A spell checker with a user model for Swedish dyslexics

Marie Gustafsson
Department of Computer Science
Lund University
Sweden
marie@katastrof.nu

Abstract

Dyslexics pose a great challenge to spelling checking programs. They are among the ones who need the programs the most, they make diverse and complicated errors, and they may have trouble picking out the intended word from a long list of suggestions. The idea behind the program described in this article is to start with a spell checker based on a noisy channel model and allow for multiple transformations of deletion, insertion, substitution and reversal between the typo and the intended/suggested word. A user model consists of matrices of how often the user makes these transformations for the different letters of the alphabet. When the user chooses a suggestion from the spell checker, the typo/correction pair is used to further update these matrices.

1 Introduction

Dyslexia is a subject of much debate, both regarding definition and diagnosis. Because spelling is a large burden for dyslexics, most agree that a spell checker can be of great advantage. Alas, most spell checkers are designed for correcting typing errors, or typos, made by fairly able spellers. However, dyslexic spellers make more diverse errors, including compound errors consisting of a sequence of mistakes. Many dyslexics experience that spell checkers are not able to suggest words for their misspellings. Further, a small number of suggested corrections is important, since it may be difficult for the dyslexic to select from a long list (Spooner and Edwards, 1997).

A computer can have a user model for predicting how a user thinks and behaves. Ordinary spell checkers have minimal if any user model. It is thought that a user model can improve a spell checker by being more tuned to the kinds of mistakes that a certain user tends to make. For example, one user might tend to confuse b and d, while another might have problems with p and b. Knowing about these individual confusions can improve the suggestions given by the spell checker.

This article will look at the possibilities of applying a simple user model to a spelling correction program presented by Kernighan et al. (1990) , which is based on a noisy channel model. More specifically, the possibilities of applying this user model to Swedish dyslexics.

2 Dyslexia

The definition of dyslexia has been much discussed, as well as whether or or not dyslexia should be defined at all. Many definitions have focused on a discrepancy between the ability to read and write and the other intellectual abilities of a person. In 1994, the Orton Society decided on the following definition(Hoien and Lundberg, 1999):

> Dyslexia is one of several distinct learning disabilities. It is a specific, language-based disorder of constitutional origin characterized by difficulties in single word

decoding, usually reflecting insufficient phonological abilities.

The causes of dyslexia are disputed, but research has shown that phonological training in early schooling can be helpful. While many dyslexics eventually manage to read at a fairly high level, the troubles with spelling are more persistent.

3 User modeling

A user model defines the way a computer believes a person using it will behave. Dynamic user model generally refers to a set of stored numbers indicating how a particular person behaves on a number of scales. The field of user modeling has been around for about twenty years, starting with student modeling in the early eighties. The cognitive processes that underlie the user's actions and the user's behavioral patterns or preferences are some things that user models may wish to describe. Another is the difference between the user's skills and expert skills (Webb et al., 2001).

Among others, the following structures and processes are often included in a user modeling system (Kobsa, 2001):

- the representation of assumptions about user characteristics in models of individual users, such as assumptions about knowledge, misconceptions, goals and preferences;
- the representation of common characteristics of users, grouping them into subgroups, or stereotypes;
- the classification of users as belonging to one or more subgroups, along with the integration of typical characteristics of these subgroups into the current individual user model;
- the recording of users' behavior, especially their past interaction with the system;
- the formation of assumptions about the user based on the interaction history.

Observing the user's behavior can provide examples for training the user model, which can be used to make a model to predict future actions. There are however problems: the need for large data sets; the need for labeled data; concept drift; and computational complexity (Webb et al., 2001). Further, if user modeling profiles are to be created, a sufficient amount of time is required before they can be of any use. A solution to this might be the incorporation of stereotypes. Ideally, these should be used for initializing the user model, until there is more information about the individual user. Also, it should be regularly checked whether or not the right stereotype is activated (Virvou and Kabassi, 2002).

4 Approaches to spell checking

The traditional spell checker will go through a text and for each word check if it is in its dictionary. If it is, the spell checker moves on to the next word. If it is not, the spell checker tries inserting, removing, substituting and swapping (or reversal of) letters to see if it can find any words from the dictionary. These four changes represent major error types. A limitation of this approach is that only words in the dictionary are considered correct, which may lead to both false negatives and false positives. Another limitation is that no account is taken for the surrounding words. Further, many spell checkers only check for errors at one place in the word (or at least this used to be the case, e.g. (Kernighan et al., 1990)).

4.1 Levenshtein distance

Levenshtein distance (LD), also called edit distance, is a measure of similarity between two strings. The distance is the number of deletions, insertions, or substitutions required to transform the source string, *s*, into the target string, *t*. For example, if *s* is "thing" and *t* is "thing", then *LD(s,t) = 0*, because no transformations are needed. The strings are already identical. If *s* is "thing" and *t* is "think", then *LD(s,t) = 1*, because one substitution (change "g" to "k") is sufficient to transform *s* into *t*. A greater Levenshtein distance, means more different strings (Gilleland,).

The Levenshtein distance can be found by (Gilleland,):

1. Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.

2. Initialize the first row to 0..n. Initialize the first column to 0..m.

3. Examine each character of s (i from 1 to n).

4. Examine each character of t (j from 1 to m).

5. If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.

6. Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.

7. After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

4.2 Noisy Channel Model

The noisy channel model of Shannon (1948) has been applied successfully to many different problems, spell checking among them. The models has two components: a source model and a channel model. In applying this to the production of natural language text, it is assumed that a person choses a word *w* to output, but that the noisy channel induces the person to output string *s* instead.

Kernighan et al. (1990) describe how probability scores for candidate corrections can be found using a noisy channel model. Using a Bayesian argument, the intended correction, *c*, can often be recovered from the typo, *t*, by finding the correction *c* that maximizes *Pr(c) Pr(t|c)*. The first factor, *Pr(c)*, is a prior model of word probabilities. *Pr(t|c)* is a model of the noisy channel that accounts for spelling transformations on letter sequences, such as insertions, deletions, substitutions and reversals.

The first step of Kernighan et al. is proposing candidate corrections, which means finding words that differ from the typo t by a single insertion, deletion, substitution or reversal. These transformations are named from the point of view of the correction, not the typo. For example, for the typo *simpe*, could be the word *simple* transformed by a noisy channel by replacing the *l* with nothing at position 5.

When a list of candidates have been generated they are scored as described above. *Pr(c)* is estimated as *(freq(c) + 0.5)/N*, where *freq(c)* is the number of times that a word *c* appears in the training corpus and *N* is the number of words in that corpus. The conditional probabilities are computed from four confusion matrices: *del[x,y]*, the number of times that the characters *xy* (in the correct word) were written as *x* in the training set; *add[x,y]*, the number of times *x* was written as xy; *sub[x,y]*, the number of times that *y* was written as *x*; and *rev[x,y]*, the number of times that *xy* was written as *yx*. From these matrices probabilities are estimated by dividing by *chars[x,y]* or *chars[x]*, the number of times that *xy* and *x* appeared in the training set, respectively.

4.3 General issues for spell checkers

What size should the dictionary be? A larger one is of course preferred, but with many unusual words there is an increased risk that they will match a misspelled word. A larger dictionary takes longer to search, and on smaller devices storage might be an issue. Compiling a dictionary is not a trivial task, as possible text sources may contain errors and many proper nouns.

5 Implementation

This paper will describe the implementation of a spell checker that uses a noisy channel model, which allows for more than one deletion, insertion, substitution or reversal between the typo and the correction (Kernighan et al. only have one), and where the word chosen as the correction by the user updates the programs confusion matrices. The spell checker is written in java, and the interface is Mac OS X's cocoa.

5.1 What the user sees

The user is met with a text editor window and a spell checker window. When the "check spelling" button (in Swedish: *kolla stavning*) is pressed, the first word that the spell checker considers to be misspelled is selected (highlighted) and the user is provided with five suggestions along with the misspelled word, in the spell checker window (see Figure 1). To correct a word, the user selects a word from the list and presses the "correct" button (*rätta*). There are several other buttons in the spell checker window. Guess (*gissa*) makes new suggestions based on the word in the text box. This is use-

ful if the user wants to make changes to the original misspelling and get new suggestions, if the intended word was not in the list. For more on this, see the description of the CHECK system in the conclusion. Ignore (ignorera) adds the selected word to a temporary list of words which will not be regarded as misspelled. Add word (*lägg till ord*) will add the selected word to the program's dictionary. Find next (*sök nästa*) will skip the selected word and move on to select the next misspelled word.

5.2 Program structure

The SpellChecker class handles all real spell checking, while the SpellInterface class handles the interaction with the interface. The other main classes are Corrector, Scorer, and Trainer. LevDistance calculates the Levenshtein distance and gives operations associated with this distance. There are several classes for holding information about words and corrections: error/correction types (Correction), a word that is a possible correction (CorrectionWord), and a typo and its possible corrections (Typo).

When the "check spelling" button is pressed, for each word, SpellInterface asks SpellChecker whether or not it is misspelled. If SpellChecker finds that the word is not in the dictionary, it asks Corrector for a list of words which are possible corrections for the typo. Corrector first get a list of words which are within a certain range of the typo's length. It then uses LevDistance to calculate the edit distance between these words and the typo, and gets a list of operations necessary to transform the correctly spelled word into the typo. A CorrectionWord holds this word and the list of operations. SpellChecker then sends the list of CorrectionWords it receives from Corrector to Scorer asking for the n best ones. Scorer holds the confusion matrices and the character matrices described above in section 4.2. For each CorrectionWord, Scorer calculates a score, based on the Bayesian argument described in the same section. The probability of is taken to be the probability of the suggested word times the product of the conditional probabilities of the operations needed to transform the suggested word into the typo. The probability of the suggested word, *Pr(c)* was estimated as *(freq(c) + 0.5)/N* by Kernighan et al. (1990). Unfortunately, this prior model cannot yet be estimated for this program, as a corpus has not yet been gone through for this purpose. Scorer thus considers Pr(c) to be 1. This should be rectified. Scorer returns to SpellChecker a list of n CorrectionWords, sorted by score. If there are not five non-zero scores, words are added to the list according to edit distance. Finally, SpellChecker makes a Typo with the misspelled word, its list of all possible CorrectionWords and the list of suggestions that scorer provided. These suggestions are then displayed to the user.

The Trainer class is used to initially fill and update the confusion and character matrices. For the initial training, a list of common misspellings in Swedish has been used. The matrices continue to be updated as the user uses the spelling checker. Feedback is sent to Trainer every time the user selects a word as a correction.

6 Evaluation

Unfortunately, this spell checker has yet to be tested on texts by dyslexics. Hopefully this will be done in the near future. A corpus for use in calculating prior probabilities and for enlarging the dictionary should also be incorporated.

For the initial training of the confusion matrices more training material is needed, since the program is of no use if these are not representative of common errors in spelling in Swedish. However, filling the matrices is not easily done, since texts or lists with both typos and corrections are needed, or it has to be done by hand. Further, it is difficult to obtain samples of dyslexic writing, since dyslexic people tend to write less and be less inclined to share their writing. Getting a lot of text from one person is even more difficult. And since it cannot be said that all dyslexics make the same mistakes, a set of matrices suitable to one dyslexic person might not be suitable for another.

There are some problems associated with updating the confusion matrices based on the users' chosen correction. One is that the user might have chosen the wrong correction, meaning that the matrices are not updated correctly. However, if this does not occur very often it should have little impact on the scores calculated from the matrices, given that the matrices are well filled. A larger quantity of mistakes might actually lead to more efficient training.

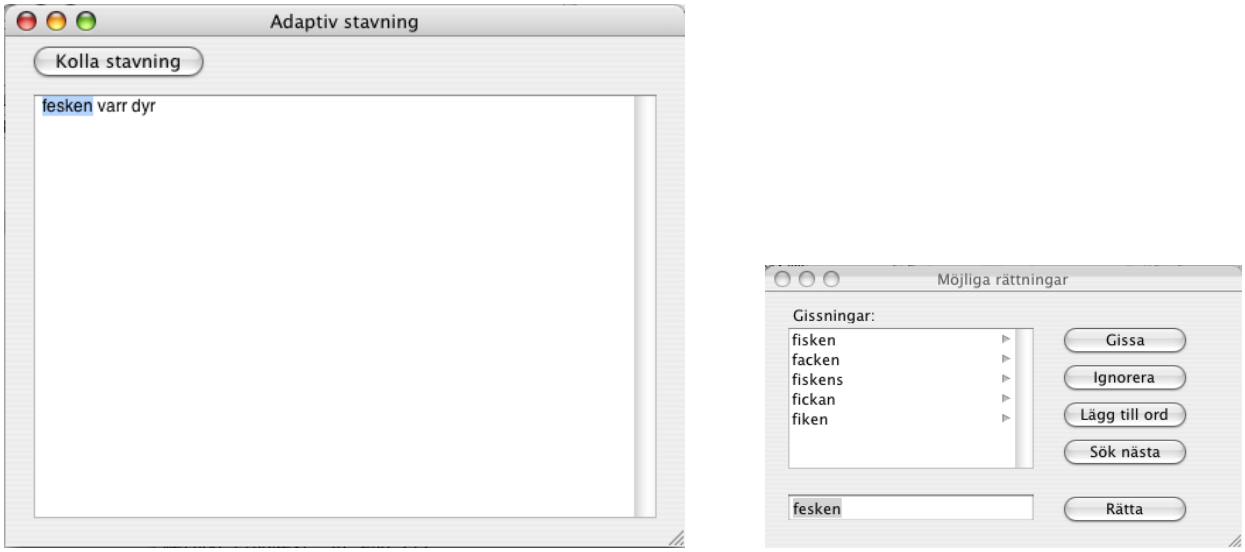

Figure 1: The spell checker in action

The approach used here assumes that there is some consistency to an individual's spelling patterns. More research should be looked at to decide whether or not this is actually the case. Also, a major short coming in this implementation is that it only to some extent addresses phonetic misspellings. That is, only phonemes represented by one letter can be dealt with.

The Scorer always returns a list of n possible corrections. It might be better if it only returned those who had scored above zero. However, given the limited confusion matrices in this version of the program, this may lead to no suggestions being made. In a future version, a more sophisticated selection based on score and variations in scores might be used.

The dictionary is stored in java's hashtable. More efficient methods of storage might be preferred. For example Stava (Kann et al., 1998) uses Bloom filters. Other necessary improvements to make the spell checker useful is to incorporate inflections so all don't have to be in the dictionary, and to have some way to check compound words. Another desirable extension is some kind of context sensitivity.

This work will not address some important aspects of spell checkers for Swedish, such as how to deal with compound words and the inflection of verbs. Another issue which will not be addressed, but which is a problem in most languages, is that a word might be misspelled for the intended meaning while being in the dictionary as it is a correct word. For example, ”witch house?” would be viewed as correct by a spell checker even though the intended sentence was probably ”which house?”. These mistakes are difficult to discover without a grammatical analysis or a more direct listing of words which sound alike

7 Conclusion

The application of a simple user model for a spelling correction program based on a noisy channel model seems promising. While the spell checker is by no means complete, this rudimentary structure does surprisingly well. Of course, this said before the program has been put to a real test. The idea of using the confusion matrices as a rudimentary user model is promising, and is not limited to use for dyslexic people, though they might need an adaptive spell checker more. The existence of research on whether or not dyslexic people make different spelling mistakes should be looked into.

Ashton describes the CHECK strategy, developed to help students use spell checkers more effectively and independently. This strategy makes use of the ”change to” box that many spell checkers have and that most let the user type in. It should be explained to the student that they can make changes in the word in this box and then press the ”guess” button

(or one with equivalent function) to generate a list of suggestions for the new word. If the new word is closer to the intended word, it might appear in the new list. This can be repeated as many times as is necessary, but it is best if only one type of change is made at a time. CHECK stands for

- Check the beginning sound of the word
- Hunt for the correct consonants
- Examine the vowels
- Changes in suggested word lists may give hints
- Keep repeating steps one through four

One possibility is to include instructions such as these in the program, if the user requires extra help.

Right now there are five words in the list of suggested words given to the user. If the intended word is not in that list, the user has to employ a method similar to the CHECK strategy. Ideally, the spell checker should find the target word in the first pass, but if it does not, a ”more suggestions” button, which adds five more suggestions to the list, might be useful.

The user model described by Spooner and Edwards (1997) is derived from a cognitive model of language production. More complex rules are used, which intend to describe permutations of errors typically made by dyslexic writers. The choices the user makes from the list of suggestions is used as feedback to measure and improve the user model's accuracy. Spooner (1996) claims that an attempt to extend the methods of checking for the four classic errors in multiple combinations (i.e. allowing for an edit distance larger than one) would both take longer and produce more suggestions. This is exactly, though perhaps not in the most efficient way, what has been done in this implementation, without any problems with speed.

Brill and Moore (2000) offer improvements on the work of Kernighan (1990), by using a more generic error model of string-to-string edits and so modeling substitutions of up to 5-letter sequences (e.g. ent being mistyped as ant, ph as f, etc.) within the framework of a noisy channel model. They find that this handles phonetic errors better than previous methods. It does however make residual errors, many which have to do with word pronounciation. To address this, Toutanova and Moore (2002) build two different error models using the Brill and Moore algorithm, one letter-based one and one based on a phone-sequence-to-phone-sequence error model. Since the better correction phonetic errors is especially interesting for a spell checker for dyslexics, their methods should certainly be looked into.

Apart from more support from research on the kinds of spelling mistakes that dyslexics make, getting a program that handles phonetic errors better is vital. Having more filled confusion matrices is also very desirable, so that one might have several sets that can serve as the beginning for different user groups, such as ”regular users”, dyslexics, or people who speak Swedish as a second language, where the spell checker could be adjusted after mother tongue.

Acknowledgements

Thanks to Pierre Nugues for all advice in developing this project. Also, I would like to thank Åsa Wengelin for getting me interested in the subject of spell checkers to begin with.

References

Tamarah Ashton. Making technology work in the inclusive classroom: A spell checking strategy for students with learning disabilities.

Eric Brill and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. *Proceedings of ACL-2000*.

Michael Gilleland. Levenshtein distance, in three flavors.

T. Hoien and I. Lundberg. 1999. *Dyslexi från teori till praktik*. Natur och Kultur.

Viggo Kann, Rickard Domeij, Joachim Hollman, and Mikael Tillenius. 1998. Implementation aspects and applications of a spelling correction algorithm.

Mark D. Kernighan, Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 205–210.

Alfred Kobsa. 2001. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11:49–63.

Claude Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423.

Roger I. W. Spooner and Alistair D. N. Edwards. 1997. User modelling for error recovery: A spelling checker for dyslexic users. *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 147–158.

Roger Spooner. 1996. Dphil thesis proposal a computerised writing aid for dyslexic people.

Kristina Toutanova and Robert C. Moore. 2002. Pronounciation modeling for improved spelling correction. *Proceedings of the 40th Meeting of the Association for Computational Linguistics(ACL-2002)*, pages 144–151.

Maria Virvou and Katerina Kabassi. 2002. Improving agent control for user modeling. *First International IEEE Symposium on Intelligent Systems*, pages 73–78.

Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. 2001. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):19–29.