Noisy channels and Bayesian autocorrection

Diego Contreras, Gowtham Arumugam, Kavish Purohit and Walid El Mouahidi

June 8, 2022

Abstract

This paper studies the problem of non-word spell checking by characterizing mistyping in the English language as an example of a noisy channel. It combats the noisy distortion of words by implementing a Bayesian spellcheck model, which generates a conditional probability distribution over possible corrections. A major feature of this model is the usage of the Levenshtein distance, a string metric that quantifies the distance between words. We train our model and apply it to several misspellings to evaluate this correction strategy. Finally, we discuss possible improvements and contextualize our study within the broader field of language processing.

Introduction

Writing is one of the most important parts of communication through which we humans can express our feelings, views, opinions as well as our intellect. Even typing has now become a crucial and necessary part of digital communication. But no one can ignore errors. Mistakes are supposed to happen, whether it be humans or machines. Sometimes, letters in words are misplaced, misspelled, or mispronounced, and the connotation of the whole phrase or sentence is altered. Words in the English language are often misspelled or written incorrectly, resulting in misinterpretation errors; this problem is as old as language itself, arising from inaccuracies in reading, writing, and transmission.

There are several types of errors that occur in the English language, whether it be grammatical or contextual. The most common type of error that occurs is the misspelling of a word; these can be traced to single-character transformations such as deletions, insertions, substitutions, and transpositions. Besides syntactical errors, we can further filter errors down to two types. The first is where a misspelled **word is a non-word,** a word that does not belong to the English dictionary. The second is a contextual type of error, **the misspelled word is an actual word but it is not the desired word**.

This paper focuses on the former category of errors. It characterizes error in the English language as an instance of a noisy channel and combats this “noise” by implementing a Bayesian spell checking model. Particularly, it computes a channel model using the Levenshtein distance and uses this in conjunction with prior probabilities from our datasets to generate a probability distribution over candidates for correction. Insights regarding the English language are drawn using the results, and observations are made in further directions.

Data Sets used

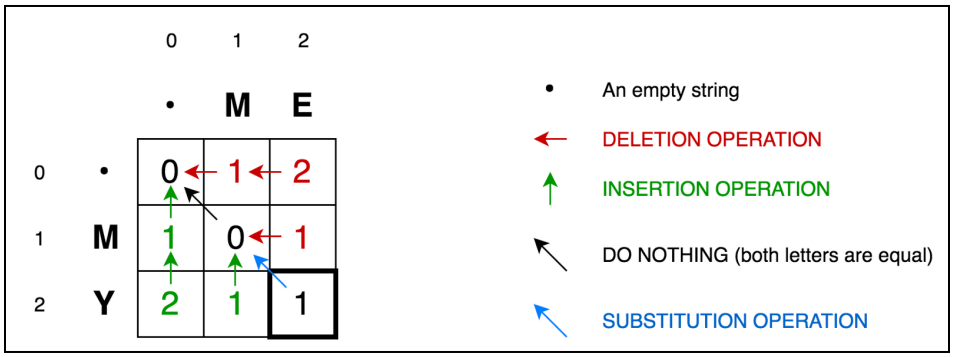
Two main datasets are used in this paper. The first (Data A) is a set of the most commonly used words on the Internet; it was composed of ~300,000 words but was reduced to 70,650 to exclude non-words and proper nouns. The set was used as our directory for “valid”

words and contains frequency data for each word, which is normalized to give our **prior** probability. Figure 1.1 (Fig. 1.1) contains 6 graphs that illustrate the distribution of the words by plotting the probabilities of the most common words on the x-axis and the corresponding words on the y-axis. Plotting 70,000 words on the graph was tedious, so the given 6 graphs each use a different range of words starting from word 1 for better detail. The most frequent words were 'the', 'and', 'of'; these were the only words with a probability greater than 0.01. All the other words have relatively less probability.

The second dataset (Data B) is a set of common misspellings made by English Wikipedia editors, each paired with the intended correct word. The number of correct words was 1,922, corresponding to a total of 2,455 misspellings; it was processed similarly to Data A, yielding a dataset of 1,662 unique correct words with a total of 2,127 misspellings. This dataset was further limited to errors that were strictly an edit distance of 1 away from the desired word. This dataset was used to 'train' our model in quantifying the likelihood of different kinds of error.

Levenshtein Distance

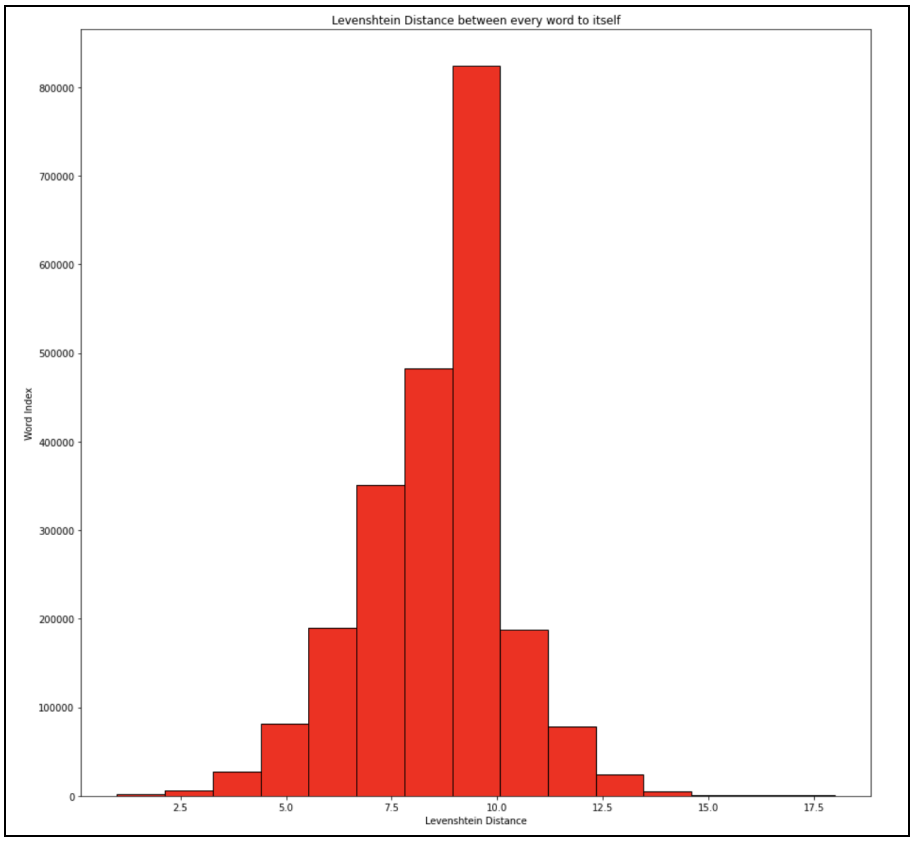
The metric we use to quantify the amount of error between a word and its misspelling is the Levenshtein Distance. It is a string metric used to measure the edit distance between two words, which are not necessarily the same length. The Levenshtein distance measures the minimum number of single-letter changes (i.e. substitutions, insertions, deletions) needed to transform an input word into a target word. Because Levenshtein distance tracks single-letter changes, it does not include transpositions (swapping the positions of two words). Fig 2.1 below shows a diagram of how the Levenshtein distance works. It checks all the letters in the first sequence with all the letters in the second sequence. In the example given, the first letter 'M' is the same so the matrix (1,1) will not have any changes made. But now, the letter 'E' in the first sequence needs to be changed to the letter 'Y' so the matrix (2,2) now has to have the substitution operation. And therefore, we have the target word "MY". So, the Levenshtein distance in this example is one because it has only one change (substitution) that was made.



Exploring data using the Levenshtein distance

The main purpose of using the Levenshtein distance was to build our auto-correction model. However, we can also use this metric to make some preliminary observations on the two datasets we have, particularly on the error tendencies of the English language as displayed by our data:

Data A



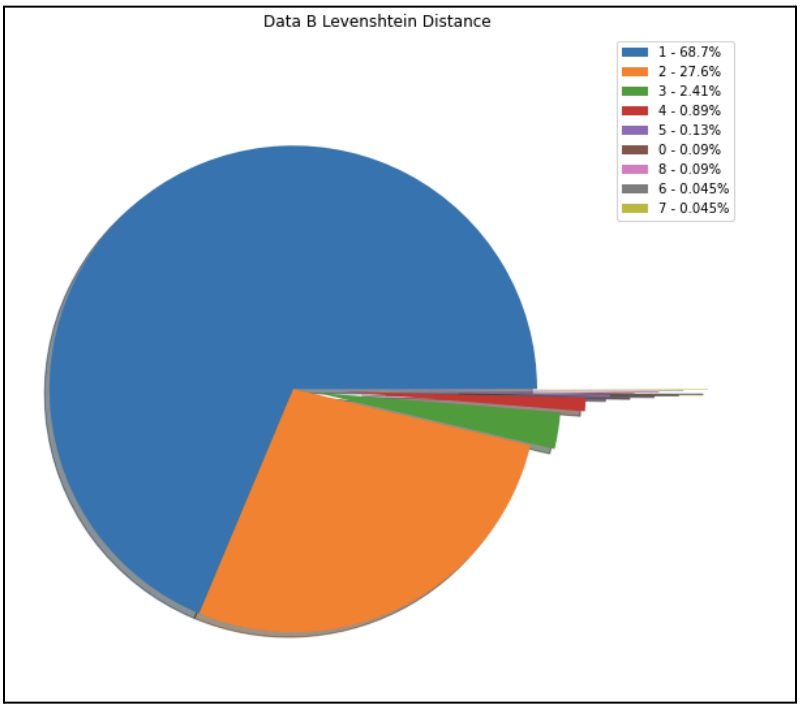
The two types of error that we previously read about were the non-word error and the real-word error. In our first dataset (Data A), we have taken the most common words in the English language used online.

The graph Fig. 2.2 (shown on the left) indicates what the distribution of the Levenshtein distance is to the number of words in our dataset. We took the dataset, calculated and calibrated the Levenshtein distance between each of the 70,000 words.

The results of the graph indicate that the edit or the Levenshtein distance between all of the 70,000 words in our data set centers between lengths 7 and 11. That is, a majority of the words in the English language need to be edited with at least 7 letters. The highest number of the Levenshtein distances in the dataset is 9. So, we can conclude from this graph that in the English language, the probability of getting one correct word, which is not the desired word, for another correct word, which is the desired word, as an error is very low. For most of the words in our dataset, we have to make at least 7 changes, which are a combination of insertions, deletions, and substitutions to get the target word. In simple terms, if you want the desired word ‘observe’ and you took the wrong word ‘obtain’ but it still stands on its own. So, both the words have two

common letters. So, to get ‘observe’ from ‘obtain’, we need to delete the four letters ‘-tain’ and append ‘-serve’. So, it is 4 deletions and 5 substitutions/replacements, giving a total of 9.

Data B



Our second dataset (Data B) is a set of non-word misspellings and their corresponding correct words. The pie chart Fig. 2.3 (shown on the right) displays all the Levenshtein distances between each misspelling and its correction as a proportion of the entire set of misspellings. We find that the vast majority of misspellings are a low edit distance away from the correct word, and hence they are ‘close’ to the word they were meant to be; i.e, errors in typing result in only minor distortions. For example, the most common kind of error appears to be a single character being transformed (Levenshtein distance 1) and occurs significantly more than errors of greater edit distances. Edit distances that are greater than 3 are much more rarely observed. This indicates that fewer character transformations are more worthwhile to analyze in building a spellcheck model; hence, the model described in this paper focuses on the most common kind of error, single-character errors.

The Autocorrect Model

To build our autocorrect model we view the misspelling problem as a noisy channel. The noisy channel takes in a word as an input and transforms it using a single letter substitution, deletion, or insertion to yield a misspelled word. We ignore transpositions for simplicity since transpositions can be thought of as an insertion followed by a deletion.

For our formulation of the autocorrect model, we assume that the misspelling produced by the noisy channel is a **non-word** (i.e. it is not a valid word in the English language). We attempt to combat this noise by implementing a spellcheck model that uses Bayesian inference.

Bayesian inference uses real-world data to make predictions. It computes the conditional probability of an event 'a' given event 'b' given training data and compares it to our test data. In this project our model uses this formula to find the conditional probability of a word *w* being the correct word, given that our misspelling is some non-word *x*:

$$P(w|x) = \frac{P(x|w)P(w)}{P(x)}$$

Here $P(w)$ is our **prior** probability; it is the probability of the occurrence of *w*, obtained from the frequencies in Data A. $P(x)$ is the probability of the misspelled word. $P(x|w)$ is our **channel model**; it is our measure of the likelihood of different kinds of errors that may transform a desired (valid) word into a misspelling; this is computed using our training data.

The strategy of our model is to take a misspelled word as input and generate a conditional probability distribution over several candidate words; it then picks the highest-ranked word as the likely correction. The candidates are valid words that are a Levenshtein distance of 1 away from the misspelled input. Note that since for any given input, $P(x)$ is a constant across all candidates; it is therefore dropped. Our computation for the model is then:

$$\hat{w} = \max_{w \in C} P(x|w)P(w)$$

where C is the set of candidate words and $\hat{w}$ is the final word recommended by our model.

The strategy to generate our channel model $P(x|w)$ is to create **confusion matrices**. These are 27x26 matrices, one for each single-character transformation (insertion, substitution, deletion) giving three in total. The 27 rows correspond to every character in the alphabet with the addition of a null character to define deletions and insertions; the columns correspond to simply the 26

alphabetical characters. We iterate through a portion of our training data and count the different kinds of character transformations; for example, if we encounter a substitution where an ‘a’ was mistyped as a ‘b’, we add 1 to the entry at index [0,1] for the matrix corresponding to substitution. Similar operations correspond to the insertion and deletion transformations. The three resulting populated matrices are then normalized to give us P($x|w$) for each candidate word.

Results/Inferences

The Bayesian Inference model works by entering words that are 1 edit distance away from the misspelled word. In the case of “hillo”, “hello, hilly, hill, access” are all words that are 1 edit distance away from the misspelled word - “hillo”. Our model produces probabilities for all these words about how probable the candidate word is the actual word intended.

This type of model is highly sensitive to the data that is fed into it. The bigger the training set the more accurate the model since it will be exposed to more types of errors and how frequent they are in the real world. Our training data was fairly sparse, meaning that some of the entries of the confusion matrix were 0. This means that we did not have recorded errors of that type in our data and could not provide an accurate prediction for some types of errors.

In the tables below we have misspelled words and candidates for possible words that were intended, ranked by the highest probability.

```
misp("hillo")
```

✓ 0.5s

	correct	prob word	prob word g	prob of mispel	total prob
1	hello	0.267426	6.249820e-05	0.028669	1.791757e-06
3	hilly	0.004148	9.694409e-07	0.000683	6.617344e-10
0	hill	0.519337	1.213702e-04	0.000000	0.000000e+00
2	hills	0.209089	4.886449e-05	0.000000	0.000000e+00

```
misp("accress")
```

✓ 0.5s

	correct	prob word	prob word g	prob of mispel	total prob
1	actress	0.031156	0.000013	0.000683	9.073172e-09
0	access	0.968844	0.000413	0.000000	0.000000e+00

```
misp("poen")
```

✓ 0.5s

	correct	prob word	prob word g	prob of mispel	total prob
2	poem	0.058858	1.689280e-05	0.000683	1.153092e-08
4	poon	0.001749	5.018892e-07	0.003413	1.712933e-09
5	peen	0.000373	1.071331e-07	0.015017	1.608825e-09
0	porn	0.788140	2.262025e-04	0.000000	0.000000e+00
1	pen	0.108159	3.104253e-05	0.000000	0.000000e+00
3	poet	0.042721	1.226133e-05	0.000000	0.000000e+00

The spell checking algorithm outputs the list of the possible correct words that can correspond to the misspelled one, and each one has four different probabilities:

- “prob word” is the probability of the word inside the sample space made by the suggested correct words.
- “prob word g” is the general probability of the word based on the Data A.
- “prob of mispel” is the probability of the correct word being misspelled in our input misspelled word.
- “total prob” is the product of the “prob word g” and “prob of mispel”.

The “total prob” column values are the most important of all as they take into account both the probability of a spelling error happening and the probability of the correct word being utilized.

Further directions

Our study focused on a very specific spell-checking problem; it looked at spell checking of non-words and was built using only single-edit-distance words. Improvements to this model could involve consideration of broader kinds of spell-checking problems that included real-world errors as well as more ‘distant’ errors. Additionally, our model focuses purely on a single word passed as input; however, state-of-the-art methods such as NLP also take into account the sentence that this word is embedded in to infer context. For example, the strategy in modern autosuggestion is to generate conditional probability distributions for a given word based on the *previously* observed words in the sentence. Additionally, the type of data our model was trained on was typed errors; an interesting tangent would be to use error data from handwritten words, which would shed light on the kinds of errors involved in human handwriting.

Code

https://github.com/dfcontra/final_project

Bibliography

Journals:

Sharma, S., & Gupta, S. (2015). A Correction Model for Real-word Errors. *Procedia Computer Science*, *70*, 99–106. https://doi.org/10.1016/j.procs.2015.10.047

Kernighan, M. D., K. W. Church, & W. A. Gale. (1990). A spelling correction program based on a noisy channel model. *COLING*, *volume II*. https://aclanthology.org/C90-2036.pdf

Jurafsky, Daniel James Martin. "Spelling Correction and the Noisy Channel." *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed., PEARSON INDIA, 2022, pp. 1–14.

Websites:

Wikipedia contributors. "Noisy Channel Model." *Wikipedia*, 17 Apr. 2022, en.wikipedia.org/wiki/Noisy_channel_model#:%7E:text=The%20noisy%20channel%20model%20is,been%20scrambled%20in%20some%20manner.

Wikipedia contributors. "Levenshtein Distance." *Wikipedia*, 14 Apr. 2022, en.wikipedia.org/wiki/Levenshtein_distance.