Multi-Level Feature Extraction for Spelling Correction

Johannes Schaback
Institut für Softwaretechnik
Technische Universität Berlin
Strasse d. 17 Juni 135, Germany
`schabby@cs.tu-berlin.de`

Fang Li
Department of Computer Science
Shanghai Jiao Tong University
Dong Chuan Road 800, China
`li-fang@cs.sjtu.edu.cn`

Abstract

For an advanced implementation of spelling correction via machine learning, a multi-level feature-based framework is developed. In order to use as much information as possible, we simultaneously include features from the character level, phonetic level, word level, syntax level, and semantic level. These are evaluated by a support vector machine to predict the correct candidate. Our method allows to correct non-word errors as well as real-word errors simultaneously using the same feature extraction methods, and it closes the gap separating isolated error correction techniques from context-sensitive methods. In contrast to previous approaches, our technique is not confined to correct only words from precompiled lists of "confused" words. Regarding the correction capabilities of our system, we outperform *Microsoft Word*, *Google*, *Hunspell*, *Aspell* and *FST* in recall by at least 3% even if confined to non-word errors. The recall of our system ranges from 90% for the first candidate to 97% for all five candidates presented.

***Index Terms*— context-sensitive spelling correction, lexical disambiguation, machine learning, isolated error correction.**

I. Introduction

Since the 1990's, research on spell checking has focused either on non-word errors or on real-word errors [13]. While non-word errors such as `ohuse` for `house` can easily be detected by validating each word of a given text against a lexicon, their correction, however, is not trivial. Increasingly difficult to handle are real-word errors resulting in valid but unintended words as for instance `out` in the sentence `I am going our tonight`. Fortunately, the majority of these real-word errors provokes syntactic inconsistencies over sentences and can be reliably found by grammar checkers. Thus *correction* is a much bigger challenge than *detection*, both for real-word and non-word errors.

Since real-word error checkers mainly depend on *lexical disambiguation* techniques while non-word error checkers apply *approximate string-matching* methods which usually involves establishing a character substitution metric to find the candidate with the shortest distance to the typo, both research areas were strictly separated so far[1]. Here, we combine both approaches by introducing an abstract framework in which the techniques of both areas contribute information for candidate discrimination via feature extraction methods that were derived from *lexical disambiguation* as well as from *approximate string-matching*. Moreover, current context-sensitive spelling correctors that are required for real-word errors mainly rely on two kinds of features: *collocations* and *co-occurrences* [19; 2]. Since both types generate a high-dimensional feature space, they have been used only for short, handcrafted lists of commonly confused words. List-based methods are very successful if the lists are large enough [4], but we want to cover a wide list-independent range of words and therefore we present a general collocation and co-occurrence model that evaluates each frequent word as a candidate against a context and thereby corrects real-word errors as well as non-word errors. Finally, we apply *candidate discrimination* by removing unlikely candidates from a candidate list based on features extracted either from the misspelling itself (on the character level) or from the context (on the word level, syntactic level, and semantic level) surrounding the misspelling. After narrowing down the candidate list to five candidates, the final candidate is selected by a support vector machine.

In this particular work we address the problem of errors made in human written texts, not errors made by *optical character recognition* (OCR) although it is a most intriguing research area. Errors made by OCR systems are typically very different from those made by humans. While OCR errors are often due to similar shapes of characters, like `in` where `m` would have been correct, or `op` instead of `qo`, human errors are usually based on motor coordination slips, phonetic similarities or spelling ignorance [13]. Thus, from a statistical point of view, the underlying, error generating distributions of the two sources are very different. As a consequence, human errors result more often in real words than OCR errors which requires to condition the models especially for either OCR errors or human errors. In [12] Kolak et. al. propose "an end-to-end process in the noisy channel framework" that combines segmentation and spell

[1]There are some exceptions however, such as an approximate string-matching algorithm that can detect subtokens in strings [14].

checking on a per string level rather than on a a per word level and is especially tailored for OCR errors. It is worth mentioning that our framework can be applied to the very interesting case of OCR induced errors as well, although with different feature extraction methods, such as for shape information and recognition confidence values. Since this requires a lot of parameter estimation we cannot discuss the problem of OCR errors here in order to keep this paper self-contained.

The following section contains a technical overview including a level-wise description of our multi-level feature extraction methods. The final section provides experiments and results, showing that our multi-level technique performs better than previous single-level methods.

II. System Overview

A. Architecture

Our framework can be roughly partitioned into two parts: the isolated error correction part and the consecutive context-sensitive error correction part as shown in Figure 1. In the isolated error correction part, we basically generate the candidate set and apply a first and rough discrimination. We employ two algorithms for candidate generation. The first one is an instance of a phonetic algorithm that selects all those words from the lexicon that are similarly pronounced. The second algorithm adds all words to the candidate set that are one edit operation away from the misspelling. The resulting candidate set is then ranked by using a modified improved error model that was originally proposed by Brill and Moore [3]. All candidates that are not among the five highest ranked are dismissed from the candidate set. The five remaining candidates are passed to the context-sensitive error correction part. There, we gather further discrimination evidence from the word level, syntactic level and semantic level. The evidence is evaluated by a support vector machine which re-ranks the five remaining candidates where the highest ranked candidate is regarded as the correction.

B. Phonetic Level

The major part of the initial candidate set is made up of those words that are similarly pronounced like the misspelling. To find these words we employ the well-known *Soundex* algorithm. Like every phonetic algorithm, *Soundex* maps a given string to a phonetic code, e.g. $soundex(fisikle) = F224 = soundex(physical)$. The mapping is implemented by a number of simple character replacement rules. Specifically, we compute the phonetic code for the given misspelling and retrieve all words from the lexicon that are mapped to the same code. This can be seen as a computationally simple approximation to the idea of pronunciation modeling [18]. The words are efficiently retrieved by traversing a precompiled trie of phonetic codes generated from all words in the lexicon. A list of links to the words having the same phonetic code is stored in each leaf of the trie. Since *Soundex* codes are not longer than four characters, the space and computational effort is low compared to the size of the lexicon. The very rough character replacement rules cause *Soundex* to map on a small domain such that many words

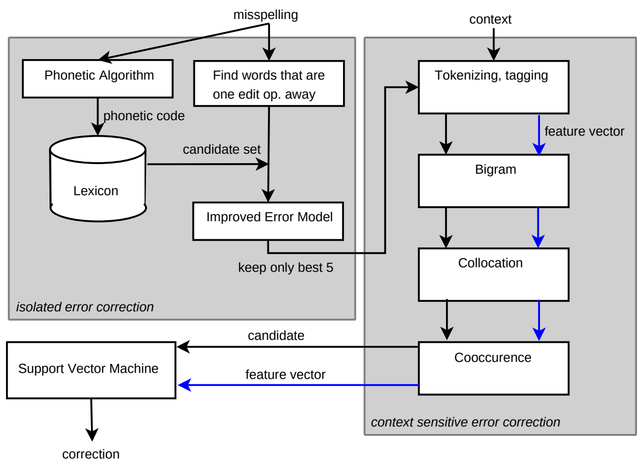

Fig. 1. The architecture of our system. The isolated error correction part takes the misspelling and outputs a candidate set of five candidates which are passed to the context-sensitive part. There, the context undergoes several feature extraction methods that stores their feature values in a feature vector for each candidate. The SVM finally performs a one-versus-all classification to decide which candidate is the correction.

have the same phonetic code. Concretely, *Soundex* maps 87% of all misspellings out of our set to the same phonetic code as the correction. Although successors of *Soundex* such as *Metaphone* retrieve fewer words, their recall falls below 87%. Since recall is more important than precision in the candidate creation phase we rely on *Soundex*.

C. Character Level

Since we only consider words as possible corrections that were included into the candidate set, the choice of the candidate set is crucial to the overall performance of the correction algorithm. By adding all words that are one edit operation away (*insertion*, *deletion*, *single character substitution*, *swap*) we improve our candidate set accuracy to 97% with an average size of 122 candidates per misspelling. This is motivated by Damerau's [6] finding that about 80% of all misspellings are single instances of insertion, deletion or substitution.

To reduce the size of the candidate set we apply a slightly modified instance of the improved error model proposed by Brill and Moore [3]. While the common edit distance [7] only considers one edit operation at a time, the improved error model trains a metric for multiple character edit operations [15; 17] as an instance of the noisy channel spelling correction framework [11]. Specifically, the probabilities for multiple character edit operations are evaluated via string substitutions of a particular partition of the misspelling and the correction. The product of these probabilities measures how far the misspelling is from the candidate under the condition that both input strings are completely partitioned and that the product probability is maximized over all possible partitions. For example the misspelling `fisikle` and candidate `physical` may be partitioned as $p(f\,|\,ph)p(i\,|\,y)p(s\,|\,s)p(i\,|\,i)p(k\,|\,c)p(le\,|\,al)$.

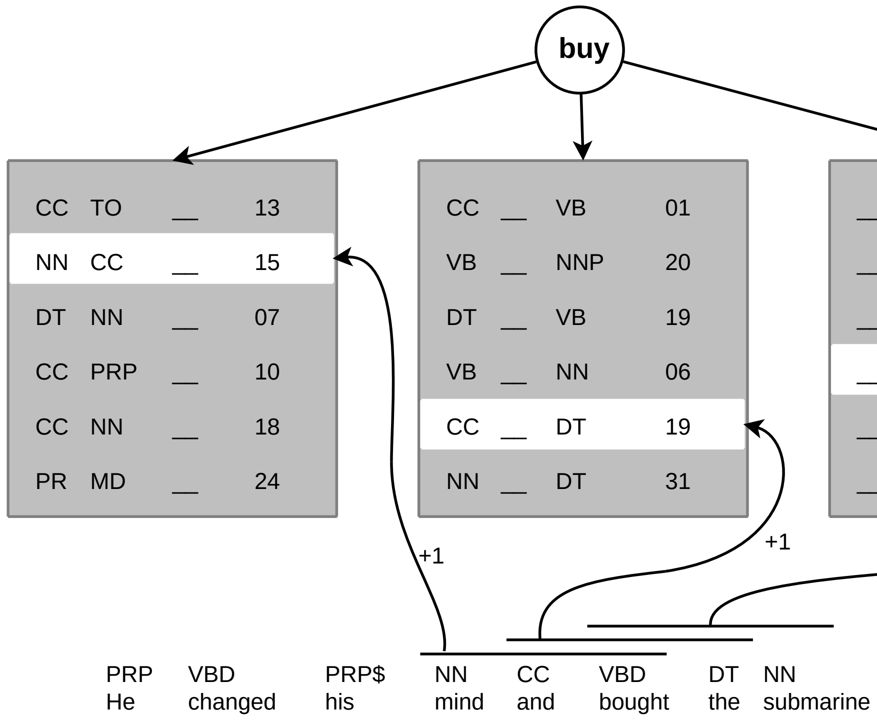

Fig. 2. Example of our collocation model for the target word `to buy`. A window of three POS tags is shifted over the text such that three POS patterns are extracted for each target word. Each pattern is added to one of three lists according to the position of the target word in the window.

While the underlying mathematics remains the same, we devised a different dynamic programming algorithm to find the optimal partition and the accumulated substitution probabilities, without requiring a lexicon that is pre-compiled into a trie.

D. Word Level

On the word level we use a word sequence bigram as a simple language model to extract evidence on how well a candidate fits into its immediate environment. Suppose we have a sentence $w_0 w_1 \cdots$, where w_i marks the misspelling, we evaluate the bigram for each candidate over the word left to the misspelling and over the word to the right. Formally, we compute $p_{bi}(w_i \,|\, w_{i-1})$ as well as $p_{bi}(w_{i+1} \,|\, w_i)$ where p_{bi} computes the bigram frequency with *add-one smoothing*. Both values are stored in the feature vector for the corresponding candidate along with the frequency of the candidate in our training corpus such that the classifier can level out the potential bias of very frequent candidates. In contrast to higher order n-grams, bigrams do not suffer so much from the sparse-data problem. Consequently, the bigram also returns values for less common candidates.

E. Syntactic Level

The bigram models explicit relationships over word pairs. That is why candidates that are equally probable to appear in text can often not be discriminated by the bare order and frequency of words. On the syntactic level, however, we are often able to find further discrimination evidence. Therefore, we devised a collocation model that has the same underlying idea as [2; 19] but differs in the way it is trained and applied. The basic idea of a collocation model is to match patterns of part-of-speech (POS) tags and words around the misspelling. Following an example of [2], in order to discriminate `whether` against `weather`, we try to match the pattern

__ TO VERB

immediately following the misspelling. If the pattern matches - as for instance in `I don't know whether to laugh or cry` - we can imply that `whether` is the correct candidate. Collocations model the syntactic use of words which is more abstract than explicit word pairs but since common candidates are often used in different syntactic environments, collocation models provide important additional discrimination evidence.

However, our collocation model is fixed to the length of three POS tags, does not use explicit words and is constructed in a lemma-centric manner: We draw the 14.000 most frequent lemmata from our lexicon which form the set L of target lemmata. We initiate three empty lists of POS triples for each target lemma. In the preprocessing phase which is required for training as well as for applying the model, we tag the sentence $w_0 w_1 \cdots$ containing the misspelling as a sequence $t_1 t_2 \cdots$ of POS tags where t_i is the tag at the position of the misspelling. Further, we have a function $lemma$ that maps a given word to its lemma form. In the training phase we fill the aforementioned lists of each target lemma. After tagging the training text, we shift a window of three POS tags over the sequence. If a lemma $lemma(w_i) \in L$ is a target lemma, we add the three triples $t_{i-2}t_{i-1}t_i$, $t_{i-1}t_it_{i+1}$ and $t_it_{i+1}t_{i+2}$ to the lists of the target lemma. The position of the target lemma tag within the triple decides to which list the triple is added such that the occurrences of triples are simply counted in the associated list as illustrated in Figure 2. We do not prune the lists because the complete collocation model still requires only 13 MB in memory on our training corpus. Note that the lists form a vector space model in which we can compute distances over POS triples. Given a misspelling w_i in a context we map the context into the vector space model of $lemma(c)$ for each candidate c and measure how well c fits into the particular syntactic environment.

Formally, the frequencies of a triple for a specific lemma can be retrieved through the functions $freq_{left}$, $freq_{mid}$ and $freq_{right}$. To apply the model, we replace the misspelling by each correction candidate c and re-tag the sentence such that $w_i = c$ gets the tag t_i assigned. We can finally compute how likely it is that the candidate c appears in this syntactic environment. With $lemma(c) = b$ we have

$$\begin{aligned} p_{colLeft}(c \mid t_i, t_{i+1}, t_{i+2}) &= \frac{freq_{left}(b, t_i, t_{i+1}, t_{i+2})}{K} \\ p_{colMid}(c \mid t_{i-1}, t_i, t_{i+1}) &= \frac{freq_{mid}(b, t_{i-1}, t_i, t_{i+1})}{K} \\ p_{colRight}(c \mid t_{i-2}, t_{i-1}, t_i) &= \frac{freq_{right}(b, t_{i-2}, t_{i-1}, t_i)}{K} \end{aligned}$$

where $K = 10$ is an empirical normalization constant. The three likelihoods are stored in the feature vector of c.

F. Semantic Level

A residual class - mainly of nouns such as `dessert` and `desert` - cannot be discriminated very well by neither the bigram nor our collocation model. The reason is that the words are too general for the bigram and syntactically equal such that the collocation model can not make a clear statement either. On the semantic level, however, human readers can easily select the correct candidate because they follow a lexical chain while reading the sentence and can easily pick the candidate that makes the most sense in the given context. Our approach is to simulate this ability with a modified co-occurrence model, again inspired by the co-occurrences described in [2; 19; 9]. Technically, we train a vector space model as a bag-of-words for each target lemma $b \in L$. An individual axis measures the number of occurrences of the according context word.

Before training, we transform each word of our training text into its lemma form and remove stop-words such that we gain an ordered set C. For example, consider the original text:

```
Lenny adopted a child that lost his
         mother and father.
```

After removing stop-words and replacing the remaining words by their lemma form we gain

```
Lenny adopt child lose mother father.
```

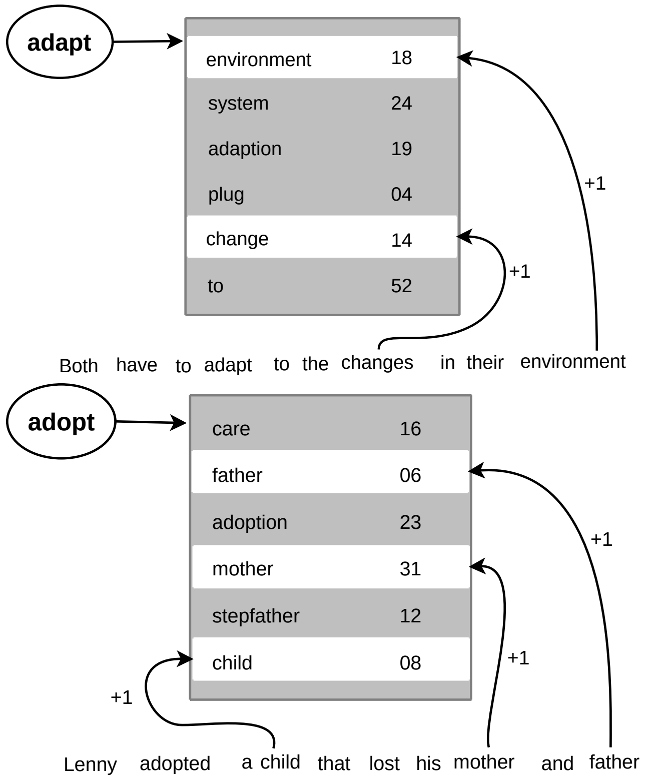

Fig. 3. Co-occurrence model for the two target lemmata `adapt` and `adopt`.

which forms the ordered set C.

From that C we collect all context lemmata that appear in the radius of 10 words around the target lemma b. In the vector space model of b we also store the number of times we encountered a context word in the vicinity of b as depicted in Figure 3. The function $freq_{co}(q, b)$ returns how often the context lemma q appeared in the vicinity of b. The bag-of-words for b can then be defined as

$$B_b = \{ y \mid y \in L : freq_{co}(y, b) > 0 \}$$

To reduce noise and prune the model at the same time, we remove those context words that appeared less frequent than the mean frequency of all context words of the target lemma b. Formally, we keep $\widehat{B_b}$ with

$$\widehat{B_b} = \{ y \mid y \in B_b : mean(b) > freq_{co}(y, b) \}$$

where

$$mean(b) = \frac{1}{|B_b|} \sum_{k \in B_b} freq_{co}(k, b)$$

The pruned model consumes 20MB after training. To apply the model, we define two distance functions d_1 and d_2 that estimate how well a given lemma b fits into a given context. The context shall be here C again, because it undergoes the same preprocessing steps as in the training phase.

$$d_1(C,b) = \frac{|\{q \mid q \in C : freq_{co}(q,b) > 0\}|}{\left|\widehat{B_b}\right|}$$

$$d_2(C,b) = \sum_{q \in C} \frac{freq_{co}(q,b)}{K_{cw}}$$

The function d_1 returns the ratio of actually found context words in C and possibly findable context words. This describes how similar the trained context and the given context are for candidate b. In contrast to that, d_2 considers how significant the found context lemmata are by summing the normalized frequencies of the context lemmata. The constant $K_{cw} = 250$ is found empirically, but the result is not very sensitive to this constant anyway. As a third feature, we use $d_3(b) = |\widehat{B_b}|\frac{1}{K_{cb}}$ that simply measures how big the vector space model is for b. It helps the SVM to estimate the confidence for the values computed for $d_1$and d_2. The normalization constant K_{cb} is the maximum over all vector space model sizes. It ranges between 180 and 500 according to from which radius we pick the context words.

G. Classifier

As mentioned above, the feature values that have been extracted for a candidate are stored in the candidate's feature vector. Recall that only the last five candidates are considered. The feature vectors are passed to a support vector machine employing a simple radial basis function kernel with $\gamma = 1$ because all feature values are already normalized to values between 0 and 1. We employed Joachims SVM^{light} [10] implementation. All together we use the following features:

Improved Error Model	rank
bigram left	$p_{bi}(w_i \mid w_{i-1})$
bigram right	$p_{bi}(w_{i+1} \mid w_i)$
candidate frequency	frequency / 10.000
bigram product	$p_{bi}(w_{i+1} \mid w_i) \times p_{bi}(w_i \mid w_{i-1})$
collocation left	$p_{col}(c \mid t_{i+1}, t_{i+2})$
collocation middle	$p_{col}(c \mid t_{i-1}, t_{i+1})$
collocation right	$p_{col}(c \mid t_{i-2}, t_{i-1})$
collocation product	$p_{col}(c \mid t_{i+1}, t_{i+2}) \times$ $p_{col}(c \mid t_{i-1}, t_{i+1}) \times$ $p_{col}(c \mid t_{i-2}, t_{i-1})$
co-occurence	$d_1(C,b)$
co-occurence	$d_2(C,b)$
co-occurence	$d_3(b)$

We train the SVM model on a data set extracted from our training corpus that consists of (*candidate*, *feature vector*, *label*) triples. The label is either +1 for the correction and -1 for false candidates. In prediction mode, the SVM sets the labels to values between -1 and +1 such that we can rank the last five candidates accordingly. The highest ranked candidate is finally taken as the correction.

III. Training and Test Corpus

Our corpus from which we spawned the training and test corpus consists of the plain text English Wikipedia (1.5 GB) and a list of (*misspelling*, *correction*) pairs. The 11.000 pairs were drawn from the Wikipedia misspelling project, ASpell's official evaluation list and the Birkbeck Spelling Corpus compiled by Mitton in 1986. From this list, we removed duplicates and corrections that were not found in our lexicon. We split both the list and the English Wikipedia into a training corpus and a test corpus. The latter contains the last 20% of the English Wikipedia 402 randomly selected pairs of misspellings and corrections from the list. Among the 402 are about two dozen real-word errors. For each correction of the test list we found an appropriate sentence in the test part of the Wikipedia while we found a sentence in the training part for each correction in the training list.

We trained the improved error model on the complete training list of misspellings and corrections and fine-tuned the model with four iterations of the expectation-maximization algorithm. While the bigram was trained only on the first 10 MB of the training part of the Wikipedia, we used the complete training text for the collocation model and co-occurrence models. Both models required to find very many occurrences of the target lemmata such that a big corpus was necessary. The SVM model was trained on 2000 selected items (*correction*, *misspelling*, *context*) from the training corpus. Note that each such triple generates five labeled feature vectors, one for each candidate. The data flow of the training and testing procedures is illustrated in Figure 4.

IV. Experiments and Results

To show that each feature extraction method contributes important discrimination information to the overall performance, we first trained the candidate generation process and then subsequently added the individual feature extraction methods to the system. Each time we added and trained a feature extraction method we re-trained the SVM model to optimally adapt the system to the additional information. We based our evaluation on precision and recall measures as if spell checking were a retrieval task. Since we have only one valid result for a "query" in spell checking, we compute an *accumulated* precision and recall. Recall is calculated as the number of found corrections divided by the number of misspellings while precision is the number of times the correction has been found divided by the sum of the sizes of the returned sets. Thus, 0.2 is the maximal precision for methods offering always 5 candidates.

We first evaluated the improved error model with our candidate generation method and found this preliminary system to have about the same performance as [3] on our test corpus. Without our candidate preselection, the improved error model performed in isolation slightly worse over the entire lexicon (123.000 entries). When focusing on the best ranked candidate only, we found a recall of 86%. Considering the first and second ranked candidate, the correction is in 92% of the cases among the candidates. If more candidates are generated, the maximal observed hit rate of 97% is reached by presenting the five highest ranked candidates. This justifies why we only kept the five highest ranked candidates for further processing.

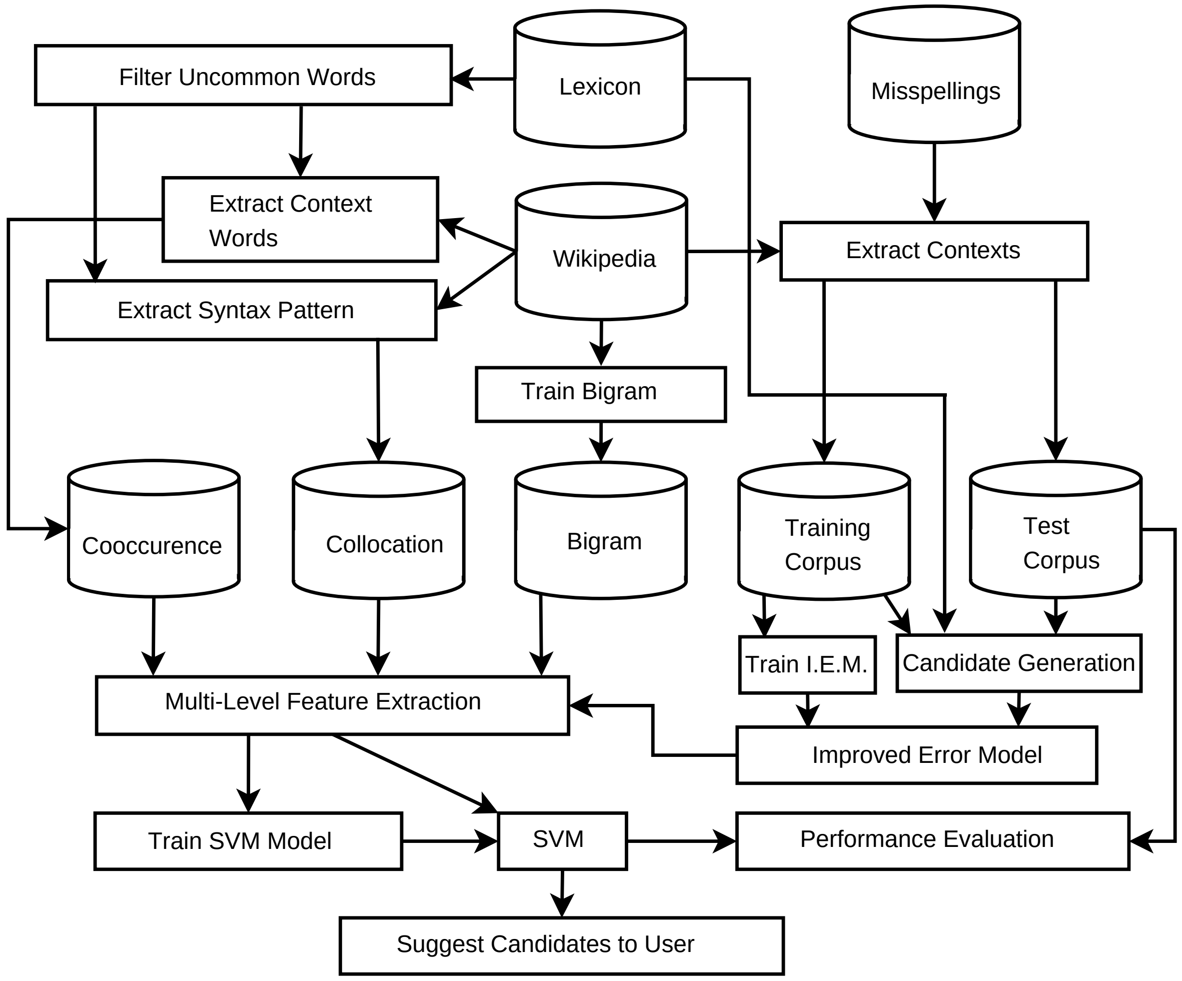

Fig. 4. Data flow diagram of our system. The misspelling corpus is split in the training corpus and the test corpus after it got annotated with contexts. The training corpus is directly used to condition the improved error model. Isolated from the character level related data, the co-occurence model, collocation model and bigram model are trained over that part of the Wikipedia that has not been used to annotate the spelling corpus. All models contribute together to the multi-level feature extraction part which forwards its data to the SVM to make the final decision.

After adding the bigram technique we gained an performance increase of 3% for the first candidate. Furthermore, adding the collocation model and co-occurrence model gave another 4% increase each. Figure 5 shows the performance improvement over five candidates for the bigram, collocation and co-occurrence after they were subsequently added. It shows that each additional feature contributes to the overall performance such that the sum of the features performs better than a single language model. The exact and final values are shown in Table I where *hits* indicates the number of correctly found candidates. Since the re-trained SVM now uses the full feature map, the upper limit of 97% is even reached within the first four candidates.

We compared our system to competitive spell checkers that are considered state-of-the-art. We selected spell checkers that use different technologies and thereby cover a wide range of techniques in the field. Among them are *Google*, *MS Word*, *Aspell*, *Hunspell* and *FST* [8]. We evaluated these third-party correctors on the test data we used for our system. But since most of these programs do not accept other input than single words (and experiments with *MS Word* led to the observation that the spell checker of *MS Word* seems also to be isolated-correction based), this comparison was not possible in full generality, since we could only use the test list of (*misspelling*, *correction*) pairs. To be as fair as possible to each competing program, we eliminated pairs for which the spell checker's lexicon did not contain the

TABLE I
CO-OCCURRENCE + COLLOCATION + BIGRAM + I.E.M

N-Best	*Hits*	*Precision*	*Recall*
1-Best	364	0.90	0.90
2-Best	382	0.47	0.95
3-Best	389	0.32	0.96
4-Best	390	0.24	0.97
5-Best	390	0.19	0.97

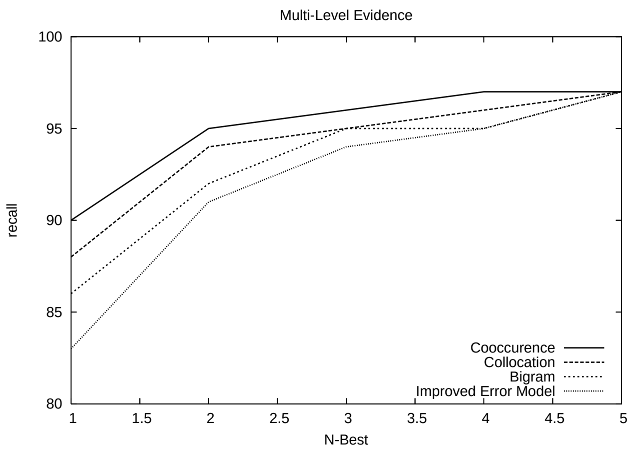

Fig. 5. Result of the experiment showing the performance increase as we added the bigram, collocation model and co-occurrence model. This shows that all employed feature extraction methods contribute to the overall performance.

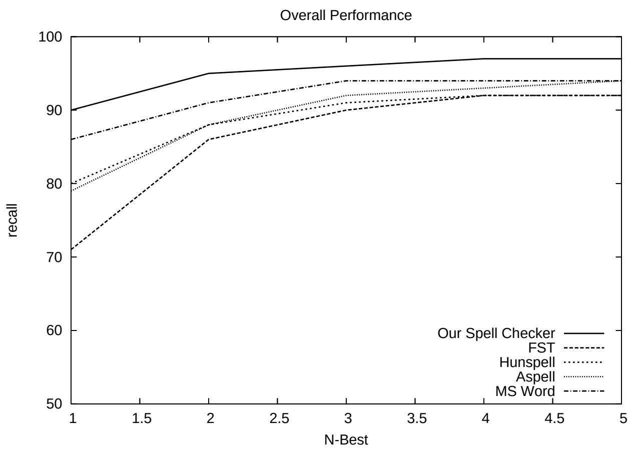

Fig. 6. Comparison of competitive spell checkers with regard to an increasing number of considered suggestions.

TABLE II
CORRECTION ABILITY OF 3RD-PARTY SPELL CHECKERS

Spell Checker	*Hits*	*Precision*	*Recall*
Our System	390	0.24	0.97
MS Word	248	0.46	0.94
Aspell	380	0.08	0.94
Hunspell	374	0.40	0.93
FST	376	0.52	0.92
Google	316	0.78	0.78

correct word, and those where the spell checker considered the misspelling as a correct word. Thus the spell checkers were tested on the range covered by their respective lexica yielding the result shown in Table II. Figure 6 shows the results over the first five candidates. It clearly shows that all features contribute to the overall performance and are not superfluous. However, this comparison does not show how well our system performs exclusively on real-word errors.

It is worth mentioning that Google as an instance of the more recent online spell checkers only returns one correction candidate. It has thus a high precision which is arguably mainly to correct search queries that are typically low on context and require a single candidate for correction. While most spell checkers rely on sparse statistics about the use and frequency of a word drawn from some limited corpus or dictionary, spell checkers that are associated with a search engine can define how popular a particular spelling variant is based on occurrences of the variant in user queries [5; 1] and on the web. This also enables to spell check proper names, places and identifiers and words that are in no dictionary.

V. Conclusions

We propose a general feature based framework that reduces spell checking to a plain machine learning task where previous methods have the classification hard-wired into the feature extraction models. Furthermore, we are the first who introduce the powerful concept of kernel machines into the field of spell checking. Another goal was to close the gap between isolated error correction and context-sensitive error correction which has been achieved by abstracting both concepts to two independent feature extraction methods. Both methods were then easily combined in our framework. This way we are able to combine the powerful ideas of the improved error model, phonetic algorithms, n-grams, collocations and co-occurences as feature extraction methods without being confined to confusion sets.

Our work shows that there is a lot to gain by combining context-sensitive and isolated error correction methods and to apply them in order to correct real-word errors as well as non-word errors. In contrast to other methods which are confined to confusion sets, we can now correct every frequent word occurring in text. The fundamental idea is to handle the problem of spell checking as a machine learning task. Several feature extraction modules provide discrimination evidence. This way we are able to combine the powerful ideas of an improved error model, phonetic algorithms, n-grams, collocations, and co-occurrences. This allows a good candidate ranking based on multi-level features. Furthermore, we seem to be the first to introduce kernel machines into the field of spell checking.

In future work we will evaluate the performance of our system for the post-correction of misclassified words for OCR, similar to [16; 20] but not only on the character level. Further training with carefully selected genuine real-world (non-Wikipedia) data should allow to increase the performance of our system as well as to enable to evaluate the real-word error correction ability more appropriately. Finally, we could improve the candidate ranking provided by the SVM by introducing a confidence value which allows to neglect other candidates if the value is relatively high. This would improve overall precision as a result.

Acknowledgements

This work has been supported by the *Verein zur Förderung der Softwaretechnik e.V.* (VFSWT) and a travel grant from

IJCAI-07.

References

[1] Ahmad, F., and Kondrak, G. Learning a spelling error model from search query logs. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (Vancouver, British Columbia, Canada, October 2005), Association for Computational Linguistics, pp. 955–962.

[2] Andrew R. Golding, D. R. A winnow-based approach to context-sensitive spelling correction. *Machine Learning (Special Issue on Natural Language Learning)* (October 1998).

[3] Brill, E., and Moore, R. C. An improved error model for noisy channel spelling correction. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics* (Morristown, NJ, USA, 2000), Association for Computational Linguistics, pp. 286–293.

[4] Carlson, A., Rosen, J., and Roth, D. Scaling up context sensitive text correction, 2001.

[5] Cucerzan, S., and Brill, E. Spelling correction as an iterative process that exploits the collective knowledge of web users. *Conference on Empirical Methods in Natural Language Processing (EMNLP). Barcelona* (2004).

[6] Damerau, F. J. A technique for computer detection and correction of spelling errors. *Commun. ACM 7*, 3 (1964), 171–176.

[7] Damerau, F. J. A technique for computer detection and correction of spelling errors. *Commun. ACM 7*, 3 (1964), 171–176.

[8] Eisele, A., and vor der Brück, T. Error-tolerant finite-state lookup for trademark search. In *27th Annual German Conference on AI, KI 2004, Ulm, Germany, September 20–24, 2004, Proceedings.* (2004). Springer Best Paper Award.

[9] Ginter, F., Boberg, J., Järvinen, J., and Salakoski, T. New techniques for disambiguation in natural language and their application to biological text. *J. Mach. Learn. Res. 5* (2004), 605–621.

[10] Joachims, T. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*, A. S. B. Schölkopf, C. Burges, Ed. MIT Press, Cambridge, MA, 1998.

[11] Kernighan, M. D., Church, K. W., and Gale, W. A. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics* (Morristown, NJ, USA, 1990), Association for Computational Linguistics, pp. 205–210.

[12] Kolak, O., Resnik, P., and Byrne, W. A generative probabilistic ocr model for nlp applications. *In HLT-NAACL* (2003).

[13] Kukich, K. Techniques for automatically correcting words in text. *ACM Comput. Surv. 24*, 4 (1992), 377–439.

[14] Lopresti, D., and Tomkins, A. Block edit models for approximate string matching. *Theoretical Computer Science 181*, 1 (1997), 159–179.

[15] Oncina, J.; Sebban, M. Learning unbiased stochastic edit distance in the form of a memoryless finite-state transducer. *International Joint Conference on Machine Learning* (2005). Workshop: Grammatical Inference Applications: Successes and Future Challenges.

[16] Ringlstetter, C., Schulz, K. U., Mihov, S., and Louka, K. The same is not the same - post correction of alphabet confusion errors in mixed-alphabet OCR recognition. In *ICDAR* (2005), pp. 406–410.

[17] Ristad, E. S., and Yianilos, P. N. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 5 (1998), 522–532.

[18] Toutanova, K., and Moore, R. C. Pronunciation modeling for improved spelling correction. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (Morristown, NJ, USA, 2001), Association for Computational Linguistics, pp. 144–151.

[19] Yarowsky, D. Decision lists for lexical ambiguity resolution: application to accent restoration in Spanish and French. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics* (Morristown, NJ, USA, 1994), Association for Computational Linguistics, pp. 88–95.

[20] Zhuang, L., and Zhu, X. An ocr post-processing approach based on multi-knowledge. *International Conference on Knowledge-Based Intelligent Information & Engineering Systems* (2005), 346–352.