Touchscreen keyboard correction

Matriculation no.: 100003610

Introduction

Nowadays, touchscreen devices are largely used and have evolved considerably in the last decade, but some issues still exist. One of the main concern is the lack of haptic feedback from a flat display, leading to a mistyping rate increase. Typing accuracy decreases also with the size of the screen and how far apart keys are layed out.

The aim of this practical is to build an N-grams model to correct letters that are mistyped. The built model is used against a given word and if a proposed correction exists, then this is proposed to the end-user. For example, if the prefix *exo* is entered, the model should propose *exp*, since in the English language there exist many more words starting with *exp* than with *exo*.

Achieved

This submission fulfills the following requirements:

- construction and use of a bigram (letter) model
- construction and use of an n-gram (letter) model
- generalisation of the mistyped neighbouring keys through the use of Confusion Matrices
- correction of letters in the word other than the last typed one
- accuracy measurement and testing

Assumptions

On building an n-gram model, given a certain Confusion Matrix, I assume that a key is always mistyped (substitution), rather than not entered (deletion) or mistakenly entered (insertion).

Two Confusion Matrices are used: a simple one, that takes into consideration only left-right neighbouring keys; and a 2-dimensional one built using a rule-of-thumb.

Design

In this section, the main components of this practical are discussed.

N-grams and Chaining

In this section I will very briefly explain how I used N-grams in the proposed solution, rather than explaining in details the theory of N-grams.

One of the main advantages of N-grams is that one needs to look only at the last N-characters. However, applying the chaining rule one could calculate the probability of the entire sequence of characters (a word). Applying the chaining rule just adds a constant factor, when comparing possible corrections of a word. However, using the chaining rule allows the model to be able to predict mistyped keys other than the last one typed.

The probability of a given sequence of words is multiplied by the probability that the modified key is mistyped (or the correct key is entered). This step is fundamental in order to introduce bias to the ideal model.

Probabilities are smoothed using LaPlace. The code, however, is written so that new smoothing functions can be easily added.

Confusion Matrices

In the solution here presented, I created two confusion matrices. The first confusion matrix, which I will call *Simple Confusion Matrix* (**SCM**) for simplicity, just considers the probability of typing a key to the right or to the left of the intended key (p_l and p_r). The SCM can be easily created on the fly, assuming arbitrary values for the mistyping probabilities are given. In the tests I have run, I assumed $p_l = p_r$, but the model can accept also cases when $p_l != p_r$. Allowing different probabilities may seem counterintuitive, but it could be useful if for instance someone who is left-hand shows tendency to mistype a key more to the right. In addition, probabilities p_{ll} and p_{rr} are used when keys have only one left or right neighbour. In my model, I assumed that $p_{ll} = p_{rr} > p_l = p_r$, on the basis that a keyboard has no keys other than latin letters and that users would not type outside of the keyboard. The probability of a key being typed correctly is $1 - p_l - p_r$ (or $1 - p_{ll}$ or $1 - p_{rr}$), as suggested by the specification.

Following the SCM, I researched confusion matrices in the 2D space xy. The only interesting result I found came from Kernighan (1990), who generated a substitution confusion matrix of correct keys vs typed keys.

This leads to

$$P(typed|correct)=\frac{subMatrix[typed\ ,correct]}{count[correct]}$$

However, count[correct] information was not provided and the AP Newswire 1988 corpus used by Kernighan is not available. A workaround would be to use a corpus containing similar content (i.e. Reuter 21578) and normalise the count by multiplying each count[$correct_i$] by the ratio between the size of the AP corpus and the surrogate corpus.

Nonetheless, while trying to incorporate the above solution in the model the following issues were raised:

- Kernighan collected errors frequencies from a corpus written using standard QWERTY keyboards (not touchscreen)
- Errors present in the AP Newswire 1988 corpus were minimal, since articles are usually checked multiple times before submission.

Given these two conclusions, I decided not to use the confusion matrices used by Kernighan, and rather created my own.

The second type of confusion matrix is a 2-Dimensional Confusion Matrix (**2DCM**), which I created by identifying common patterns in a standard QWERTY keyboard (see *Illustration 1*[1])

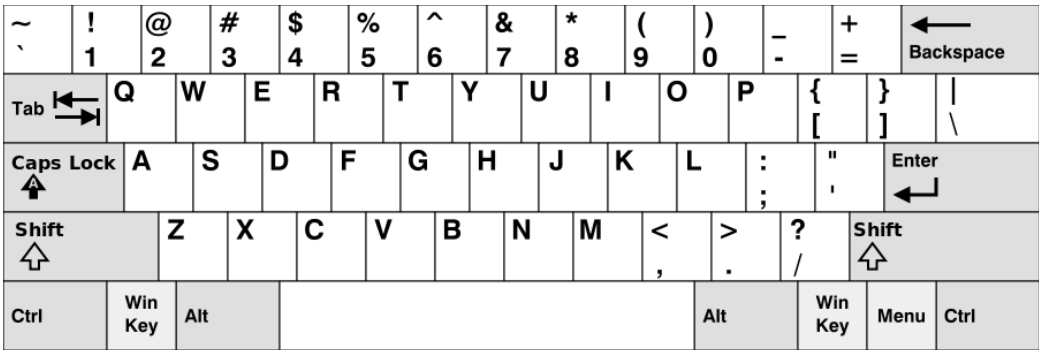

~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl

Illustration 1 Typical QWERTY keyboard

1 http://scilogs.com/

Patterns were identified by observing common keys relative positions (see Appendix A, *Table 1*). For each pattern, I assigned probabilities to the keys belonging to the pattern. The rule of thumb I used was to assign a high probability (> 0.77) to the main key of the pattern (*intended key*), probabilities 0.07-0.11 to keys to the left or to the right of the intended key and lower probabilities (< 0.04) to keys up or down to the intended key. Generally, probability of a key decreases as the position of such key is further away from the intended key. Symmetry was also used across the patterns, to create a more stable confusion matrix.
The reason I assigned higher probabilities to keys to the left or to the right of the intended key, compared to keys up or down to the intended one, is that users tend to move fingers more horizontally than vertically. I arrived to this conclusion simply by personal experience with touchscreen QWERTY keyboards.

Experiments

This section discusses the experimental part of this project. The main focus of the experiments was to observe how accuracy changes with N. I ran two main experiments: one where the model tries to correct only the last letter in a word and one where it tries to correct any letter in a word. For each experiment I used both a SCM and a 2DCM.

A mistyped letter, in a word, was introduced according to the confusion matrices probabilities. A uniform PDF was used to establish what letter in a word to modify.

I used the Brown corpus. k-fold CV, with k set to 10, was used to split the corpus in a training and a validation set. The experiments evaluated N from 1 to 6. Larger values of N were not considered, since most of the English words are at most 5-6 characters long.

Experiment 1

The results of the first experiment show that the optimal choice of N is three, when trying to correct only the last letter of a word. *Illustration 2* shows that models using a SCM and a 2DCM have analogous behaviour. However, I think it is interesting to notice that the accuracy is always higher when using the 2DCM. My hypothesis is that the use of the 2DCM introduces a beneficial bias toward the correction of additional keys, improving the overall performance of the model.

Experiment 2

In the second experiment accuracy was traced as N grows, when the model tries to correct any letter in a word. Interestingly, the optimal value of N is four in this experiment. My hypothesis is that since the model is trying to correct letters at any position in a word, N-grams with slightly large N are able to cover larger sequences of characters at once and therefore perform better. The fact that accuracy degrades at a lower rate after its optimal value, compared to Experiment 1, is a positive evidence toward my hypothesis.

A final observation is that accuracy in correcting any letter in a word is generally lower compared to correcting only the last letter, as expected, and the use of a 2DCM allows bigrams to almost as accurate as trigrams and 4-grams.

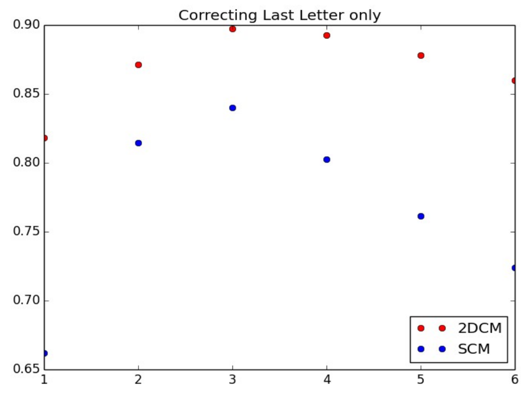


Illustration 2

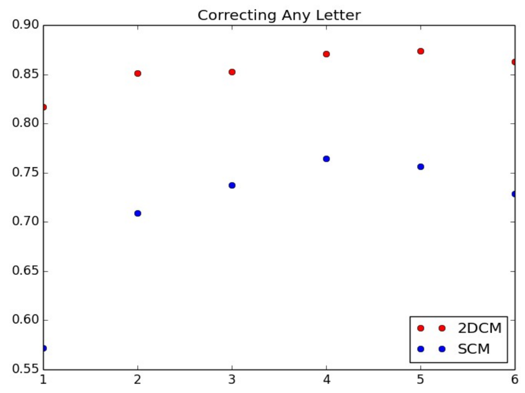


Illustration 3

Conclusion and Evaluation

In conclusion, I found that accuracy in the correction model does not always increases with N, given that N-grams are used.
The proposed solution could be improved by implementing a more advanced smoothing function, interpolation, or Katz Backoff. An evaluation could be pursued to find the best function/technique and prove that Laplace smoothing is too crude with characters sequences that occur too often in the training set.
Finally, I think it would be interesting to set up an experiment in order to generate appropriate confusion matrices and confirm the model accuracy.

Bibliography

Kernighan, M. D., Church, K. W., & Gale, W. A. (1990, August). A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*(pp. 205-210). Association for Computational Linguistics.

http://scilogs.com/, QWERTY Keyboard, available at: http://www.scilogs.com/scientific_and_medical_libraries/files/QWERTY-keyboard-US-version.png, last retrieved on the 12th March 2014

Appendix A

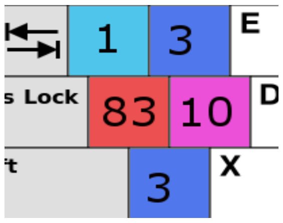
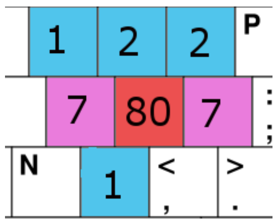
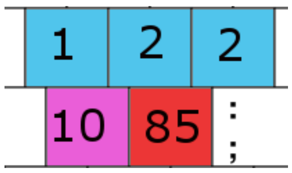
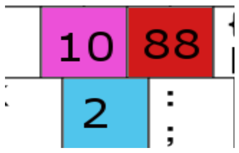
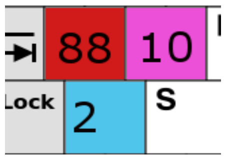
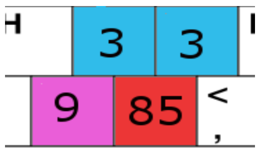
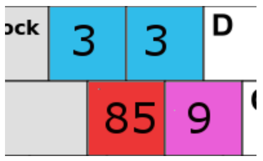
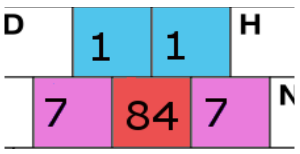
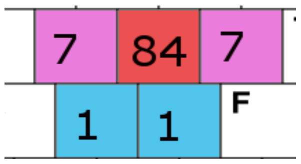
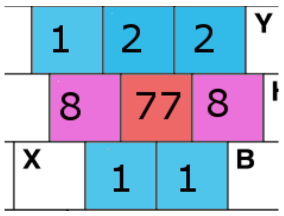
Letters	Pattern	Letters	Pattern
A			
K			
L			
P		Q	
M		Z	
B, C, N, V, X		E, I, O, R, T, U, W, Y	
D, F, G, H, J, S			

Table 1 Patterns in the same row are equivalent by symmetry. White keys do not belong to the pattern.