Correction of keyboard typos with an Hidden Markov Model

Ilaria Pigazzini, Cezar Angelo Sas, Andrea Vidali

June 27, 2017

Contents

1 Introduction **3**

2 The model **3**
2.1 Inference task: Viterbi 3

3 Model Parameters **3**
3.1 Transition matrix 3
3.2 Emission matrix 3

4 Tools and libraries **4**

5 Data Analysis **4**

6 Test **7**

7 Demo and GUI **7**

8 Prediction Task **7**

9 Conclusions **7**

1 Introduction

2 The model

2.1 Inference task: Viterbi

The chosen task to infer the model is Most Likely Sequence. The Hidden Markov Model library(ref) used in our project implements the viterbi algorithm.

3 Model Parameters

Following, the parameters given to the model:

States: alphabetical characters of the $QWERTY$ keyboard

Observables: alphabetical characters of the $QWERTY$ keyboard

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

Transition Table: bigram frequencies of the English language. See Section 3.1 for further information.

Emission Table: 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 3.2 for further information.

3.1 Transition matrix

3.2 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 of press a digit on the keyboard and all the digits, the emission matrix describes the probability of a digit to be press given the will of press the intended digit ("fat finger typo"). For each digit, this matrix contains relevant values when the probability refers to the neighbor digits of the intended digit. 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 1). 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 press. This simplified configuration led to poor results, since there is no emphasis on the fact that the intended digit will be press correctly most of the time. This

Figure 1: Neighbor matrix

4 Tools and libraries

5 Data Analysis

We partitioned the output data by assigning each data three boolean attributes:
1) Perturbed, which indicates whether the data 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 data matches the ground truth. In Figure 2 we reported the representation of how we divided the data. the circle contains all data which were corrected(*Corrected* = 1) by our model. It is divided in:

- **corrected-right**: perturbed data which were corrected and match the ground truth;

Table 1: 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.

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

In Table 2, all the possible combination of attributes which define each data set.

Table 2: Data attributes

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

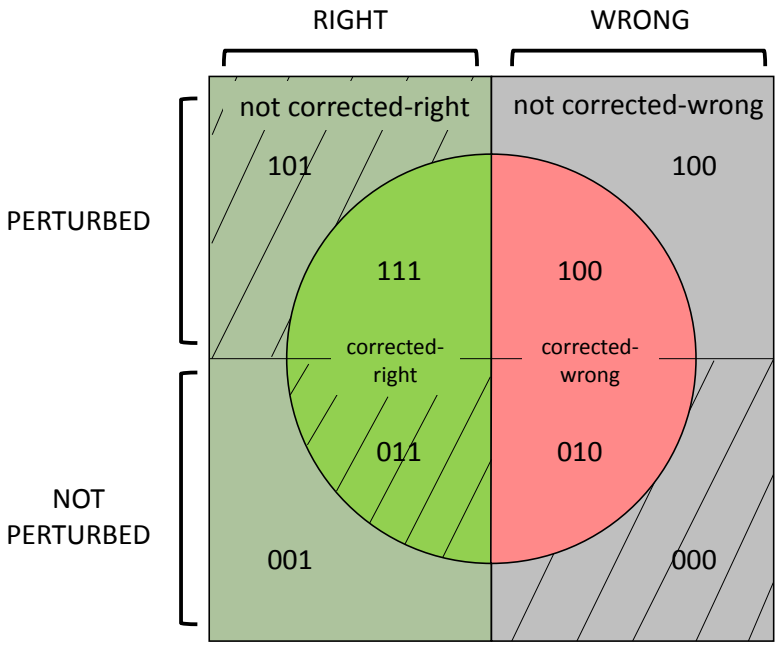

Figure 2: Data representation

6 Test

7 Demo and GUI

8 Prediction Task

9 Conclusions