Word AutoCorrection Implementation from scratch

Riccardo Caprile (4370774)

4370774@studenti.unige.it

# Heading 1: Introduction

Auto Correction is a Natural Language Processing feature we use every day constantly, for example when we use the smartphone keyboard while we are writing a message , when we search something on Google or when we write an essay on Microsoft Word.

The auto Correction algorithm will help us to understand how all of these devices and applications analyze and know that the word written by us is misspelled and suggests us a correct word that appears in the dictionary.

Autocorrect can save a lot of time for users by fixing common misspellings. It can also improve communication by catching errors that could change the meaning of what we are trying to write.

Therefore, it could be interesting realize what’s behind this algorithm and implement it from scratch without the support of particular Python libraries like Natural Language Toolkit aka NLTK , one of the most important libraries in NLP . The implementation of the algorithm and all the function implemented will be analyzed step by step in this report.

The language of the set of words used is going to be English but it could be changed easily by just modifying the content of the input .txt file.

## Heading 1.1: Prerequisites

* Python IDE : Jupyter Lab , Jupyter Notebook etc…
* Python libraries : Pandas , numpy , re , string …
* A .txt file that contains a set of words ( chapters of a books , reports , articles or paragraphs ). It can be chosen by the user.

# Heading 2 : Data Preprocessing

The first step for implementing this kind of feature is preprocessing our data. The dataset chosen is the book “The Hunger Games” in .txt extension.

After the importation of the necessary libraries we can start with the preprocessing.

The first function implemented is text\_file\_processing(book\_file) , which takes in input a .txt file , read it , converts all the letters in lower case and return them as a list.

Obviously the list of words will contain duplicates and we need to get rid of them. We can simply convert the list in a python set , which by definition set items are unordered ( they can appear in different order every time you use them) , unchangeable ( we cannot change the items after the set has been created ) and do not allow duplicate values .

Afterwards, some test print are implemented for checking that is all correct, for example first 20 words of the dataset and firs 20 unique words.

For this file , there are 7603 unique words total.

Then , we need to count how many times a word appears in the text. This can be done easily implementing a function named count\_words(list\_words).

This function takes in input the list of all the words in the dataset , creates an empty dictionary and fill it with the words and how many they appear in the text.

The key of the dictionary is the word , whereas the value is the number of times the word is in the text given.

As before , there are some test print : the first one checks which are the top 20 words present in the corpus and it was easily predictable that are words like conjunctions , possessive adjectives ,auxiliary verbs , pronouns and articles.

Moreover, there is a check that tells us how many keys are in the dictionary , and the number is equal to the unique words , so the dictionary is correct.

# Heading 3 : Word probabilities.

Next step is the computation of word probabilities.

We got a dictionary with all the unique words and the number of times they appear in the document , so we can calculate the word’s probability to get picked up randomly.

The probability is given by a simple formula , that we have seen during the course.

The probability to get a certain word is : P(‘word’) = number of times ‘word’ appears in the corpus / length(document).

So , the function word\_probabilities(dictionary\_count) compute for every word , its probability and returns a probabilities dictionary.

The code continues , as before , with some checking prints (top 20 words with the highest probability , sum of all the probabilities).

# Heading 4 : String edits (Insert , Delete , Swap , Replace)

There are 4 operations we can use to edit a given string. Let’s analyze and implement them.

## Heading 4.1 : Insert operation

Insert\_letter(word) function takes in input a single word. The objective of the function is to insert every letter of the alphabet in every position of the word, at beginning , in the end , in the middle and store every word combination in list , which it is going to be the output list.

Example : word = “the” , output list = [ athe , bthe, tahe……]

## Heading 4.2: Delete operation

Delete\_letter(word) function takes in input a single word. The objective of this function is delete just one letter and return a list of words formed after deleting that letter

Example : word = “the” , output list = [he , te , th]

## Heading 4.3: Swap operation

Swap\_letter(word) function takes in input a word. His objective is to switch adjacent letters from the string one by one. The function will return the list of all possible adjacent letter swap combination.

Example : power , swap list [ opwer, pwoer, poewr, powre]

## Heading 4.4: Replace operation

Replace\_letter(word) , same input as before. It replace every letter in the word with every letter in the alphabet forming a new word. The list of new formed words is the output. Duplicates are removed.

Example : word = “the”, output list = [ahe,bhe,che ……]