Correcting the Autocorrect: Context-Aware Typographical Error Correction via Training Data Augmentation

Kshitij Shah, Gerard de Melo
Department of Computer Science
Rutgers University–New Brunswick
kshitij.shah@rutgers.edu, gdm@demelo.org

Abstract

In this paper, we explore the artificial generation of typographical errors based on real-world statistics. We first draw on a small set of annotated data to compute spelling error statistics. These are then invoked to introduce errors into substantially larger corpora. The generation methodology allows us to generate particularly challenging errors that require context-aware error detection. We use it to create a set of English language error detection and correction datasets. Finally, we examine the effectiveness of machine learning models for detecting and correcting errors based on this data. The datasets are available at `http://typo.nlproc.org`.

Keywords: Corpus, Error Generation, Deep Learning

1. Introduction

With the emergence of the Internet as the preferred medium for communication and the rise of social media, the number of words typed by an average human being has increased manyfold in the last decades. The majority of this communication is casual in nature, with less attention and time dedicated by the writer compared to formal writing. This leads to a plethora of typographical errors in such text. This problem is exacerbated by the comparably small size of the keyboard on hand-held devices. Hence, typographical errors, which were somewhat more limited and likely to be corrected immediately in past decades, are now ubiquitous.

Traditional solutions such as dictionary-based spelling correction are often inadequate. First of all, online writing tends to make liberal use of neologisms, slang words, and so on, that are lacking in pre-existing dictionaries. Any automated means to mine dictionaries from social media risk also incorporating frequent misspellings such as **seperate* into the dictionary. Second, a higher error rate also entails a larger likelihood of a typographical error leading to a form that happens to be a legitimate word in the dictionary. This problem exists even with more advanced spelling correction software, as indeed, many such errors are now *caused* by autocorrection software.

To overcome these challenges, an alternative is to draw on advances in machine learning to identify and correct such typographical errors. While autocorrection typically considers just the current and possibly the last few words, deep models that can account for both sides of a larger context window have the potential to more accurately determine whether a word fits in context. Unfortunately, deep learning generally requires large annotated corpora for effective training. Such large annotated datasets are difficult to procure, as the labeling process tends to be time-consuming and expensive (Qiu et al., 2020). There have been significant efforts in the last decade to overcome this gap for the task of grammatical error correction (Dale and Kilgarriff, 2011; Ng et al., 2013; Ng et al., 2014). However, no sufficiently large datasets exist for typographical error correction.

In this paper, we aim to generate realistic typographical errors based on statistical distributions collected from a relatively small annotated seed dataset. Our method captures the error patterns from the seed data and generates similar error distributions on error-free target corpora of arbitrary size. An important special case, particularly when typing on hand-held devices, is the abundance of *real-word errors*, also known as *atomic typos*. These occur when misspelled words happen to also be valid words in the dictionary, but used in the wrong context, requiring *context-aware spelling correction*. We examine inducing such errors automatically via a dictionary-based spelling corrector. Finally, we explore the effectiveness of deep neural networks to detect and correct such errors. While the experiments in this paper are limited to English, our method is applicable to any language with a restricted (true) alphabet.

2. Related Work

Due to the high cost and difficulty of labeling large text datasets for error correction, several studies explored generating artificial grammatical mistakes (Foster and Andersen, 2009; Felice and Yuan, 2014; Ng et al., 2013; Rei et al., 2017; Kasewa et al., 2018). For example, Foster and Andersen (2009) achieved this by moving, substituting, inserting, and removing words in a sentence, and investigated part-of-speech tags to induce more realistic errors. Felice and Yuan (2014) used probability word-level statistics computed on the CoNLL 2013 shared task data (Ng et al., 2013) to introduce artificial grammatical errors at the word level. Rei et al. (2017) used a statistical machine translation system to translate correct text to ungrammatical text. For training, they relied on the annotated dataset from CoNLL 2014 (Ng et al., 2014) but considered in inverted order: They feed the corrected version as input and the ungrammatical one as the output. These works aim to mimic the phenomenon of grammatical errors, i.e., when linguistic utterances are deemed lacking in grammatical acceptability with respect to a linguistic standard.

Only very little research has been conducted on simulating errors originating from basic typographical mistakes made by a writer. While these may stem from a lack of knowledge of the authoritative spelling or appear as a symptom of conditions such as dyslexia, they may also be caused by

simple key entry mistakes such as those referred to as *fat finger errors*.

In the past, such errors were often corrected by means of a dictionary-based spelling corrector, but with the proliferation of hand-held devices, this no longer seems sufficient. Indeed, many such devices invoke autocorrection tools, which may lead to entirely new errors that are very hard to detect. Bigert et al. (2003) developed a tool called 'Missplel', which could introduce character-level errors, among others. However, those errors were not modeled upon real-world data. Whitelaw et al. (2009) repurposed Web data as a noisy corpus for spelling correction and used a limited corpus with artificially inserted misspellings to tune classifiers. Ghosh and Kristensson (2017) created a synthetic dataset of incorrect key strokes by sampling from a Gaussian at each key location on a virtual keyboard. They also created another dataset by replacing correct words with their misspellings, as given by an annotated typo corpus (Aramaki, 2010). Their paper noted the lack of datasets for this problem, which led them to create their own to train and test their model. Our work differs from their approach in that we induce a noise model from text that can then be used to introduce a wide range of errors, instead of replacing a small set of words by their misspellings. Baba and Suzuki (2012) studied different error categories arising during typing.

Methods for context-aware spelling correction have mostly focused on small sets of words typically confused in human writing: Many of the most well-known approaches to this task rely on predefined *confusion sets* to solve it (Carlson and Fette, 2007; Golding and Roth, 1999; Carlson et al., 2001; Banko and Brill, 2001). However, with the prominence of autocorrect systems on mobile devices, this approach is ineffective. Hence, we explore generating realistic spelling errors requiring context-aware correction in this paper.

3. Method

Our method for artificial error generation consists of two phases. First, we compute probabilities for different types of errors based on a small annotated corpus. Subsequently, we rely on these statistics to generate errors in the target text.

3.1. Model Induction

We classify typographical errors into four categories and induce a model consisting of a series of probability distributions. Errors of each of these types are counted in an annotated corpus and a specific set of rules are used to calculate probabilities for each category. In many cases, probability distributions are computed by counting a specific error pattern with regard to different characters, and then normalizing by dividing counts by the total number of errors of that category.

Substitution Errors. A substitution error occurs when another character replaces the correct character. Intuitively, a character is more likely to be replaced by a nearby character on a QWERTY keyboard. Moreover, some characters are more likely to be mistyped compared to others. We empirically determine the probability of each character c being substituted $P(\text{substitution} \mid c)$, as well as the probability of each character c' in the character set C replacing c, i.e., $P_{\text{substitution}}(c' \mid c)$.

$$P(\text{substitution} \mid c) = \frac{\text{f}_{\text{substitution}}(c)}{\text{f}(c)}$$

$$P_{\text{substitution}}(c' \mid c) = \frac{\text{f}_{\text{substitution}}(c', c)}{\sum_{\overline{c} \in C} \text{f}_{\text{substitution}}(\overline{c}, c)}$$

where $\text{f}_{\text{substitution}}(x)$ denotes the frequency of character x being replaced, while $\text{f}_{\text{substitution}}(y, x)$ denotes the frequency of character y replacing character x.

Insertion Errors. Insertion errors occur when an additional character is inserted by mistake. We assess the probability of a given inserted character depending on the adjacent characters, assuming that these typically result from nearby keys being activated near-simultaneously. However, the added character could be attributed to either the previous or to the next character. We resolve this by attributing it to the adjacent character that is nearer to it on a virtual keyboard. If the distance to both neighbors on the virtual keyboard is the same, we assign it to one of them randomly. The computed statistics include probabilities of any character being inserted before or after a given character c, i.e., $P(\text{insertion} \mid c)$, and the individual probabilities for the inserted character, given by $P_{\text{insertion}}(c' \mid c)$. The probabilities for insertion before and after a given character are computed separately.

$$P(\text{insertion} \mid c) = \frac{\text{f}_{\text{insertion}}(c)}{\text{f}(c)}$$

$$P_{\text{insertion}}(c' \mid c) = \frac{\text{f}_{\text{insertion}}(c', c)}{\sum_{\overline{c} \in C} \text{f}_{\text{insertion}}(\overline{c}, c)}$$

where $\text{f}_{\text{insertion}}(x)$ is the frequency of any character being inserted adjacent to x, and $\text{f}_{\text{insertion}}(y, x)$ denotes the frequency of character y being inserted adjacent to character x.

Another caveat is when the inserted character is the same as its neighbor, i.e., the same character is typed twice. In this case, we cannot easily decide which of these two occurrences ought to be considered as the added one. To address this issue, we define the distinct subcategory of replication errors, which we measure separately.

Replication Errors. A replication error occurs when a character is repeated twice. We compute a separate probability of replication $P(\text{replication} \mid c)$ given each character.

$$P(\text{replication} \mid c) = \frac{\text{f}_{\text{replication}}(c)}{\text{f}(c)}$$

Deletion Errors. A deletion error occurs when a user misses a particular character that should have been entered. Some characters exhibit a higher tendency of getting missed. Hence, we compute the probability $P(\text{deletion} \mid c)$ of each character c getting missed.

$$P(\text{deletion} \mid c) = \frac{\text{f}_{\text{deletion}}(c)}{\text{f}(c)}$$

Transposition Errors. A transposition error is registered when two consecutive characters are typed out of order. Only consecutive character pairs are considered for this type of error and the probability of transposition for every possible sequence of two characters c_1 and c_2 is represented as $P(\text{transposition} \mid c_1c_2)$.

$$P(\text{transposition} \mid c_1c_2) = \frac{\text{f}_{\text{transposition}}(c_1c_2)}{\text{f}(c_1c_2)}$$

3.2. Error Generation

Corpus Cleaning. Before introducing errors, we seek to ensure that the original corpus does not have substantial typographical errors. Thus, in a preprocessing step, given a dictionary vocabulary V, documents with out-of-vocabulary words not in V are discarded.

Error Induction. Our error generation process is based on the assumption that typing proceeds progressively character by character and that errors may occur at every keystroke. Hence, the errors are generated at each character based on the precomputed statistics. Each error category is considered individually in random order with its respective probability multiplied by a weighting coefficient for that error category. The weighting coefficients allow us to control the relative frequency of each error category individually as well as the overall error rate. Retaining a uniform distribution of coefficients mimics the original distribution observed on annotated data. These coefficients also enable us to the use of character frequency distributions from a corpus other than the one used to compute error statistics. This later allows us to compute probability values based on the character distribution of the target corpus we are inducing errors on, while still using the error distribution from the annotated corpus. In the case of insertion or substitution errors, if an error is determined to be generated, the candidate for insertion or substitution is selected based on the computed statistics. If a particular error category is applied to a character, the remaining error categories are no longer considered for it.

Error Replacement. Having generated the corrupted text, we invoke a dictionary-based spelling correction algorithm to generate confused words that are valid words but unlikely to fit in the context. Specifically, confusion is enforced by selecting the highest-ranked suggestion that is not correct.[1] However, if Enchant makes only one suggestion for a given error, it is accepted without forcing confusion.
This results in hard to detect errors calling for context-aware spelling correction.

Output Dataset. During this entire process, the number of tokens is kept the same as in the original text. For single character words and rarely for other short words, it is possible that an entire word is removed during the character-level corruption. In such cases a placeholder token is used in its place. The placeholder is a special symbol such as <UNK>. If a word is split in two by the spelling corrector, the longer one is chosen and the shorter one discarded. For instance, if *blike* is corrected as *be like*, we only consider *like* and discard *be*. This constraint facilitates working with our data, because the labels are on every word and are easier to process if the number of words does not diverge between the actual text and our generated version. Thus, in the final error detection datasets that we induce, each word has a binary label associated with it. Introduced errors are labeled as positive, while original words are labeled as negative. We also provide the original word as the ground truth for the task of error correction.

[1] Another variant of the data, which picks the best suggestion without forcing confusion is also generated, but not used in the subsequent experiments presented in this paper. It will be available to the community as part of the dataset.

4. Analysis of Error Statistics

We derive error statistics from the Twitter Typo Corpus (Aramaki, 2010), which contains 39,171 pairs of words with typographical errors along with the correct word. While the dataset only includes lower-case English letters, our method can generalize to any character set.
Figure 1 shows the distribution of errors among the error types described in Section 3 in the Twitter Typo Corpus (Aramaki, 2010) used for error induction. We found that the error distribution is dominated by substitution, insertion, and deletion errors, while the replication and transposition errors are relatively scarce. Note that the latter two can be considered derivative. For instance, replication errors can be viewed as insertion errors with the inserted character being the same as an adjacent character.

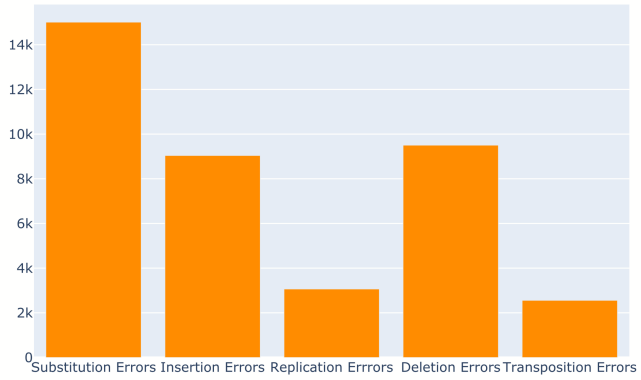

Figure 1: Distribution of Error Types

Figure 2 shows the frequency distribution of inserted characters. We observe several patterns emerging from the data. The characters that on standard QWERTY keyboards are located near the finger tips in a natural typing position are more likely to be inserted by mistake. For example, ‘a’, ‘d’, ‘e’, and ‘i’ exhibit high insertion error frequencies. Figure 3 shows the frequencies for deletion errors, which accounts for a certain character being missed by the writer. The deletion frequency is highly correlated with the natural occurrence frequency of characters. Frequently used vowels such as ‘a’, ‘e’, ‘i’, ‘o’ show a higher deletion frequency. However, this does not imply that they are more likely to be missed. Our model combines these statistics with the occurrence frequencies and error rate hyperparameter in order to generate probabilities of deletion errors. A noticeable characteristic that emerges is that the characters in the middle row of QWERTY keyboards such as ‘d’, ‘f’, ‘g’, ‘j’, ‘k’ are less likely