A visual and interactive tool for optimizing lexical postcorrection of OCR results

Christian Strohmaier
SfS – Univ. of Tübingen
strohmai@sfs.uni-tuebingen.de

Christoph Ringlstetter, Klaus U. Schulz*
CIS – Univ. of Munich
{kristof,schulz}@cis.uni-muenchen.de

Stoyan Mihov*
LPDP – Bulgarian Academy of Sciences
stoyan@lml.bas.bg

Abstract

Systems for postcorrection of OCR-results can be fine tuned and adapted to new recognition tasks in many respects. One issue is the selection and adaption of a suitable background dictionary. Another issue is the choice of a correction model, which includes, among other decisions, the selection of an appropriate distance measure for strings and the choice of a scoring function for ranking distinct correction alternatives. When combining the results obtained from distinct OCR engines, further parameters have to be fixed. Due to all these degrees of freedom, adaption and fine tuning of systems for lexical postcorrection is a difficult process. Here we describe a visual and interactive tool that semi-automates the generation of ground truth data, partially automates adjustment of parameters, yields active support for error analysis and thus helps to find correction strategies that lead to high accuracy with realistic effort.

1 Introduction

The quality of recognition results that are achieved with actual commercial OCR engines is generally high. Still, a non-neglectable number of errors occur even for document layouts with simple fonts and normal font sizes. The number of errors often grows in a significant way for document parts where small font sizes and italic style are used. In this situation, techniques for lexical postcorrection may help to reduce the error rate and to achieve high accuracy [9, 17, 6, 2, 3, 11, 16].

Unfortunately, in most cases no real benefits are achieved with a naive postcorrection strategy. The success of techniques for lexical postcorrection crucially depends on a considerable number of design decisions and parameter settings, and a correction strategy that works well for a given recognition task may fail to produce satisfactory results when treating another group of documents. Since a whole bunch of design decisions and parameters may influence the quality of the final correction result, the adaption of a given correction algorithm to a new application is a complex and time consuming task.

A first crucial decision is the selection of a suitable background dictionary. Since scanned corpora often belong to specific thematic areas, general purpose dictionaries usually fail to reflect vocabulary and word frequencies of a given text. A partial improvement is obtained by using an appropriate selection of specialized dictionaries. Examples are dictionaries for proper names, geographic names, for terminological expressions from fixed thematic areas, acronyms, etc. Another suggestion, discussed in [15], is to derive for each application an additional "dynamic" dictionary that is obtained via analysis of web pages from the given thematic area retrieved by web search engines. In a given application we may then look for an optimal combination of a base dictionary with special dictionaries and a dynamic dictionary. Note that it would be naive to expect optimal results from a dictionary with maximal size, e.g., due to "false friend" errors (s.b.). The question arises of how to find a good selection of dictionaries within the complete set of dictionaries that are at our disposal.

A second step is the design of a scoring function that may be used to rank correction candidates retrieved from the joint dictionary for a given garbled word. A natural way to achieve such a ranking is based on a combination of a distance value and a frequency information. The distance value measures the similarity between the garbled word and a given lexicon entry. The frequency value describes the number of occurrences of a given dictionary entry in a very large standard corpus. Many different options exist for defining a suitable distance function. For example, various variants of the Levenshtein (edit) distance can be used. A second task is to find a good balance between distance values and

*Funded by VolkswagenStiftung

frequency values.

It is not realistic to assume that the dictionary contains each token of the text. For example, in earlier experiments with German documents we only reached a lexical coverage between 93.07% and 98.42% even with very large dictionaries that are enriched with vocabulary from thematic web pages ([15]). Hence, for automated correction tasks a correction candidate should only be used to replace the original OCR result in the presence of additional confidence. In our approach, the correction candidate is only used if its ranking value exceeds a given threshold. The problem arises of how to select an optimal threshold.

If several OCR engines are at our disposal, the question arises of how to combine the recognition/postcorrection results that are achieved with the single engines. In our case we would like to combine the results on the level of words (tokens) that are recognized. Again this process typically evolves several design decisions and fine tuning of parameters.

In this paper we describe a visual and interactive tool that helps to find a good path through the labyrinth of selections, design decisions and parameter adaptions that are needed to fine tune a given system for a new application. For this process we use ground truth data. Hence, in terms of machine learning, the tool facilitates training. For a fixed sample of the document the original text is reconstructed. The sample is analyzed with each OCR engine. Using these data we first search for an optimal postcorrection model for each single OCR engine (Phase 1). Afterwards the models obtained for the single OCR engines are combined into a joint postcorrection model (Phase 2) which is then used (application phase) for analyzing the complete text.

Phase 1 is further divided into three steps. In Step 1 the original text of the sample is automatically aligned with the output of the given OCR engine on the sample. In Step 2 a set of files is created where potential correction candidates for each token recognized by the OCR engine are listed. The correction candidates are selected in the given dictionaries, ignoring that we know the original word. Step 3 is used to search a good postcorrection model for the output of the given OCR engine. This process, which is partially automated and supported by visual tools, includes the selection of dictionaries, the selection of a distance measure, the definition of a scoring function for correction candidates, an interactive error analysis and further points. In order to measure the quality of a given model in terms of correction accuracy we make use of the fact that we know the correct tokens of the original text. Details are described below.

The correction models obtained at the end of Phase 1 produce for each token recognized by an OCR engine a list of correction suggestions where each correction suggestion comes with a confidence value. In Phase 2, which represents work in progress, the correction candidates and the confidence values obtained for the distinct OCR engines are combined, and a joint list of correction candidates is built which is finally used for automated or interactive correction.

Our tool is based on an open architecture. This means that it can deal with any number of dictionaries, computable distance measures for strings, OCR engines, etc.

In the following sections, after a brief discussion of the kind of resources that are needed in our approach, each of the three steps of Phase 1 is described in more detail. The emphasis is on the central Step 3. Section 6 briefly comments on Phase 2. Section 7 describes implementation details and visualization techniques. Section 8 gives evaluation results that were achieved when analyzing texts from distinct thematic areas.

2 Basic resources

We face a situation where several **OCR engines** are used to recognize a printed text. OCR software is considered as a black box with textual output. For postcorrection of the recognition results, a collection $\mathcal{D}$ of **dictionaries** is at our disposal. In these dictionaries, each entry v comes with a value $f(v) \in [0,1]$ representing a **normalized frequency information**. Furthermore we assume that a collection $\mathcal{M}$ of **distance measures** for strings is available, together with suitable algorithms for computing these distances. One distance function, the *basic distance measure* d_0, is distinguished. It is used for preselecting correction candidates (cf. Section 4).

Usually $\mathcal{D}$ and $\mathcal{M}$ are fixed. However, the interactive error analysis in Step 3 of Phase 1 (s.b.) might indicate that better correction results can be expected from a new kind of dictionary, or from another distance measure. In such a situation we may decide to extend/modify the initial ressources.

In our actual system, two commercial OCR engines are integrated. The collection of dictionaries contains large-scale lexica for English, German and Bulgarian language, special subdictionaries for proper names, geos, abbreviations and acronyms. We also use dynamic dictionaries obtained via analysis of web vocabulary, cf. [15]. The standard Levenshtein distance[1] is used as basic distance measure. Furthermore we use the length-sensitive Levenshtein distance[1]. We can also use variants where edit costs depend on specific symbols. However, the implementation of algorithms for optimizing weights is not finished, hence we do not have evaluation results for these distances.

[1] The *standard Levenshtein distance* [13] between two words W and V, denoted $d_L(V,W)$, is the minimal number of letter insertions, deletions and substitutions that are needed to transform V into W. The *length-sensitive Levenshtein distance* is $d'_L(V,W) := d_L(V,W)/(|V|+|W|)$ where $|U|$ denotes the length of the word U.

3 Step 1 of Phase 1: Alignment

In Phase 1, as a first step the output received from the OCR engine is aligned with the reproduced original text of the sample. We compute the **alignment file** AF which contains tuples of the form $\langle w, w^{ocr} \rangle$. Here w is a correct token of the original sample, w^{ocr} represents the recognition result for this token obtained with the given OCR engine. For automated alignment we use a modified version of the usual dynamic programming approach. The alignment problem becomes more difficult for multi-column documents if the natural reading order of the original sample is permuted in the output of the OCR engine. In this case, an analysis of local neighbourhoods is used to reconstruct the proper alignment with the original text. As a matter of fact also this method is limited and there are cases where an automated alignment is impossible. Even for single column texts in general we do not reach a complete alignment of *all* tokens due to recognition errors of the OCR engine where tokens are merged or split. In the following discussion, alignment problems are ignored.

4 Step 2 of Phase 1: Computation of correction files

For each token w^{ocr} recognized by the given OCR engine and each dictionary $D \in \mathcal{D}$ we **preselect** a list of n **correction candidates** in D. Here n is an (adjustable) parameter. For preselection, the basic distance measure d_0 is used. Entries v of D where $d_0(w^{ocr}, v)$ is small are preferred. If $d(w^{ocr}, v) = d(w^{ocr}, v')$ for two distinct entries v, v' of D_i, then the more frequent entry is preferred. Preselection of correction candidates only represents a filtering step that narrows down the search for good correction suggestions in Step 3. The choice of the value n yields a compromise between recall and computational efficiency. From an algorithmic point of view, techniques described in [14] are used to preselect correction candidates, cf. Section 7.

Using the alignment file and the preselected correction candidates we produce for each dictionary $D \in \mathcal{D}$ a **correction file** CF_D. Let $\mathcal{M} = \{d_1, \ldots, d_m\}$ denote the set of distances. CF_D contains all tuples of the form $\langle w, w^{ocr}, \chi, v_1, \vec{d}[1], f(v_1), \ldots, v_n, \vec{d}[n], f(v_n) \rangle$ where w represents a correct token of the original sample, w^{ocr} represents the recognition result of the OCR engine, the v_l represent the preselected correction candidates for w^{ocr} in D, the boolean value χ indicates if w occurs in D, $f(v)$ describes the normalized frequency of v, and $\vec{d}[l]$ is shorthand for $d_1(w^{ocr}, v_l), \ldots, d_m(w^{ocr}, v_l)$ $(1 \leq l \leq n)$. The set of correction files for all dictionaries represents the basis for the interactive computations in Step 3.

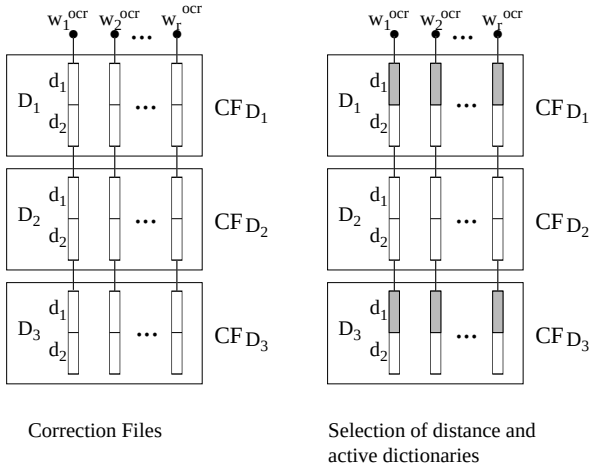

Figure 1. Selection of a configuration.

5 Step 3 of Phase 1: Interactive optimization of correction model and error analysis

Step 3 is split into rounds. Each round begins with the **selection of a configuration**. A configuration is given by a subcollection $\mathcal{D}' \subseteq \mathcal{D}$ of **active dictionaries** and an **active distance measure** $d \in \mathcal{M}$. In order to select a configuration, a subcollection $\mathcal{D}'$ and a distance measure $d \in \mathcal{M}$ may be activated using buttons of the graphical user interface. Figure 1 illustrates the selection of a configuration in a situation where three dictionaries D_1, D_2 and D_3 and two string distances d_1, d_2 are at our disposal. Measure d_1 is used and dictionaries D_1 and D_3 are activated.

For a given configuration $(\mathcal{D}', d)$, an interactive and semi-automated process is used to determine a unary correction model which leads to optimal correction accuracy. With a **unary correction model** we mean any deterministic procedure that computes for each recognized token w^{ocr} a unique **correction suggestion** w^{cor} which is w^{ocr} itself or an (other) entry of an active dictionary. **Correction accuracy** is defined as the percentage of tokens w^{ocr} where w^{cor} coincides with the correct token w. At this point it should perhaps be stressed that the selection of a *unique* correction suggestion primarily serves as a basis for comparing correction accuracy. Values for the correction accuracy reached for distinct configurations are compared in order to determine a configuration that leads to optimal accuracy. Any unary model obtained in this way can be extended to a k-ary model (s.b.).

In our system, unary correction models are completely determined by the actual configuration $(\mathcal{D}', d)$ and values for two parameters α, τ that are described below. In the sequel we assume that distance values $d(v, w^{ocr})$ are trans-

lated into similarity values $s(v, w^{ocr}) \in [0, 1]$ where high values mean high similarity.

Balance. The **score** of a correction candidate v for a token w^{ocr} is $sc(v) := \alpha s(v, w^{ocr}) + (1 - \alpha) f(v)$. **Balance parameter** α is a value in $[0, 1]$ that determines the relative weight of similarity versus frequency. Since frequency and distance values are of distinct nature and normalized in distinct ways, this gives only a basic intuition. For selecting a value for α the graphical user interface offers two options. The value can be defined interactively, using a scroll bar. Alternatively we may automatically compute a value α that leads to optimal correction accuracy (s.b.).

Threshold. Assume that a balance parameter α has been fixed. Consider a recognized token w^{ocr}. Using the correction files, among all correction candidates for w^{ocr} that have been preselected in *active* dictionaries we select the entry v^{cand} with the highest score. Collecting the entries v^{cand} for all tokens w^{ocr} we create a list *Cands* with tuples of the form $\langle w, w^{ocr}, v^{cand}, sc(v^{cand})\rangle$. *Cands* is sorted in descending score order, The sorted list is divided into two parts. The "active part" contains an initial interval where entries have high scores. For the tokens w^{ocr} of tuples in the active part we define $w^{cor} := v^{cand}$. As a matter of fact, very often w^{ocr} is found in an active dictionary and we get $w^{cor} := v^{cand} = w^{ocr}$. The remaining "passive part" contains tuples with low scores. Here we generally define $w^{cor} := w^{ocr}$. The **threshold parameter** τ defines the minimal score for elements of the sorted list *Cands* that belong to the active part. The value of the threshold parameter can be assigned interactively, using a scroll bar. Alternatively we may automatically determine a value τ that leads to optimal correction accuracy.

Optimization. Appropriate values for the parameters α and τ can be computed using a simple hill climbing procedure. For an initial pair of values (α_0, τ_0), which is selected arbitrarily, the system automatically computes the correction accuracy that is obtained. Fixing the value for τ_0 we use the system to compute a value α_1 that leads to optimal correction accuracy for the given threshold τ_0. Fixing then α_1 we compute a value τ_1 that leads to optimal correction accuracy for the given value of the balance parameter, α_1. In this way we continue until a local maximum is found. We have started to fully automatize the complete procedure.

Error analysis. Consider a situation where we have fixed a unary correction model in terms of a configuration $(\mathcal{D}', d)$ and suitable values for the parameters α, τ. Let $\hat{D}$ denote the union of all active dictionaries. For simplicity we assume that $w^{cor} = w^{ocr}$ as soon as $w^{ocr} \in \hat{D}$, which holds in all realistic cases. With a **correction error** we mean any triple (w, w^{ocr}, w^{cor}) where $w \neq w^{cor}$. In our system, correction errors are classified using seven disjoint categories:

1. "false friends": $w \neq w^{ocr} \in \hat{D}$ and $w^{cor} = w^{ocr}$.
2. The OCR-result is actively corrected, with wrong result: $w^{ocr} \notin \hat{D}$ and $w^{cor} \neq w^{ocr}$. We distinguish three subcategories:
   (a) "wrong candidate" errors where $w \in \hat{D}$ (here $w \neq w^{ocr}$),
   (b) "infelicitous correction" errors where $w \notin \hat{D}$ and $w = w^{ocr}$, and
   (c) "no chance I" errors where $w \notin \hat{D}$ and $w \neq w^{ocr}$.
3. The OCR-result is left unmodified, with wrong result: $w^{ocr} \notin \hat{D}$ and $w^{cor} = w^{ocr}$. We distinguish three subcategories:
   (a) "too cautious" errors where $v^{cand} = w$,
   (b) "wrong candidate and threshold" errors where $v^{cand} \neq w$ but $w \in \hat{D}$, and
   (c) "no chance II" errors where $v^{cand} \neq w$ and $w \notin \hat{D}$.

Our system automatically provides the user with an **error analysis** where for each of the seven categories the number (and percentage) of correction errors is depicted. These kind of statistics can be used to recognize deficits of the current correction model. For example, "false friend" errors can only be avoided with a smaller dictionary. In contrast, a large number of "no chance" errors indicates that the coverage obtained by the set of active dictionaries is yet insufficient. A large number of "wrong candidate" errors typically indicates that the selected distance measure is not optimal or the number n of preselected correction candidates per dictionary is too small. "Too cautious" errors can be avoided with a more liberal threshold. However, if the threshold is already optimal a larger value will lead to more "infelicitous corrections". For any combination of error categories, the list of all errors triples (w, w^{ocr}, w^{cor}) of the given error types can be displayed, separating triples from the active part from triples in the passive part, which may yield further insights about error sources and possible improvements.

Extension to k-ary model. Assume that eventually we have found a configuration $(\mathcal{D}', d)$ and values for the parameters α, τ that lead to optimal correction accuracy. As a final step that prepares the combination of the correction models obtained for the single OCR engines into a global correction model, the unary correction model for the current OCR engine is automatically extended. With a **k-ary correction model with confidence values** we mean a deterministic procedure that computes, given w^{ocr}, a ranked list of k correction suggestions with confidence values. In our approach, possible correction suggestions are w^{ocr} itself and the correction candidates that have been preselected in active dictionaries. Among the latter elements we select the k elements with the highest score. Let $C_0(w^{ocr})$ denote

the resulting set. For the final selection of correction suggestions, where w^{ocr} may replace an element of $C_0(w^{ocr})$, we use confidence values.

As a preparatory step we compute the *border score* $sc_0 := (sc_a + sc_b)/2$ where $sc_a = \tau$ (resp. sc_b) is the score of the last (first) tuple of the active (passive) part of the sorted list *Cands*. Note that the border score does not depend on a given token w^{ocr}. The **confidence** $conf(v_i)$ of a correction suggestion v_i in a list $C_0(w^{ocr})$ is defined as $sc(v_i)/sc_0$. Note that $conf(v^{cand}) > 1$ for all entries $\langle w, w^{ocr}, v^{cand}, sc(v^{cand})\rangle$ in the active part of *Cands*. The elements v^{cand} occurring in *Cands* represent the top-scored correction suggestions of the lists $C_0(w^{ocr})$. For the entries in the passive part we receive confidence values < 1. For all tokens w^{ocr} that do not occur in $C_0(w^{ocr})$ we define $conf(w^{ocr}) := 1$. Eventually we select for each token w^{ocr} as our set of correction suggestions the k elements in the set $C_0(w^{ocr}) \cup \{w^{ocr}\}$ with the best confidence values.

The rational behind this choice of a confidence value can be explained as follows. Since we use a limited amount of training data we can only approximate an optimal value for the threshold parameter τ. If a correction suggestion v has small confidence > 1, then already a minor change of the value for τ can lead to a situation where v is positioned in the passive part where we use w^{ocr} instead of v as correction suggestion in the unary model. In contrast, a large confidence value for v indicates that even with major changes of the value of τ we will still prefer v (as opposed to w^{ocr}) in the unary model.

6 Phase 2: Combining the correction models derived for single OCR engines

Phase 2 starts with the computation of the **global alignment file** *GAF*: the alignment files for the single OCR engines are used to compute all tuples of the form $\langle w, w_1^{ocr}, \ldots, w_{n_e}^{ocr}\rangle$ where w is a correct token of the original sample and the w_i^{ocr} represent the recognition results for this token obtained with the distinct OCR engines. Here n_e denotes the number of OCR engines.

Using the output of the k-ary correction models for the single OCR engines and *GAF*, a file with entries of the form $\langle w, L_1, \ldots, L_{n_e}\rangle$ is computed. Here L_i denotes the list of correction suggestions and confidence values obtained for recognized token w_i^{ocr} using the k-ary correction model for the i-th OCR engine ($1 \leq i \leq n_e$).

The final task is now to compute a corresponding list of the form $\langle w, L\rangle$ where L is a list of k correction suggestions that represent an optimal subcollection of the correction suggestions in the lists $L_1, \ldots, L_{n_e}$. Furthermore, each suggestion v in L should have a combined confidence value that takes into account all confidence values for v in the lists L_i. In the literature, combination problems of a similar abstract form have been intensively studied [4, 5, 18, 8, 12, 7, 1, 10]. Most work on combination problems in the area of OCR recognition is devoted to the combination of symbol classifiers. Here we combine results on the level of words. We are confident that the combination techniques developed in this area can be adapted to the present situation and will lead to good solutions. The extension of the actual tool to combination of OCR engines represents ongoing work.

7 Implementation and visualization techniques

Following the steps described above, our software currently consists of five main modules: (1) alignment, (2) preselection of correction candidates in dictionaries and computation of correction files CF_D, (3) computation of lists *Cands* with parameter optimization and error analysis, (4) graphical user interface (GUI), and (5) OCR combination. For preselection of correction candidates in dictionaries we use a technique based on so-called "Levenshtein-automata" introduced in [14].

For efficiency reasons, Levenshtein-automata are implemented in C. The calculation of correction suggestions only takes around 5 milliseconds per token even for dictionaries with more than one million entries. Because of the large number of calls the selection of correction candidates still represents a bottleneck in the software architecture. We actually try to accelerate this process using precalculation and caching techniques. Furthermore we look at variants of the techniques in [14] that further reduce access times.

All other (sub)modules of the system are developed in a strictly object oriented style in Java. Java offers optimal support for programming GUIs. Using the Java Native Interface (JNI) we were able to build a wrapper around the C-tools that describe Levenshtein-automata.

Given an OCRed document our modules have the following time requirements: With a conventional PC (1GHz Pentium processor with 256 MByte main memory) we need ca. 2 seconds per page for alignment, which means that a corpus with 8000 tokens takes ca. 40 seconds. Selection of correction candidates for the same corpus using nine dictionaries takes about 6 minutes. Manipulation of model parameters via the GUI can be done interactively. In order to avoid a long loading time at the beginning of each interactive session we separated alignment and computation of correction files CF_D from interactive parameter optimization and error analysis. Computation of correction files is organized as a batch job. Both correction files CF_D and lists *Cands* are basic data structures, hence they are formulated as aggregated Java classes. For passing CF_D-contents from Module 2 to Module 3, CF_D-files are represented as XML-files that are conform to a fixed DTD. The list *Cands* is com-

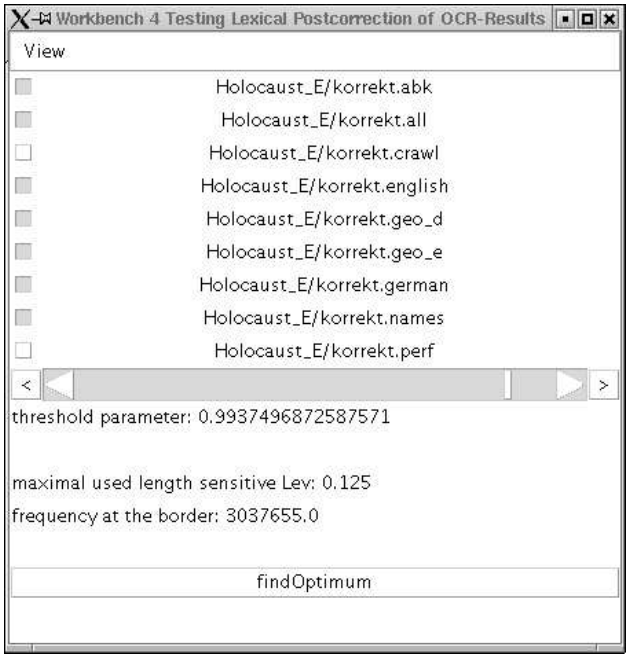

Figure 2. Selection of active dictionaries and threshold parameter.

Figure 3. Statistics for correction accuracy and error frequencies.

puted in real-time from the parsed XML files. The interactive part of the software follows the model-view-control design pattern. Currently the OCR combination module is in an early development stage.

The complete source code of the system consists of approx. 5000 lines of code.

A small collection of screen-shots should give some ideas of the current options for visualization and interactivity that are offered by the tool. Figure 2 shows the window that is used for selecting active dictionaries. The items "Holocaust_E/korrekt.abk" etc. stand for the correction files CF_D that have been computed for a document entitled "Holocaust_E" using nine distinct dictionaries D (abk,...,perf). Dictionaries, or the associated correction files, can be activated using the buttons on the left-hand side. The picture shows seven activated dictionaries. For the threshold parameter τ the value $0.9937\ldots$ has been selected. The maximal distance of a correction candidate in the unary model is indicated, as well as the frequency of the last entry of the active part. We may change the value using the scroll bar. We may also automatically compute the value for τ that leads to maximal accuracy.

In parallel we may open the window depicted in Figure 3 and see the correction accuracy and error statistics for the given selection of dictionaries and values (unary correction model). Correction accuracy is computed both with reference to all tokens and for the subset of normal tokens. A token is *normal* if it is composed of standard letters only. When modifying τ or the selection of dictionaries, new accuracy values and error values are computed and displayed in real-time.

Figure 4 shows the window that is used for fixing the balance parameter α. The same window is used to select a string distance. For the actual session, the length-sensitive Levenshtein distance has been selected (score component 0). Score component 1 is the normalized frequency value. If distinct frequency values from several ressources are available we may also select the type of frequency information that is used. We may modify the value of α using the scroll bar. When modifying α, new values for the error statistics are only computed by demand. We may use the system to compute an optimal value. In this case we automatically receive a new statistics for accuracy and error frequencies.

8 Evaluation results

In one experiment we used the tool for postcorrection of an English text T_{Bot} from the field of botany. The text contained 4478 tokens. As to layout, a single standard font without small letters was used. T_{Bot} was split into two disjoint parts, a training text T_{Bot}^{tr} of 896 tokens and an application text T_{Bot}^{appl} with 3582 tokens. Both parts were processed with a single OCR engine.

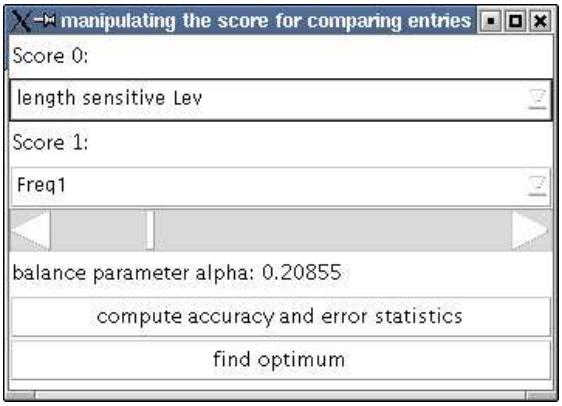

Figure 4. Selection of distance measure and balance parameter.

We used the tool in order to find an optimal selection of dictionaries and parameter values for T^{tr}_{Bot}. In this experiment, optimal results were obtained using one single "dynamic" subdictionary which was obtained via automated analysis of the vocabulary of web pages associated with the field of botany. Details of the general method for dynamic dictionary construction are described in [15]. For defining an optimal score, small values of α turned out to be useful. This effect is mainly due to the current normalization of frequency values that leads to small differences of normalized values. Still, it indicates that frequency information plays an important role. With the optimal threshold score, 86% of the list *Cands* belonged to the active part. ¿From these parameters we derived a unary correction model.

The model was then applied to T^{appl}_{Bot}. For T^{appl}_{Bot} we observed a correction accuracy of 96.89% for normal tokens using *plain* OCR results *without* postcorrection. Since correction accuracy measures the percentage of correctly recognized tokens (as opposed to symbols) this result is impressive and represents a serious challenge for any kind of automated lexical postcorrection with a unary model. In our case, after lexical postcorrection we obtained a correction accuracy of 97.43%.

We then repeated the experiment and excluded the dynamic dictionary from our set of ressources. Among the remaining dictionaries we had a dictionary with conventional English words with $315,300$ entries and a dictionary of proper names with $372,628$ entries. In this situation, regardless of the selection of active dictionaries and parameter values we could not reach a correction accuracy exceeding 96.89%. This illustrates the importance of the dictionary.

In a second experiment we treated another English text T_{RE} about the Roman Empire. T_{RE} contains 10000 tokens, layout is of the same form as above. T_{RE} was split into a training part with T^{tr}_{RE} with 2000 tokens and an application part T^{aapl}_{RE} with 8000 tokens. Both texts were scanned and analyzed with a single OCR engine. Our tool was used in order to find an optimal selection of dictionaries and parameter values for T^{tr}_{RE}. As before, optimal results were obtained using one single dynamic dictionary with vocabulary from web pages with contents associated with the history of the Roman empire. The optimal value for α was again small. With an optimal choice of the threshold score τ, 84% of the list *Cands* belonged to the active part. ¿From these parameters we derived a unary correction model.

We then applied the model to T^{appl}_{RE}. For T^{appl}_{RE} we observed a correction accuracy of 98.8% for normal tokens using plain OCR results without postcorrection. The value is close to optimal. In such a situation, most unary models for automated postcorrection will in fact lead to reduced accuracy. In this case, after lexical postcorrection we obtained a correction accuracy of 98.94%. We then looked at other values of α. For most values it turned out to be impossible to reach a correction accuracy beyond 98.8% on T^{appl}_{RE}. This result again illustrates the importance of frequency information.

The results, which are of course only preliminary, show that already in the actual form the tool helps to find good models for lexical postcorrection. We hope that better results can be obtained when using a variant of the Levenshtein distance where edit costs depend on the particular symbols that are used in the edit operations. Even more significant improvements can probably be obtained when combining results from several OCR engines. Both points represent work in progress.

As a matter of fact, fully automated postcorrection is always difficult in situations where OCR already yields brilliant results. In these cases suitable techniques for interactive correction are more relevant. We briefly comment on an experiment that illustrates the potential power of approaches based on multiple OCR engines. When using a single OCR engine, the problem of detecting false friends - a kernel problem for interactive OCR correction - is very difficult. Here the use of a second OCR engine turns out to be extremely helpful. Table 1 shows the number of false friends that have been obtained as the result of a lexical postcorrection of a text from the area of neurology with 5691 tokens. We used four dictionaries $D_1, \ldots, D_4$ of ascending size and coverage. The first line FF(1) counts the number of false friends for OCR engine 1 (the number of false friends for OCR engine 2 was slightly higher). The second line counts the number of tokens that led to false friends for both OCR engines. The last line counts the number of tokens that led to identical false friends for both OCR engines. The dramatic reduction shows that most false friends can be found simply by comparing the results of both OCR engines.

Dictionaries	D_1	D_2	D_3	D_4
FF(1)	22	27	28	32
FF(1) $\wedge$ FF(2)	2	4	6	7
FF(1) = FF(2)	1	2	3	4

Table 1. Single and double false friends using two OCR engines.

9 Concluding remarks

We described a tool that helps to optimize lexical postcorrection of OCR results, both for automated and interactive correction scenarios. In our description we concentrated on the steps and routines that are applied during the training phase. At the end of this phase, once an optimal configuration is selected and the correction model is fixed, all active dictionaries can be compiled into a single dictionary, which simplifies the actual computation of correction suggestions during the application. For each token w^{ocr} recognized by an OCR engine we may directly select k correction candidates with optimal scores, which means that the preselection step is not necessary. Generalizing ideas from [14] we are currently working on a fast algorithm that traverses a given dictionary and selects the k best correction suggestions subject to a given score.

Our experiments, which describe training and application phase, show that with the models derived for fully automated postcorrection in the training phase we can achieve an improvement w.r.t. correction accuracy even in cases where a single OCR engine is used and the plain OCR result is already excellent. Further progress can be expected from suitable techniques for combining postcorrection results for several OCR engines and from the use of improved distance measures. Additional work will be necessary to develop an extension of the current tool that provides optimal support for visual and interactive error correction.

References

[1] K. Chen, L. Wang, and H. Chi. Methods of combining multiple classifiers with different features and their applications to text-independent speaker identification. *Int. J. of Pattern Recognition and Artificial Intelligence*, 1997.

[2] A. Dengel, R. Hoch, F. Hönes, T. Jäger, M. Malburg, and A. Weigel. Techniques for improving OCR results. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[3] G. Ford, S. Hauser, D. X. Le, and G. R. Thoma. Pattern matching techniques for correcting low confidence OCR words in a known context. In *Proceedings of SPIE, Vol. 4307, Document Recognition and Retrieval VIII*, pages 241–249, 2001.

[4] T. Ho. A theory of multiple classifier systems and its application to visual word recognition. Technical Report Technical Report 92-12, State University of New York at Buffalo, 1992.

[5] T. Ho, J. Hull, and S. Srihari. On multiple classifier systems for pattern recognition. *IEEE Trans. Patt. Anal. Machine Intell. PAMI*, 16(1):66–75, 1994.

[6] R. Hoch and T. Kieninger. On virtual partitioning of large dictionaries for contextual post-processing to improve character recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(4):273–289, 1996.

[7] Y. Huang, K. Liu, and C. Suen. The combination of multiple classifiers by a neural network approach. *Int. J. of Pattern Recognition and Artificial Intelligence*, 9:579–597, 1995.

[8] Y. Huang and C. Suen. Combination of multiple classifiers with measurement values. In *Proc. of the Second International Conference on Document Analysis and Recognition (ICDAR 93)*, pages 598–601, 1993.

[9] K. Kukich. Techniques for automatically correcting words in texts. *ACM Computing Surveys*, pages 377–439, 1992.

[10] L. Lam, Y.-S. Huang, and C. Suen. Combination of multiple classifier decisions for optical character recognition. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[11] T. A. Lasko and S. E. Hauser. Approximate string matching algorithms for limited-vocabulary OCR output correction. In *Proceedings of SPIE, Vol. 4307, Document Recognition and Retrieval VIII*, pages 232–240, 2001.

[12] D. Lee and S. Srihari. A theory of classifier combination: The neural network approach. In *Proc. of the Third International Conference on Document Analysis and Recognition (ICDAR 95)*, pages 42–45, 1995.

[13] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 1966.

[14] K. U. Schulz and S. Mihov. Fast String Correction with Levenshtein-Automata. *International Journal of Document Analysis and Recognition*, 5(1):67–85, 2002.

[15] C. Strohmaier, C. Ringlstetter, K. U. Schulz, and S. Mihov. Lexical postcorrection of OCR-results: The web as a dynamic secondary dictionary? Technical report, University of Munich, 2003. submitted to ICDAR 03.

[16] K. Taghva and E. Stofsky. OCRSpell: an interactive spelling correction system for OCR errors in text. *International Journal of Document Analysis and Recognition*, 3:125–137, 2001.

[17] F. Weigel, S. Baumann, and J. Rohrschneider. Lexical postprocessing by heuristic search and automatic determination of the edit costs. In *Proc. of the Third International Conference on Document Analysis and Recognition (ICDAR 95)*, pages 857–860, 1995.

[18] L. Xu, A. Krzyzak, and C. Suen. Methods of combining multiple classifiers and their application to handwritten numeral recognition. *IEEE Trans. on Systems, Man, and Cybernetics*, 22:418–435, 1992.