Correction of keyboard typos with an Hidden Markov Model

Ilaria Pigazzini, Cezar Angelo Sas, Andrea Vidali

July 3, 2017

Contents

1 Introduction

HMMispelling is a demo application to correct input keyboard typos. During the development of our project we made some studies on the problem: we defined an Hidden Markov Model, ran the Viterbi algorithm on it and analyzed the results by comparing different configurations and inputs.

2 The model

We modeled the problem as an hidden markov model, where the hidden states are the characters of the text which should have been typed; the emissions are the actual typed characters.

The chosen task to infer the correct typed text is Most Likely Sequence. The Hidden Markov Model library(see Section 3, Hidden-Markov Model) used in our project implements the Viterbi algorithm.

2.1 Model Parameters

Following, the parameters given to the model:

States: alphabetical characters of the $QWERTY$ keyboard

Emissions: alphabetical characters of the $QWERTY$ keyboard

Prior Probability Matrix: relative frequencies of letters in the Englis language.

Transition Matrix: bigram frequencies of the English language. See Section 2.3 for further information.

Emission Matrix: the probability of the digit to be correct or uncorrect. The uncorrect digits for every QWERTY alphabetical character are its neighbors with distance one. See Section 2.4 for further information.

Since the correction of mispelling will be applied on the alphabetical characters(lowercase and uppercase, plus whitespace) of the $QWERTY$ keyboard, in the following sections we will refer to keyboard input as “digit”.

2.2 Prior probability matrix

To obtain this matrix we computed for every digit the frequency of the bigram [whitespace, digit]. This matrix and the transition matrix were trained on the same datasets (see Section 2.5).

2.3 Transition matrix

The transition matrix describes the probability of a digit to be followed by another digit in the input text. The values of this matrix are bigram frequencies of the English language, where a bigram is a sequence of two digits. We trained this matrix on three datasets: *Swift*, *Twitter* and the sum of the two, named *Hybrid*(see Section 2.5).

2.4 Emission matrix

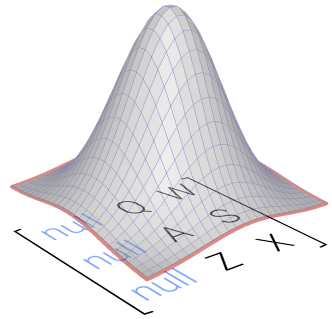
Consider a digit of the keyboard. In our model we assume that its neighbors are all digit placed at distance "1 digit" in every direction on the keyboard. For instance, digit "*S*" is at distance 1 from digit "*A*" but it is at distance "2" from digit "*F*".

Given the intention to press a digit on the keyboard, the emission matrix describes the probability of any digit to be pressed instead of the intended digit ("fat finger typo"). For each digit, this matrix contains relevant values when the probability refers to the neighbor digits. The other probabilities are set to constant $\epsilon = 10^{-5}$.

We modeled the neighbors of every digit as a 3x3 neighbor matrix where the intended digit is placed in position $[2, 2]$ (See Figure 1a). If a digit is on the edge of the keyboard, the position contains a *null* value. Given a neighbor matrix, we compute a bidimensional distribution centered on the intended digit position. We tested three different kinds of distribution: uniform, gaussian and custom. For what concerns the uniform distribution, we assumed that every neighbor digit and the intended one have the same probability to be pressed. This simplified configuration led to poor results, since there is no emphasis on the fact that the intended digit will be pressed correctly most of the time. This problem was solved by introducing the gaussian distribution with *mean* equals to 0 (Figure 1b). We explored different settings for the *variance* in order to find the best configuration (see Section 4). Even if the gaussian distribution seems to offer good values for the emission matrix, it should be used to model continous variables.

Finally, we tested our custom distribution where the probability of pressing the intended digit is definitely higher than the probability of pressing its neighbors, which is uniform. This distribution gave the best results.

(a) Neighbor matrix



(b) Gaussian distribution

Figure 1: Emission distribution example

2.5 Training

Table 1 reports the datasets used to realize our project: the data were used to train the model parameters. In this report we refer to them as *Twitter* (collection of tweets about apple and trump), *Swift* (collection of news, tweets and blogs text) and *Hybrid* (Tweet + Swift).

All datasets were previously cleaned from all characters different from the alphabetical ones(lowercase and uppercase), plus whitespace. Moreover, urls and mentions were removed from each tweet in order to avoid noisy samples.

Name	**Source**	**#characters**
news	Swift Key	204233394
twitter	Swift Key	164456394
blogs	Swift Key	207723793
apple	Twitter	2376252
trump	Twitter	11849767
Swift	news, twitter, blogs	576413581
Twitter	apple, trump	14226019
Hybrid	Swift, Twitter	590639600

Table 1: Dataset information

2.6 Dictionary

In order to support and enhance our algorithm, we added a function which verifies word existence on a dictionary. After the correction of every word, this function checks whether the corrected word or the original one is included in the dictionary and returns the matching one, giving priority to the corrected word in case both are equally present/absent in the dictionary. The dictionary was taken from NLTK (see `www.nltk.org`).

3 Tools and libraries

Library	Source	Description
Hidden-Markov Model	`github.com/Red-devilz/hidden_markov`	Python implementation of the hidden markov model
Autowrong	`github.com/pwrstudio/autowrong`	Introduces keyboard typos into a string
Tweepy	`www.tweepy.org`	Python library for accessing the Twitter API.
Django	`www.djangoproject.com`	High-level Python Web framework.

Table 2: Libraries

4 Test and results

In the following section we report the results obtained from the test conducted on our application. First we introduce our test dataset, then we describe the semantic of our output data, we list our performance metrics and finally we discuss the results. Notice that for what concerns the testing we divided the output data depending on whether the correction was launched having the whole tweet as input or single words in tweets as input. On the contrary, when evaluating the output data we only checked if every word of the output text was equals to every word in the ground truth, types of correction(on whole tweets/on single words).

4.1 Data Analysis

We partitioned the output data by assigning to each word in the analyzed text three boolean attributes:

1) Perturbed, which indicates whether the word was perturbed by Autowrong or not; *2) Corrected*, set to 1 if the model tried to correct it; *3) True*, set to 1 if the output word matches the ground truth. In Figure 2 we reported the representation of how we divided the data. the circle contains all word which were corrected($Corrected = 1$) by our model. It is divided in:

- **corrected-right**: perturbed word which were corrected and match the ground truth;
- **corrected-wrong**: word which were corrected badly whether they were perturbed or not;
- **not corrected-right**: word not corrected and match the ground truth;
- **not corrected-wrong**: missed correction of word.

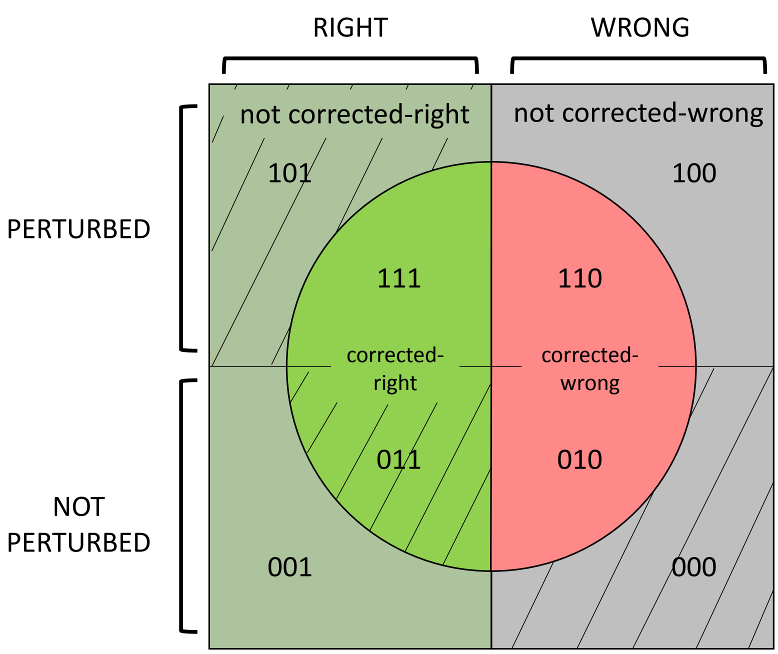


Figure 2: Data representation

In Table 3, all the possible combination of attributes which define each data set.

Observation				
Perturbed	**Corrected**	**True**	**Description**	**Evaluation**
0	0	0	-	impossible
0	0	1	not altered observation	correct
0	1	0	unnecessary correction	wrong
1	0	0	missed correction	wrong
1	0	1	-	impossible
1	1	0	wrong correction	wrong
0	1	1	-	impossible
1	1	1	right correction	correct

Table 3: Data attributes

4.2 Performance measure

The combinations of attributes in Table 3 were used to generate the confusion matrix in Table 4. From this matrix we define:

$$Precision = \frac{111}{111 + 100} \tag{1}$$

$$Recall = \frac{111}{111 + 010 + 110} \tag{2}$$

$$Accuracy = \frac{111 + 001}{111 + 100 + 010 + 110 + 001} \tag{3}$$

$$F1 - measure = 2 * \frac{precision * recall}{precision + recall} \tag{4}$$

4.3 Performance evaluation

The performances were valued on tweets from dataset *Apple* composed by 5700 tweets containing 69 000 words talking about Apple Inc.. *Apple* dataset was previously corrupted with a 5% error probability thanks to Autowrong, which leads to 25% corrupted words. The errors simulate the wrong typing of a digit by replacing the intended digit with one of its neighbors.

We ran the Viterbi algorithm with different input values and measured the performance variation. In particular, we experimented different configurations of Emission

		Predicted condition	
		T	F
True Condition	T	111(TP)	100(FN)
	F	010, 110(FP)	001(TN)

Table 4: Confusion matrix

Matrix(EM) and Transition Matrix(TM). All experiments were conducted twice by running the algorithm on the text of the whole tweets first and on every single word of the tweets then. In the following section we will show the results in both cases.

Our goal was to establish the best parameter configuration for the analysis and the model. The considered parameters are:

- *Analysis*: defined on {tweets, words}, indicates whether the analysis was conducted by running the algorithm on the text of the whole tweets or on every single word of the tweets;
- *Transition Matrix training(TM tr)*: defined on {*Twitter*, *Swift*, *Hybrid*}, indicates the dataset on which the transition matrix was trained;
- *Emission Matrix distribution(EM distr)*: defined on {uniform, gaussian, custom}, indicates the distribution of the values in the emission matrix;
- *EM parameter*: defined on [0.05, 0.95] with a step of 0.05, in case of custom EM distr = custom indicates the probability of pressing the intended digit; in case of gaussian EM distr = gaussian indicates the variance value.

The possible parameters combination are $2 * 3 * 2 * 19 + 2 * 3 = 234$. In order to find the best configuration, we first looked for the combination of "Analysis", "TM tr" and "EM distr" which gives the highest value of *Accuracy*. For every parameter p, we fixed the remaining two and for every combination we verified which value of p maximizes the *Accuracy*. In Table 5, 6 and 7, the best parameter value is called "Winner".

From these Tables we assessed that the best combination for the three parameter is $\{Analysis =$ tweets, $TMtr =$ Hybrid, $EMdistr =$ custom$\}$.

For what concerns the gaussian/custom distribution of emissions, we tested them with different *EM parameters* in order to understand how the performances change. In Figure 3 we resume the results obtained with different variance settings word analysis

Combination		
Analysis	**TM**	**EM Winner**
tweet	Swift	Custom
tweet	Twitter	Custom
tweet	Hybrid	Custom
words	Swift	Custom
words	Twitter	Custom
words	Hybrid	Custom

Table 5: EM best parameter versus Analysis and TM

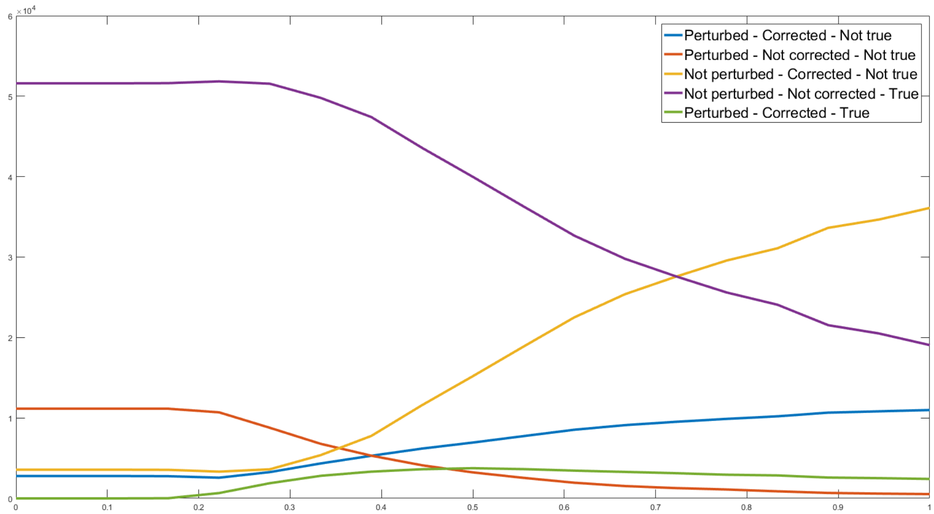
Combination		
Analysis	**EM**	**TM Winner**
tweets	Custom	Hybrid
words	Custom	Hybrid
tweets	Gaussian	Hybrid
words	Gaussian	Hybrid
tweets	Uniform	SwiftKey
words	Uniform	SwiftKey

Table 6: TM best parameter versus Analysis and EM

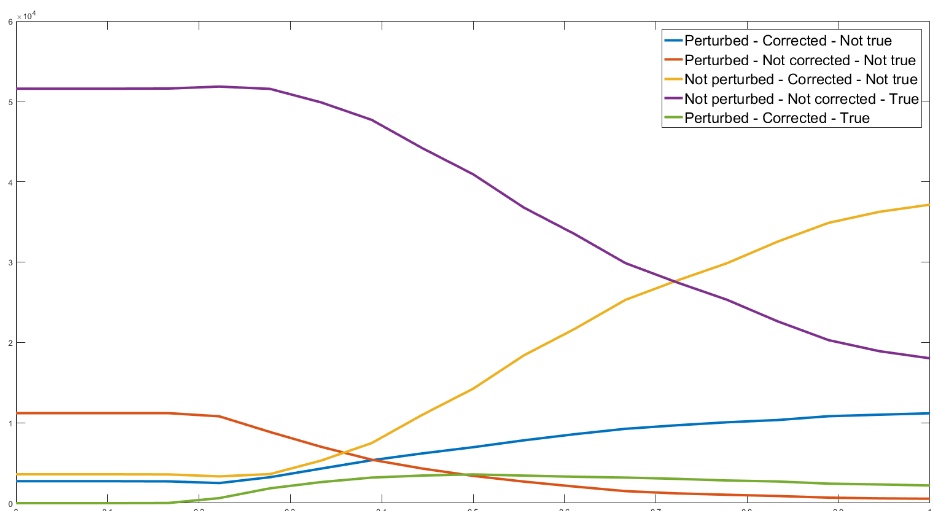
Combination		
EM	**TM**	**Analysis Winner**
Swift	Custom	tweets
Swift	Gaussian	tweets
Swift	Uniform	tweets
Hybrid	Custom	tweets
Hybrid	Gaussian	tweets
Hybrid	Uniform	tweets
Twitter	Custom	tweets
Twitter	Gaussian	tweets
Twitter	Uniform	tweets

Table 7: Analysis best parameter versus EM and TM

and tweet analysis. The performance valued was the number of words in each output data set(see Section 4.1).



(a) Analysis: tweets - TM training set: Hybrid



(b) Analysis: words - TM training set: Hybrid

Figure 3: Gaussian distribution - variance influence on output data

Figure 4 shows how the probability of pressing the intended digit affects the performance word analysis and tweet analysis. The performance valued was the number of words in each output data set(see Section 4.1).

After our experiments we established that the best value for the parameter is $EMparameter = 0.35$, in case of $EM\ distr =$ gaussian; $EMparameter = 0.95$, in case of $EM\ distr =$ custom.

In Table 4.3, the best results obtained with the best parameters configuration.

Analysis	**TM tr**	**EM distr**	**EM p**	**Prec**	**Rec**	**F1**	**Acc**
tweets	Hybrid	custom	0.95	0.2942	0.3547	0.3216	0.7784

Table 8: Best results on dataset apple

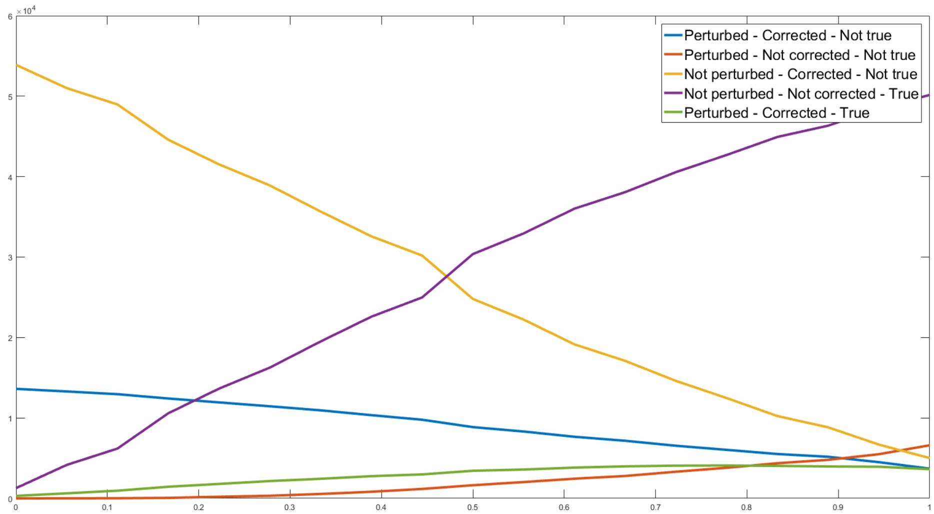
The test presented was conducted without the support of our dictionary(see Section 2.6). After its introduction, we observed a linear increase of the performance metrics. Table 9 shows a sample of the final results that we obtained from testing dataset *Apple*.

Rank	**Analysis**	**Dict**	**TM tr**	**EM distr**	**EM p**	**Prec**	**Rec**	**F1**	**Acc**
1	tweets	NLTK	Twitter	custom	0.95	0.3364	0.3688	0.3519	0.8006
2	tweets	NLTK	Swift	custom	0.95	0.3303	0.3432	0.3366	0.7991
5	tweets	NLTK	Twitter	gaussian	0.35	0.2761	0.3107	0.2924	0.7900
19	tweets	-	Hybrid	custom	0.95	0.2942	0.3547	0.3216	0.7784
23	words	-	Hybrid	custom	0.95	0.2799	0.3272	0.3017	0.7754
24	words	NLTK	Twitter	gaussian	0.35	0.2480	0.2981	0.2708	0.7754
30	words	NLTK	Hybrid	gaussian	0.3	0.2110	0.1684	0.1873	0.7736
31	tweets	-	Hybrid	gaussian	0.3	0.2152	0.1769	0.1942	0.7734
339	tweets	NLTK	Swift	uniform	-	0.0326	0.6054	0.0619	0.4814
632	tweets	-	Swift	uniform	-	0.0227	0.9559	0.0444	0.1625

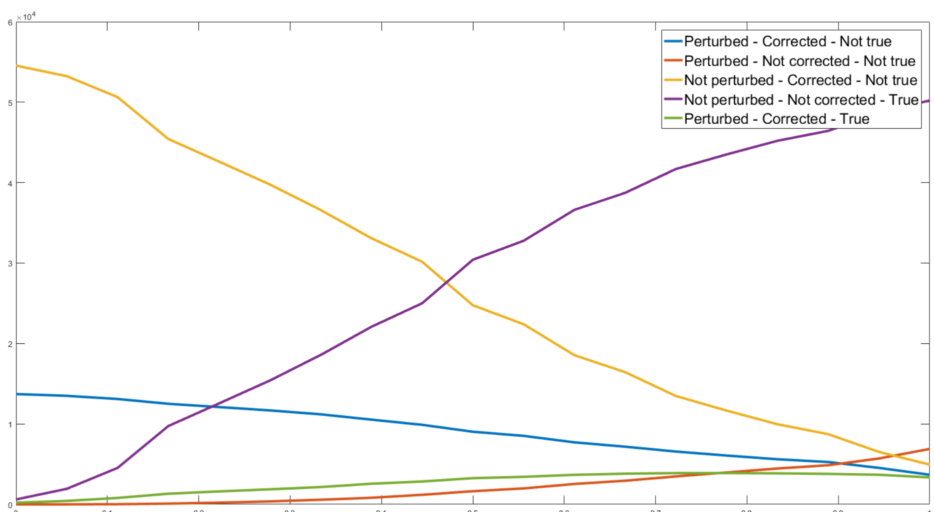
Table 9: Final results on Apple dataset

4.4 Word prediction analysis

In this section we show some examples of how our model corrects or nor frequent and infrequent words.



(a) Analysis: tweets - TM training set: Hybrid



(b) Analysis: words - TM training set: Hybrid

Figure 4: Custom distribution - intended digit probability influence on output data

The most frequent word is “Apple”. This word is perturbed in different ways: some of them are corrected by our model, some remain corrupted. For instance, “Apole”, “Appoe” and “Alple” are all samples of uncorrected perturbation. In this case, even if the probability of ’pl’ Is higher that ’po’ and ’pp’, the sequence probability is higher for ’Appp’ and ’Apol’ than ’Appl’, hence our model doesn’t correct the words. On the other hand, the model succeds in the correction when the word is perturbed as “Aople”, “Apple” or “Wpple”: in these cases the error is not exploting the model weaknesses.

Another example of common wrongly corrected word is “you”, perturbed as “yoh” or “yoj”. Our model corrects it as “yon” because the emission probability of the ’n’ and ’u’ are equal, but the transition probability is higher for bigram ’o-n’ than ’o-u’.

The last example regards an uncommon word which is not corrected when perturbed. “Liveroooo”, perturbation of “Liverpool”, is not corrected because the bigram ’oo’ is a common one, and our system is unable to handle errors of triple or more letters repeated.

5 HMMispelling demo

As result of our project, we introduce our demo “HMMispelling”. The demo offers a Graphic User Interface and two modes: an interactive mode where the user can type his message and check the real time correction and an automatic one which shows the real time correction of a stream of real tweets.

The pipeline of our application is divided in ..steps:

- Takes the input text
- Infers the correction of the eventual typos from the model
- returns the corrected text

6 Conclusions

In our project we developed an application able to correct keyboard typos thanks to the use of an Hidden Markov Model. We trained the model parameters and tested the possible parameters combinations in order to find the best configuration for the model. Moreover, we added a dictionary to support the correction task. Finally we analyzed the results. We don’t have high performances due to our model weakness: the fact that it is based on bigram frequencies and not on the whole word probability often leads the Viterbi algorithm to accept perturbed words; if all the bigrams in a word have high probability to occur, the model can't establish if it is a perturbed or a corrected word. Future enhancements could be to use a Markov Chain and replace the bigram frequencies with longer engrams.