Correct your Text with Google

Stéphanie Jacquemont, François Jacquenet, Marc Sebban

Correct your text with Google*

Stéphanie Jacquemont, François Jacquenet, Marc Sebban
Laboratoire Hubert Curien (UMR CNRS 5516)
18 rue Benoît Lauras - 42000 Saint-Etienne - France
{jacqstep,jacquene,sebbanma@univ-st-etienne.fr}

Abstract

With the increasing amount of text files that are produced nowadays, spell checkers have become essential tools for everyday tasks of millions of end users. Among the years, several tools have been designed that show decent performances. Of course, grammatical checkers may improve corrections of texts, nevertheless, this requires large resources. We think that basic spell checking may be improved (a step towards) using the Web as a corpus and taking into account the context of words that are identified as potential misspellings. We propose to use the Google search engine and some machine learning techniques, in order to design a flexible and dynamic spell checker that may evolve among the time with new linguistic features.

1 Introduction

The idea behind this paper comes from some everyday practices of text correction. Indeed, in our everyday life, when one wants to find the correct spelling of words, many people are used to ask some help from Google. Wondering which of the various spellings of a word is the right one, we apply with Google, the simple following heuristic: the more pages containing the spelling, the best spelling. We think this could be the basic idea of a more sophisticated tool we decided to design trying to automate the manual process we used to have every day. The problem with such an approach is that while a human being asking some help from Google knows the various spellings he wants to test, in the case of a machine-based process, the various spellings must be automatically provided to the machine.

At the moment, there are mainly three well known spell checkers that can be used by end users to correct their texts: *Ispell*, *Aspell* and *Word*.

Ispell, designed by GNU, or *Word* spell checker rely on a very basic principle: when they detect a misspelling, they propose all the possible words that are at some edit distance [21] from this misspelling. *Aspell*, also a GNU project, has been designed to improve *Ispell*. The main contribution of the checker is that it uses the metaphone algorithm of Lawrence Phillips [13]. This algorithm is able to calculate some suggestions based on soundslike similarity. Hence, it converts the misspelled word to its soundslike equivalent. It finds all words that have a soundslike at small edit distances from the original word's soundslike. Then it finds misspelled words that have a correctly spelled replacement by the same previous criteria. Finally, *Aspell* scores the result list and returns the words with the lowest score (the score is roughly the weighed average of the weighed edit distance of the word to the misspelled word and the soundslike equivalent of the two words).

The performances of *Ispell*, *Aspell* and *Word* may be compared using two main criteria. The first one, called *Suggestion Intelligence* (SI), corresponds to the rate of good suggestions proposed by the spell checker with respect to all the misspellings of test file:

$$SI = \frac{Total\ correct\ suggestions}{Total\ misspellings} * 100.$$

The second criterion, called *Suggestion Intelligence First* (SIF), represents the ability of the spell checker to find the good suggestion at the first position of the list of all the suggestions:

$$SIF = \frac{Total\ correct\ suggestions\ found\ first\ on\ list}{Total\ misspellings} * 100.$$

We ran *Aspell*, *Ispell* and *Word* on a set of documents containing 500 identified misspellings (cf section 4) and we get the results of table 1.

*This work has been supported by the Web Intelligence project of the French Region Rhône-Alpes.

	Aspell	Ispell	Word
SI	79	78	78
SIF	58	39	65

Table 1. Comparative study of state of the art spell checkers.

Concerning *Suggestion Intelligence*, the three tools are quite equivalent at approximately 78%. This result shows the potential improvement we can achieve. Concerning the *Suggestion Intelligence First* criterion, Ispell is the worst checker, then Aspell is a little bit better and Word is the best of the three, but the scores remain quite low. One possible explanation is that these tools mainly focus on the misspellings and not on their contexts. Thus, to increase the *Suggestion Intelligence* and *Suggestion Intelligence First* scores, we think it would be interesting, while calculating some suggestions to correct a misspelling, to take into account the whole sentence (or at least some part of it) that contains this misspelling. That is the vision we are presenting now.

First, in the next section, we show that the Web has already been considered as a corpus by other researchers in various projects of various domains. In section 3, we present our system, its architecture, the way it works and the machine learning techniques it integrates. In section 4, we present some experiments we did with our system and compare its performance with *Aspell*, *Ispell* and *Word* in terms of *Suggestion Intelligence* and *Suggestion Intelligence First*. Finally, we conclude and propose some future explorations.

2 Using the Web as a Corpus

In this section, we first explain why the Web can be considered as a corpus and we describe several examples of applications where the Web has been used with that purpose.

2.1 Can we consider the Web as a corpus?

Since the beginning of the nineties and the advent of the Internet, the amount of documents that can be accessed on the Web has considerably increased and those ones are more and more diversified. These features make the Web an interesting tool for researchers. Indeed, at the same time, it has been more and more urgent to design linguistic corpora in the context of various researches such as natural language processing, machine translation, etc. To built such corpora, many people have focus on the Web. First, let us recall what is a corpus using the definition of McEnery and Wilson [12]:

"Any collection of more than one text can be called a corpus, (corpus being Latin for "body", hence a corpus is any body of text). But the term "corpus" when used in the context of modern linguistics tends most frequently to have more specific connotations than this simple definition. The following list describes the four main characteristics of the modern corpus: sampling and representativeness, finite size, machine-readable form, a standard reference."

This definition gives us some basis to explain the reasons why it is possible to consider the Web as a linguistic corpus. Let us consider each previous features.

If we consider *sampling*, as the Web, while voluminous, is not an exhaustive representation of the various languages, it has to be considered as a sample of those ones. Moreover, as the various search engines only index a part of the whole Web, we usually use only a sample of it.

Concerning *representativeness* of the Web as a corpus, the main question is to know if the Web is enough representative of natural languages. In other words, is the Web representative enough to replace classic corpus?

[23, 6, 10] tried to answer this question studying the Web in the context of learning the frequencies of trigrams, that is, the probability estimate of three consecutive words, with the objective to model natural languages. The objective was to compare the probabilities of 3-grams calculated using a corpus and those calculated from the Web. Several search engines such as $Google$, $AltaVista$, $Lycos$ and $Fast$ have been used for that study. They have shown that 3-grams calculated from the Web could improve the word error rate with respect to frequencies calculated on the classic corpus, such as the British National Corpus. The frequencies calculated on the Web have been used in various applications of various domains and it has been proved that the results obtained using the Web were significantly better than those obtained with the British National Corpus.

These studies show the evidence of the interest to replace classical corpus by the Web or a sample of it. That leads us to conclude the Web really has the characteristic of representativeness.

It is obvious that the Web has a *finite size*, even if this one is not constant. The number of words on the Web is countable even if it evolves among the time. Grefenstette and Nioche [3] for example have estimated the number of words that can be accessed by some search engines.

The third characteristic of a corpus is to be *machine readable*. It is clear that, by definition, the documents on the Web are electronic files, thus easily usable by computers and their softwares.

Finally, concerning the fact that a corpus has to be a *standard reference*, it is difficult to say at the moment that it is the case. Indeed, the various experiments that have been done until now do not always use the same samples of the

web. Nevertheless, more and more researches have chosen to use the Web as a corpus that cannot be ignored and we can think it will become a standard in a few years.

Those considerations show us that the definition of a corpus proposed by McEnery et Wilson [12] may be applied to the Web. It is important to note that many other features are attached to the Web such as its linguistic diversity, its evolution, its accessibility, which are considerable advantages with respect to classical corpus. This is for these reasons that we have chosen to use the Web as a corpus in the context of spell checking.

2.2 Various domains where the Web has been used

Obviously there are many applications related to natural language processing that rely on the Web [8].

In machine translation, several researches have used the Web as a corpus. Classically, there are two ways to use the Web. First, it can be used to choose the best translation among several candidate translations. Second, it can be used as a database of translations.

Using the Web to choose the best translation has been done for example by Way and Ghough [22] who developed WEBMT, a tool that uses classical translation rules and then submits the various possibilities to a search engine. Probabilities are calculated thanks to the number of occurrences returned by the engine. The chosen translation is the one that has the greatest probability. Experiments show that in the case of three different translation systems, the use of the Web may increase their performance of 64%.

Using the Web as a database of translations has been done by Resnik and Smith [17, 18] with the STRAND project, that aims at building a parallel corpus, that is couples of texts that are exact translations of each others. Kraaij et al. [9] also use the Web to build statistical translation models.

The Web has also been used for the Word Sense Disambiguation task which is a difficult problem. In [19], Santamaria et al. try to associate the meaning of words to directories of thematic search engines such as $Yahoo$. Thanks to this work, some meanings have been discovered and for some rather difficult words, good results have been obtained. Some limitations have been shown for some too general words such as adverbs. This study could be used to complete Wordnet, thanks to the Web diversity.

Tanguy and Hathout have used the Web to complete the french *Verbaction* lexicon [4] which is a very useful lexicon for natural language processing in french. Language being in constant evolution, it is necessary to be able to make this lexicon evolve, thus they have chosen to base their system on the Web to ensure that. Markert et al. have used the Web to solve anaphora [11]. The basic idea of the system is to search on the Web for some couple anaphora/antecedent to validate the best antecedent. Ghani et al. [2] have used the Web to build corpora of representative documents for minority languages. Radev et al. have designed a tool, called PROFILE [15], able to collect on the Web some functional descriptions of entities (i.e. part of sentences describing something or someone) tagged lexically or syntactically. These ones are then used to generate summaries of news. In the domain of Question Answering systems, they also designed the QASM (Question Answering using Statistical Model) system [16] which is able to ask some questions and provides some answers from search engines in natural language. QASM translates questions in natural language in requests for classical search engines. The idea is to learn, using some machine learning techniques, the best sequences of transformations from a natural language request to search engine one.

We can say, at the end of this study, that using the Web as a corpus is not really a new idea and has been at the core of various successful projects. Thanks to its linguistic diversity it makes it possible to perform efficient machine translation, and to build other corpus even for minority languages. Its thematic diversity makes it possible to complete well known corpus. More generally, its size allows it to be used as a training sample for machine learning algorithms.

We are going to see now that the Web is also the core of our new efficient spell checker.

3 The WebSpell system

If we consider a standard spell checker, its basic principle is quite simple. Let $T_1 \ \ldots \ T_n$ the constitutive tokens (or words) of a document. When the spell checker detects a token T_i that is not in its associated dictionary, i.e. in fact a misspelling, it searches in this dictionary for some tokens CS_{ij} that could be suggested to the user in order to correct this misspelling. Since each of these candidate suggestions are more or less appropriate, a crucial task to achieve is to rank them accurately. The next two sections aim at presenting the way we perform that.

3.1 Basic principle of the system

The basic principle of the WebSpell system is described in figure 1.

The candidate suggestions come from three different sources. First, WebSpell collects, from its dictionary, the tokens at an edit distance smaller than a fixed value (usually equal to 1) of the misspelling. Another class of candidate suggestions is made up of the frontier words [1] also calculated with the help of the dictionary. Frontier words are

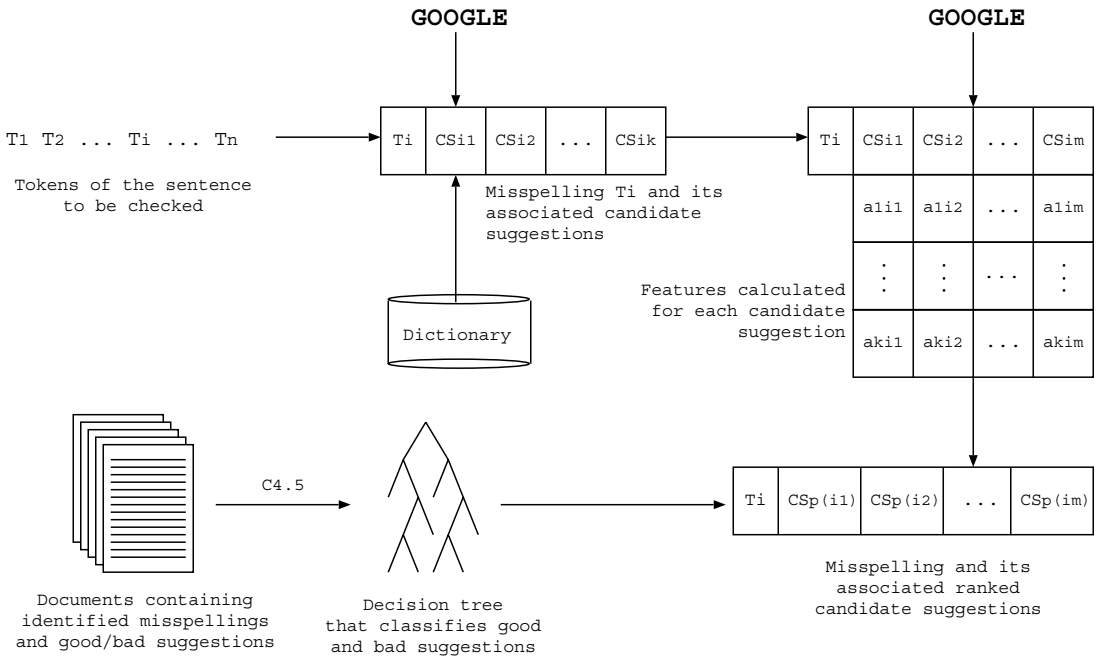

Figure 1. Principle of the WebSpell system

defined by the presence or absence of a space between two words. Suggestions that we will consider as frontier words are strings built by adding or deleting a space. For example, "below" is a frontier word because it is the concatenation of two existing words "be" and "low". The last category of candidate suggestions corresponds to the false misspellings, i.e. tokens not encountered in the dictionary while they belong to the language of the considered document. To find them, we search for a misspelling with Google. If the answer is "Did you mean: S" we consider the misspelling is really an error and S is added to the set of candidate suggestions, else we add the misspelling to this list. When the set of candidate suggestions CS_i is built, we calculate, for each candidate suggestion CS_{ij} $(j = 1, \ldots, |CS_i|)$ of a misspelling T_i a vector of values a_{kij} corresponding to the features. Each candidate suggestion is then submitted to a decision tree (see below) that classifies it as a good or a bad candidate. Good candidate suggestions, which become suggestions from that time, are then ranked according to their distance to the separator hyperplane provided by the decision tree. That means we build a permutation p such that $CS_{p(i1)} \ldots CS_{p(ik)}$ is the ranked list of suggestions.

3.2 Features attached to suggestions

First, let us explain how such a decision tree can be built. From a corpus of *learning* documents, we carry out the spell check task by searching for the set of candidate suggestions. Given a misspelling and its corresponding candidate suggestions, we are able to manually select the good candidate, which is then labelled as positive example, while the others are then automatically labelled as negative examples. Doing the same thing for all the documents, we can design a learning set LS composed of positive and negative examples. The C4.5 algorithm [14] may then be used to build a decision tree from LS able to linearly separate the good candidates from the bad ones.

In order to learn this decision tree, we have to define a set of features associated with each candidate suggestion CS_{ij} supposed to correct a misspelling T_i. After many investigations for designing a relevant representation of a misspelling, we have chosen to define five features related to the Web and six other features not linked to the Web, resulting in a feature vector $A = \{a_{kij} | k = 1, \ldots, 11\}$.

3.2.1 Features that do not use the Web

Size: We have chosen to take into account the size, that is the number of letters, of the misspelling. This very simple feature is useful to take into account the distribution of natural language tokens with respect to this size. Indeed the more we consider long tokens, the less they exist in natural language. A short token will have more candidate suggestions for correction than a longer one.

Distance: The second feature we considered is the distance between the candidate suggestion CS and the misspelling T. It is based on the classical edit distance [21], that is the number of operations (deletion, insertion, substitution of letters) used to change one token into another one. In order to normalize the results, we decided to divide the value calculated with the classical edit distance by the size of the token it is applied on. Indeed, it is more likely to make two misspellings in a long token rather than in a short one:

$$Distance(CS, T) = \frac{\text{edit distance}(CS, T)}{Size(T)}.$$

In fact, this feature is one of the most important and is used in every spell checker.

Position of error: The third feature gives the position of the first error in the misspelling T with respect to the candidate suggestion CS. This feature is important because some studies have shown that errors was more frequent in some places of the tokens [7]. Here again, we have chosen to normalize this value by the size of the misspelling:

$$Position(T, CS) = \frac{\text{position of error}(T, CS)}{Size(T)}.$$

Punctuation tag: This tag indicates if the sentence we are considering contains some punctuation or not. It is useful during the correction in order to distinguish the sentences that have to be corrected.

Frontier tag: This feature is useful to determine if a suggestion is a frontier word or not.

Keyboard distance: The value of this feature is calculated by taking into account the distance between keys on a keyboard. Here again, this value is normalized by the size of the misspelling. To compute this value, we have built a matrix containing the distance of all the keys of a keyboard. Due to space restriction, it is not possible to detail here the way this distance is calculated, the reader may refer to [5].

3.2.2 Features that use the Web

We now propose five features whose values have to be calculated using the Web. Among them, the probability of n-grams allows us to take into account the context of the misspellings. We have decided to limit this context for algorithmic reasons because those probabilities are calculated using a lot of requests to the Google search engine.

We have chosen to introduce smoothing during the calculation of frequencies to avoid the problems of dividing by zero during the calculus of probabilities.

Frequency of a candidate suggestion: The first feature we have considered is the frequency of the candidate suggestion on the Web. We search with Google, the number of times the candidate suggestion is indexed by this search engine.

Probability of 2-grams: This feature corresponds to the probability of the 2-gram made up of the candidate suggestion CS and the preceding token T_{before}. To estimate this probability we use the classical Bayes formula:

$$P_{2gram}(T_{before}\ CS) = P(CS/T_{before}) \times P(T_{before}).$$

In fact $P(T_{before}) = \frac{\#Occ(T_{before})}{\#Occ(*)}$ where $\#Occ(*)$ is the total number of tokens indexed by Google. The problem is that it is difficult to compute this number. To overcome this drawback, we decided to estimate $P(T_{before})$ in that way:

$$P(T_{before}) = \frac{\#Occ(T_{before}\ CS))}{\#Occ(*\ CS)}$$

where $\#Occ(*\ CS)$ is the number of occurrences of the candidate suggestion CS when it is preceded by another token, whatever it is.

To compute $P(CS/T_{before})$, we use the conditional probabilities formula:

$$\begin{aligned} P(CS/T_{before}) &= \frac{P(T_{before} \cap CS)}{P(T_{before})} \\ &= \frac{\#Occ(T_{before}\ CS)}{\#Occ(T_{before})}. \end{aligned}$$

So, we deduce that:

$$P_{2gram}(T_{before}\ CS) = \frac{\#Occ(T_{before}\ CS)}{\#Occ(T_{before})} \times \frac{\#Occ(T_{before}\ CS)}{\#Occ(*\ CS)}.$$

Probability of left 3-grams: On the same principle as 2-gram, this feature represents the probability of 3-gram, noted P_{3gram1}, made up of the candidate suggestion CS and the two preceding tokens T_{b1} and T_{b2}:

$$\begin{aligned} P_{3gram1}(T_{b1}\ T_{b2}\ CS) &= P(CS/T_{b1}\ T_{b2}) \\ &\times\ P(T_{b2}/T_{b1}) \\ &\times\ P(T_{b1}). \end{aligned}$$

Using the same reasoning steps as for the probability of 2-grams, we get:

$$P_{3gram1}(T_{b1}\ T_{b2}\ CS) = \frac{\#Occ(T_{b1}\ T_{b2}\ CS)}{\#Occ(T_{b1}\ T_{b2})} \times \frac{\#Occ(T_{b1}\ T_{b2})}{\#Occ(T_{b1})} \times \frac{\#Occ(T_{b1}\ T_{b2}\ CS)}{\#Occ(*\ T_{b2}\ CS)}.$$

Probability of centered 3-grams: The fourth feature, noted P_{3gram2}, is the probability of the 3-gram made up of the candidate suggestion CS, the token T_b before CS and the token T_a after CS.

It is computed in the same way as the two previous probabilities:

$$P_{3gram2}(T_b\ CS\ T_a) = \frac{\#Occ(T_b\ CS\ T_a)}{\#Occ(T_b\ CS)} \times \frac{\#Occ(T_b\ T_a)}{\#Occ(T_b)} \times \frac{\#Occ(T_b\ CS\ T_a)}{\#Occ(*\ CST_a)}.$$

Probability of co-occurrence: Finally, the last feature, noted $Cooc$, allows the system to measure if the couple of tokens (T, CS) made up of the misspelling T and a candidate suggestion CS, appears frequently or not in the same pages on the Web. Indeed, if such a couple appears frequently, we can suppose that the suggestion has good chance to be a good one. The value of this feature is processed in that way:

$$Cooc(T, CS) = \frac{\#Occ2(T, CS)}{\#Occ(CS)}$$

where $\#Occ2(T, CS)$ is the number of pages, returned by Google, where T and CS appear simultaneously.

4 Experiments

In this section, we are going to make some experiments in order to compare our WebSpell system with state of the art spell checkers using the *Suggestion Intelligence* and *Suggestion Intelligence First* criteria.

4.1 Experimental protocol

We used various English corpora found on the Web in order to get a good diversity in the linguistic style and origins of the documents that have been considered.

Those corpora led us to build a file containing 700 positive examples, i.e. good candidate suggestions constituted by vectors of eleven feature values. As we always get more bad suggestions than good ones, those corpora led us to generate 30,000 negative examples, i.e. bad suggestions constituted by vectors of eleven feature values. Thus, the resulting set of positive and negative examples is unbalanced, that is known to be a difficulty for C4.5 to make it learn an efficient decision tree. To overcome this drawback, we ran some prototype selection techniques based on those proposed by Sebban et al. in [20] resulting in a final learning set of 1100 examples.

We ran C4.5[1] on this learning set and get a decision tree modelling good and bad candidate suggestions.

Using this decision tree, we ran WebSpell on a test set made up of new sentences containing 500 identified misspellings. This set has also been built using English texts found on the Web. Hence, WebSpell can count, for each misspelling, the number of times the good suggestion appears in the list of suggestions whatever its position (which provides the *Suggestion Intelligence* score) and at the first position (which provides the *Suggestion Intelligence First* score).

4.2 Comparison with state of the art spell checkers

Table 2 summarizes the results obtained by the various spell checkers on the test corpus we built in the context of this experiment. To assess the relevance of Web-based features of our system, we run WebSpell using either all the eleven features (WebSpell v1), or only those not issued from the Web (WebSpell v2).

Spell Checker	SI	SIF
Ispell	78	39
Aspell	79	58
Word	78	65
WebSpell v1	**91**	**72**
WebSpell v2	**87**	**54**

Table 2. Comparative study of WebSpell with state of the art spell checkers

We can see that WebSpell significantly outperforms other spell checkers in terms of *Suggestion Intelligence* using the features based on the Web or not. Concerning *Suggestion Intelligence First*, without the Web, our system performs badly compared with Aspell and Word, but we can see that Web-based features have a significant impact on the efficiency of the system. This was expected at the beginning

[1] We used the weka library at http://www.cs.waikato.ac.nz/ml/weka/

of our investigations and we may see it has been verified by the experiments.

5 Conclusion

In this paper, we have presented the WebSpell system which is able to spell check a document thanks to the help of the Web and more precisely the Google search engine. Our experiments have shown that WebSpell outperforms *Ispell*, *Aspell* and *Word* on the two classical criteria *Suggestion Intelligence* and *Suggestion Intelligence First*. In fact, thanks to the Web, WebSpell is able to take into account the context of the candidate suggestions in order to efficiently choose the good ones using a learned decision tree and ranking them according to their distance to the separator hyperplane. WebSpell is easy to train and one of its major issues is that it may evolve in the same way the Web evolves. That means the system may linguistically evolves because the evolution of the content of millions of Web pages reflects linguistic evolutions among the time. WebSpell works whatever the language we use provided that it is sufficiently represented on the Web.

One new way of research we want to explore now is the construction of a corpus of contexts of tokens on the Web. Such a corpus, which will obviously be huge, would be really valuable to be able to take into account even more the context of tokens while correcting a text. Indeed, as Office2007 is beginning to do, non misspelling tokens could be pointed out by the system as non correct tokens thanks to their contexts. In fact, that will lead us to built a kind of semantic error checker.

References

[1] M. A. Elmi and M. Evens. Spelling correction using context. In *Proceedings of the International Conference on Computational Linguistics*, pages 360–364. Morgan Kaufmann, 1998.

[2] R. Ghani, R. Jones, and D. Mladenic. Using the web to create minority language corpora. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 279–286. ACM Press, 2002.

[3] G. Grefenstette and J. Nioche. Estimation of english and non-english language use on the WWW. In *Proceedings of the RIAO International Conference*, pages 237–246, 2000.

[4] N. Hathout and L. Tanguy. Webaffix : a tool for finding and validating morphological links on the WWW. *Proceedings of the International Conference on Linguistic Resources and Evaluation*, May 2002.

[5] S. Jacquemont, F. Jacquenet, and M. Sebban. Webspell, a machine learning approach to spell checking based on the web. Technical report, Univ. of Saint-Etienne, France, 2006.

[6] F. Keller and M. Lapata. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29:459–484, 2003.

[7] M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the International Conference on Computational Linguistics*, pages 205–210, 1990.

[8] A. Kilgarriff and G. Grefenstette. Introduction to the special issue on the web as a corpus. *Computational Linguistics*, 29:333–347, 2003.

[9] W. Kraaij, J. Nie, and M. Simard. Embedding web-based statistical translation models in cross-language information retrieval. *Computational Linguistics*, 29:381–419, 2003.

[10] M. Lapata and F. Keller. The web as a baseline: Evaluating the performance of unsupervised web-based models for a range of NLP tasks. In *Proceedings of the Conference of the Association for Computational Linguistics*, pages 121–128, 2004.

[11] K. Markert, N. Malvinam, and N. Modjeska. Using the web for nominal anaphora resolution. In *Proceedings of the European Assocation of Computational Linguistics Workshop on the Computational Treatment of Anaphora*, pages 39–46, 2003.

[12] T. McEnery and A. Wilson. Corpus linguistics. *Edinburgh University Press*, 1996.

[13] L. Philips. Hanging on the metaphone. *Computer Language*, 7(12):39–43, dec 1990.

[14] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[15] D. R. Radev and K. R. McKeown. Building a generation knowledge source using internet-accessible newswire. In *Proceedings of the Conference on Applied Natural Language Processing*, 1997.

[16] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. Prager. Mining the web for answers to natural language questions. In *Proceedings of the International Conference on Information and Knowledge Management*. ACM Press, 2001.

[17] P. Resnik. Mining the web for bilingual text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 527–534, Maryland, USA, 1999.

[18] P. Resnik and N. A. Smith. The web as a parallel corpus. *Computational Linguistics*, 29:349 – 380, 2002.

[19] C. Santamaria, J. Gonzalo, and F. Verdejo. Automatic association of web directories with word senses. *Computational Linguistics*, 29:485–502, 2003.

[20] M. Sebban and R. Nock. Combining feature and prototype pruning by uncertainty minimization. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 533–540. Morgan Kaufmann, 2000.

[21] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[22] A. Way and N. Gough. wEBMT: developing and validating an example-based machine translation system using the world wide web. *Computational Linguistics*, 29:421–457, 2003.

[23] X. Zhu and R. Rosenfeld. Improving trigram language modeling with the world wide web. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, 2001.