



Complex Engineering Problem
Roman Urdu Spell Checker and Corrector

Submitted by

Ammara Noor	2020-CE-38
Ayesha Shafique	2020-CE-39
Hira Sohail	2020-CE-41
Aneeza Sabir	2020-CE-42
Aima Naseer	2020-CE-43
Anyar Yasir	2020-CE-44

Submitted to

Dr. Yasir Saleem

Course

CMPE-341 Artificial Intelligence

Semester

5th

Date

13th December 2022

Department of Computer of Engineering

University of Engineering and Technology, Lahore

Problem Statement

The goal of this assignment was to implement a spell corrector for roman Urdu, using Noisy Channel based on Bayes Theorem.

Abstract

Spell Checker is used to identifying and correct mistakes made by users while writing text and the mistakes are generally spelling mistakes. An intelligent spelling correction system is proposed to automatically correct spelling mistakes in text-editor or text documents using contextual information of the confusing words. The system is capable to correct words belonging to the set of confusing words fed into it if they are contextually wrong. In this technique, an algorithm to identify and correct real-world errors is proposed.

Introduction

Writing is one of the predominant and vital ways of communication through which humans can express their views to others and keep a record. It is one of the most effective forms of language representation. Writing is composed of vocabulary, semantics, and grammar. The scalability of writing text has increased, due to which many issues such as spelling mistakes also evolved. Not every person is proficient in representing language. Some carry out the task of formal writing and others do free writing, depending on their respective goals. Free writing is a task where a writer writes while ignoring grammatical and spelling mistakes, which means the chances of making mistakes, are very high. Spelling correction is an application used to identify and correct spelling mistakes in the text written by the user.

Problem Analysis

A wrong word will be fed to the algorithm and it should pick the best possible correct word out of some possible candidate words, which will be generated using the error model and n-gram language model.

$$\hat{w} = \operatorname{argmax}_{w \in \text{candidates}} P(x|w) * P(w)$$

Here:

- $P(x|w)$ is the error model
- $P(w)$ is the language model
- “x” is the possible wrong word
- “w” belongs to candidate words

Data

The implementation requires two data files **data.txt** which will serve as the corpus and **misspellings.txt** for generating error model tables. data.txt is a collection of Roman Urdu sentences and misspellings.txt contains a list of words and their misspellings. One user input file **file.txt** will be required which is supposed to be checked and corrected. One helper file **wrong.txt** will be used which stores all incorrect words in file.txt.

History- Noisy channel

Noisy Channel for spelling proposed around 1990

1. IBM

Mays, Eric, Fred J. Damerau and Robert L. Mercer. 1991. Context-based spelling correction. Information Processing and Management, 23(5), 517–522

2. AT&T Bell Labs

Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. Proceedings of COLING 1990, 205-210

Noisy Channel

The intuition of the noisy channel model is to treat the misspelled word as if a correctly spelled word had been “distorted” by being passed through a noisy communication channel.

This channel introduces “noise” in the form of substitutions or other changes to the letters, making it hard to recognize the “true” word. Our goal, then, is to build a model of the channel. Given this model, we then find the true word by passing every word of the language through our model of the noisy channel and seeing which one comes the closest to the misspelled word.

Besiyan Interface

This noisy channel model is a kind of Bayesian inference. We see an observation x (a misspelled word) and our job is to find the word w that generated this misspelled word. Out of all possible words in the vocabulary, we want to find the word w such that $P(w|x)$ is highest. We use the hat notation $\hat{}$ to mean “our estimate of the correct word”.

$$\hat{w} = \operatorname{argmax}_{w \in C} \overbrace{P(x|w)}^{\text{channel model}} \overbrace{P(w)}^{\text{prior}}$$

Edit distance

Edit distance expresses the difference between two strings as a nonnegative real number by counting edit operations that are required to transform one string into another. The two most commonly used edit distances are the **Levenshtein edit distance** and the **Damerau–Levenshtein distance**. Levenshtein identifies atomic edit operations such as

- Substitution: replaces one symbol with another;
- Deletion: removes a symbol (or replaces it with an empty string); and
- Insertion: adds a symbol or replaces an empty string with a symbol.

In addition, the Damerau–Levenshtein distance adds the operation of

- Transposition, which exchanges two subsequent symbols.

The significant difference between the Levenshtein distance (LD) and the Damerau–Levenshtein distance (DLD) is that the Levenshtein distance does not consider letter transposition. The edit operation set proposed by Levenshtein did not consider transposition as an edit operation because the transposition of two subsequent letters can be substituted by deletion and insertion or by two substitutions. We are implementing the Damerau-Levenshtein distance algorithm for the generation of candidate words having an edit distance equal to one.

Language Model

The most common form of a language model is the n-gram language model, calculated from the frequency of word sequences of size n. It gives the probability of a candidate word given its history of words. If the given n-gram sequence is not presented in the training corpus, the probability is calculated by a back-off that considers shorter contexts. The n-gram language model only depends on previous words, but other classifiers can make use of arbitrary features in any part of the context. The language model is usually trained on a training corpus that represents language with correct spelling. We are implementing Unigrams and Bigrams.

Confusion Matrices

A simple model might estimate, for example, $p(\text{kitaab}|\text{kitaob})$ just using the number of times that the letter a was substituted for the letter o in some large corpus of errors. To compute the probability for each edit in this way we'll need a confusion matrix that contains counts of errors. In general, a confusion matrix lists the matrix number of times one thing was confused with another. Thus for example a substitution matrix will be a square matrix of size 26x26 that represents the number of times one letter was incorrectly used instead of another. We'll use four confusion matrices:

- deletionMatrix [x;y]: count(xy typed as x)
- insertionMatrix [x;y]: count(x typed as xy)
- substitutionMatrix [x;y]: count(x typed as y)
- transpositionMatrix [x;y]: count(xy typed as yx)

Estimation of Channel Model

Here we do our implementation of the error model. Once we have the confusion matrices, we can estimate $P(x|w)$ as follows (where w_i is the i th character of the correct word w) and x_i is the i th character of the typo x :

$$P(x|w) = \begin{cases} \frac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Implementation

The spelling corrector contains four main components:

1. Language Model

The language model gives the probability of each word occurring in the vocabulary determined using the corpus given in data.txt. A simple unigram model should be trained using the provided file data.txt. Assuming that all words given in data.txt are correctly spelled.

2. Error Model

This model will be used to estimate the probability of typing x when w was intended. This error is to be modeled using character-level insert, delete, transpose and substitute tables. Generating these tables using the provided list of common misspellings in Roman Urdu.

3. Candidate Words Generation

The candidate model will give you all the possible words w which could have been mistyped as x . We consider words that are one edit away from x and exist in our vocabulary determined by data.txt. Note that there could be many possible words for example for "kitaab" deleting the b would give kitaa which does not exist in Roman Urdu. Thus, for each error word we will be generating a set of words from the vocabulary.

4. Selection Model using argmax

In the case of automatic selection, the final selection using argmax will be responsible for choosing the candidate word w which has the highest probability.

In the case of manual selection, the final selection will be done by the user by choosing appropriate candidate words.

Solution Design

The technique to solve this problem in a step-wise manner is as follows:

- The first objective was to read the corpus, tokenize the corpus into words using the spaCy library, and save the results to lists.
- The second step was to read the misspellings.txt and make a list of correct words and their possible misspelled versions, this was again done using the spaCy.
- The next objective was to obtain the language models $P(w)$:
 - ✓ UNI-GRAM Model was generated which will be later used in Noisy Channel.
 - ✓ Character Level BI-GRAM Model was generated which was later used for confusion matrices.
- After that, it was time to generate the Confusion Matrices:
 - ✓ Four matrices: insertion, deletion, substitution, and transposition matrices were generated with initial values: zeros (later to be updated)
 - ✓ A Function “find_edit_type” was implemented to find the type of edit required to convert a wrong word into the correct word.
 - ✓ Using the above function and lists generated from misspellings.txt, all the confusion matrices were updated. (See RESULTS section for outputs)
- The next objective was to obtain the Error Model $P(x|w)$, It was generated using the following equation:

$$P(x|w) = \begin{cases} \frac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

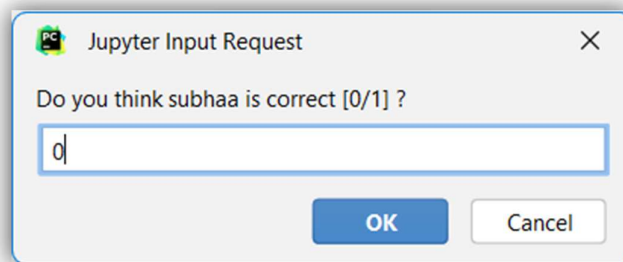
Here:

- w belongs to candidate words
 - x is the possible wrong word
- The next step was to generate a Candidate Words list, for which all the possible words in vocabulary that were 1 edit-distance away from the wrong word were picked.
 - Now, the final objective was to implement the Selection Model:
 - ✓ This model parsed the candidate words list and sent each candidate word to both error and language models to get the probabilities.
 - ✓ Then Multiplied those probabilities with a power of 10 (to get rid of very small numbers)
 - ✓ Then applied argmax to choose the word with the highest probability and returned it as the output.

Output of Implementation

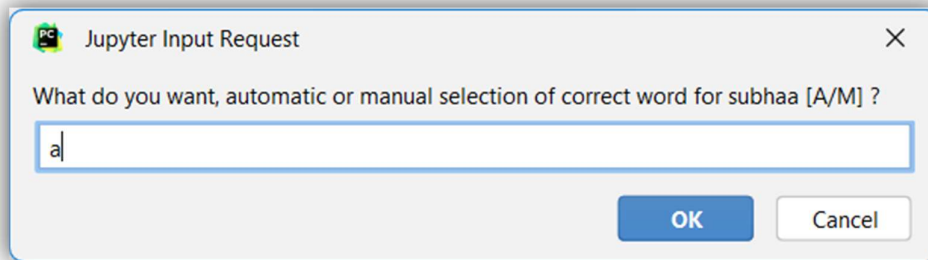
```
Input file Text:
aj me subhaa jldai uti test tiyar kia namaz prhe pir tyar hoi or university a gai
```

```
Misspelled Words are :
subhaa
jldai
prhe
```



A Jupyter Input Request dialog box with a title bar containing a PC icon and the text 'Jupyter Input Request'. The main text asks 'Do you think subhaa is correct [0/1] ?'. Below the text is a text input field containing the character '0'. At the bottom are two buttons: 'OK' and 'Cancel'.

```
Do you think subhaa is correct [0/1] ?
>>>>> 0
```



A Jupyter Input Request dialog box with a title bar containing a PC icon and the text 'Jupyter Input Request'. The main text asks 'What do you want, automatic or manual selection of correct word for subhaa [A/M] ?'. Below the text is a text input field containing the character 'a'. At the bottom are two buttons: 'OK' and 'Cancel'.

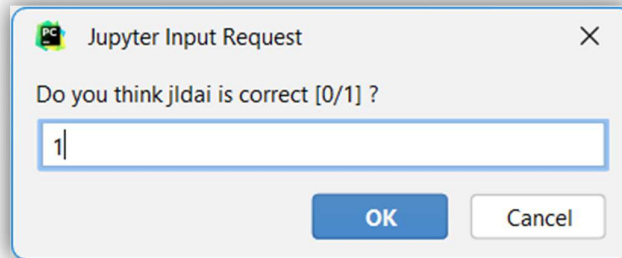
```
What do you want, automatic or manual selection of correct word for subhaa [A/M] ?
>>>>> Automatic
```

Candidate words for subhaa:

Candidates	x w	P(x w)	P(w)	P(x w)*P(w) *10^9
subha	a #	0.017793594306049824	4.856726566294317e-05	864.1862217605548
subhan	a n	0.019155627561231296	2.4283632831471587e-05	465.16822635335825

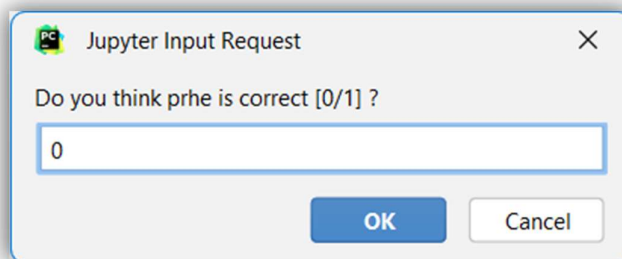
Automatic Best Correction for: subhaa is: subha

```
Do you think jldai is correct [0/1] ?  
>>>> 1
```



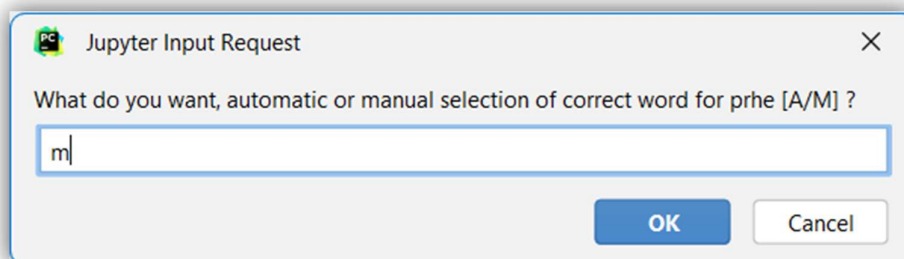
A dialog box titled "Jupyter Input Request" with a close button (X) in the top right corner. The text inside asks "Do you think jldai is correct [0/1] ?". Below the text is a text input field containing the number "1". At the bottom right are two buttons: "OK" and "Cancel".

```
jldai is added to the dictionary
```




A dialog box titled "Jupyter Input Request" with a close button (X) in the top right corner. The text inside asks "Do you think prhe is correct [0/1] ?". Below the text is a text input field containing the number "0". At the bottom right are two buttons: "OK" and "Cancel".

```
Do you think prhe is correct [0/1] ?  
>>>> 0
```




A dialog box titled "Jupyter Input Request" with a close button (X) in the top right corner. The text inside asks "What do you want, automatic or manual selection of correct word for prhe [A/M] ?". Below the text is a text input field containing the letter "m". At the bottom right are two buttons: "OK" and "Cancel".

```
What do you want, automatic or manual selection of correct word for prhe [A/M] ?  
>>>> Manual
```


 Jupyter Input Request

Select a candidate word from this list: rhe; prha; parhe; prhi; arhe; pre; pdhe; prh!!!

OK Cancel

 Jupyter Input Request

Select a candidate word from this list: rhe; prha; parhe; prhi; arhe; pre; pdhe; prh!!!

OK Cancel

Candidate words for prhe:

Candidates	x w	P(x w)	P(w)	P(x w)*P(w) *10 ⁹
rhe	p #	0.02779996804601374	2.4283632831471587e-05	675.0842167560403
prha	e a	0.016741232501082406	2.4283632831471587e-05	406.5379432025839
parhe	# a	0.002859866539561487	2.4283632831471587e-05	69.44794899372236
prhi	e i	0.016938110749185668	2.4283632831471587e-05	411.31886229202684
arhe	p a	0.018126713811516815	2.4283632831471587e-05	440.18246264003915
pre	h #	0.023317242007468805	2.4283632831471587e-05	566.2273435519379
pdhe	r d	0.016577447763771368	2.4283632831471587e-05	402.5606547783236
prh	e #	0.018505338078291814	2.4283632831471587e-05	449.3768353154884

Manual Correction for: prhe is: prhi

Output file Text after Correction:
 aj me subha jldai uti test tiyar kia namaz prhi pir tyar hoi or university a gai

Some Challenges Faced

- Implementation of the Unigram Model was initially taking $O(N*N)$ which was never-ending on the huge corpus, it was reduced to $O(N)$ later.
- Finding the type of edit required to make the wrong word correct was challenging, so the technique used to solve this was Divide and Conquer.
- Computation of the Error Model, where the character level bigram model is used, was challenging to implement.

Conclusion

Despite all the challenges and difficulties faced throughout the implementation, observing the results were satisfying.

References

- [1] Mark D. Kernighan, Kenneth W. Church, and William A. Gale.1990. [A Spelling Correction Program Based on a Noisy Channel Model](#). In *COLING 1990 Volume 2: Papers presented to the 13th International Conference on Computational Linguistics*.
- [2] M. A. Qureshi et al. 2022. [Sentiment Analysis of Reviews in Natural Language: Roman Urdu as a Case Study](#). In *IEEE Access*, vol. 10, pp. 24945-24954