labelled as correct. Since $A+B+C+D=1$, any three parameters uniquely determine the fourth.

sorted rank	true class	CA	CR
1	0	6/6	0/4
2	0	6/6	1/4
3	1	6/6	2/4
4	0	5/6	2/4
5	1	5/6	3/4
6	1	4/6	3/4
7	0	3/6	3/4
8	1	3/6	4/4
9	1	2/6	4/4
10	1	1/6	4/4
–	–	0/6	4/4

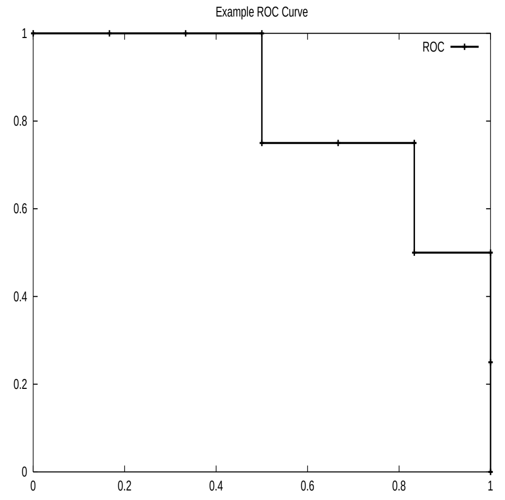


Figure 2.5: Example of a sorted data set and ROC curve (with CA on the horizontal axis, and CR on the vertical axis.)

can be plotted on a graph. Figure 2.5 gives an example of a sorted data set and the resulting graph.

IROC

ROC curves provide for a qualitative analysis of classifier performance; a related quantitative metric is IROC, defined as the area under an ROC curve. From figure 2.5 it is obvious that ROC curves are staircase-shaped: either CA or CR will change with each new point, but never both simultaneously. This makes it easy to calculate the area by simply summing fixed-width columns in the graph. If $\mathcal{D}$' is a data set sorted by assigned score[3] this calculation is:

$$\mathrm{IROC}(\mathcal{D}') = \sum_{i=1}^{n} \delta(c^{(i)}, 1) \sum_{j=1}^{i-1} \delta(c^{(i)}, 0)/n_0 n_1$$

where n_0 and n_1 are the numbers of incorrect and correct examples as before.[4] In other words, each correct example contributes a term that is proportional to the number of incorrect examples with lower ranks.

The geometric interpretation makes it obvious that IROC takes on values in $[0, 1]$, with 0.5 corresponding to a random separation of correct and incorrect examples, 1.0 corresponding to a perfect separation (all incorrect examples before correct ones), and 0.0 the opposite. It is less obvious that the baseline value of 0.5 is not affected by the prior probability of correctness in the data set, as is CER, for example. Clearly, a classifier cannot exploit knowledge of prior probabilities to improve IROC directly, since assigning all examples to the most frequent class will not affect their ranking. However, it might still be the case that the expected value of IROC across all possible rankings depends on the prior.

To show that this is not the case, it is helpful to rewrite IROC in terms of the sequence $\mathcal{R}$ of the ranks of the correct examples in $\mathcal{D}'$: $\mathcal{R}(\mathcal{D}') = \{r|c^{(r)} = 1\}$ Then:[5]

$$\mathrm{IROC}(\mathcal{R}) = \sum_{j=1}^{n_1} (r_j - j)/(n_0 n_1). \tag{2.8}$$

Letting Q_{n,n_1} be the set of all rankings $\mathcal{R}$ for data sets with n_1 correct examples out of n total examples, and assuming that all rankings are equally likely, the expected value of IROC across all rankings is:

$$E[\mathrm{IROC}(\mathcal{R})] = \sum_{\mathcal{R} \in Q_{n,n_1}} \mathrm{IROC}(\mathcal{R})/|Q_{n,n_1}|. \tag{2.9}$$

[3]Here we view the original data set $\mathcal{D}$ as a set of triples $(\mathbf{x}, c, i)$, where $i \in \{1, \ldots, n\}$ is the unique index assigned to each example. $\mathcal{D}$' is simply a re-indexing of $\mathcal{D}$ in which higher indexes correspond to higher assigned scores.

[4]Note that IROC is undefined if either n_0 or n_1 is 0; we assume throughout this section that this is not the case.

[5]It is interesting to note that this statistic is similar to the Wilcoxon rank sum [38].

Defining the complement of $\mathcal{R}$ to be $\overline{\mathcal{R}} = \{r | n + 1 - r \in \mathcal{R}\}$, we have:

$$\begin{aligned}
\text{IROC}(\mathcal{R}) + \text{IROC}(\overline{\mathcal{R}}) &= \sum_{j=1}^{n_1} (r_j + n + 1 - r_j - 2j)/(n_0 n_1) \\
&= (n_1 n + n_1 - n_1(n_1 + 1))/(n_0 n_1) \\
&= (n - n_1)/n_0 = 1.
\end{aligned}$$

Clearly then, $\sum_{\mathcal{R} \in Q_{n,n_1}} \text{IROC}(\mathcal{R}) + \text{IROC}(\overline{\mathcal{R}}) = |Q_{n,n_1}|$. Because the complement of each $\mathcal{R}$ in Q_{n,n_1} is also in Q_{n,n_1}, and because there is a one-to-one mapping between rank sets and their complements, $\{\overline{\mathcal{R}} | \mathcal{R} \in Q_{n,n_1}\} = Q_{n,n_1}$. Therefore:

$$\begin{aligned}
\sum_{\mathcal{R} \in Q_{n,n_1}} \text{IROC}(\mathcal{R}) + \text{IROC}(\overline{\mathcal{R}}) &= |Q_{n,n_1}| \\
\sum_{\mathcal{R} \in Q_{n,n_1}} \text{IROC}(\mathcal{R}) &= |Q_{n,n_1}|/2
\end{aligned}$$

Substituting this result into (2.9) gives $E[\text{IROC}(\mathcal{R})] = .5$. This is a nice property because it means that IROC values from different data sets can be compared directly.

IROC versus CER

An interesting question is the extent to which the two scalar measures of performance we use—IROC and CER—are correlated. It turns out that whenever one classifier dominates another (having a systematically higher ROC curve and therefore a higher IROC), it also has a lower CER (assuming the threshold for CER is optimized on $\mathcal{D}$). This is significant because different classifiers usually have clearly separated ROC curves, rather than ones that intertwine. Whenever this common pattern occurs, we can conclude that the dominant classifier is better according to both metrics.

Formally, classifier A dominates classifier B if the two give different rankings for $\mathcal{D}$ and, for identical values of CA, A's maximum CR value is always at least as high as B's maximum CR value. To show that this leads to lower CER values, we express classifier accuracy as a function of CA and CR:

$$\text{ACC} = (n_1 \text{CA} + n_0 \text{CR})/n.$$

Now let CA_B and CR_B be the CA and CR values that correspond to B's maximum accuracy threshold, and let CR_A be the maximum CR value that results when A operates at a threshold that makes its CA equal to CA_B. If ACC_A is the accuracy of A at this threshold, and ACC_B is B's maximum accuracy, then from the definition of dominance:

$$\text{ACC}_A - \text{ACC}_B = n_0(\text{CR}_A - \text{CR}_B)/n \geq 0.$$

This establishes the property, since ACC = 1 − CER.

2.5.3 Bootstrapping Error Bars

We end this section with a short presentation of the bootstrapping technique that we used to derive error-bars on some of the experimental results.

Let us consider that we are interested in estimating the "performance" of a model. Here "performance" may be classification error, IROC, average precision, F-score, etc. If we wish, for example, to compare two models, or have guarantees on the average performance of a model, it is necessary to better characterise the variability of the performance, in order to be confident that an observed difference is not due to chance variations on a few examples. This may be done by obtaining a distribution of the performance, or some summary statistic on this distribution, such as confidence intervals. For some of these performance measures, it is easy to derive confidence intervals on the performance. However, in most cases where the performance measure is a bit complicated, there is no analytical expression, and we must estimate error bars in some other way.

In the following, we use the term "error bar" to mean symmetric confidence interval. For example, the 95% error bars on an IROC value of 0.80 may be 0.05, indicating that $[0.75, 0.85]$ is a 95% confidence interval for the IROC of this model.

The idea with estimating error bars on the performance of a model is the following. If we could sample exhaustively from the distribution of the data, $(\mathbf{x}, c) \sim P(\mathbf{x}, c)$, we may be able to calculate the "true" performance S of our model (eg the "true" IROC). Unfortunately, we can not do that for a variety of reasons including the fact that the true distribution $P(\mathbf{x}, c)$ is usually unknown and that we do not have infinite computing power.

However, we have a sample $\mathcal{D} = \{\mathbf{x}^{(i)}, c^{(i)}\}$ containing examples with associated labels. From this sample, we can obtain an estimate of the performance $\widehat{S} = f(\mathcal{D})$. How far from S is our estimate $\widehat{S}$?

Obviously, if we could repeatedly sample from $P(\mathbf{x}, c)$ different samples $\mathcal{D}$, we could replicate this process, obtain numerous estimates $\widehat{S}$ and therefore better characterise the distribution of $(\widehat{S} - S)$. However, we usually can't do that, for the same reason that prevents us from calculating S directly.

The bootstrap principle [9] consists in replacing the unknown $(\widehat{S} - S)$ by the empirically obtained $(S^* - \widehat{S})$. Having a possibly large collection of estimates S^* from many bootstrap samples, we can derive statistics on $(S^* - \widehat{S})$ and apply them to $(\widehat{S} - S)$. For error bars, we will simply find the value Δ such that $P(|S^* - \widehat{S}| < \Delta) = 1 - \alpha$ based on the bootstrap samples. We then use Δ as an error bar for $\widehat{S}$. Our final estimate of the performance $\widehat{S}$, with error bars, is therefore $\widehat{S} \pm \Delta$.

More general details about the bootstrap may be found in several textbooks about the bootstrap, eg [9, 14].

For all experiments reported in subsequent chapters for which confidence intervals are given, the boostrap technique was used with 20 resampling runs and a confidence interval of 95%.

Chapter 3

Sentence-Level Experiments

In this chapter we describe the experiments carried out at the sentence level. Given a source sentence and a machine-generated translation, we are interested in making a judgement about the correctness of the entire translation. As discussed in the introduction, since the vast majority of machine-generated translations are simply wrong, our strategy here is to redefine “correctness” as having an MT evaluation score higher than a certain threshold.

The chapter is divided into three main parts. Section 3.1 describes the features we used. Section 3.2 gives the results of experiments to test various confidence estimation techniques based on these features, using the evaluation metrics described in section 2.5. Finally, section 3.3 reports on experiments to determine if CE techniques can be used to improve the output of the base MT system.

3.1 Sentence-Level Confidence Features

In this section we describe the 91 features we used for sentence-level CE. Each description is accompanied by tags that classify the feature in two separate ways:

- By dependence on the base model (B) or not (N). Features are considered dependent on the base model if they express some information that could not be inferred just by examining the source/target pair to which they apply. For example, the length of a target hypothesis is not considered a model-dependent feature, even though the hypothesis has of course been generated using the model. On the other hand, the average length of all hypotheses in an nbest list *is* considered model dependent because it is not an intrinsic property of the particular hypothesis under consideration.

- By dependence on the source text (S), the target text (T), or both (ST). Features are tagged as S or T if they are intrinsic to either the source or target texts—ie, can be calculated from them in isolation—otherwise ST. For instance, the average length of the target sentences in an nbest list would be tagged S, because it can be calculated from the source sentence alone (using the base SMT system to generate the nbest list), and does not depend on the current target hypothesis.

This classification is used in the feature-attribution experiments described in section 3.2. To facilitate cross-referencing, each (scalar) feature is also assigned a unique index. When a group of closely-related features is described together, a range of indexes like (5–9) will be indicated. Table 3.3 at the end of the section contains a list of all features, along with their indexes, names, and classifications.

In general, features that pertain to the source sentence are listed first in this section, followed by those that pertain to the target sentence, and finally by those that apply to both. However, this is only a loose ordering which is violated when necessary in order to group features that were implemented together or that are related in some other way. Also, note that feature indexes are not assigned in order of the listing here, but rather in the order the features were implemented.

3.1.1 Base Model Feature Functions

As described in [31], the ISI SMT system is based on a maximum entropy model defined over a variety of feature functions, listed below. We re-used these feature functions as confidence features in our CE models.

- `BaseScore`: a log-linear combination of all of the features listed below. This is the base model's main probability estimate, and is used to rank alternatives in the nbest list. (B, ST, 35)
- `CostsAlTemp`: log probability for the translation model. (B, ST, 36)
- `CostsAlTempPenalty`: Because longer alignment templates (ATs) can be more descriptive and dependable in translation, this penalty encourages fewer (and longer) ATs over more (and shorter) ATs. (B, ST, 37)
- `CostsRuleBased1`: dependent on the number of a times a translation rule was used outside of the AT model. For example, rules were used in all cases of date, time, or digit translations. (B, ST, 38–39)
- `CostsLanguageModel`, `CostsLanguageModel2`, `CostsLanguageModel3`, `CostsLanguageModel4`: the log-probability of a target hypothesis based on one of four trigram language models, constructed identically but with different training corpora. (B, T, 40–43)
- `CostsJump`: captures non-monotonicity in source-to-target alignment by counting the number of jumps in word-to-word alignment within alignment templates. (B, ST, 44–45)
- `CostsSWLex`: log-probability of the target hypothesis based on a single-word lexicon rather than alignment templates. (B, ST, 46)
- `CostsWordPenalty`: To help keep the translation system from automatically favoring shorter (and therefore more "probable") hypotheses, this integer word penalty score is monotonically decreasing as hypothesis lengths grow. (B, T, 47)

3.1.2 Simple Nbest Features

The following features were all extracted from translation system log files in an attempt to classify a more confident and accurate translation procedure.

1. N-best translation results
   - Rank in N-best list. (B, ST, 1, 48)

- Average hypothesis length: the average length of hypotheses within the N-best list. (B, S, 2)
- N-best list density: the number of target words of all sentences in the N-best list divided by the number of source words. (B, S, 3)
- The length N of the N best list. (B, S, 4)

2. Ratio of hypothesis base score and best score. For each hypothesis, its base-model score (= negative logarithm of the probability) is divided by the base-model score of the best (first in N-best list) hypothesis, to give a normalized score that can be compard across different source sentences. (B, ST, 5)

3.1.3 Search Based Features

The beam search algorithm of the Alignment Template system performs pruning steps for each cardinality of the coverage vector. All partial hypotheses with the same number of covered source positions are compared and the one with the highest probability among them is determined. Then, all hypotheses which have a probability close to this best value (i.e. which lie within the 'beam') are kept for further expansions, whereas the others which have a lower probability are discarded. Furthermore, all those hypotheses that are equal with respect to the translation and language model states are recombined in order to reduce the size of the search space.

We computed several features related to this pruning: for each pruning step, i.e. for each cardinality of the coverage vector, we are given information such as the number of pruned hypotheses. Those values are averaged over all pruning steps and taken as features. We extracted the following features:

1. Number of active hypotheses after pruning and recombination: This is the number of partial hypothesis that were kept for further expansion, after all indistinguishable hypotheses have been recombined. (B, S, 6)

2. Number of pruned hypotheses: This is the number of hypotheses that have a low probability (compared to the best one) and are discarded at this step in the search. (B, S, 7)

3. Base score of best hypothesis in beam: This is the negative logarithm of the highest probability among all considered partial hypotheses. (B, S, 8)

4. Base score of worst hypothesis in beam before pruning: This is the negative logarithm of the lowest probability among all partial hypotheses with the same number of translated source words before the pruning is performed. (B, S, 9)

5. Base score of worst hypothesis in beam after pruning: This is the negative logarithm of the lowest probability in the beam after pruning has been performed. These scores are lower or equal to the one in 4. (B, S, 10)

Note that most of the features in this and the previous section are the same for all translations of one source sentence. Thus, they give an estimate of how hard this source sentence is to translate for the SMT system.

3.1.4 Sentence Length Features

We used three sentence-length-based confidence features:

1. source sentence length (N, S, 81)

2. target translation length (N, T, 82)

3. source / target length ratio (N, ST, 83)

These features are trivial but could be indicators of the intrinsic difficulty of the source sentence and of the potential mismatch between source and target sentences.

3.1.5 Source N-gram Frequency Statistics

The n-gram frequency features are meant to reflect how common the words and word sequences in a given source sentence are on average. The intuition behind them is that if a large percentage of the n-grams in the source sentence have often been seen in a large corpus, then the translations produced for the sentence may be more accurate.

For these experiments we used the Chinese side of bi-text corpus, initially used for training the base ISI SMT models. The set of unigrams, bigrams, and trigrams seen in the Chinese side of the corpus was extracted and sorted in order of increasing frequency. Start- and end-of-sentence tokens were also included in this sorting. To save memory, all singleton n-grams were not included in final counts and divisions. Source sentences were then broken into sets of n-grams, and each n-gram found in the sentence was mapped to the

Statistics	Unigrams	Bigrams	Trigrams
# n-grams seen in corpus	102,625,259	102,025,258	67,319,946
# distinct n-grams seen in corpus	126,725	4,553,400	13,847,674
# distinct non-singleton n-grams	58,616	2,254,454	4,842,677
Max abs. frequency	5,905,550	735,878	253,921
Ave abs. freq $\overline{x}$	1749.67	44.23	12.04
# distinct n-grams with freq $> \overline{x}$	3773	199,219	582,357
Lower quartile bound	3	2	2
Median frequency	8	4	3
Upper quartile bound	62	11	6
# n-grams in each quartile	∼14,680	∼560,000	∼1,200,000

Table 3.1: Corpus Frequency Statistics

absolute corpus frequency previously recorded. General corpus frequency statistics can be found in table 3.1.

Many typical measures of frequency do not effectively express the sentence n-gram distribution. n-gram frequency arithmetic or geometric means are heavily affected by the appearance of a few commonly seen n-grams, regardless of the remainder of the distribution. As can be seen in table 3.1, only 5.7%, 8.8%, and 12% of non-singleton unigrams, bigrams, and trigrams respectively were seen more frequently than the mean count. A second possible approach of simply noting the median n-gram frequency of a sentence would not give enough detail about the sentence as a whole. Taking this into consideration, we decided to use a quartile range measure. We divided the corpus n-grams into four quartiles containing approximately an equivalent number of distinct n-grams. The number of source sentence n-grams within each frequency quartile was then counted and normalized over the number of n-grams in the sentence.

The final result of this approach was 12 parameters per source sentence showing the percentage of 1-, 2-, and 3-grams in each of the four frequency quartile ranges. (N, S, 49–60)

3.1.6 Source Language Model Features

Using the same Chinese corpus as in section 3.1.5, a trigram language model with Kneser-Ney discounting as backoff was built using the SRI Toolkit. These features were also used in our sub-sentential CE experiments. For each source sentence, three features were calculated:

- Log-probability of the source sentence (N, S, 61)

- Perplexity 1 – includes end-of-sentence marker (N, S, 62)
- Perplexity 2 – does not include end-of-sentence marker (N, S, 63)

3.1.7 Averaged Target Word Statistics

This section describes a set of features based on statistics that pertain to individual target words. Each feature is the average value of the corresponding statistic over all words in the current target hypothesis. Statistics are:

1. Number of occurrences of the target word within this sentence. (N, T, 74)
2. Relative frequency of the word in the N-best list. (B, ST, 75)
3. Rank-weighted frequency of this word in the N-best list. (B, ST, 76)
4. Frequency of this word in the N-best list, weighted by the base-model score; this is analogous to the calculation of word posterior probabilities over N-best lists. (B, ST, 77)

In addition to these, we also calculated the values in 2 – 4 over the set of words in the nbest list occuring in exactly the same target position as the current one. (B, ST, 78–80)

3.1.8 Target Language Model Features

We tried two methods of training target language models (in addition to those described in section 3.1.1): using an external corpus, and using the current nbest list.

External Training Corpus

Using the English half of the training corpus described in section 3.1.5, an English trigram language model with Kneser-Ney discounting as backoff was built using the SRI Toolkit. As in 3.1.6, for each target sentence, three confidence features were calculated:

- Log-probability of the target sentence (N, T, 64)
- Perplexity 1 – calculated using all input tokens (N, T, 65)
- Perplexity 2 – calculated using all token but the end-of-sentence marker (N, T, 66)

These features were also used for sub-sentential CE experiments.

Using the Nbest List as Training Corpus

The idea here is to capture the homogeneity of translations within nbest lists. For each source sentence, we used the words in the corresponding nbest list to build a target "vocabulary" V_{nbest}, and then trained 1-, 2-, and 3-gram language models. As probabilities were calculated for sentences based on LMs trained on the same sentences, smoothing techniques were unnecessary and therefore not used. Using the resulting vocabulary and models, we computed the following nine confidence features for each hypothesis in the nbest list:

- Sentence 1-gram log-probability (B, ST, 11)
- Sentence 2-gram log-probability (B, ST, 12)
- Sentence 3-gram log-probability (B, ST, 13)
- Sentence 1-gram log-prob divided by sentence length (B, ST, 14)
- Sentence 2-gram log-prob divided by sentence length (B, ST, 15)
- Sentence 3-gram log-prob divided by sentence length (B, ST, 16)
- 1-gram perplexity (B, ST, 17)
- 2-gram perplexity (B, ST, 18)
- 3-gram perplexity (B, ST, 19)

In addition to the language-model based features above, we also calculated three related features that pertain to the nbest list as a whole:

- average length of the target sentences (B, S, 20)
- vocabulary size divided by average sentence length (B, S, 21)
- vocabulary size divided by the length of the source sentence (B, S, 22)

Note that the final feature is a kind of average "fertility" of the words in the current source sentence.

3.1.9 Center Hypothesis Features

These features are similar in spirit to the ones described in the previous section in that they capture the degree of similarity between the current hypothesis and others in the nbest list. The Levenshtein (edit) distance between each of the top 1000 hypotheses was calculated and, without taking nbest rank into account, the hypothesis with the smallest average edit distance from all others was selected as the center hypothesis.

Confidence features included:

1. Edit distance, h_n, from the center hypothesis. (B, ST, 67)

2. The percentage of the other 1000-best hypotheses with an edit distance less than $|h_n| * 0.5$. (B, ST, 68)

3. The percentage of the other 1000-best hypotheses with an edit distance less than $|h_n| * 0.35$. (B, ST, 69)

The second and third features were meant to highlight hypotheses which may have differed greatly from a few outliers but deviated little from much of the hypothesis population. In preliminary testing, text generated from hypotheses having the greatest value here seemed to improve the BLEU score as much as or more than text generated from the center hypotheses themselves accomplished.

3.1.10 Basic Syntax Check

This basic syntax check looked to highlight hypotheses with mismatched parentheses and/or quotation marks. If such a syntax error was identified, these features would trigger and become non-zero. (N, T, 72–73)

3.1.11 IBM Model 1

In [5], five standard SMT methods were presented. While our base SMT system is derived from IBM Model 4, we can examine the resulting translations with the much simpler Model 1. Model 1 uses what is known as a "bag of words" translation model, meaning that its calculations are not tied to any specific alignment structure (apart from the basic one-to-many source-target correspondence assumed by all IBM models). Rather, for each source/target hypothesis pair, we find the sum of probabilities of all possible alignments. This captures a sort of topic or semantic coherence in translations.

As defined in [5], Model 1 gives a probability of any given translation pair with the formula

$$p(\mathbf{f}|\mathbf{e}) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j|e_i),$$

where e_0 is the "empty" word, introduced to capture source token alignments that correspond to no actual target token.

The WS03 SMT group trained models of this form using the tool Giza++, producing two distinct distributions: $p(\mathbf{f}|\mathbf{e})$ and $p(\mathbf{e}|\mathbf{f})$. Using our set of source and target sentences, we used these trained distributions to find the Model 1 log probability of the source sentence given each target sentence, as well as of each target sentence given the corresponding source sentence. Each of these results was considered as a confidence feature. (N, ST, 90–91)

3.1.12 Alignment Discontinuities

In this section we describe both sentence-level and word-level confidence features that identify *non-contiguities* within the word alignments produced by the baseline SMT system. These are *many-to-many* alignments: any source word can be aligned to an arbitrary number of target words and vice-versa. For language pairs such as English-French or English-Chinese, where alignments between the source an the target sentence tend to be monotone, correct translations should in general result in contiguous word sequences aligned with each other. The presence of non-contiguities within the word alignments could therefore be an indicator of alignment and/or translation problems. Jumps can occur both within the source-to-target and within the target-to-source alignments. The baseline SMT system already captures this property as a feature function, described in section 3.1.1 as CostsJump (and described in [31] as the *Phrase Alignment Feature Function*). This feature simply sums over jump distances in the source language of alignment templates which are consecutive in the target language. The features we implemented explore this idea further in the following ways:

1. bidirectionality: we compute the features both over target and source language
2. maximum jump: we consider the maximum jump as a sentence-level or word-level confidence feature
3. average-jump: at the sentence-level we normalize the sum of jumps by the number of words in the sentence

Moreover, we compute these features on a word-level basis, which allows them to be used for word-level confidence estimation. The sentence-level features are simply the average and maximum of the alignment-jump features over all words in the sentence.

Maximum and average jumps have the advantage of being robust to sentence length: the original sum of jumps feature function is perfectly valid within the SMT translation framework, where the focus is on comparing translation alternatives for a given source sentence and are typically comparable in length. However, for confidence estimation it is important that confidence features generalize well from one sentence to another, regardless of its length.

We give the formal definition of these features with respect to source-to-target alignments. The definitions for the target-to-source alignments are analogous. Consider an arbitrary source sentence $\mathbf{f} = f_i^J = (f_1, \ldots, f_J)$ consisting of a sequence of tokens f_j and a corresponding target translation $\mathbf{e} = e_i^I = (e_1, \ldots, e_I)$. For each source token f_j there is a (possibly empty) sequence of k aligned target tokens, noted by

$$\mathrm{A}(f_j) = (e_{i_1}, \ldots, e_{i_k}),$$

where the indexes $i_1, \ldots, i_k$ range over $\{1, \ldots, I\}$ and are given in ascending order. A jump in the alignment of the source token f_j occurs if any two consecutive target tokens $(e_{i_x}, e_{i_{x+1}})$ within $\mathrm{A}(f_j)$ are such that $i_{x+1} > i_x + 1$. The alignment jump for f_j is defined as:

$$\mathrm{J}(f_j) = \sum_{x \in \{1, \ldots, k-1\}} i_{x+1} - i_x - 1.$$

We define the maximum alignment jump for f_j by:

$$\mathrm{J_M}(f_j) = \max\nolimits_{x \in \{1, \ldots, k-1\}} i_{x+1} - i_x - 1.$$

These word-level features can then be extended to the sentence level as follows: the maximum source alignment jump is the maximum alignment jump over all source tokens:

$$\mathrm{J_M}(\mathbf{f}) = \max\nolimits_{j \in \{1, \ldots, J\}} \mathrm{J_M}(f_j) \qquad \text{(B, ST, 84)}$$

and the average source alignment jump is:

$$\mathrm{J}(\mathbf{f}) = \sum_{j \in \{1, \ldots, J\}} \mathrm{J}(f_j)/J. \qquad \text{(B, ST, 85)}$$

3.1.13 Translation Consistency Features

These features use the alignments available from the base model to capture the degree to which target words or phrases (alignment templates) are translated the same way throughout the nbest list. For each target word or phrase in the current hypothesis, we determine the source word to which it is aligned most often in the nbest list, and divide the number of times it is aligned to that word by the total number of potential alignments (ie, the size of the nbest list). These normalized counts are used as-is for sub-sentential features, but averaged over all words or phrases in the target sentence to yield two sentential features:

- average target word consistency (B, ST, 71)
- average target phrase (= alignment template) consistency (B, ST, 70)

3.1.14 Phrase-Based Language Models

This set of twelve features was calculated for purposes and in a manner very similar to that depicted for the nbest language models in section 3.1.8. However, whereas that procedure generated a vocabulary of target words and word-based language models, our components here focussed on a vocabulary of alignment templates V_{nbest_AT} used throughout a source sentence's N-best list.

To achieve this, we represented each Alignment Template and the source positions which are aligned as a unit in each hypothesis sentence. For example, if according to the translation model the target words "the first two months of this year" of a certain hypothesis were aligned to the first three Chinese terms of the source sentence, we would add to the vocabulary V_{nbest_AT} the single concatenated term

"`the+first+two+months+of+this+yearA0+1+2`"

Repeating this concatenation process for all alignments in the full nbest list, we were then able to train AT-based (unsmoothed) 1-gram, 2-gram, and 3-gram language models. Using these models, the same twelve features as described for nbest word LMs in 3.1.8 were computed. (B, ST, 23–34)

3.1.15 Semantic similarity features

A pervasive problem in Machine Translation is the existence of multiple possibilities for correct output. Whereas in ASR there is a unique string of text corresponding to the sound-stream, acceptable translations for a source sentence can vary wildly in length, word order, and lexical choice. This problem has dire consequences for the language and translation models; the true probability associated with a particular word will be divided among its synonyms, thereby assigning inappropriately large weights to words with fewer synonyms. We would like our confidence estimation to not be fooled by this divergence of possible meaning representations; we would like the system to be able to identify that even though a pair of sentences may share few words, if the semantic content is similar, they ought to be assigned similar confidence scores.

The approach employed here is to construct a function that will quantify the degree of semantic similarity between a pair of sentences. This metric can then be used for the same purposes as the other sentence-level metrics discussed in this paper (e.g. Levenshtein distance) In this section, then, we discuss briefly some current ideas in creating word-level semantic metrics, then discuss how they can be used to build a sentence-level metric, and what its relevance is for machine translation.

Semantic features have thus far been left out of the system, both because of their inherent difficulty, and because the reliance on human-constructed sources of semantic data is somewhat antithetical to the data-driven approach of statistical machine translation. Nevertheless, the potential gains from this kind of an analysis are exciting enough to merit its investigation.

Word-level semantic similarity metrics

The use of automatic metrics for semantic similarity at the word level is a well-studied problem, and in certain domains and for certain applications, the results are very encouraging. Numerous techniques have been proposed for solving this problem: for word sense disambiguation and malapropism detection, approaches based on information content using corpus statistics [37] and approaches involving some kind of weighted path length in a lexical taxonomy such as WordNet [16]. Vector-space techniques are also often employed in information retrieval [10].

Using these metrics in a machine translation context places certain restrictions on the type of metric that is employable. Because they rely on WordNet's IS-A hierarchy which is only defined for nouns, the information content approaches cannot be used; we will need to be able to compare instances of other word-classes in translation hypotheses, and it is often the case that a translation application will hypothesize translations of the same word using more than one part of speech. Speed is also a concern because of the number of word-pairs that the application will need to compare.

The word-similarity function that was used here uses a dictionary-based approach developed by Banerjee & Pedersen [2]. WordNet, in addition to representing the interconnections between words, provides a gloss (definition) for each of them. In this algorithm, similarity between a pair of words is represented by the degree of overlap between the words used in their glosses; two words that mean about the same thing, according to this intuition, would be defined using the same words. N-gram overlaps between glosses are weighted heaver proportional to their length.

Relying solely on the glosses of the words we are interested in comparing in this manner is unlikely to yield good results: glosses are typically only a couple words long, and the choice of words used in them is highly arbitrary. To make the data a little less sparse, the algorithm also defines a set of *relation pairs* that indicate other related words that are to be combined–WordNet has a huge graph structure that can be exploited by comparing words close to the target words in the graph. For example, the algorithm would compare the hypernyms (things that the word in question is a specific kind of) of the words in question. The total similarity score of a word-pair is a weighted combination of the scores for all the relation pairs.

Computation of this word-similarity metric was done in perl using a module written by Siddharth Patwardhan and Ted Pedersen [34].

It is important to note that the nodes in WordNet's graph correspond not to words but to meanings, or word-senses. Most words that people use have many senses, and coming across them in MT output, we do not have the luxury of knowing which sense was the intent of the translator. To deal with this problem we define the word similarity to be simply the maximum similarity over all senses, a reasonable assumption given that the words we are comparing are intended to mean the same thing. In practice this may not always turn out to be the pair of word senses that were intended by the translator, but we take it as a reasonable approximation.

Table 3.2 gives a few examples of the kinds of outputs that this function produces, taken mostly from the test set. While we did no formal testing aside from a cursory examination of the figures, it seems from this table and from other similar experiments that this metric is a good one. It is difficult to evaluate the magnitudes from this kind of experiment—should *construction* really be three times as far from *development* as *accomplishments* is from *achievements*?—but it seems at least that the ordering of the word-pairs is appropriate. One problem that seems apparent is the bias towards identity: in the examples above, the similarity between the two instances of *economic* is nearly twice as high as any of the other pairs, even those belonging to the

Word pair	Similarity
economic - economic	4.048
accomplishments - achievements	2.142
cities - municipalities	1.963
China - Chinese	1.314
construction - development	0.746
remarkable - remarkably	0.492
apples - oranges	0.338
accomplishments - successful	0.085

Table 3.2: Sample word similarity scores

same WordNet synset (*accomplishments* and *achievements*). While such a metric might conceivably be desirable in the context of information retrieval, where an exact match to the query is potentially more valuable than a match on a similar word, in machine translation it would be ideal if synonyms were given nearly the same score as pairs of the same word, since using a true synonym yields an equivalent translation. This is especially true because the metric does not apply translingually and hence only derived outputs can be compared; we don't want to bias in favor of particular a particular word choice if that word choice is merely the output of the system.

Sentence-level semantic similarity

The algorithm described above outputs a number representing the semantic similarity of two words. In order to generate confidence features for the entire sentence, we need a way of combining these word-level scores into a function that measures semantic similarity between entire sentences.

The main problem in doing this is deciding which words to compare. We can take advantage of the word alignments generated by the translation system–every word in the target translation is associated with one or more words of the source sentence. We cannot compare source words to target words directly, but we can compare two translation hypotheses for the same source by comparing the words aligned to the same source word. Since the alignment is many-to-many, we compute a score corresponding to each source word that is the maximum word-similarity over all pairs of words aligned to that source word.

In symbols, suppose we have two translation hypotheses E and E' whose words are represented in the form e_i or e'_i. We define the alignment B by $B_{i,j} = 1$ if f_j aligns to e_i and $B_{i,j} = 0$ otherwise. B' is defined likewise.

We can then compute a similarity vector S with one entry per source word $f_j \in F$ according to:

$$S_j = \max_{i,k}[\text{ sim } (e_i, e'_k) \cdot B_{i,j} \cdot B'_{i,k}] \tag{3.1}$$

In the case that there are no content words aligned to a given source word, that source word is ignored altogether as not contributing to the semantic content of the sentence.

Note that this use of the alignments to the source makes the sentence-level metric weakly model-dependant. Intuitively, such a function shouldn't depend on the model—it is intended to represent the similarity in meaning of the two sentences, not anything about the way that the model processed it. It would in theory be possible to get away without using alignments or the source sentence at all, computing instead the similarity of each word in one sentence with each word in the other and assuming that the word-pairs with maximum similarity are most likely aligned to each other. This has the significant disadvantage of dramatically increasing the processing time, a serious problem given the fact that the word-similarity function is already very slow, so we avoid it.

The most natural way to convert the vector of similarities S into a sentence-level score would be to simply average the values. This doesn't do a good job of capturing the relevant idea, however. Ideally the highest scores will be given to sentence pairs for which all the words are very similar, and sentence pairs with most of the words very close and a couple words completely off should be judged worse than sentence pairs with all the words only fairly close. The averaging function ought also to compensate for the dilation of the range of the word-level function near the upper end noted above.

Three techniques were employed to deal with these issues. The first uses a log-linear average rather than a straight average:

$$\text{sim}_{log}(E, E') = \exp\left(\frac{1}{\#\text{ content words}} \sum_j (\log S_j)\right) \tag{3.2}$$

This dampens the upper end of the range, mediating the influence of identical words in the sum, and penalizing more heavily sentences containing words having little similarity.

Another approach is to simply use the harmonic mean instead of the arithmetic mean, effectively defining semantic distance as the reciprocal of similarity and averaging distance instead. Since it is possible to have zero similarity, we have to add a parameter α which specifies the extent to which we want to bias our scale in favor of low numbers. For the experiments we ran we set $\alpha = .2$.

$$\text{sim}_{harm}(E, E') = \left(\frac{1}{\#\text{ content words}} \sum_j \left(\frac{1}{S_j + \alpha} \right) \right)^{-1} \tag{3.3}$$

Also, we can eliminate the impact of large word-similarity values by quantization; we can pick a threshold value θ and define the sentence similarity as the percentage of content words in the source that are associated with output words having similarity greater than that threshold:

$$Q_i(\theta) = \begin{cases} 1 & \text{if } S_j \geq \theta \\ 0 & \text{if } S_j < \theta \end{cases} \tag{3.4}$$

and

$$\text{sim}_{thresh}(E, E') = \frac{1}{\#\text{ content words}} \sum_i (Q_i) \tag{3.5}$$

Our experiments used $\theta = 1$.

The following table gives some sample outputs of the log-linear sentence-level similarity metric. Results from the other two functions tend to be mostly monotonic with this one.

Sentence	Sim
comparing to: *china 's 14 open border cities marked economic achievements*	
china 's 14 open border cities marked economic achievements	1.444
china 's 14 open border cities economic achievements marked	1.444
china 's 14 open border cities significant economic achievements	1.302
china 's 14 open border cities economic achievements significant	1.302
china 's 14 open border cities economic achievements remarkable	1.301
china 's 14 open border cities achievements marked	1.290
china 14 open border cities marked economic achievements	1.281
china 's 14 open border cities achievements significant	1.148
china 's 14 open border cities achievements remarkable	1.147
china 's 14 open border cities achievements significantly	1.135
china 's 14 open border cities economic remarkable achievements	1.083
china 's 14 open border cities economic construction remarkable achievements	1.072
china 's 14 open border cities significant achievement in economic construction	1.026
china 14 open border cities achievements remarkable	0.983
china 's 14 open border cities building remarkable achievements	0.940
china 's 14 open border cities construction remarkable achievements	0.939
china 's 14 open border cities , remarkable achievements	0.918
china 14 open border cities building remarkable achievements	0.777

Semantic similarity for confidence estimation

The functions described above quantify the degree of similarity between a pair of sentences. This adds another to our growing list of sentence-similarity metrics: word-error rate, NIST, etc, and could in principle be employed in the same manner as any of them. One could, for example, use the semantic metric to find the center hypothesis in an N-best list and then let the similarity of this hypothesis to each sentence in the list be a confidence feature. One could even try using it to compare outputs to the reference translations and evaluate the system, though given the high degree of arbitrariness inherent in the measure, it seems very unlikely that this will produce useful results.

Due to the fact that this metric is extremely computationally expensive, the only feature that we computed was, for each sentence, the average of the similarity of that sentence to each of the top three sentences in the N-best list. The idea is that the top sentences are probably reasonably good translations, and so we'd like to identify sentences near in meaning to them, and we use three of them in hope of capturing some of the variance in the space of acceptable translations. This yields three features for each sentence, corresponding to the three averaging functions above.

Because computing the similarity between words is such a slow process, we employ a cache. Every pair of words that is compared is written into the cache along with its similarity value to minimize the time required to process them the next time they are encountered; since the words we are going to be processing are multiple translations of the same sentence, we can expect to encounter the same pairs of words many times, and thus save a lot of time by cacheing. For each N-best list, we clear the cache and start a new one; that way, the size of the cache indicates the semantic variance of the set of translation alternatives, roughly proportional to the total number of different content words in the entire N-best. As such, this cache size is sent to the ML layer as another confidence feature, taking the same value for every sentence of the list.

The features described in this section can be summarized as follows:

- log-linear average semantic score (B, ST, 86)
- harmonic mean semantic score (B, ST, 87)
- proportion of strongly-related words (B, ST, 88)
- word-pair cache size (B, ST, 89)

Not surprisingly, the semantic features do little to improve confidence estimates. Capturing quivalences between words, which this technique sets out to capture, is at the core of what the statistical MT system does. Attempting to do this *a priori* at least with this approach requires a great deal of guesswork and intuition, so we expect it to be very noisy. The cache size feature tends to be slightly more useful than the semantic similarity ones, probably largely because cache size is also roughly proportional to sentence length, a feature whose relevance is well-established.

3.1.16 Feature Summary

Table 3.3 contains a list of all features described in this section. The *sect* column gives the subsection in which each feature is described, and the B/N and S/T columns give its base-model and source/target dependence classifications.

	feature	sect	B/N	S/T		name	sect	B/N	S/T
1	nbestfeat.0	2	B	ST	48	NBestRank.0	2	B	ST
2	nbestfeat.1	2	B	S	49	all-quartile.0	5	N	S
3	nbestfeat.2	2	B	S	50	all-quartile.1	5	N	S
4	nbestfeat.3	2	B	S	51	all-quartile.2	5	N	S
5	nbestratiofeat.0	2	B	ST	52	all-quartile.3	5	N	S
6	searchfeat.0	3	B	S	53	all-quartile.4	5	N	S
7	searchfeat.1	3	B	S	54	all-quartile.5	5	N	S
8	searchfeat.2	3	B	S	55	all-quartile.6	5	N	S
9	searchfeat.3	3	B	S	56	all-quartile.7	5	N	S
10	searchfeat.4	3	B	S	57	all-quartile.8	5	N	S
11	word-ngram-0-6.0	8	B	ST	58	all-quartile.9	5	N	S
12	word-ngram-0-6.1	8	B	ST	59	all-quartile.10	5	N	S
13	word-ngram-0-6.2	8	B	ST	60	all-quartile.11	5	N	S
14	word-ngram-0-6.3	8	B	ST	61	all-lmscore.0	6	N	S
15	word-ngram-0-6.4	8	B	ST	62	all-lmscore.1	6	N	S
16	word-ngram-0-6.5	8	B	ST	63	all-lmscore.2	6	N	S
17	word-ngram-6-6.0	8	B	ST	64	all-en_lmscore.0	8	N	T
18	word-ngram-6-6.1	8	B	ST	65	all-en_lmscore.1	8	N	T
19	word-ngram-6-6.2	8	B	ST	66	all-en_lmscore.2	8	N	T
20	word-ngram-6-6.3	8	B	S	67	wordcenthyp.0	9	B	ST
21	word-ngram-6-6.4	8	B	S	68	wordcenthyp.1	9	B	ST
22	word-ngram-6-6.5	8	B	S	69	wordcenthyp.2	9	B	ST
23	atal-ngram-0-6.0	14	B	ST	70	align-atal.0	13	B	ST
24	atal-ngram-0-6.1	14	B	ST	71	align-word.0	13	B	ST
25	atal-ngram-0-6.2	14	B	ST	72	PsAndQs.0	10	N	T
26	atal-ngram-0-6.3	14	B	ST	73	PsAndQs.1	10	N	T
27	atal-ngram-0-6.4	14	B	ST	74	avg-nbestwordfeat.0	7	N	T
28	atal-ngram-0-6.5	14	B	ST	75	avg-nbestwordfeat.1	7	B	ST
29	atal-ngram-6-6.0	14	B	ST	76	avg-nbestwordfeat.2	7	B	ST
30	atal-ngram-6-6.1	14	B	ST	77	avg-nbestwordfeat.3	7	B	ST
31	atal-ngram-6-6.2	14	B	ST	78	avg-nbestwordfeat.4	7	B	ST
32	atal-ngram-6-6.3	14	B	ST	79	avg-nbestwordfeat.5	7	B	ST
33	atal-ngram-6-6.4	14	B	ST	80	avg-nbestwordfeat.6	7	B	ST
34	atal-ngram-6-6.5	14	B	ST	81	sent-length.0	4	N	S
35	BaseScore.0	1	B	ST	82	sent-length.1	4	N	T
36	BaseFeatures.0	1	B	ST	83	sent-length.2	4	N	ST
37	BaseFeatures.1	1	B	ST	84	word-align.0	12	B	ST
38	BaseFeatures.2	1	B	ST	85	word-align.1	12	B	ST
39	BaseFeatures.3	1	B	ST	86	semsim.0	15	B	ST
40	BaseFeatures.4	1	B	T	87	semsim.1	15	B	ST
41	BaseFeatures.5	1	B	T	88	semsim.2	15	B	ST
42	BaseFeatures.6	1	B	T	89	semsim.3	15	B	ST
43	BaseFeatures.7	1	B	T	90	model1.0	11	N	ST
44	BaseFeatures.8	1	B	ST	91	model1.1	11	N	ST
45	BaseFeatures.9	1	B	ST					
46	BaseFeatures.10	1	B	ST					
47	BaseFeatures.11	1	B	T					

Table 3.3: List of all features used for sentence-level CE.

3.2 Evaluation

In this section we present the results of experiments carried out to test the performance of various confidence estimation techniques, independent of any particular application. The experiments are divided into four categories:

1. Comparison of MLPs (the most poweful classification technique we used), trained on all features, under different problem settings and model configurations.
2. Learning curves for a subset of the MLPs.
3. Feature-attribution experiments designed to assess the contribution of individual features and various classes of features.
4. Comparison of MLP and naive Bayes classification techniques.

These experiments are described in detail in the following four sections.

3.2.1 MLP Experiments

All experiments described in this section were carried out on the standard training and test sets described in section 2.1, with nbest list sizes limited to at most 1000. Tests were run under all four problem settings described in section 2.2: using the NIST and WERg error metrics to measure translation quality, with thresholds set so as to give 5% and 30% correct examples in the training corpus. Table 3.4 shows the actual thresholds, and gives the corresponding proportions in the *test* corpus, which constitute baseline values for the classification error rate scores we quote below.

metric	thresh	% corr in training	% corr in test
NIST	10.023	5	3.21
	8.149	30	32.50
WER	0.4193	5	5.65
	0.5720	30	32.50

Table 3.4: Problem settings used for MLP experiments: thresholds and resulting proportions of translations deemed correct.

All models were trained with the full set of features described in the previous section, except the two IBM1-based ones (which were implemented too late for inclusion in these tests). Model configurations were varied in the following ways:

- source-specific feature normalization (as described in section 2.2.3) or lack thereof
- classification versus regression
- number of hidden units: 0, 1, 5, 10, or 20

The classification models were evaluated using the IROC, CER, and NCE metrics described in section 2.5; regression models were evaluated using only IROC and CER. For completeness, results for all tests are presented in tables 3.7 (for classification) and 3.8 (for regression) at the end of this section; each table is organized into four blocks, corresponding to the four problem settings. We discuss various aspects of the results under the headings that follow.

Source-Specific Feature Normalization

The most striking result is that normalizing features separately for each source sentence (lines with Y in the *norm* columns in tables 3.7 and 3.8) does not appear to be a good idea. Models trained with this technique are systematically and significantly worse across all configurations and metrics. This suggests that features having wide variance over source sentences are useful when attempting to judge quality according to an absolute standard. A somewhat stronger version of this conclusion is that it may occasionally be useful for the model to discard *all* generated translations for a given source sentence if some highly-indicative criterion such as base model probability is low enough. We have not investigated this hypothesis in detail.

Classification versus Regression

Table 3.5 shows the performance of the best configurations for classification and regression on all four problem settings. Classification gives better IROC scores on all problems, with significant differences in three out of four cases. This is not surprising, given that the classification MLPs were specifically trained for the given thresholds, whereas the regression MLPs are specific only to the error metric. For CER, the results are much less clear: the classification MLPs are better with 30% thresholds, and worse with 5% thresholds, but only one of these differences is significant (30% NIST). Although neither class of MLPs was trained to optimize CER, the squared-error loss for the regression MLPs appears to be able to partially compensate for lack of prior knowledge of the threshold for determining correctness.

metric	thresh	IROC	CER	ΔCER	NCE
NIST	5% corr	.800 ± 0.002	3.21 ± 0.03	0.0%	9.18 ± 0.61
	30% corr	.763 ± 0.001	27.10 ± 0.10	16.6%	15.62 ± 0.09
WER	5% corr	.818 ± 0.002	5.57 ± 0.05	1.4%	16.94 ± 0.21
	30% corr	.734 ± 0.001	28.95 ± 0.06	10.9%	12.09 ± 0.10
NIST	5% corr	.757 ± 0.002	3.10 ± 0.03	3.4%	
	30% corr	.762 ± 0.001	27.20 ± 0.10	16.3%	
WER	5% corr	.741 ± 0.002	5.54 ± 0.05	1.9%	
	30% corr	.723 ± 0.001	29.04 ± 0.04	10.6%	

Table 3.5: Best results for classification (above double line) and regression. The ΔCER column gives relative improvement in CER over the baseline. Note that the results on each line in this table are not necessarily generated by a single model.

Hidden units

It is difficult to discern a clear pattern of performance with number of hidden units. Looking only at the results without feature normalization, there appears to be a very slight trend across all models towards better CER and NCE results with higher numbers of hidden units, with the peak somewhere between 1 and 5. For IROC, this trend is reversed, with performance generally tailing off when more hidden units are used, and the overall peak at 0. Many of the differences involved in these comparisons are not statistically significant. Since MLPs with 0 hidden units form a distinct class of models (having linear decision surfaces), it would be interesting to show that they give clearly better or worse performance than MLPs with one or more hidden units on this problem. Unfortunately, it does not seem possible to substantiate any general concusions about this from our data.

Problem Settings

Direct comparison of results for the NIST and WERg settings is difficult with the 5% threshold experiments, because the test-set priors are quite different (3.21% and 5.65% respectively). On the 30% threshold experiments, NIST seems slightly easier to learn than WERg, with better results given by the optimal configurations for all error metrics. Comparing the 5% and 30% experiments, we note that, as expected, the different priors have no systematic effect on IROC and NCE measurements. CER is of course much higher for the more difficult 30% experiments; however, as can be seen from table 3.5, the relative improvement over the baseline is uniformly better in this setting than for the 5% experiments.

metric	thresh	model	IROC	CER	NCE
NIST	5% corr	C 20	0.800 ± 0.002	3.21 ± 0.03	9.18 ± 0.37
	30% corr	C 01	0.763 ± 0.001	27.10 ± 0.10	15.62 ± 0.09
WER	5% corr	C 01	0.818 ± 0.002	5.60 ± 0.05	16.94 ± 0.21
	30% corr	C 00	0.734 ± 0.001	28.95 ± 0.06	12.00 ± 0.09

Table 3.6: Best-performing MLP model for each problem setting. The *C* in the *model* column stands for classification, and the number beside it gives how many hidden units were used.

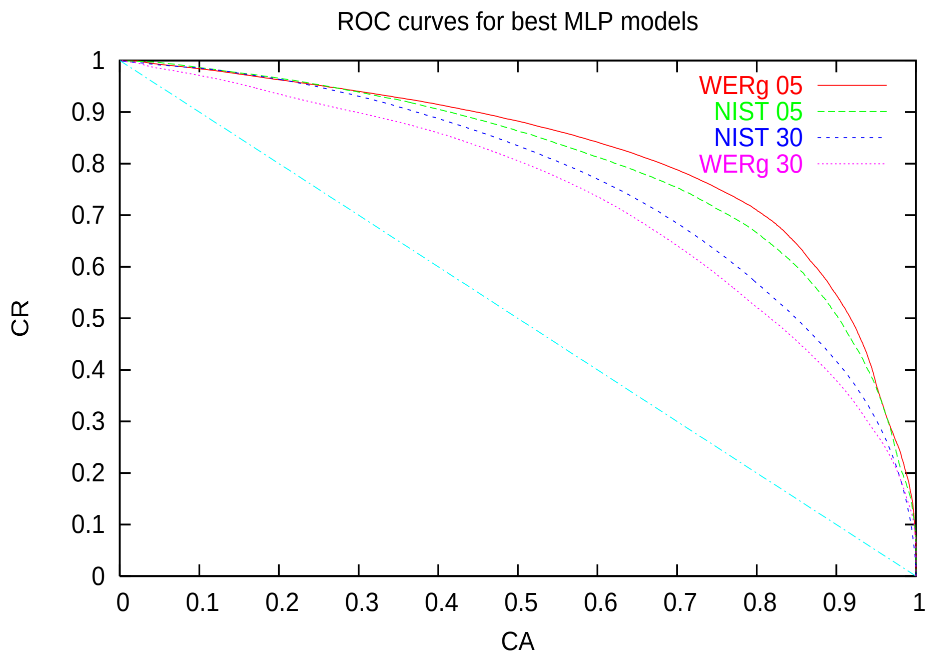


Figure 3.1: ROC curves for the models shown in table 3.6.

Best Results

Table 3.6 shows the performances of the best models for each problem setting, and figure 3.1 contains the corresponding ROC curves. Although the three evaluation metrics we used are fairly well correlated, choosing a single best model for each problem setting required resolving some conflicts among them. To do this, we averaged all three metrics (normalized appropriately) to give a combined measure. For regression models, we assumed an NCE value for the purposes of this comparison that was the average over all values generated by the classification models. Despite this, all of the winning models turned out to be classification models; this result does not change when NCE is dropped entirely from the comparison. In all cases, except CER for the NIST 5% setting, results are significantly better than the baseline.

metric	thresh	norm	nhu	IROC	CER	NCE
NIST	5% corr	N	00	0.795 ± 0.003	3.21 ± 0.03	4.49 ± 0.61
		N	01	0.789 ± 0.003	3.21 ± 0.03	8.30 ± 0.37
		N	05	0.796 ± 0.003	3.21 ± 0.03	8.44 ± 0.37
		N	10	0.794 ± 0.003	3.21 ± 0.03	7.67 ± 0.39
		N	15	0.785 ± 0.002	3.21 ± 0.03	7.44 ± 0.43
		N	20	●0.800 ± 0.002	3.21 ± 0.03	●9.18 ± 0.37
		Y	00	0.698 ± 0.004	3.21 ± 0.03	-2.44 ± 0.50
		Y	01	0.696 ± 0.004	3.21 ± 0.03	0.52 ± 0.31
		Y	05	0.706 ± 0.004	3.21 ± 0.03	0.49 ± 0.35
		Y	10	0.705 ± 0.004	3.21 ± 0.03	0.72 ± 0.33
		Y	15	0.709 ± 0.004	3.21 ± 0.04	1.17 ± 0.34
		Y	20	●0.716 ± 0.005	3.21 ± 0.03	●1.51 ± 0.40
	30% corr	N	00	0.762 ± 0.001	27.1 ± 0.1	15.01 ± 0.09
		N	01	●0.763 ± 0.001	●27.1 ± 0.1	●15.62 ± 0.09
		N	05	0.739 ± 0.001	28.6 ± 0.1	12.41 ± 0.12
		N	10	0.742 ± 0.001	28.3 ± 0.1	11.94 ± 0.14
		N	15	0.728 ± 0.001	29.1 ± 0.1	10.15 ± 0.09
		N	20	0.726 ± 0.001	28.5 ± 0.1	9.67 ± 0.19
		Y	00	0.732 ± 0.001	28.5 ± 0.1	10.96 ± 0.10
		Y	01	●0.733 ± 0.001	●28.3 ± 0.1	●11.83 ± 0.09
		Y	05	0.728 ± 0.001	28.8 ± 0.1	11.60 ± 0.11
		Y	10	0.720 ± 0.002	29.3 ± 0.1	10.87 ± 0.12
		Y	15	0.733 ± 0.001	29.5 ± 0.1	11.77 ± 0.11
		Y	20	0.711 ± 0.001	30.0 ± 0.1	9.60 ± 0.10
WER	5% corr	N	00	●0.818 ± 0.002	5.59 ± 0.05	15.99 ± 0.23
		N	01	0.818 ± 0.002	5.60 ± 0.05	●16.94 ± 0.21
		N	05	0.808 ± 0.002	5.65 ± 0.05	15.26 ± 0.19
		N	10	0.803 ± 0.002	●5.57 ± 0.05	13.80 ± 0.22
		N	15	0.813 ± 0.002	5.65 ± 0.05	14.23 ± 0.24
		N	20	0.815 ± 0.002	5.65 ± 0.05	15.30 ± 0.26
		Y	00	●0.799 ± 0.002	5.65 ± 0.05	●13.89 ± 0.20
		Y	01	0.789 ± 0.002	5.65 ± 0.05	13.70 ± 0.14
		Y	05	0.758 ± 0.001	5.65 ± 0.05	9.09 ± 0.12
		Y	10	0.774 ± 0.001	5.65 ± 0.05	11.31 ± 0.15
		Y	15	0.781 ± 0.002	○5.64 ± 0.05	11.78 ± 0.17
		Y	20	0.776 ± 0.001	5.64 ± 0.05	11.06 ± 0.16
	30% corr	N	00	●0.734 ± 0.001	●28.95 ± 0.06	12.00 ± 0.09
		N	01	0.730 ± 0.001	29.31 ± 0.06	●12.09 ± 0.10
		N	05	0.723 ± 0.001	29.25 ± 0.12	11.10 ± 0.16
		N	10	0.719 ± 0.001	30.09 ± 0.06	10.49 ± 0.14
		N	15	0.718 ± 0.001	29.50 ± 0.07	10.29 ± 0.15
		N	20	0.702 ± 0.001	29.76 ± 0.08	7.00 ± 0.20
		Y	00	●0.715 ± 0.001	29.87 ± 0.07	●10.17 ± 0.10
		Y	01	0.708 ± 0.001	29.94 ± 0.08	9.72 ± 0.09
		Y	05	0.702 ± 0.002	30.79 ± 0.06	9.01 ± 0.10
		Y	10	0.694 ± 0.002	30.51 ± 0.08	8.09 ± 0.14
		Y	15	0.712 ± 0.001	●29.31 ± 0.09	9.82 ± 0.14
		Y	20	0.705 ± 0.002	29.79 ± 0.06	9.62 ± 0.13

Table 3.7: MLP classification results. Circles mark best results; these are shaded when significantly better than the baseline.

metric	thresh	norm	nhu	IROC	CER
NIST	5% corr	N	00	0.740 ± 0.002	3.21 ± 0.03
		N	01	0.753 ± 0.001	3.21 ± 0.03
		N	05	•0.757 ± 0.002	•3.10 ± 0.03
		N	10	0.754 ± 0.001	3.21 ± 0.03
		N	15	0.743 ± 0.002	3.21 ± 0.03
		N	20	0.720 ± 0.002	3.20 ± 0.03
		Y	00	•0.675 ± 0.001	3.21 ± 0.03
		Y	01	0.670 ± 0.002	3.21 ± 0.03
		Y	05	0.658 ± 0.002	○3.21 ± 0.03
		Y	10	0.653 ± 0.001	3.21 ± 0.03
		Y	15	0.649 ± 0.002	3.21 ± 0.03
		Y	20	0.641 ± 0.003	3.21 ± 0.03
	30% corr	N	00	•0.762 ± 0.001	•27.2 ± 0.1
		N	01	0.758 ± 0.001	27.9 ± 0.1
		N	05	0.755 ± 0.001	27.8 ± 0.1
		N	10	0.741 ± 0.001	27.6 ± 0.1
		N	15	0.733 ± 0.001	28.7 ± 0.1
		N	20	0.713 ± 0.001	29.6 ± 0.1
		Y	00	•0.738 ± 0.001	•28.1 ± 0.1
		Y	01	0.736 ± 0.001	28.3 ± 0.1
		Y	05	0.733 ± 0.001	28.4 ± 0.1
		Y	10	0.714 ± 0.001	30.1 ± 0.1
		Y	15	0.714 ± 0.001	29.6 ± 0.1
		Y	20	0.705 ± 0.001	30.0 ± 0.1
WER	5% corr	N	00	•0.741 ± 0.002	5.65 ± 0.05
		N	01	0.725 ± 0.002	5.65 ± 0.05
		N	05	0.722 ± 0.004	•5.54 ± 0.05
		N	10	0.714 ± 0.003	5.65 ± 0.05
		N	15	0.690 ± 0.003	5.64 ± 0.05
		N	20	0.689 ± 0.003	5.65 ± 0.05
		Y	00	•0.729 ± 0.003	○5.64 ± 0.05
		Y	01	0.725 ± 0.003	5.65 ± 0.05
		Y	05	0.701 ± 0.003	5.65 ± 0.05
		Y	10	0.692 ± 0.003	5.65 ± 0.05
		Y	15	0.681 ± 0.003	5.65 ± 0.05
		Y	20	0.673 ± 0.003	5.65 ± 0.05
	30% corr	N	00	•0.723 ± 0.001	29.10 ± 0.11
		N	01	0.710 ± 0.001	29.89 ± 0.06
		N	05	0.708 ± 0.001	•29.04 ± 0.04
		N	10	0.686 ± 0.001	30.25 ± 0.05
		N	15	0.674 ± 0.001	30.82 ± 0.11
		N	20	0.674 ± 0.001	30.81 ± 0.12
		Y	00	•0.700 ± 0.002	29.95 ± 0.07
		Y	01	0.695 ± 0.002	30.03 ± 0.08
		Y	05	0.682 ± 0.002	•29.75 ± 0.06
		Y	10	0.679 ± 0.001	30.54 ± 0.07
		Y	15	0.662 ± 0.002	31.02 ± 0.08
		Y	20	0.660 ± 0.002	31.39 ± 0.08

Table 3.8: MLP regression results. Circles mark best results; these are shaded when significantly better than the basline.

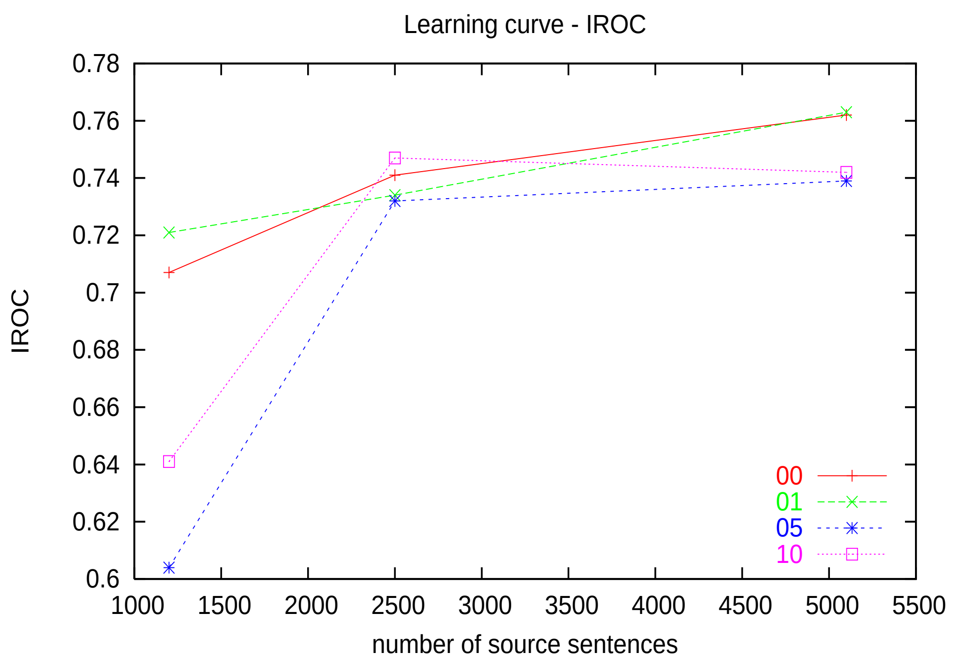
training size	nhu	IROC	CER	NCE
1200	00	0.707	30.26	-6.53
	01	0.721	29.10	1.92
	05	0.604	31.57	-39.81
	10	0.641	31.74	-28.18
2500	00	0.741	28.24	12.00
	01	0.734	28.48	12.25
	05	0.732	28.77	11.31
	10	0.747	28.04	13.64
5100	00	0.762	27.12	15.01
	01	0.763	27.09	15.62
	05	0.739	28.61	12.41
	10	0.742	28.29	11.94
5100(2k)	00	0.761	27.23	13.87
	01	0.757	27.30	14.53
	05	0.733	28.99	8.03
	10	0.741	28.59	9.32

Table 3.9: Learning curve data for the NIST 30% setting. The first column gives the number of source sentences used to generate nbest lists, and the second the number of hidden units in the models. The block labelled 5100(2k) shows results for nbest lists of maximum size 2,000 instead of the standard 1,000.

3.2.2 Learning Curves

Although our training data consisted of on the order of a million individual examples, these were generated from a much more modest collection of only 5100 source sentences. An interesting question is therefore the extent to which we might expect performance to increase with more training material. To assess this, we trained MLPS with 0, 1, 5, and 10 hidden units on two smaller subsets of our corpus, consisting of 1200 and 2500 source sentences. We also trained on the full 5100-sentence set, but using 2000-best lists rather than the usual 1000-best lists. All these experiments were performed with the same feature set as in the last section, with the exception of the 2000-best tests, which are minus three features that were too expensive to generate for the longer nbest size. To facilitate analysis of the results, we ran these tests only for the NIST 30% problem setting.

Results are shown in table 3.9, and learning curves are plotted in figure 3.2 for IROC and NCE (the curves for CER display similar behaviour). For 1000-best lists, all metrics exhibit a common pattern in which the curves



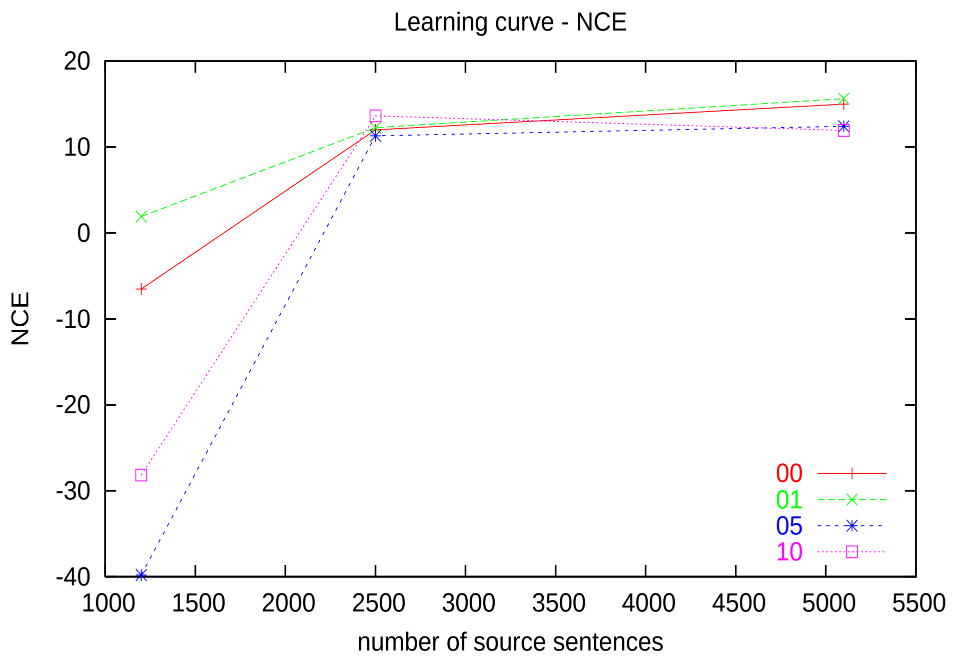


Figure 3.2: Learning curves for IROC and NCE metrics. Each curve corresponds to a model with a given number of hidden units, as indicated on the label.

for the models with 0 and 1 hidden units are similar, as are those for the models with 5 and 10 hidden units. The 5/10 curves appear to flatten at 5100 source sentences, whereas the 0/1 curves continue to grow slowly. Although more data points would be required for certainty, this behaviour is opposite to what one would expect if the higher-capacity models were overfitting the training sets to a greater degree than the lower capacity ones. Since we attempted to control overfitting by early stopping, this may indicate that the measures we took were applied too aggressively to the higher-capacity models on larger data sets. In any case, it is encouraging that at least the lower-capacity models show the potential to benefit from more data.

Using 2000-best lists leads to a slight drop in performance compared to 1000-best lists for the same number of source sentences (5100). The almost twofold increase in the quantity of data appears to be outweighed by the dissimilarity between the last 1000 candidates in these longer lists and the 1000-best lists which comprise our test corpus.

3.2.3 Feature Attribution

One of the central goals of our workshop was to try to determine which features or classes of features are most valuable in discriminating between correct and incorrect MT output. In this section we describe three experiments to gauge contributions:

- Using single features as discriminant scores.
- Training MLP models with single features removed.
- Training MLP models using the groups of features identified in section 3.1.

Each of these is described under one of the headings below. In all cases, tests were run only for the NIST 30% problem setting.

Single Feature Discrimination

Since all our features are numeric, they can be used directly as confidence scores. This approach constitutes an important baseline, because unlike the methods discussed previously, it is an unsupervised technique and hence applicable even in the absence of examples labelled with correctness information. As with regression, only the IROC and CER metrics can be calculated for these tests, since raw feature values do not provide probability estimates.

Table 3.10 shows the 20 best and 20 worst features according to IROC. (Features that are negatively correlated with MT quality—having an original IROC value less than .5—are shown with negative scores = 1−original IROC.) Several observations are noteworthy. First, the best single-feature IROC score is 0.648, a 15% relative degradation compared to the best MLP model for this setting. CER scores appear to roughly correlated with IROC scores, and are uniformly low; the majority are not below the baseline, and the lowest value is 30.22%, significantly higher than the 27.1% achieved by the best MLP model.

In terms of the feature classes identified in section 3.1, the clearest pattern is that three of the best features are target-dependent (class T), whereas none of the worst features are in this class. In fact, the majority of the best features (including all the top 5) apply directly to the target hypothesis, although most are not formally classified as T because they exploit supplementary information such as the nbest list. Another observation is that all of the top 10 models depend on the base model, although only one feature in the top 20 is a direct base-model score. Finally, 16 of the best 20 models use the nbest list in some way, but only 8 of the worst 20 do. To summarize these observations: successful features tend to be ones that rely on information derived from the base model, particularly those that look at the whole nbest list; and they also tend to focus on properties of the target sentence.

Complemented-Feature Models

The results in the previous section measured the worth of each feature on its own. Another way to assess the value of a feature is to measure its contribution when used with other features. To do this, we trained models in which single features were excluded. The difference in performance between the full model and one in which a particular feature is excluded gives an indication of how much this feature contributes in the presence of all others.

Table 3.11 displays the 20 best and 20 worst features on this test, as measured by IROC. In this case, since we are concerned with the *decrease* in IROC when each feature is omitted, lower IROC scores indicate better features. Interestingly, the intersection between the 20 best features here and the 20 best on the single-feature tests is fairly small: only six features appear in both lists. The complemented-feature test tends to favour both the base model's feature functions and features that are completely independent of the base model; this is understandable, as features in both these classes are a priori more likely to be orthogonal to each other and to other features in the model.

	feature	class	IROC	CER
26	atal-ngram-0-6.3	B ST	0.648 ± 0.001	32.14 ± 0.08
78	avg-nbestwordfeat.4	B ST	-0.647 ± 0.001	32.51 ± 0.10
75	avg-nbestwordfeat.1	B ST	-0.647 ± 0.001	32.51 ± 0.10
79	avg-nbestwordfeat.5	B ST	-0.647 ± 0.001	32.51 ± 0.10
76	avg-nbestwordfeat.2	B ST	-0.647 ± 0.001	32.51 ± 0.10
04	nbestfeat.3	B S	0.645 ± 0.014	30.22 ± 0.10
69	wordcenthyp.2	B ST	-0.644 ± 0.001	32.51 ± 0.10
18	word-ngram-6-6.1	B ST	0.639 ± 0.001	32.44 ± 0.10
21	word-ngram-6-6.4	B S	0.637 ± 0.001	31.51 ± 0.11
68	wordcenthyp.1	B ST	-0.637 ± 0.003	32.51 ± 0.10
19	word-ngram-6-6.2	B ST	0.631 ± 0.001	31.72 ± 0.10
82	sent-length.1	N T	-0.626 ± 0.002	32.51 ± 0.10
20	word-ngram-6-6.3	B S	-0.625 ± 0.001	32.51 ± 0.10
02	nbestfeat.1	B S	-0.625 ± 0.001	32.51 ± 0.10
47	BaseFeatures.11	B T	0.624 ± 0.002	32.43 ± 0.10
91	model1.1	N ST	-0.624 ± 0.001	32.51 ± 0.10
06	searchfeat.0	B S	0.623 ± 0.001	32.51 ± 0.10
64	all-en_lmscore.0	N T	0.622 ± 0.001	32.41 ± 0.10
22	word-ngram-6-6.5	B S	0.621 ± 0.001	32.19 ± 0.09
11	word-ngram-0-6.0	B ST	-0.621 ± 0.001	32.51 ± 0.10
86	semsim.0	B ST	-0.528 ± 0.001	32.51 ± 0.10
33	atal-ngram-6-6.4	B ST	0.527 ± 0.001	32.31 ± 0.07
90	model1.0	N ST	-0.526 ± 0.001	32.51 ± 0.10
05	nbestratiofeat.0	B ST	0.526 ± 0.001	32.51 ± 0.10
29	atal-ngram-6-6.0	B ST	-0.523 ± 0.001	32.51 ± 0.10
54	all-quartile.5	N S	0.523 ± 0.014	32.32 ± 0.29
25	atal-ngram-0-6.2	B ST	-0.521 ± 0.001	32.51 ± 0.10
39	BaseFeatures.3	B ST	0.518 ± 0.017	32.51 ± 0.10
35	BaseScore.0	B ST	0.517 ± 0.001	32.51 ± 0.10
51	all-quartile.2	N S	0.512 ± 0.018	32.35 ± 0.26
67	wordcenthyp.0	B ST	-0.512 ± 0.007	32.51 ± 0.10
48	NBestRank.0	B ST	0.511 ± 0.001	32.51 ± 0.10
01	nbestfeat.0	B ST	0.511 ± 0.001	32.51 ± 0.10
10	searchfeat.4	B S	0.506 ± 0.001	32.46 ± 0.10
08	searchfeat.2	B S	0.505 ± 0.001	32.51 ± 0.10
24	atal-ngram-0-6.1	B ST	-0.505 ± 0.001	32.51 ± 0.10
36	BaseFeatures.0	B ST	0.503 ± 0.001	32.51 ± 0.10
09	searchfeat.3	B S	0.502 ± 0.001	32.46 ± 0.09
55	all-quartile.6	N S	0.501 ± 0.005	32.51 ± 0.10
52	all-quartile.3	N S	0.500 ± 0.005	32.51 ± 0.10

Table 3.10: Single-feature discrimination results. Features are ordered by descending IROC, with the best 20 in the top box, and the worst 20 in the bottom box.

	feature	class	IROC
72	PsAndQs.0	N T	0.746 += 0.001
58	all-quartile.9	N S	0.748 += 0.001
59	all-quartile.10	N S	0.748 += 0.001
60	all-quartile.11	N S	0.748 += 0.001
30	atal-ngram-6-6.1	B ST	0.748 += 0.001
06	searchfeat.0	B S	0.749 += 0.001
31	atal-ngram-6-6.2	B ST	0.749 += 0.001
48	NBestRank.0	B ST	0.749 += 0.001
46	BaseFeatures.10	B ST	0.749 += 0.001
71	align-word.0	B ST	0.750 += 0.001
19	word-ngram-6-6.2	B ST	0.750 += 0.001
05	nbestratiofeat.0	B ST	0.750 += 0.001
40	BaseFeatures.4	B T	0.751 += 0.001
47	BaseFeatures.11	B T	0.751 += 0.001
20	word-ngram-6-6.3	B S	0.751 += 0.001
21	word-ngram-6-6.4	B S	0.751 += 0.001
35	BaseScore.0	B ST	0.751 += 0.001
39	BaseFeatures.3	B ST	0.751 += 0.001
22	word-ngram-6-6.5	B S	0.751 += 0.001
23	atal-ngram-0-6.0	B ST	0.751 += 0.001
83	sent-length.2	N ST	0.757 += 0.001
80	avg-nbestwordfeat.6	B ST	0.757 += 0.001
91	model1.1	N ST	0.757 += 0.001
61	all-lmscore.0	N S	0.758 += 0.001
90	model1.0	N ST	0.758 += 0.001
77	avg-nbestwordfeat.3	B ST	0.758 += 0.001
56	all-quartile.7	N S	0.758 += 0.001
09	searchfeat.3	B S	0.758 += 0.001
29	atal-ngram-6-6.0	B ST	0.758 += 0.001
12	word-ngram-0-6.1	B ST	0.758 += 0.001
81	sent-length.0	N S	0.758 += 0.001
32	atal-ngram-6-6.3	B ST	0.758 += 0.001
89	semsim.3	B ST	0.758 += 0.001
08	searchfeat.2	B S	0.758 += 0.001
84	word-align.0	B ST	0.759 += 0.001
88	semsim.2	B ST	0.759 += 0.001
63	all-lmscore.2	N S	0.759 += 0.001
57	all-quartile.8	N S	0.760 += 0.000
86	semsim.0	B ST	0.760 += 0.000
87	semsim.1	B ST	0.760 += 0.001

Table 3.11: Complemented-feature discrimination results. Features are ordered by *ascending* IROC, with the best 20 in the top box, and the worst 20 in the bottom box.

model	features
all	1-91
base model	36-47
base-dependent	1-10,23-48,70-71,84-89
base-independent	11-22,49-69,72-83,90-91
source	2-4,6-10,49-63
target	1,5,11-22,35,40-43,47-48,64-69,73-83
source and target	23-34,36-39,44-46,70-72,84-91

Table 3.12: Feature sets used for grouping experiments.

model	IROC	CER	NCE
all	0.763 ± 0.001	27.10 ± 0.10	15.62 ± 0.09
base model	0.746 ± 0.001	27.82 ± 0.09	13.97 ± 0.08
base-dependent	0.754 ± 0.001	28.02 ± 0.10	14.83 ± 0.10
base-independent	0.712 ± 0.001	29.88 ± 0.09	9.76 ± 0.09
source	0.687 ± 0.002	30.41 ± 0.07	7.23 ± 0.08
target	0.751 ± 0.001	27.91 ± 0.08	14.48 ± 0.09
source and target	0.746 ± 0.001	28.47 ± 0.10	13.62 ± 0.07
26 atal-ngram-0-6.3	0.648 ± 0.001	32.14 ± 0.08	

Table 3.13: Results for models trained on different feature groups.

Feature Groups

The final experiment described in this section involves training MLPs on various groups of features to determine which group achieves the best performance on its own. The original intent was to use the classes identified by the labels given in section 3.1 and table 3.3. Unfortunately, due to a misunderstanding, somewhat different groupings were used when the models were trained. The difference arises mainly from ignoring reliance on the nbest list when assigning a feature to a class. For instance, features 11-22 are considered to depend on the base model in the section 3.1 classification because they are ngram language models derived from the nbest list; but in the grouping used for the experiments described below, they are considered independent. Similar considerations affected the source/target distinctions. For reference purposes, table 3.12 lists the exact feature set used for each model in this experiment.

Table 3.13 presents the results for the feature-group models, together with the best result from the single-feature tests for comparison; figure 3.3

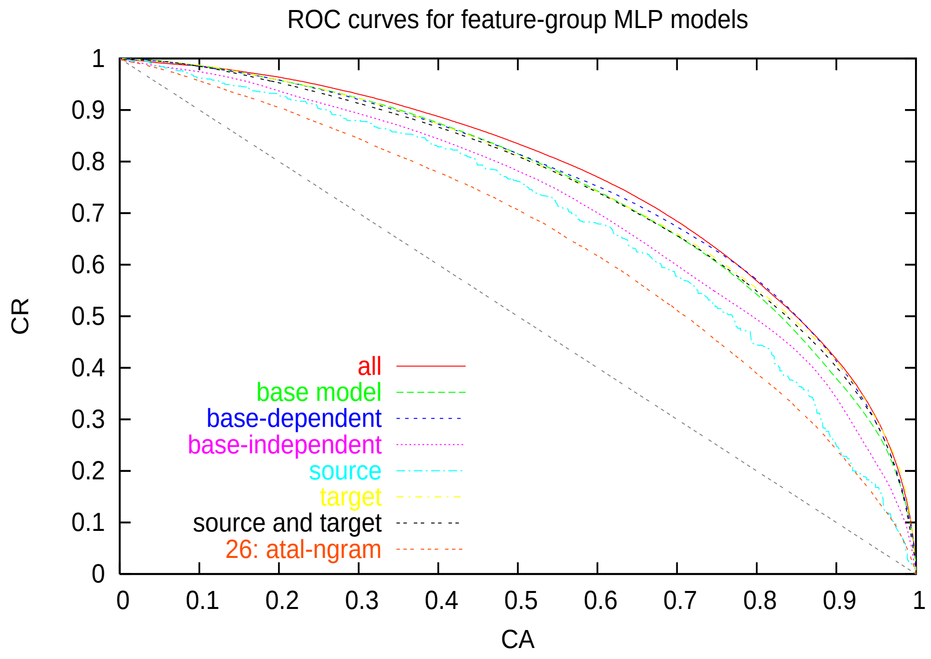


Figure 3.3: ROC curves for models trained on different feature groups. Note that the "wiggly" appearance of the curve for the *source* group is due to the fact that these features are invariant over all entries in a given nbest list, which leads the model to assign large blocks of examples exactly the same probability of correctness.

shows the corresponding ROC curves. The most striking aspect of these results is that the model trained on only the output from the twelve feature functions in the base model is almost as good as those trained on many more features; compared to the model trained on *all* features, it is only about 2% worse in relative IROC and CER, and 10% worse in relative NCE. The ROC curves reflect this small difference, especially for higher CA values.

Despite substantial differences in the class definitions, the results also strongly corroborate the main conclusions from the single-feature tests: features that depend on the base model are more useful that those that do not, and features that apply to the target hypothesis are more useful than ones that apply only to the source sentence (as well as, to a much lesser extent, those that apply to a source/target pair). Another clear conclusion is that a model that has been trained on labelled data—regardless of the feature set used—is better at discriminating than any single feature on its own.

model	IROC	CER
Naive Bayes	0.61043	23.42
MLP 00	0.73816	21.97
MLP 05	0.70336	22.48
MLP 10	0.67681	22.74
MLP 15	0.68898	22.83
MLP 20	0.69211	21.79

Table 3.14: MLP versus Naive Bayes

3.2.4 MLP versus Naive Bayes Models

In attempting to compare MLP and naive Bayes models on the full feature set, we ran into a practical problem with our implementation of naive Bayes: multiplying such a large number of class-conditional probabilities created precision problems that caused the model to essentially act as a switch and assign probabilities of either 1 or 0 to most examples. Because of this, we show in table 3.14 results generated from an earlier stage of the workshop, involving only about 40 features. These are not directly comparable to any other results in this section, but they do serve to indicate the clear superiority of the MLP models on this problem, a trend which is corroborated in the next chapter. This is not a completely surprising result, given that many sets of features described in the previous section clearly violate the independence assumption underlying the naive Bayes model.

3.3 Applications

The most important goal of confidence estimation is to enable filtering of correct or incorrect translation outputs of a SMT system. Therefore, confidence estimation can be considered succesful if the produced confidence scores are more discriminative then the scores produced by the base MT system. Discriminability results have already been presented in the previous section. In this section we investigate possible ways to use CE for a diferent goal, which is to improve the quality of the translation output. This is a much harder task and results are less promising.

We tried two different applications for sentence-level CE: *rescoring* and *model combination*. We performed our experiments using N-best translation lists provided by two different SMT systems (ISI and CMU, see section 2.1 for details) on parallel corpora.

Recall that N-best lists consist of up to N most probable translation alternatives for a given source sentence according to the SMT system. Translation alternatives are ranked according to a probabilistic score produced by the underlying SMT models. In our experiments $N = 1000$, although due to pruning of the search space certain N-best lists contained less then 1000 alternatives. We used these N-best lists either separately, for the rescoring experiments, or in parallel, for the model combination applications.

For each of the underlying SMT systems we trained separate sentence-level CE models to produce a confidence score for each individual translation alternative. The confidence scores are estimates of the posterior probability of correctness of a given translation alternative, obtained using the methods described in section 2.2. The data we used for the training and validation of our CE models were N-best translation lists as described in section 2.1.

We should note first that there was a fundamental problem in our use of the CMU data: we compared N-best translations tokenized according to the CMU tokenization scheme with reference translations tokenized according to the ISI tokenization algorithms. We realised only too late that there were considerable differences between the two tokenization methods. Thus, the evaluation of the CMU translation was artificially penalized. This problem affected both training and testing stages for both the reranking and model-combination experiments, and calls into question the validity of the results presented below for the CMU system. Results on the standard ISI base model are unaffected however.

ISI	BLEU	NIST	aps-NIST	WERg
Baseline	31.58 (± .84)	9.29 (± .11)	7.47	0.612
CE-NIST	31.08 (± .90)	9.20 (± .12)	7.67	0.619
CE-WERg	30.78 (± .85)	9.14 (± .12)	7.48	0.620
Oracle aps-NIST	41.77 (± .92)	9.21 (± .11)	9.51	0.538
Oracle WERg	39.62 (± .88)	9.21 (± .12)	8.56	0.465

Table 3.15: Rescoring on ISI N-best lists

3.3.1 Rescoring

The rescoring experiments were conducted separately for each SMT system. We reranked the N-best translation hypotheses according to their sentence-level confidence score. We then measured the impact on translation quality according to overall BLEU and NIST scores as well as on the average per-sentence NIST, named aps-NIST,[1] and WER-g scores.

Tables 3.15 and 3.16 present the results obtained. The baselines are performances of the ISI and CMU base SMT systems. The CE-NIST (respectively CE-WERg) results were obtained by rescoring according to the confidence score obtained by MLPs trained on a 5%-correct threshold over per-sentence NIST (respectively WERg) scores. Oracle results were obtained by systematically picking the optimal translation alternative according to a given evaluation metric and represent upper bounds on the possible performance improvement.

The results are rather disappointing: basically, no significant improvement was obtained by rescoring. This is not completely surprising given that the objective functions used for CE training are not the same functions used to measure performance on the reranking tests. It is noteworthy, however, that there *is* a small increase over the baseline ISI system on the metric (aps-NIST) that was most similar to the criterion used for CE training (in the case of NIST thresholding). If we had a true gold standard for MT correctness, and used it to train a CE system as well as to evaluate MT quality, we would expect far better results from CE-based reranking.

3.3.2 Model Combinations

For the model-combination experiments we compared two very basic schemes based on maximum base-model score or maximum confidence score voting: for each source sentence we compare the N-best alternatives procuced by

[1]Per-sentence NIST is the version used throughout the rest of this report.

CMU	BLEU	NIST	aps-NIST	WERg
Baseline	17.39 (± .81)	7.50 (± .11)	6.89	0.700
CE-NIST	17.86 (± .76)	7.18 (± .11)	6.73	0.721
CE-WERg	17.39 (± .78)	7.31 (± .12)	6.64	0.715
Oracle aps-NIST	22.96 (± .83)	8.59 (± .11)	8.55	0.675
Oracle WERg	21.17 (± .79)	7.86 (± .11)	7.52	0.608

Table 3.16: Rescoring on CMU N-best lists

the two SMT systems independently and picked the one with (1) the highest base-model score or (2) the one with the highest confidence score. As in the rescoring experiments, we determined any potential translation accuracy gain by measuring overall BLEU and NIST scores as well as the average per-sentence NIST and WER scores on the resulting translations.

These combination schemes are evidently sub-optimal. A lot of work has been done in statistical model combination and a more sophisticated approach, such as stacking [44] could yield much better results than simple maximum score voting.

We expected maximum base-model score voting scheme to degrade the translation accuracy compared to the output of the single best base SMT system. This is the case because sentence scores produced by the base SMT systems are not really comparable, despite that fact that they are supposed to be estimates of the same probability. What we hoped to achieve with the maximum confidence scoring scheme was to improve upon the accuracy of the best of the two individual systems. This hope was based on the fact that we expected the confidence scores to be better estimates of the probability of correctness than the sentence scores of the underlying models and therefore to be more reliable for comparing the quality of translations. Similar work presented in [11] yielded positive results. However, the tokenization mismatch problem in our setup appears to have had a severe effect on the combination results presented in 3.17 irrelevant. This problem is also reflected in the fact that the oracle scores for the combined nbest lists are only slightly higher than those for the ISI system on its own (no different at all for the WERg metric).

ISI + CMU	BLEU	NIST	aps-NIST	WER-g
Baseline	31.58 (± .84)	9.29 (± .11)	7.47	0.612
Norm. base score	17.63 (± .83)	7.53 (± .11)	6.90	0.619
CE-NIST	22.31 (± .99)	7.90 (± .14)	7.36	0.684
CE-WER	28.37 (± .91)	8.87 (± .13)	7.14	0.641
Oracle aps-NIST	41.77 (± .99)	9.52 (± .11)	9.80	0.538
Oracle WERg	39.62 (± .88)	9.21 (± .12)	8.61	0.465

Table 3.17: Maximum CE-score Model Combination

Chapter 4

Subsentence-Level Experiments

4.1 Motivation

Apart from sentence level confidence estimation, we also investigated confidence estimation on the word level. The motivation behind this was that often the sentence as a whole might be incorrect, but contain correct parts. For example, only 30% of the translations were rated 4 or 5 (i.e. acceptable or perfect) in our human evaluation exercise described in the next chapter. Nevertheless, they contain correct parts which we do not want to discard as incorrect. Moreover, the classification as correct or incorrect is easier on the sub-sentence level, because the concept of a correct translation is more intuitive on this level. There exist several possible applications for confidence estimation on the word level: In a post-editing scenario, it would be helpful to highlight incorrect words. The system output would be given to a human translator, and the words that have a low confidence would be tagged as possible errors such that the human translator could focus on them when correcting the text.
Another possibility is to apply confidence measures in an interactive translation environment where a system proposes translations of the input text and a human translator can either accept or correct them. The system then adapts its proposals according to the modifications by the human translator. In such an environment, the system would only output those words that have a high confidence and discard the others. Thus, it would spare the human translator time and effort for reading and correcting bad output. This approach to computer assisted translation is pursued for example in

the European project TransType2.
The third and most challenging application of sub-sentence level confidence estimation is that of search criteria based on confidence estimates. The translation hypotheses proposed by the system (represented e.g. in a word graph or an N best list) would be recombined in order to find a better translation than the one preferred by the translation system.

This chapter is organized as follows: First, we are going to introduce the features we implemented for the word level confidence estimation in Section 4.2. Section 4.3 contains a description of the different word error measures we investigated. Experimental results for the word level confidence estimation are presented in 4.4. We then discuss ideas for the recombination of translation hypotheses using the word level confidence estimates in Section 4.6.

4.2 Word Level Features

4.2.1 Target Language Based Features

Semantic Features

The use of the semantic data provided by WordNet was also investigated as a confidence feature at the word level. Since the semantic similarity features used at the sentence level were calculated by combining scores at the word level and it is not clear what the optimum method for combination is, we are optimistic that these semantic features will prove more useful at the word level than at the sentence level.

The first similarity feature used is the average semantic similarity from the word in question to the word aligned to the same position in each of the top three hypotheses. We denote by $B_i(n)$ the set of target positions aligned to the source word in position i in the nth best hypothesis of the N best list and by $B_j^{-1}(n)$ the set of source positions aligned to the target word in position j in the nth best hypothesis. Then we define the set $S_i(n,1)$ of words of the first sentence of the N best list that are aligned to the same position as word i of sentence n as:

$$S_i(n,1) = \{e_k | k \in B_j(1) \text{ and } j \in B_i^{-1}(n)\} \tag{4.1}$$

Then if $sim(e,f)$ is our semantic similarity metric mapping pairs of words to real numbers, then the confidence we assign to e_k of the nth sentence is

$$C(e_k) = \max_{x \in S_i(n,1)} \text{sim}(e_k, x) \tag{4.2}$$

More information about this feature, including the details of how $sim(e, f)$ is computed, can be found in section 3.1.15.

The two other features included for word-level confidence estimation come from WordNet's polysemy count. Simply put, for each word in the target sentences, this feature assigns a value equal to the number of different senses, counting all parts of speech, that the word has. The thought here is that words that can mean a number of different things might be more difficult to translate, and thus we might want to have lower confidence in them. Unfortunately, word polysemy is also a (fairly reliable) indicator of frequency–more common words tend to have more possible meanings, and so this feature ends up not being very enlightening.

The polysemy count including only senses that occur in tagged texts in WordNet's corpus is also used as a feature.

Parentheses and Quotation Marks

(`PsAndQs`) This basic syntax check looked to highlight hypotheses with mismatched parentheses and/or quotation marks. If such a syntax error was identified, these features would trigger and become non-zero.

Number of occurrences

For each word in the target sentence, we counted the number of times it occurs in the sentence. The idea behind this was that words that tend to appear more than once, such as determiners, are more likely to be correct than rare words.

4.2.2 Word Posterior Probabilities and Related Measures

The notation employed in this section is the following: We refer to a single word in the target sentence by e, and the whole target sentence is denoted as e_1^I. The source sentence is f_1^J, and the alignment mapping target positions to source positions is called B_1^I where B_i is the set of source positions aligned to target position i.

We investigated three different features that are calculated rather similarly: relative frequencies, rank weighted frequencies and word posterior probabilities. Consider a target word e and its set of aligned source positions B. We then determine those sentences in the N best list that 'match'

the pair (e, B) under certain conditions that are to be explained later in this section. Let the set of those sentences be $\mathcal{S}(e, B)$.
The relative frequency of the pair (e, B) in the N best list is then computed as

$$\frac{1}{N} \sum_{(e_1^I,\ B_1^I) \in \mathcal{S}(e,B)} 1 \ .$$

The rank weighted frequency is given as

$$\frac{2}{N(N+1)} \sum_{(e_1^I,\ B_1^I) \in \mathcal{S}(e,B)} (N + 1 - rank(e_1^I,\ B_1^I)) \ .$$

Here, we sum up the inverted ranks $N + 1 - rank(e_1^I, B_1^I)$, because we want an occurrence of the word in a hypothesis near to the top of the list to score better than on in the lower ranks. This value is normalized by the sum of all ranks in the list.
Let $p(e_1^I, B_1^I, f_1^J)$ the joint probability of source sentence f_1^J and the target sentence e_1^I with the according alignment B_1^I. The word posterior probability is calculated as the normalized sum of probabilities of all sentences in $\mathcal{S}(e, B)$:

$$\frac{1}{p(f_1^J)} \sum_{(e_1^I,\ B_1^I) \in \mathcal{S}(e,B)} p(e_1^I, B_1^I, f_1^J) \ . \tag{4.3}$$

The probability $p(f_1^J)$ is computed by summing the probabilities $p(e_1^I, B_1^I, f_1^J)$ over all sentence/alignment pairs (e_1^I, B_1^I) in the N best list.

For the set $\mathcal{S}(e, B)$ of 'matching' sentences, we implemented three different variants:

1. $\mathcal{S}(e, B) = \{(e_1^I,\ B_1^I) \mid e_i = e\}$
   That is, we sum over all sentences containing the word e in exactly the target position i. This is purely target language based and does not take the alignment B into account. This variant is the strictest one, because it requires the word to occur exactly in the given position i. Often, the same target word occurs in several hypotheses in the N best list, but in different positions due to reordering of words, insertions and deletions. This observation lead us to the second variant:

2. $\mathcal{S}(e, B) = \{(e_1^I,\ B_1^I) \mid \exists\ i : (e_i, B_i) = (e, B)\}$
   Here, all sentences are regarded where the word e is aligned to the source position(s) in B, i.e. this set of features is dependent on the translation model.

3. $\mathcal{S}(e, B) = \{(e_1^I,\ B_1^I) \mid \exists\, i : e_i = e\}$
   In this variant, all sentences are taken into account that contain the target word e, disregarding its position as well as its alignment B. This set of features depends only on the target language, but is by far less strict than 1.

Note that for variant 3, the sum of the sentence probabilities as performed in Eq. 4.3 does not result in a probability distribution. For variant 1 of $\mathcal{S}(e, B)$, we obtain a probability distribution over the pairs (e, i) of target words e and positions i in the target sentence. Variant 2 results in a probability distribution over pairs of target words e and their aligned set of source positions B.
For a detailed description of these confidence measures, see [41].

4.2.3 IBM Model 1

As described in section 3.1.11, IBM1 translation models were trained separately from the base translation model, and their probabilities were used as features at the sentence level. We also used IBM1 to derive a single feature at the word level. For a given target word e, this feature is just e's contribution to the total target probability:

$$p(e|\mathbf{f}) = \sum_{j=0}^{m} p(e|f_j)/(m+1)$$

where $\mathbf{f} = f_1, \ldots, f_m$ is the source sentence, and f_0 is the empty word.

4.2.4 SMT Model Based Features

Two features that are based directly on the bast Statistical MT model, namely the Alignment Template model ([32, 31]), were applied.

- Alignment Template containing this word: The Statistical Machine Translation system segments source and target sentence into bilingual phrases, the so-called Alignment Templates. This feature gives the identity of the Alignment Template that was applied in the translation of the current target word.

- For the translation of special phenomena such as dates and time expressions, a rule based system was integrated into the translation process. We implemented one feature specifying whether the target word was translated by this rule based system or not.

4.3 Word Error Measures

In Machine Translation, it is not intuitively clear how to classify words as correct of incorrect when comparing the translation to one or several references. We implemented a number of different measures for classifying single words in a translation hypothesis as correct or false. They were inspired by different automatic evaluation metrics like WER and PER.

- Pos: This error measure considers a word as correct if it occurs in exactly this target position in one of the reference translations.
- WER: A word is counted as correct if it is Levenshtein-aligned to itself in one of the references.
- PER: A word is tagged as correct if it occurs in one of the reference translations. Here, the reference is regarded as a bag of words, i.e. the number of occurrences per word is taken into account.
- Set is a less strict variant of PER: the number of occurrences per word is not considered, i.e. a word occurring in the translation three times is tagged as correct every time, even if the reference contains it only once or twice.
- n-gram: This metric considers the word as well as its $n-1$ predecessors in the hypothesis and labels only those words as correct that occur in the references together with this history. n was chosen to be 2, 3, and 4.

All error metrics except for n-gram exist in two variants: First, each translation hypothesis is compared to the pool of all references (i.e. four different reference translations for our corpus). Second, we determine that reference that has minimum distance to the hypothesis according to the metric under consideration and classify the words as correct or incorrect with respect to this reference. That is, under the metric PER for example, the pooled variant labels all those words as correct that occur in *any* of the references, whereas the second variant considers only those words correct that are contained in the *nearest* reference.
Table 4.2 in Section 4.4.1 shows the percentage of words that are labeled as correct according to the different error measures on the training, development and test corpora.

4.4 Experimental Results

4.4.1 Experimental Setup

For the sub-sentence level confidence estimation, we picked all those Chinese source sentences that were provided with four reference translations, leaving us with 1,871 sentences in total. The split into training, development and test corpus is given in Table 4.1. The test corpus is the same as the one used for the sentence level confidence estimation.
For each source sentence, we worked with the 1,000 best list provided by the statistical machine translation system, yielding a total of about 1.8 million target sentences in total over all three corpora.

Table 4.1: Corpus Statistics (using 1000 best lists).

	Sentences		Running Words
	Source	Target	Target
Training	700	698 082	20 736 971
Develop	293	292 870	7 492 753
Test	878	876 831	26 360 766

Table 4.2 shows the number of correct words under the different error metrics introduced in Section 4.3. We see that 'Pos' is a very pessimistic metric and considers only every fifth or sixth word correct, whereas the other metrics count much more words as correct, because they do not require them to occur in the exact position of the reference. The introduction of the Levenshtein alignment into the error measure yields a large increase in the number of words considered correct. This number grows further the more the error criterion is relaxed (see PER, Set). Naturally, the n-gram metric gets stricter the longer the history gets.
For the error measures existing in two variants, comparing to the pool of all references and the nearest reference, respectively, we see a significant reduction in the number of words labeled as correct if only the nearest reference is taken into account.

Note that those figures are not the translation errors for the system output. They are calculated for every hypothesis in the 1,000 best list (and not only for the single best translation).

Table 4.2: Correct words [%] in the corpora according to different error measures (pooled/nearest reference).

Error Measure	Training	Develop	Test
Pos	19.5 / 14.1	22.8 / 16.7	21.7 / 15.5
WER	63.1 / 42.2	61.2 / 43.4	62.3 / 42.5
PER	75.1 / 65.1	70.6 / 62.2	73.6 / 63.8
Set	81.5 / 71.0	77.4 / 67.6	80.7 / 70.0
2-/3-/4-gram	42.0/24.4/15.4	39.5/22.9/14.6	41.5/24.4/15.5

4.4.2 Performance Measures

We used the following three criteria for measuring the performance of the different confidence estimation methods and features:

- Confidence Error Rate (CER): The CER is the number of incorrectly assigned tags divided by the number of generated words in the translated sentence. The Baseline CER is determined by labeling all generated words as correct, i.e. it gives the ratio of substitutions and insertions in the translated sentence. Unlike in the previous section, we optimized the tagging threshold on a development corpus in order to get an unbiased estimate (rather than simply a measure useful for relative comparisons).

- Receiver Operating Characteristic (ROC) curve: The ROC plots the *correct rejection rate* versus *correct acceptance rate* for different values of the tagging threshold. The correct rejection rate is the number of incorrectly translated words that have been tagged as wrong, divided by the total number of incorrectly translated words. The correct acceptance rate is the ratio of correctly translated words that have been tagged as correct. These two rates depend on each other: If one of them is restricted by a lower bound, the other one cannot be restricted. See section 2.5 for more details about ROC curves.

- Area under ROC (AROC): This value specifies twice the size of the area between the ROC and the diagonal; it ranges from 0 to 1. The higher this value, the better the classifier discriminates.

4.4.3 Experimental Results for Single Features

Using the Naive Bayes classifier, we tested the performance of single features for word confidence estimation. Out of those, we chose the three features which yield best results and combined them with the same classifier. Finally, we integrated all 17 features into the Naive Bayes approach.
Table 4.3 shows the confidence estimation performance of single features in the Naive Bayes framework in terms of CER and AROC using the error measure PER. The features which yield the best results are the word posterior probability, rank weighted frequency, and relative frequency with respect to occurrence of the word in any position in the target sentence. Those three features show a very similar behavior and give a significant improvement over the baseline of more than 5% absolute in CER. The feature based on Model1 also discriminates very well, followed closely by the word posterior probabilities and frequencies with regard to the aligned source position(s). The combination of three of the best performing features, word posterior probabilities with respect to different criteria and the Model1 based feature, yields a significant improvement over the performance of any of the single features. This improvement is still increased if more information is added by combining all 17 features.

Figure 4.1 plots the ROC for the word error measure PER. It compares the MLP with 15 hidden units combining all features with the Naive Bayes classifiers using the single best, the three best and all features. It supports the analysis of the results contained in Table 4.3: The Naive Bayes approach gains from the integration of more features, but it is still outperformed by the MLP.

4.4.4 Comparison of Different Models

For word level confidence estimation, we investigated several different MLP architectures, with the number of hidden units ranging from 0 to 20.

Table 4.4 compares the tagging performance in terms of CER and AROC for MLP architectures and for the Naive Bayes classifier, including all features. We investigated them for three of the error measures described in Section 4.3: WER, PER and Set. All of those were computed with respect to the nearest reference. We see that the Naive Bayes classifier and the MLP with zero hidden units have a very similar performance for all three word error measures. But as soon as the MLP gets more complex by the addition of more hidden units, the MLP outperforms the Naive Bayes approach significantly. The MLPs with 10 or hidden 15 units perform best, but there

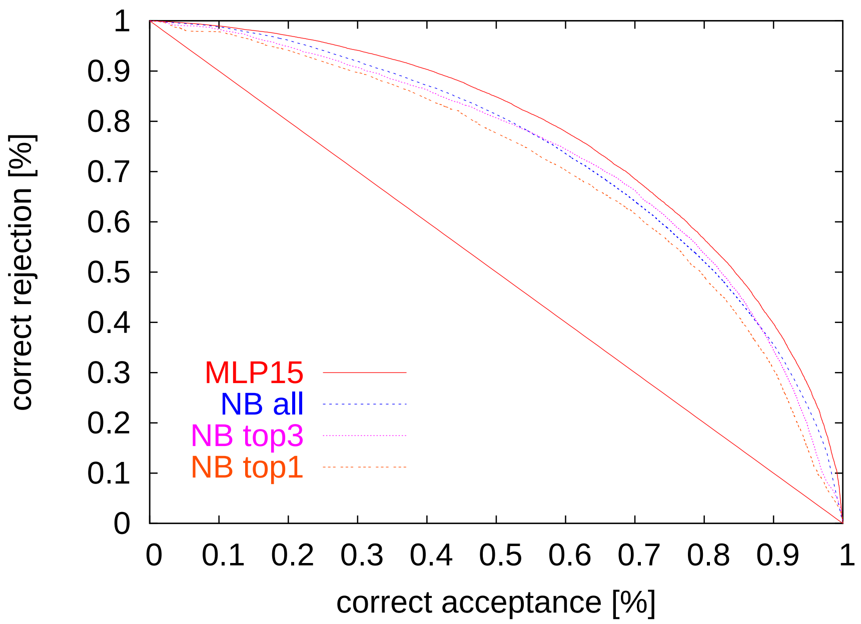


Figure 4.1: ROC for PER, Naive Bayes and MLP with 15 hidden units. The MLP combines all features, and Naive Bayes performance is shown for the single best feature, the top 3 and combination of all features.

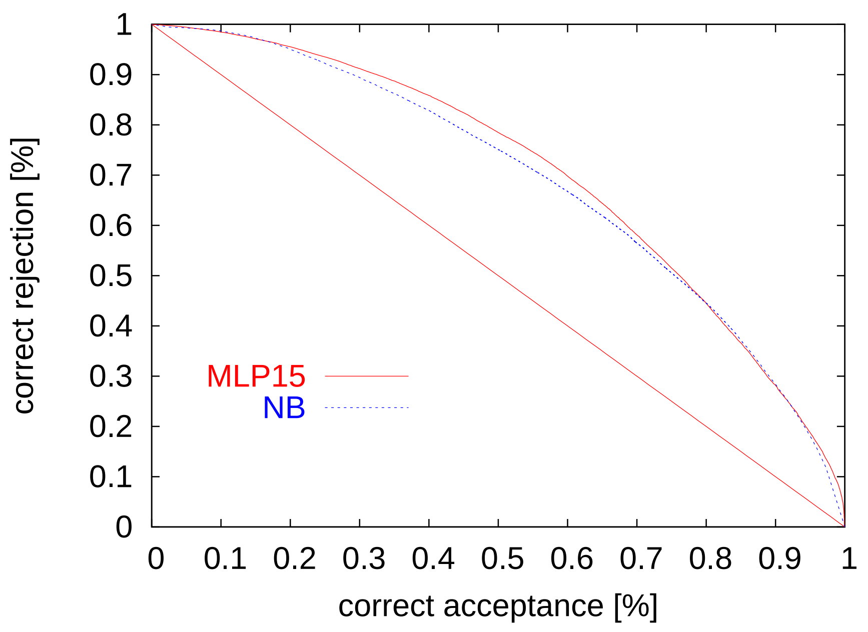


Figure 4.2: ROC for WER, Naive Bayes and MLP with 15 hidden units combining all features.

Table 4.3: CER [%] and AROC [%] for single features and their combination using Naive Bayes; Error Measure: PER. The second column denotes the section in this report describing the feature.

Feature		Section	CER	AROC
Baseline		–	36.2	–
Word posterior prob.	any target pos.	4.2.2	30.9	41.3
Rank weighted frequency	any target pos.	4.2.2	30.8	41.2
Relative frequency	any target pos.	4.2.2	30.9	41.4
Model1		4.2.3	31.2	39.7
Word post. prob.	aligned source pos.	4.2.2	31.9	39.0
Rank weighted frequency	aligned source pos.	4.2.2	31.9	38.9
Relative frequency	aligned source pos.	4.2.2	31.9	38.8
Word post. prob.	fixed target pos.	4.2.2	32.5	37.7
Rank weighted frequency	fixed target pos.	4.2.2	32.6	37.4
Relative frequency	fixed target pos.	4.2.2	32.7	37.2
AT identity		4.2.4	33.1	34.5
# occurrences		4.2.1	33.1	33.2
Rule based		4.2.4	33.1	34.1
Parentheses & Quot. marks		4.2.1	33.1	33.5
WordNet polysemy count		4.2.1	33.2	33.6
WordNet polysemy count (w.r.t. corpus)		4.2.1	33.2	33.5
Avg. semantic similarity		4.2.1	33.4	33.3
Word post.-any + Word post.-alig + Model1		–	29.2	46.6
All		–	29.6	47.2

is no significant difference to the ones with 5 or 20 hidden units in terms of AROC values .

Figures 4.2 and 4.3 plot the ROC for the best performing MLP and the Naive Bayes classifier under the word error measures WER and Set. Again, we see that the MLP discriminates better between the correct and incorrect words.

In Figure 4.4, we see ROC curves for different MLP architectures using PER as word error measure. These agree with the figures presented in Table 4.4: The MLP with zero hidden units performs worse than the more complex architectures, but there is no significant difference between the nets

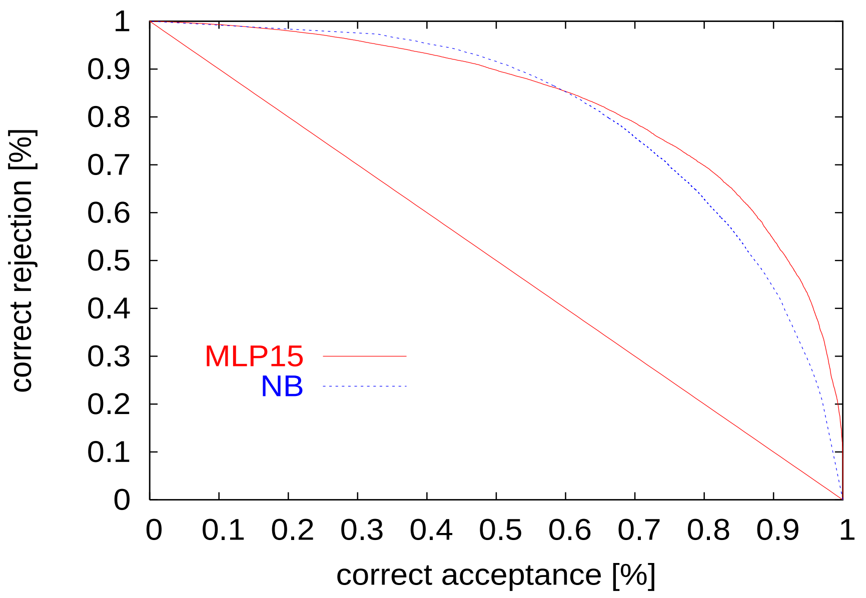


Figure 4.3: ROC for Set, Naive Bayes and MLP with 15 hidden units combining all features.

consisting of 5 to 20 hidden units.

4.4.5 Comparison of Different Word Error Measures

Table 4.5 compares the AROC values for confidence estimation using an MLP with 20 hidden units for all the error measures described in Section 4.3. We see that classification according to some of the error measures is easier to learn than according to others. For the n-grams for example, the classification gets easier the longer the history gets. This is due to the fact that especially the 4-gram is very pessimistic and labels many of the words as incorrect as Table 4.2 showed. Thus, its behavior is easier to learn.
The word error measure for which the highest AROC value is achieved is 'Set'. Analogously, the reason here is that this metric is easy to learn because it labels a high percentage of the words as correct (cf. Table 4.2.)

4.5 Conclusion

We investigated a number of different classifiers for confidence estimation on the word level: a Naive Bayes approach and MLPs with several different

Table 4.4: Comparison of AROC values [%] for different error measures (min variant) and different machine learning techniques. All features are included. The best result is shown in bold.

Machine learning method		WER	PER	Set
Naive Bayes		38.2	47.2	61.4
MLP	0 hidden units	37.7	47.6	62.3
	5 hidden units	41.0	52.3	65.5
	10 hidden units	**41.1**	**53.2**	65.7
	15 hidden units	**41.1**	**53.2**	**65.8**
	20 hidden units	40.6	53.1	65.7

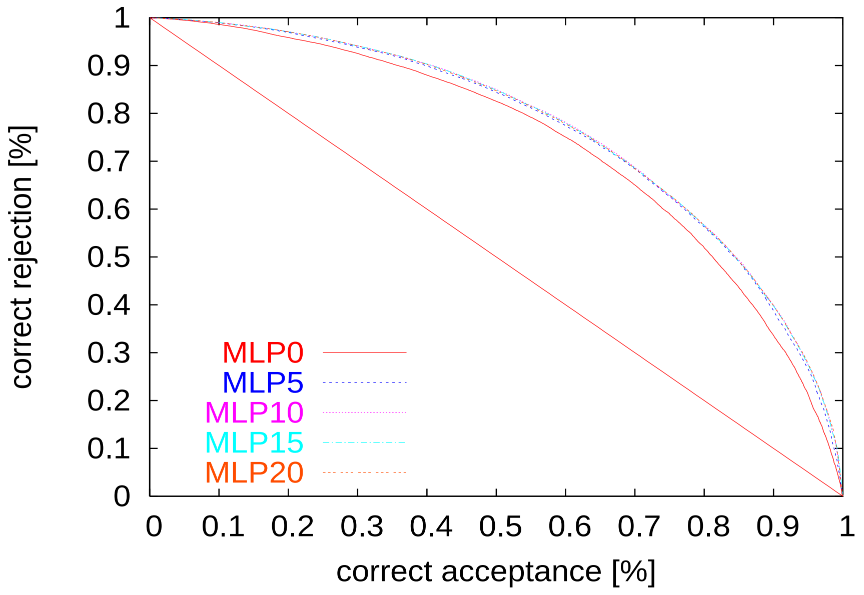


Figure 4.4: ROC for PER, MLPs with different numbers of hidden units combining all features.

Table 4.5: Comparison of AROC values [%] of an MLP with 20 hidden units for different error measures (min variant, all features).

Error Measure	Pos	WER	PER	Set	2-/3-/4-gram
AROC [%]	46.8	40.6	53.1	65.7	37.6 / 40.4 / 48.0

numbers of hidden units. We saw that the MLPs with 5 or more hidden units outperform Naive Bayes, whereas the MLP without any hidden units shows the same classification performance as Naive Bayes.
Various word level features have been implemented. Experiments showed that the combination of those features improves over the application of any single feature.
We implemented different metrics for labeling words in the generated translations as correct or incorrect, and the results were consistent over those different metrics. The performance of the confidence estimation techniques was evaluated on 1000 best lists of the same Chinese–English task as was considered for the sentence level.

4.6 Recombination

As a follow-up of this workshop. we plan to investigate a search criterion based on confidence estimation on the sub-sentence level. The idea is to recombine different translation hypotheses that are represented in the word graph or N best list. This recombination would make it possible to create new translations that are not contained in the graph or list as such.
Regarding this approach, there are still a few problems to be solved: A criterion for the selection of the sentence length has to be developed. If the algorithm would simply pick the sequence with the highest confidence, it would favor short sequences over long ones. One possible solution to this would be to normalize the confidence by the length of the generated sentence.
Another issue is finding a selection criterion for the target words in the recombination. Selecting them on basis of their sentence position will cause inconsistencies, because the same word can be chosen twice, even if it should occur in the sentence only once. If one selects the target words with regard to their aligned source words instead, it is not clear how to determine the correct word order. One possible way to solve this problem would be to represent the search space by a word graph representing all valid hypotheses.

Thus, the search would be restricted only to hypotheses without inconsistencies and with possible word orders in the target sentence. Then, one could determine the best path through this graph based on the confidence estimation.

Chapter 5

Evaluation of MT output

5.1 Sentence-Level Machine Translation Evaluation

Though the analysis of automated MT evaluation techniques was not an original goal of our project, in attempting to interpret the results of a system trained specifically to replicate the output of MT error metrics, it becomes clear that a better grasp of the meaningfulness of those metrics is illuminating. In particular, while studies establishing the corpus-level correlation between modern error metrics and human judgements have been conducted with greater scope than we could hope to reproduce, our approach relies critically on the correlation of these metrics at the *sentence level.* To quantify this correlation, we conducted a small experiment, enlisting the help of colleagues to obtain human judgements of machine translation task-adequacy with respect to a human-produced reference. Our results confirm the intuition that automatic metrics do not yet capture human judgements with any significant predictive power at the sentence level, though we demonstrate, in agreement with earlier studies, that over large corpora metric inconsistencies tend to average out and produce increased correlation. Our study is by no means definitive and we discuss the feasibility of similar, larger experiments based on our experiences.

5.2 Evaluation Protocol

In our experiment, every evaluator is presented with successive pairs of sentences, each consisting of one hypothesis (machine-produced) translation, and one reference (human-produced) translation. The two translations cor-

```
************************************************************************
    Human MT Eval Client
************************************************************************

  Hypothesis:

    ( washington ) , comprehensive report the latest issue of the new
    yorker " weekly , iraq 's intelligence agencies responsible for
    many years and 911 incident osama bin laden under the leadership of
    the al qaeda maintain close ties .

  Reference:

    comprehensive report , washington -- the latest issue of new yorker
    magazine suggests that iraqi intelligence has been in close touch
    with top officials in al @-@ qaida group for years . the al @-@
    qaida group is believed to have masterminded the 911 incident .

  Enter your rating (1-5), 'h' for help, or 'q' to quit:
```

Figure 5.1: A sample display as might be seen by evaluators during our experiment.

respond to the same Chinese source. The hypothesis is presented on screen above the reference, although we make no attempt to control the order in which evaluators read the sentences. All translations are presented in their lower-case, tokenized form. An example of this presentation, as seen by an evaluator, is shown in figure 5.1.

Given constraints of time and a relatively small number of willing participants, we developed a single five-point rating scale designed to be simple, intuitive, and to represent various levels of language tasks for which a translation would be satisfactory. Our rating scale is presented in figure 5.2, exactly as it was explained to the evaluators. The tasks for which each score 1-5 is intended to represent adequacy are described approximately as follows:

1	Universally inadequate
2	“Bag of words” quality; potentially useful for IR
3	Gisting; rough content description
4	Human post-processing; less-demanding applications
5	General/universal use

When the evaluator enters his or her rating, the next pair of sentences is immediately displayed. Evaluators continue in this fashion until they choose

```
*********************************************************************
    Human MT Eval Client
*********************************************************************

    Please rate the quality of a given hypothesis translation with
    respect to the reference on a scale from 1 to 5 as follows:

    Reference ex: bob walked the dog.

    1: Useless; captures absolutely none of the reference's meaning.
        ex: franklin is a doctor.
    2: Poor; contains a few key words, but little or no meaning.
        ex: dog banana walk.
    3: Mediocre; contains some meaning, but with serious errors.
        ex: the dog walked bob.
    4: Acceptable; captures most of the meaning with only small errors.
        ex: bob walk the dog.
    5: Human quality; captures all of the reference's meaning.
        ex: bob took the dog for a walk.

    Press return to continue...
```

Figure 5.2: Our evaluation metric, as presented to evaluators during the experiment. This is displayed at the beginning of each evaluation session, and can be redisplayed by the evaluator at any time with a single keystroke.

to stop.

5.3 Implementation and Setup

Because we do not have the resources to conduct a formal study with paid subjects, presenting the experiment to users in as painless a way as possible is, we believe, critical to their participation. To this end, we have implemented the above protocol as a live server/multi-client system, allowing evaluators to work from arbitrary locations at arbitrary times, and returning results to us in real-time for immediate analysis. The system is completely flexible, enabling users to come and go as they please, and the server guarantees that no evaluator will ever be asked to rate the same pair of sentences twice. Furthermore, as a compromise between efficiency and noise reduction (optimized when each item is rated many times), we have decided that each hypothesis translation should be rated by exactly two evaluators, and the server works actively to ensure that as many sentences as possible satisfy this requirement.

To generate a set of pairs for evaluation, we selected the top one hundred translations from the N-best lists in our test set, and paired each with a randomly chosen reference from the available four. We also included pairs for calibration purposes. Positive calibration examples consisted of all 6 combinations of two reference translations for each source sentence. Negative calibration examples consisted of 6 pairs of one randomly selected reference translation for the source sentence, and one randomly selected reference translation for another source sentence. Each type of calibration pair therefore accounts for around 6% of the evaluation set. During the experiment, sentence pairs are selected from the evaluation set at random.

The system was live for several weeks, and we encouraged voluntary participation throughout. In the end, 29 users logged a total of 20 evaluation-hours and rated 705 hypotheses (two scores per hypothesis). Seventy-two of the rated examples were calibration pairs, leaving us with two judgements on each of 633 unique MT outputs.

5.4 Vote Standardisation

The evaluation tool provides votes on a scale from 1 to 5 for each sentence pair (proposed, reference). Although we have provided indicative guidelines for assigning votes to examples, different voters have much different voting patterns. In figure 5.3, we present the profiles of two actual voters from the evaluation. The histograms of the votes are based on 67 votes (for E) and 60 votes (for F), so they have similar reliability (of the order of max ±10% for the largest bins). Clearly, voter F is much more conservative in his votes, while voter E is more generous. As a consequence, a vote of 4 does not have the same value for both voters: For F, only few translation receive a 4, so one could assume that they are of high quality; For F, a vote of 4 is more common, and may just indicate a slightly above average quality.

In order to standardise the votes on a similar scale with comparable value accross voters, we transform each vote into a quantile[1]. For each voter V and score $s \in \{1, 2, 3, 4, 5\}$, the standardised vote is:

$$x = f(s, V) = P(S < s|V)$$

where $P(S|V)$ is the distribution of scores for voter V, so $P(S < s|V)$ is the probability that voter V would assign a score lower than s. In order

[1]This idea was originally suggested by Jason Eisner during one of the workshop cookie breaks.

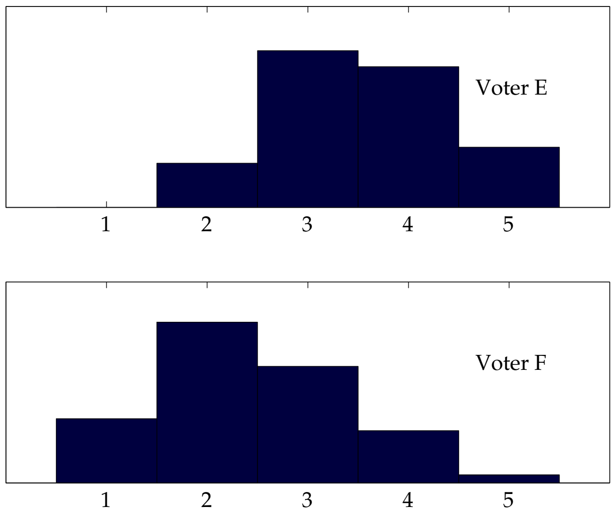


Figure 5.3: Difference in voting patterns between two voters.

to estimate this probability from our discrete 5-point scale, we accumulate all votes that are strictly below s, and half of the votes that are equal to s. The ratio of these accumulated votes to the total number of votes is the standardised score. If $n(s, V)$ is the number of votes of voter V that are equal to s, then:

$$\widehat{x} = \frac{\sum_{i<s} n(i) + n(s)/2}{\sum_{i=1}^{5} n(i)}$$

Strictly speaking, in order to estimate $P(S < s|V)$ we should only accumulate votes up to $s - 1$. On the other hand, to estimate $P(S \leq s|V)$, we should accumulate all votes up to s. Our standardised estimate may be viewed as an average of these two expressions. Alternatively, the standardisation may be viewed as follows: if we spread all the votes for a given value $i = 1, \ldots 5$ in bins centered on this value and construct the corresponding continuous cumulative distribution function (figure 5.4, left), then the value of the cumulative distribution function in s is exactly our estimate.

As illustrated on figure 5.4 (right), the same actual vote of 3, for two voters with different voting profiles, may be standardised to very different value. For voter E, it would correspond to a noticeably below average standardised value, while for voter E, it is significantly above average. This result conforms to the intuition we have about the voting patterns of E and

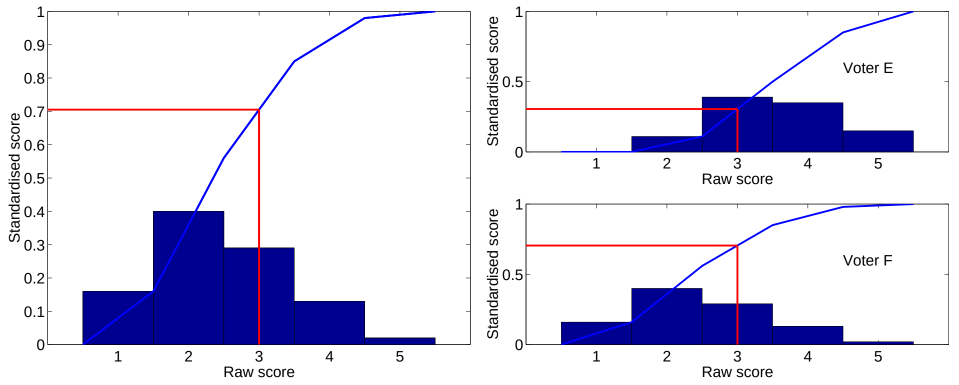


Figure 5.4: Left: Mechanism for standardising votes from a 1–5 scale to a [0,1] scale. Right: Effect on a similar vote (3) for the two voters presented in figure 5.3

F.

Vote Averaging

In order to reduce the sensitivity of sentence scores to a particular voter, and to have an idea of the inter-annotator agreement, the same sentence pair was evaluated by two different annotators. To obtain a composite score from both votes, we need to average them in a principled way.

As explained above, the standardised score is the empirical estimate of a probability that the voter scores below an actual value. The reliability of this estimate will depend on the total number of votes for this voter. Intuitively, it seems that a voter who performed very little evaluation and gave only 1, 2 and 3 to the evaluated sentences may be particularly harsh, or, if he evaluated few sentences, he may just have been unlucky and got only bad translations.

Let us consider 2 voters evaluating the same sentence. The vote for the first one is s_1, with the profile $n_1(1), n_1(2), \ldots n_1(5)$. The vote for the second one is s_2, with the profile $n_2(1), n_2(2), \ldots n_2(5)$. The standardised votes are:

$$\begin{aligned}
\widehat{x}_1 &= \frac{\sum_{i<s_1} n_1(i) + n_1(s_1)/2}{\sum_{i=1}^{5} n_1(i)} \\
\widehat{x}_2 &= \frac{\sum_{i<s_2} n_2(i) + n_2(s_2)/2}{\sum_{i=1}^{5} n_2(i)}
\end{aligned}$$

as explained above, and the average vote is:

$$\overline{x} = \frac{\sum_{i<s_1} n_1(i) + n_1(s_1)/2 + \sum_{i<s_2} n_2(i) + n_2(s_2)/2}{\sum_{i=1}^{5} n_1(i) + \sum_{i=1}^{5} n_2(i)} \tag{5.1}$$

which is a weighted average of $\widehat{x}_1$ and $\widehat{x}_2$, where the weight depends on the total amount of votes. Accordingly, the vote of voters which evaluated many sentences will contribute more to the average, because the corresponding standardised vote is inherently more reliable. Equation 5.1 also corresponds to modelling the distributions of x_1 and x_2 as Beta distributions and averaging the scores over these distributions.

5.5 Error Metrics

We compare the results of our human evaluation experiment to the sentence-level behavior of six common automatic MT evaluation metrics:

- WER: Word error rate, computed as the minimum number of insertions, deletions, and substitutions required to transform the hypothesis into any reference (Levenshtein/edit distance), normalized by reference length.
- WER-g: As above, but normalized by the total length of the alignment (insertions, deletions, substitutions, and matches).
- PER: Position-independent error rate; treats both hypotheses and references as unordered bags of words and counts the necessary operations to make them equal. Normalized by reference length.
- BLEU ([33]): The geometric mean of hypothesis n-gram precision for $1 \leq n \leq 4$, multiplied by an exponentially decaying length penalty, to compensate for short, high-precision translations ("the").
  - Smoothed precisions
  - Adjusted length penalty
- NIST ([28]): The **arithmetic** mean of hypothesis n-gram precisions, weighted by n-gram frequencies in a fixed corpus (effectively, less common n-grams receive greater emphasis). Also uses a length penalty.
- F-Measure ([24]): The harmonic mean of precision and recall, where the size of the match between hypothesis and reference is the maximum

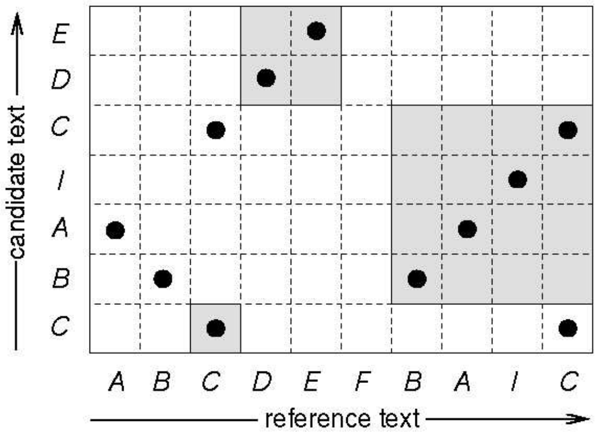


Figure 5.5: The F-measure error metric. Matching runs are grayed. When $k = 2$, the F-Measure is equal to the fraction of the grid covered by these aligned blocks.

of $\sqrt[k]{\sum |r_i|^k}$ over all sets $M = \{r_1, ..., r_n\}$ of non-conflicting matched runs of words. We use $k = 1$. See figure 5.5 for a visual interpretation of the F-measure.

5.6 Correlation with Human Judgements

We investigated the correlation of the human judgement with the automatic measure in several ways.

We want to perform several comparisons: human voter vs human voter in order to check inter-annotator agreement, human voter vs automatic score to check how well the automatic scores fit human judgements, or even one automatic score vs another in order to check how well these correlate.

Let us denote by u a reference vote (typically vote from one annotator) and by v a second vote or measure. We consider two correlation measures. First we calculate the standard correlation coefficient, which is the normalised covariance:

$$r = \frac{\sum_i (u_i - \overline{u})(v_i - \overline{v})}{\sqrt{\sum_i (u_i - \overline{u})^2}\sqrt{\sum_i (v_i - \overline{v})^2}}$$

The correlation coefficient r varies between -1 and 1, 1 indicating perfect correlation (u_i and v_i are on a line with positive slope), -1 indicating perfect anit-correlation (u_i and v_i on a line with negative slope), and 0 indicating no correlation at all. The standard correlation coefficient also has a convenient interpretation in terms of the amount of variance in one variable that is explained by the second variable, using a linear regression. For example, a value of $r^2 = 0.5$ indicates that only half the variability in v may be linearly modelled from u, corresponding to a "signal-to-noise ratio" of 1 (as much unexplained noise as there is "signal", or explained variance).

We also compute the Spearman rank correlation coefficient. This is done by sorting u_i and v_i and replacing each value by its rank. Let U_i (resp. V_i) indicate the rank of u_i (resp. v_i) in the sorted list of u's (resp. v's), with the added twist that we average the ranks of ties. For example, if the second and third elements in the sorted list are equal, they both receive a "rank" of 2.5. The Spearman rank-order correlation coefficient is then:

$$s = \frac{\sum_i \left(U_i - \overline{U}\right)\left(V_i - \overline{V}\right)}{\sqrt{\sum_i \left(U_i - \overline{U}\right)^2}\sqrt{\sum_i \left(V_i - \overline{V}\right)^2}}$$

Statistical significance test of non zero correlation coefficients are available [35]. In addition, the variance is approximately $\mathrm{Var}(s) = 1/(N-1)$ (this expression is exact when there are no ties).

As a first comparison, we look at the correlation between each automatic score and one of the human scores, compared to the inter-annotator agreement. The results plotted on figure 5.6 show that the correlations are quite poor (30–35 percent, or a SNR of about 10%). Note however that even the inter-annotator agreement is poor, as it only reaches 45%. Overall, there is little difference between the linear and Spearman correlations, suggesting that whatever small dependency there may be between the automatic and human scores is almost linear.

The best performing scores are the WER-g (best Spearman rank-correlation) and the NIST score (best linear correlation). Surprisingly, the F-measure score has the lowest sentence-level correlation.

In order to check how significant the differences in correlations are, we computed bootstrapped error bars for all correlations, including the inter-annotator agreement. Figure 5.7 shows that the inter-annotator agreement is significantly above all other correlations. On the other hand, all automatic measure correlations are within error bars of each other, indicating that the observed differences between the correlations may not actually be large enough to be significant.

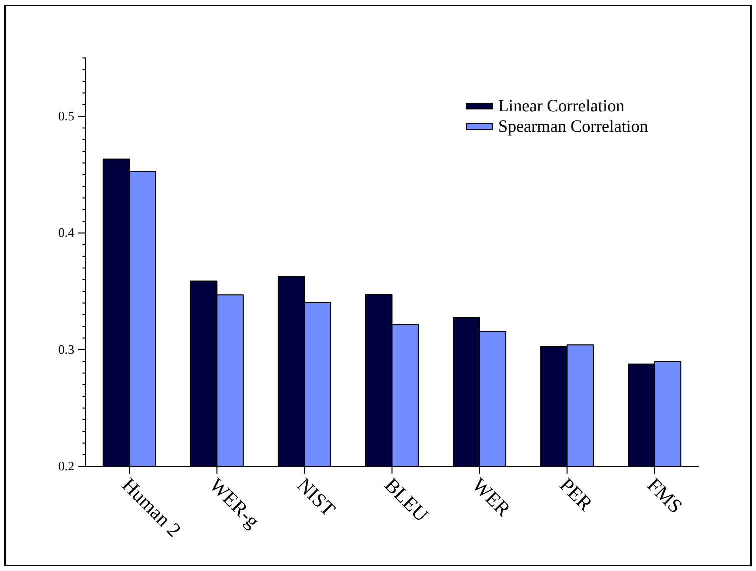


Figure 5.6: Inter-annotator agreement (correlation of Human2 vs. Human1, left) and correlation of all automatic scores with a single human judge (Human1).

As the inter-annotator agreement is low, we calculate the correlation of each automatic score with the average of the human votes. Averaging reduces somewhat the noise in the reference human score, and the resulting correlations become a bit higher, reaching around 40% (figure 5.8). Note however that these values are not comparable anymore to the inter-annotators agreement, which is calculated using the raw (ie non-averaged) human scores.

Earlier reports indicate that various automatic scores are well correlated with human judgement over entire texts. This is the case for BLEU as well as NIST, FMS or WER. In order to check that, we investigated the correlation of the automatic scores with the human scores averaged over several sentences. In figure 5.9, we split the data in five bins, equally spaced on the human average scale. Both the human average score and the NIST score are then averaged for each bin. Figure 5.9 shows that the resulting five points are in a very clear, consistant, almost linear relationship. However, for a single sentence, the fit is very poor. For example, for a NIST score of 7, the average human score would be around 0.5, but the spread is so large, that the actual human score could actually be anywhere between 0.1 and

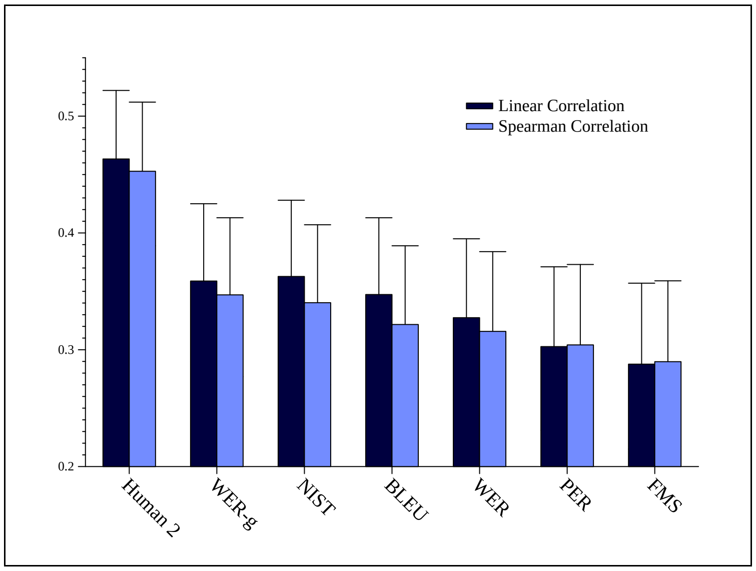


Figure 5.7: Bootstrapped error bars on the correlations from figure 5.6.

0.9 with reasonable probability.

5.7 Future Work

5.7.1 Similar Studies

While our results show a significant performance gap between current automatic metrics and human judges at the sentence level, we believe that a more extensive study of a similar nature will be feasible and revealing. A primary goal of such a study should be wider scope: evaluating a larger set of more diverse hypotheses will reduce the width of confidence intervals and potentially allow performance discrimination between individual metrics. Additionally, it will be valuable to increase the number of evaluators that examine each hypothesis; by decreasing noise in this fashion, we expect to see improved correlation of the automatic metrics and obtain more reliable results.

We observed experimentally that an average evaluator can score approximately 75 sentences per hour; while our subjects had prior experience in the field, it does not seem unreasonable to expect speeds of roughly one pair rated per minute in general, particularly if done in a controlled environment.

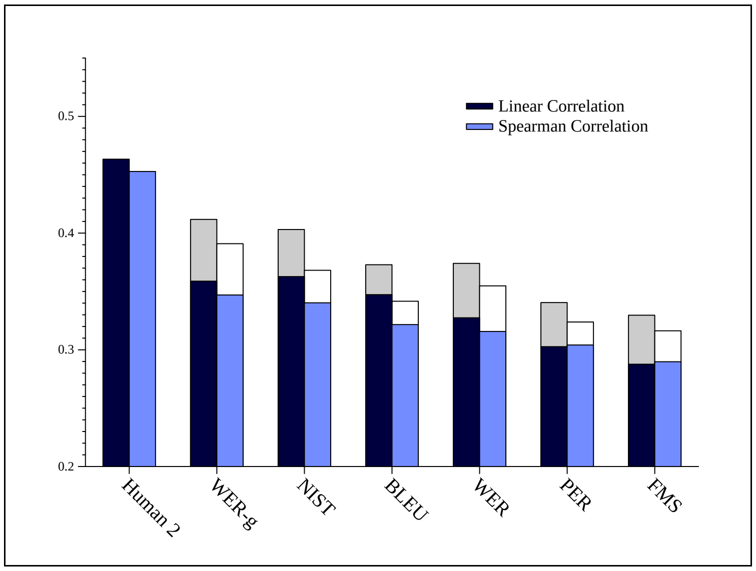


Figure 5.8: Correlation with the average of the two human judges is better for all metrics. Inter-annotator agreement is presented for reference but is not comparable to the other correlations as it does not involve any averaging.

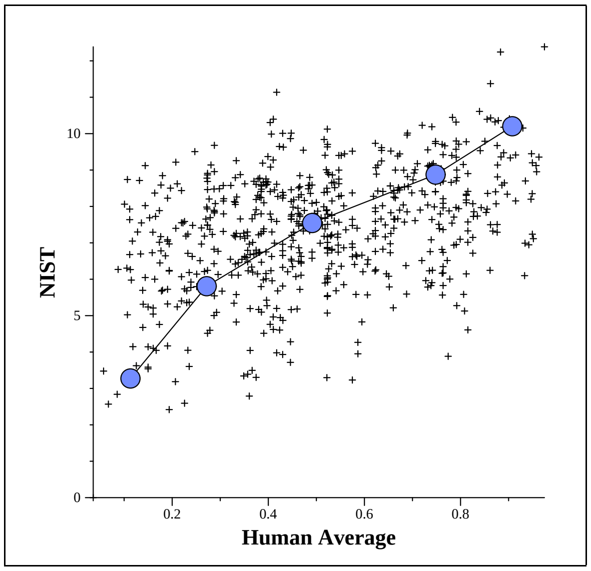


Figure 5.9: Average over five bins in the human average score. This confirms previous results of high correlation at the corpus level.

Increasing the scores per hypothesis to three then brings the cost to about 50 evaluator-hours per 1000 hypotheses, not unreasonable for a large-scale study.

5.7.2 Automatic MT Evaluation Metrics

Of course, the primary conclusion we draw from this experiment is that error metrics more successful at the sentence-level will be necessary to enable the practical success of projects like ours that rely on them. Given the large amounts of MT data that have become available in recent years, such metrics would potentially motivate a wide variety of solutions that can take advantage of large quantities of accurately evaluated output, including confidence estimation, internal hypothesis re-ranking, and improved error-analysis of modern MT outputs.

Chapter 6

Conclusion

The confidence estimation for MT workshop studied various techniques in an attempt to classify MT output (baseline data obtained from the Syntax for MT workshop) as correct or not. Classification was applied both to entire target sentences and to individual words and ngrams within target sentences. In an attempt to gain insight into what constitutes a correct translation at the sentence level, as well as to gauge the performance of automatic evaluation metrics on this scale, we also undertook a small MT evaluation exercise involving human annotation.

6.1 Summary of Results

The general conclusion from the workshop is that confidence estimation is a very difficult problem for MT. At the sentence level, high variance in the metrics used to automatically assign correctness makes it hard to learn meaningful distinctions between good and bad translations. At the subsentence level, the ability to match parts of a machine-generated translation with a reference translation significantly reduces noise, but the best way to perform this matching remains unclear, as does a good strategy for exploiting the resulting classifiers. Although there are many interesting potential applications for confidence estimation for MT, we cannot make a strong claim that any are currently feasible.

The following summary lists the most salient results obtained:

- Training a separate layer using machine-learning techniques is better than relying solely on base model scores.

- Features derived from the base model are more valuable than external ones, and should be tried first before investing effort in the implementation of complex external functions.
- Features based on nbest lists are more valuable than ones based solely on individual hypotheses.
- Features that capture properties of the target text are more valuable than those that do not.
- Multi-layer perceptrons (neural nets) outperform naive Bayes models. MLPs with more hidden units can give better performance than those with fewer.
- At the sentence level, NIST and WERg error measures have the best correlation with human assessments, although the differences among all automatic evaluation metrics are generally not statistically significant. Inter-annotator agreement is poor, but clearly distinguishable from automatic metrics.

6.2 Future Work

At the sentence level, it is clear that better automatic evaluation metrics need to be developed before progress can be made on the confidence estimation problem. We expect that advances in statistical MT will lead to improved evaluation metrics, perhaps based on sophisticated models that can be trained to distinguish between human and machine translations.

At the subsentence level, the most obvious possibility is to investigate the application of the techniques studied in chapter 4 for improving search procedures and/or to perform hypothesis recombination over nbest lists or word graphs.

Bibliography

[1] *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, July 2002.

[2] Satanjeev Banerjee and Ted Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing (CICLING)*, Mexico City, 2002.

[3] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71, 1996.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford, 1995.

[5] Peter F. Brown, Stephen A. Della Pietra, Vincent Della J. Pietra, and Robert L. Mercer. The mathematics of Machine Translation: Parameter estimation. *Computational Linguistics*, 19(2):263–312, June 1993.

[6] P. Carpenter, C. Jin, D. Wilson, R. Zhang, D. Bohus, and A. Rudnicky. Is this conversation on track? In *Eurospeech*, 2001.

[7] R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[8] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley, 2001.

[9] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, 1993.

[10] W. B. Frakes and R.Yates. *Information Retrieval: Data Structure and Algorithmics.* Prentice Hall, 1992.

[11] Simona Gandrabur and George Foster. Confidence estimation for text prediction. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL)*, Edmonton, Alberta, May 2003.

[12] L. Gillick, Y. Ito, and J. Young. A probabilistic approach to confidence measure estimation and evaluation. In *ICASSP 1997*, pages 879–882, 1997.

[13] Didier Guillevic, Simona Gandrabur, and Yves Normandin. Robust semantic confidence scoring. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP) 2002*, Denver, Colorado, September 2002.

[14] Peter G. Hall and Donald Hall. *The Bootstrap and Edgeworth Expansion.* Springer-Verlag, 1995.

[15] T. Hazen, S. Seneff, and J. Polifroni. Recognition confidence scoring and its use in speech understanding systems. *Computer Speech and Language*, pages 49–67, 2002.

[16] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 265–283. MIT Press, 1998.

[17] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(3):359–366, 1989.

[18] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression.* Wiley, 1989.

[19] T. Kemp and T. Schaaf. Estimating confidence using word lattices. In *Eurospeech*, pages 827–830, 1997.

[20] A. Stolcke L. Mangu, E. Brill. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14(4):373–400, 2000.

[21] C. Ma, M.A. Randolph, and J. Drish. A support vector machines-based rejection technique for speech recognition. In *ICASSP 2001*, 2001.

[22] B. Maison and R. Gopinath. Robust confidence annotation and rejection for continuous speech recognition. In *ICASSP 2001*, 2001.

[23] R. Manmatha and H. Sever. A formal approach to score normalization for meta-search. In M. Marcus, editor, *Proceedings of HLT 2002, Second International Conference on Human Language Technology Research*, pages 98–103, San Francisco, 2002. Morgan Kaufmann.

[24] I. Dan Melamed, Ryan Green, and joseph P. Turian. Precision and recall of machine translation. In *Proceedings of the Human Language Technology Conference (HLT)*, pages 61–63, Edmonton, Alberta, May 2003. HLT-NAACL.

[25] P. Moreno, B. Logan, and B. Raj. A boosting approach for confidence scoring. In *Eurospeech*, 2001.

[26] C. Neti, S. Roukos, and E. Eide. Word-based confidence measures as a guide for stack search in speech recognition. In *ICASSP 1997*, pages 883–886, 1997.

[27] Grace Ngai and David Yarowsky. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, October 2000.

[28] George Doddington (NIST). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Human Language Technology Conference (HLT)*, San Diego, CA, March 2002.

[29] *Proceedings of the NIST Rich Transcription Evaluation*, 2002.

[30] *Proceedings of the NIST Workshop on Machine Translation Evaluation*, 2003.

[31] Franz Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 2004. To appear.

[32] Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In ACL-02 [1].

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In ACL-02 [1], pages 311–318.

[34] Siddharth Patwardhan and Ted Pedersen. Wordnet::similarity, 2003. Perl Module, `http://search.cpan.org/dist/WordNet-Similarity/`.

[35] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2nd edition, 1992.

[36] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1998.

[37] Philip Resnick. Using information content to evaluate semantic similarity. In *14th International Joint Conference on Artificial Intelligence*, pages 448–453, Montréal, 1995.

[38] John A. Rice. *Mathematical Statistics and Data Analysis.* Duxbury Press, 1995.

[39] A. Sanchis, A. Juan, and E. Vidal. Improving utterance verification using a smoothed naive Bayes model. In *ICASSP 2003*, 2003.

[40] M. Siu and H. Gish. Evaluation of word confidence for speech recognition systems. *Computer Speech and Language*, 13(4):299–318, 1999.

[41] Nicola Ueffing, Klaus Macherey, and Hermann Ney. Confidence measures for Statistical Machine Translation. In Elliott Macklovitch, editor, *Proceedings of MT Summit IX*, New Orleans, September 2003. International Association for Machine Translation.

[42] M. Weintraub, F. Beaufays, Z. Rivlin, Y. Konig, and A. Stolcke. Neural-network based measures of confidence for word recognition. In *ICASSP 1997*, pages 887–890, 1997.

[43] F. Wessel, R. Schlüter, K. Macherey, and H. Ney. Confidence measures for large vocabulary continuous speech recognition. *IEEE Trans. on Speech and Audio Processing*, 9(3):288–298, 2001.

[44] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[45] R. Zhang and A. Rudnicky. Word level confidence annotation using combinations of features. In *Eurospeech*, pages 2105–2108, 2001.