HMM-Based Error Correction Mechanism for Five-Key Chording Keyboards

Adrian Tarniceriu, Bixio Rimoldi, Pierre Dillenbourg

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
adrian.tarniceriu@epfl.ch, bixio.rimoldi@epfl.ch, pierre.dillenbourg@epfl.ch

***Abstract*— As different text input devices lead to different typing error patterns, considering the device characteristics when designing an error correction mechanism can lead to significantly improved results. In this paper, we propose and evaluate a spelling correction algorithm based on Hidden Markov Models. It is designed for a five-key chording keyboard and uses the probabilities that one character is typed for another, named confusion probabilities. For the used evaluation text, the proposed algorithm reduces the error rate from 10.11% to 1.27%. In comparison, MsWord and iSpell reduce the error rate to 4.75% and 6.69%, respectively.**

I. Introduction

For most people, interacting with a mobile device requires visual commitment to the input mechanism. As a consequence, there are many situations in our daily life when we have to refrain from using these devices, as our vision is already committed: for instance, while texting and walking in a crowded place, one should focus on the environment and not on the phone.

A way to type fast, accurately, and with limited visual feedback is provided by chording keyboards. These devices generate a character by simultaneously pressing a combination of keys, and with five keys there are 31 combinations, enough for the letters of the English alphabet. If the keys are adequately placed, we can type with one hand and without looking at them, so we could use a mobile device even during activities for which vision is partially or entirely committed, such as walking in crowded spaces, jogging, or riding a bike.

The main drawback of such keyboards is that the users should learn the correspondence between keys and characters before being able to type. In previous studies [1], we showed that an adequately chosen mapping between key combinations and characters can be learned in less than 45 minutes and the average text entry rates after approximately 350 minutes of practice are 20 words per minute, with the maximum above 30 words per minute. Moreover, the character error rates are lower in the absence of feedback than when one can see what has been typed. Therefore, not seeing the typed text actually represents an advantage.

Considering the above-mentioned results, the proposed text input method is a viable option for situations when vision is already committed to other tasks. Automatic error correction will increase the keyboard's ease-of-use and typing speed as users will not have to stop typing in order to correct eventual mistakes.

In this paper, we continue our work from [2], [3] on error correction mechanisms for chording keyboards. If before we focused on individual words and used the maximum a posteriori rule to find the most likely candidates, now we will also consider the context. To achieve this, we will model the typing process as a Hidden Markov Model (HMM) [4].

The paper is organized as follows. Section II presents a brief overview of existing text error correction mechanisms. In Section III, we describe the proposed error correction algorithm. Section IV presents the evaluation dataset and the error correction results. Section V concludes the paper.

II. Related Work

A detailed overview of the commonly used correction techniques is presented by Kukich in [5]. Research in spelling error detection and correction is grouped in three main categories:

1) Non-word error detection: Groups of n letters (n-grams) are examined and looked up in a table of statistics. The strings that contain non-existing or highly infrequent n-grams are considered errors.
2) Isolated word error correction: Each word is treated individually and considered either correct or incorrect. In the latter case, a list of possible candidates is proposed. These candidates can be provided using techniques such as minimum edit distance, similarity key techniques, rule-based techniques, probabilistic techniques, etc.
3) Context dependent error correction: Errors can be detected by parsing the text and identifying incorrect part-of-speech, part-of-sentence, or word n-grams. Other approaches consider grammatical and inflectional rules, semantical context, and can also identify stylistic errors.

Most of the methods presented above can be applied to any typed text, regardless of the input device. As various input techniques become more popular, the classic correction techniques have been improved to also consider the device particularities. Goodman et al. [6] present an algorithm for soft keyboards that combines a language model and the probabilities that the user hits a key next to the desired key. Kristensson and Zhai [7] propose an error correction technique for stylus typing using geometric pattern matching. A strategy

that can be applied to chording text input is presented by Sandnes and Huang in [8].

Though widely used in speech processing, HMMs are not (yet) popular for text error correction. Conditional random fields (CRF), which can be seen as generalized HMMs [9], have been used for query correction and refinement [10], [11], but, to the best of our knowledge, these are the closest existing applications to spelling error correction. One drawback of CRFs is that they require training to estimate the model parameters. HMMs have a simpler structure, and the model parameters can be obtained from existing word tables.

III. Algorithm

To keep things relatively simple, we will consider that a typed word only depends on the previous one, being in the framework of Markov chains. If we know the probability of a word given its predecessor, the frequency of each word, and the probability to type word *x* when word *y* is intended, we have all the necessary ingredients to use Hidden Markov Models.

A. Hidden Markov Models

Firstly, we will provide a short description of HMMs, inspired from the paper written by Rabiner in 1989 [4].

A Markov chain can be described as a system that, at any time, is in one state S_i of a set of states $S_1, S_2, \ldots, S_N$. At every time instant, the system can switch to another state. If we denote the state at time t as q_t, then q_{t+1} will only depend on q_t and not on q_{t-1}, q_{t-2}, etc.

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \ldots) = P(q_{t+1} = S_j | q_t = S_i) = a_{ij} \quad (1)$$

The probability to go from state S_i to state S_j does not depend on the time t and is denoted a_{ij}. To obey standard stochastic constraints, $a_{ij} \geq 0$ and $\sum_{j=1}^{N} a_{ij} = 1$. The other parameters needed to completely describe the model are the initial state probabilities, $\pi_i = P(q_1 = S_i)$, for $1 \leq i \leq N$. A three-state Markov process is presented in Figure 1.

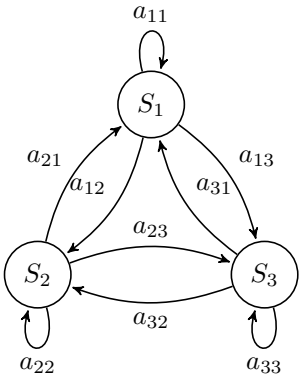

Fig. 1. Three state Markov model

Usually, for real-world processes, we do not have direct access to the states, but to some observations (considered independent of each-other) related to the state. For example, in the case of typing, we do not know what was the intended word, but we can see what has been typed. This scenario, when the states are hidden from the observer, is called a hidden Markov model. In addition to the state transition and initial state probabilities (a_{ij} and π_i), we also need the observation probabilities. If the set of possible observations is $O_1, O_2, \ldots, O_M$, we will denote by $b_j(O_k)$ the probability to observe O_k when the state is S_j:

$$b_j(O_k) = P(\text{observe } O_k \text{ at } t | q_t = S_j). \quad (2)$$

These probabilities are named emission probabilities.

B. Typing seen as a Hidden Markov Process

To correct errors using HMMs, we are interested in solving the following problem: given an observation sequence and the model parameters, estimate the optimal state sequence.

In the framework of typing, the hidden states represent the intended words and the observations are the typed words. The initial state probabilities π_i are given by the word frequencies and the state transitions a_{ij}, representing the probability of one word given its predecessor, are obtained from word tables (we used the British National Corpus [12] and the Google books corpus [13]).

The emission probabilities, b_{ij}, represent the probability of a typed word given the intended one, and depend on the confusion probabilities. These are the probabilities to type character *i* when character *j* was intended ($p(i|j)$), and were determined experimentally. For example, if *bat* is the intended word and *oat* is the typed word,

$$p(oat|bat) = p(o|b)p(a|a)p(t|t). \quad (3)$$

The above example holds for substitution errors (when one letter is switched to another letter), but other error types such as extra or deleted characters, merged or split words, etc. are treated similarly. We provide one more example for such a scenario:

$$p(t\ he|the) = p(t|t)p_{SpAdd}p(h|h)p(e|e), \quad (4)$$

where p_{SpAdd} is the probability to add a space inside a word.

The generic model is given in the top part of Figure 2 and the bottom part of the figure shows a concrete example. The number of states is the same as the number of observations. By observation, we mean any typed string of characters between two spaces, even if it is obtained by concatenating two words or by splitting one word.

Generally, each state represents a possible word, but there are two exceptions. To accommodate for concatenated words, the states can also be represented by bigrams, as for *is too* and *istoo* from the given example. Also, we introduce the NULL state to facilitate the splitting scenario, as in the case of *exercise* and *e xercise*. Now, there are two observations for the state *exercise*, and no observation related to the NULL state.

The general form of a hidden Markov model, as presented so far, cannot be used to represent the typing process because

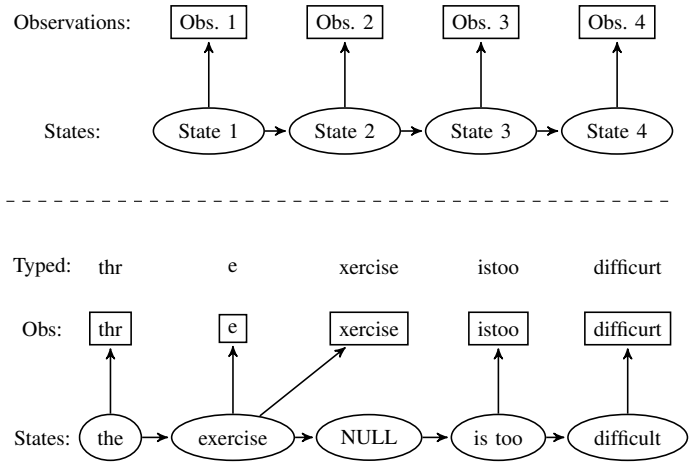

Fig. 2. Illustration of a HMM model used for typing. Top: general model; bottom: concrete example.

of practical reasons. For every observed word, any dictionary word can represent a possible state and the huge size of the state space would make any implementation unfeasible. However, there is a high probability to mistakenly type *shale* instead of *shape* and a very low probability to type *hippopotamus* instead of *shape*. In other words, if the observation is *shale*, the most probable states would be *shale*, *shape*, etc., but *hippopotamus* will have negligible probability. Considering this, we will limit the number of possible states per observation to a fixed value, N_s, and the considered states will be the words (or bigrams) with the highest posterior probabilities given the typed text. So, if y is the typed text, we will consider the candidates x that lead to the N_s highest values of $p(x|y)$, or, equivalently, to the highest values of $p(y|x)p(x)$. If the number of observations is N_o, then the total number of considered states will be $N_s \times N_o$.

It should also be considered that typing is a sequential process and from the states corresponding to observation O_i, we can only go to the states corresponding to the following observation, O_{i+1}. This leads to a left-right HMM, with the shape of a trellis.

In Figure 3, we present an example for the typed text *"thr e xercise istoo difficurt"* and all the possible paths of the trellis for $N_s = 3$. For each typed word, we will order the N_s candidate states from the most likely (top) to the least likely (bottom). Note that not all paths are allowed: from a state corresponding to a split word we can only go to a NULL state; NULL states, corresponding to the second part of a split word, can only be reached from the state containing that word, and not from other states. In the given example, this is the case for *three* being split to *thr e* and *exercise* being split to *e xercise*.

IV. Correction Results

After establishing the typing model and knowing the model parameters, we are ready to "correct" the typed text by determining the optimal state sequence. But before doing this, we should define what "optimal" means.

We will consider two optimality criteria. The first one chooses the states that are individually most likely and maximizes the expected number of correct individual states. The second criterion estimates the most likely state sequence, or trellis path. The algorithms used to implement these criteria are the Forward-Backward and the Viterbi algorithm, respectively, and are described in [4].

A. Evaluation Text

In order to gather enough data to evaluate the proposed algorithm, we asked 10 students from our university to type using a chording keyboard prototype.

The total amount of data gathered during the experiment consists of 40 640 words, out of which 4109 (10.11%) contain errors. Of these, 3120 (75.93%) are substitution errors. The remaining 989 errors occurred when people did not type a letter, typed an extra letter, the space between words was missing, when whole words were missing or added, etc.

The total number of typed characters is 220 910, from which 6428 are errors (2.91%). We used these characters to determine the confusion matrix, containing the probabilities of one letter being typed instead of another.

B. Results

The correction mechanisms were implemented in MATLAB and Python. The number of possible states for each observation, N_s, was set to 5. Experimentally, we noticed that increasing it above this value would only increase the number of required computations, without providing any improvements in the error correction.

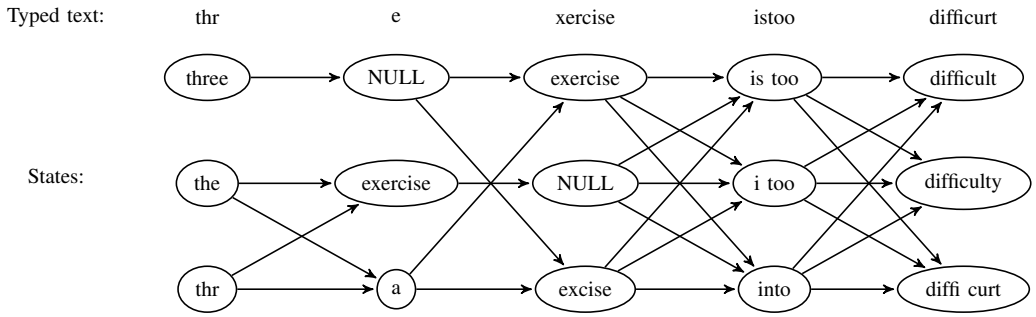

Fig. 3. Trellis example

To have a reference, we compared the results to the method that we previously proposed in [3], which only focuses on substitution errors for individual words and does not consider context information, to MsWord, and to iSpell. The results are presented in Figure 4.

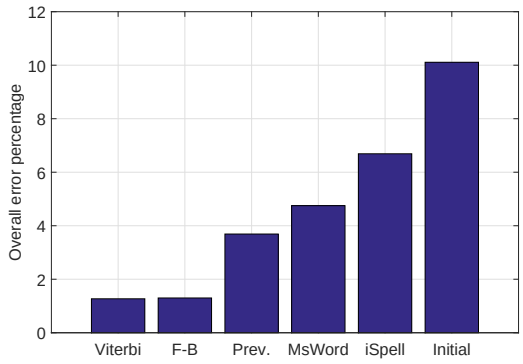

Fig. 4. Overall error rates for the HMM based algorihms (Viterbi and Forward-Backward implementations), previous work, MsWord, and iSpell

The Viterbi method decreases the error rate from 10.11% to 1.27% and the Forward-Backward method to 1.30%. Our previously proposed method provides a final error rate of 3.69%, while MsWord and iSpell lead to 4.75% and 6.69%, respectively.

The performances of the proposed algorithm are clearly better than when using MsWord or iSpell. It is not surprising that the context dependent error correction method is more efficient than analyzing only individual words, and the price for the lower error rate is increased complexity.

The Viterbi based algorithm is slightly better than the Forward-Backward algorithm, but the difference is not too big. The explanation is that the Forward-Backward algorithm estimates the most likely state for each observation, but the resulting state sequence may not be a valid succession of words in natural language (or a very unlikely word sequence).

V. Conclusion

In this paper, we presented a context-dependent error correction mechanism. The typing process was modeled using HMMs, where the typed words represent the observations and the possible candidate words represent the hidden states. We decided to use this approach because we know all the parameters of such a HMM: the initial state probabilities are actually the word frequencies, the state transition probabilities are given by the probability of a word given its predecessor, and the emission probabilities are the probabilities to type word x when word y was intended.

For a given typed text (observation sequence), we determined the optimal candidates (hidden states) using the Viterbi and the Forward-Backward algorithms. These methods reduce the overall error rate from 10.11% to 1.27% and 1.30%, respectively. This is significantly better than what is achieved by MsWord and iSpell (4.75% and 6.69%, respectively).

This advantage is due to the HMM algorithm, which takes into account word statistics for the English language and to the use of the device-dependent confusion probabilities (in [3], we showed that using more accurate probabilities leads to lower error rates). Even if the presented method was designed for a specific keyboard, it can be easily applied to other input devices by updating the confusion probabilities.

Naturally, we would like to further reduce the error rate, but doing this is rather difficult. Using a grammatical model might avoid errors such as *"a duck quack"* instead of *"a duck quacks"*, but, because the error rate is quite low, we feel that the benefits will not compensate the increased complexity. Moreover, there will always be some residual errors that cannot be corrected: - We cannot do anything about errors such as missing or extra words, or when most letters of a word were modified. - It is unavoidable that some correctly spelled words are changed by the algorithm. - The algorithm cannot do anything for discourse errors, as when the typed text is *"he likes the movie"* and the intended text is *"he liked the movie"*.

References

[1] A. Tarniceriu, P. Dillenbourg, and B. Rimoldi, "Single-handed eyes-free chord typing: A text-entry study," *International Journal On Advances in Intelligent Systems*, vol. 7, no. 1,2, pp. 145–155, 2014.

[2] A. Tarniceriu, B. Rimoldi, and P. Dillenbourg, "Error correction mechanism for five-key chording keyboards," in *The 7th International Conference on Speech Technology and Human-Computer Dialogue*, ser. SpeD '13, Cluj-Napoca, Romania, October 2013.

[3] A. Tarniceriu, B. Rimoldi, and P. Dillenbourg, "Fine-tunning a map error correction algorithm for five-key chording keyboards," in *ECUMICT 2014*, ser. Lecture Notes in Electrical Engineering, L. De Strycker, Ed. Springer International Publishing, 2014, vol. 302, pp. 153–165.

[4] L. R. Rabiner., "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 267–297, 1989.

[5] K. Kukich, "Techniques for automatically correcting words in text," *ACM Comput. Surv.*, vol. 24, pp. 377–439, December 1992.

[6] J. Goodman, G. Venolia, K. Steury, and C. Parker, "Language modeling for soft keyboards," in *Proceedings of the 7th International Conference on Intelligent User Interfaces*, ser. IUI '02. New York, NY, USA: ACM, 2002, pp. 194–195.

[7] P.-O. Kristensson and S. Zhai, "Relaxing stylus typing precision by geometric pattern matching," in *Proceedings of the 10th International Conference on Intelligent User Interfaces*, ser. IUI '05. New York, NY, USA: ACM, 2005, pp. 151–158.

[8] F. Sandnes and Y.-P. Huang, "Non-intrusive error-correction of text input chords: a language model approach," in *Fuzzy Information Processing Society, 2005. NAFIPS 2005. Annual Meeting of the North American*, june 2005, pp. 373 – 378.

[9] C. Sutton and A. Mccallum, *Introduction to Conditional Random Fields for Relational Learning*. MIT Press, 2006.

[10] J. Guo, G. Xu, H. Li, and X. Cheng, "A unified and discriminative model for query refinement." in *SIGIR*, S.-H. Myaeng, D. W. Oard, F. Sebastiani, T.-S. Chua, and M.-K. Leong, Eds. ACM, 2008, pp. 379–386.

[11] Y. Li, H. Duan, and C. Zhai, "A generalized hidden markov model with discriminative training for query spelling correction," in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '12. New York, NY, USA: ACM, 2012, pp. 611–620.

[12] URL: http://www.kilgarriff.co.uk/bnc-readme.html [accessed: 2015-03-10].

[13] URL: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html [accessed: 2015-03-10].