AutoCorrect Keyboard with python

Under the guidance of Prof. Kaspar.

Abstract

The project aims to develop an auto correction system for keyboard typing errors using Python and machine learning techniques. The system analyzes the text input in real-time and applies machine learning algorithms to predict and correct typing errors, thereby improving the accuracy and efficiency of the typing process. The project involves training a machine learning model using a large corpus of text data to learn the patterns and rules of language usage. The trained model is then integrated into the keyboard interface to provide automatic correction suggestions to the user. The results of the project demonstrate the potential of machine learning in enhancing the functionality and usability of keyboard interfaces. The project involves training a machine learning model using a large corpus of text data and applying NLP techniques such as tokenization, stemming, and part-of-speech tagging to preprocess the text data. The trained model is then used to identify and correct typing errors by comparing the input text with the learned patterns of language.

Keywords

- Python libraries
- Natural Language Processing(NLP)
- word probabilities
- Relative Frequency of words

Introduction

An autocorrect keyboard is a software tool that automatically corrects misspelled words or typos as the user types on a keyboard. Python is a popular programming language that can be used to create an autocorrect keyboard. One of the features that use Natural Language Processing (NLP) is the Autocorrect function. It is specially programmed to generalize all the correct words in the dictionary and looks for the words that are the most comparable to those words not in the vocabulary.

Methodology

- Preprocessing
- Building Language Model
- Implementing Spell Checker
- Integration with Keyboard
- Testing and Evaluation

Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of Artificial Intelligence that allows computers to understand and process natural human language. NLP uses a programming language that enables computers to evaluate and interpret large volumes of natural language data. Whether the language is spoken or written, natural language processing uses artificial intelligence to take real-world input, process it, and make sense of it in a way a computer can understand. At some point in processing, the input is converted to code that the computer can understand.

Autocorrect feature.

The Autocorrect model is programmed to correct spellings and errors while inputting text and locating the most comparable related words. It is completely based on NLP that compares the words in the vocabulary dictionary and the typed words on the keyboard. If the typed word is found in the dictionary, the autocorrect feature assumes you typed the correct term. If the word does not exist, the tool identifies the most comparable words in our smartphone's history, as it indicates. When building this model/feature, the following steps are involved:

Identifying misspelled word

A word is misspelled if the text is not found on the vocabulary of the corpus (dictionary), then the autocorrect system flags out for correction.

- INSERT - a letter should be added. Example "to" => "top", "two"
- DELETE - removes a letter. Example "hat" => "at", "ha", "ht"
- SWAP - swaps two adjacent letters. Example "eta" => "tea", "eat"
- REPLACE - changes one letter to another. Example "jaw" => "jar", "paw"

Order filtered candidates based on word probabilities.

The probabilities of the words are calculated based on the following formula:

$$P(w) = C(w)/V$$

- P(w)- the probability of a word w.
- C(w) - number of times (frequency) word appears in the vocabulary dictionary.
- V - the total sum of words in the dictionary.

Building autocorrects feature.

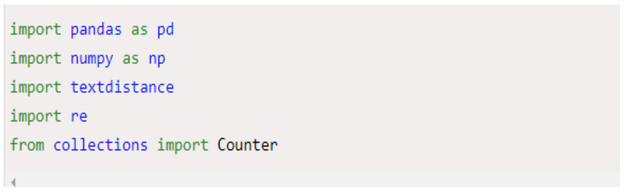
We will need a dictionary to develop an autocorrect system where the smartphone uses history to match the typed words to see if they are correct or not. For this tutorial, we will use a sample.txt file found in the project folder containing the 1000 most used vocabularies.

Libraries

Panda, NumPy, text distance, re, matplotlib, seaborn, surprise

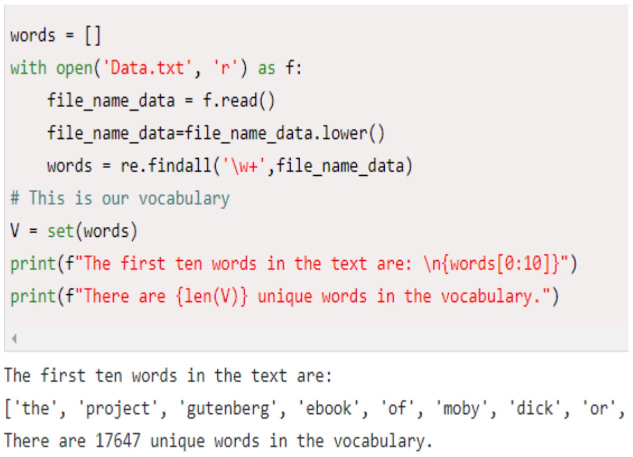
Python Code for Autocorrection

1.First we call all necessary libraries :



```
import pandas as pd
import numpy as np
import textdistance
import re
from collections import Counter
```

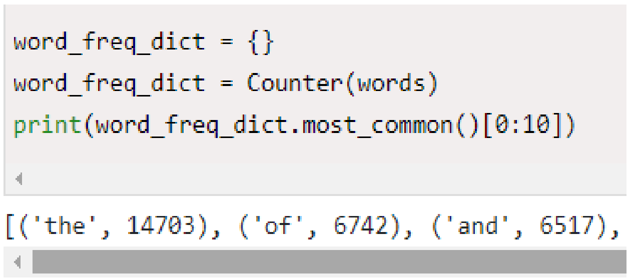
2. Now we read the Data.txt file f:



```
words = []
with open('Data.txt', 'r') as f:
    file_name_data = f.read()
    file_name_data=file_name_data.lower()
    words = re.findall('\w+',file_name_data)
# This is our vocabulary
V = set(words)
print(f"The first ten words in the text are: \n{words[0:10]}")
print(f"There are {len(V)} unique words in the vocabulary.")
```

```
The first ten words in the text are:
['the', 'project', 'gutenberg', 'ebook', 'of', 'moby', 'dick', 'or',
There are 17647 unique words in the vocabulary.
```

3.In the above code, we made a list of words, and now we need to build the frequency of those words, which can be easily done by using the counter function in Python :

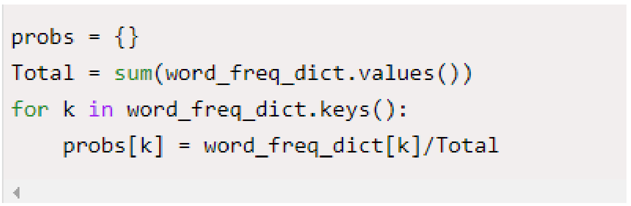


```
word_freq_dict = {}
word_freq_dict = Counter(words)
print(word_freq_dict.most_common()[0:10])
```

```
[('the', 14703), ('of', 6742), ('and', 6517),
```

4. Relative Frequency of words:

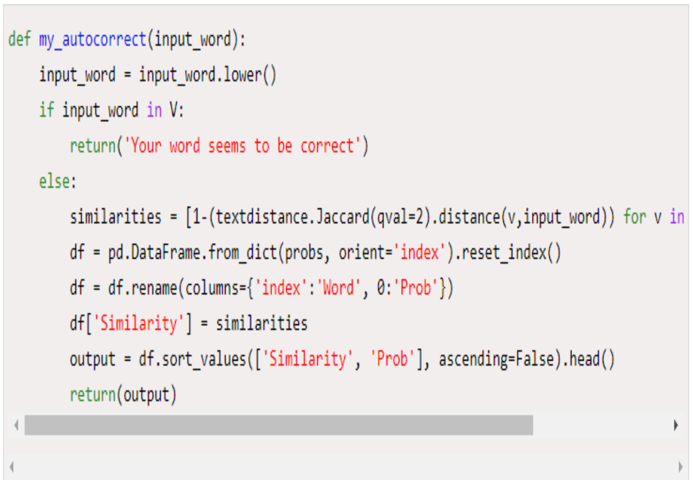
Now to get the probability of occurrence of each word, this equals the relative frequencies of the words:



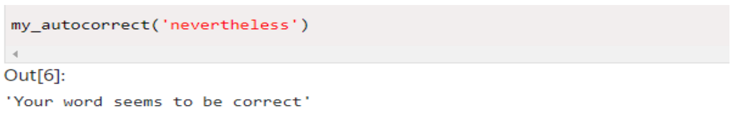
```
probs = {}
Total = sum(word_freq_dict.values())
for k in word_freq_dict.keys():
    probs[k] = word_freq_dict[k]/Total
```

5. Finding Similar Words:

Now we will sort similar words according to the Jaccard distance by calculating the 2 grams Q of the words. Next, we will return the 5 most similar words ordered by similarity and probability:



```
def my_autocorrect(input_word):
    input_word = input_word.lower()
    if input_word in V:
        return('Your word seems to be correct')
    else:
        similarities = [1-(textdistance.Jaccard(qval=2).distance(v,input_word)) for v in
        df = pd.DataFrame.from_dict(probs, orient='index').reset_index()
        df = df.rename(columns={'index':'Word', 0:'Prob'})
        df['Similarity'] = similarities
        output = df.sort_values(['Similarity', 'Prob'], ascending=False).head()
        return(output)
```

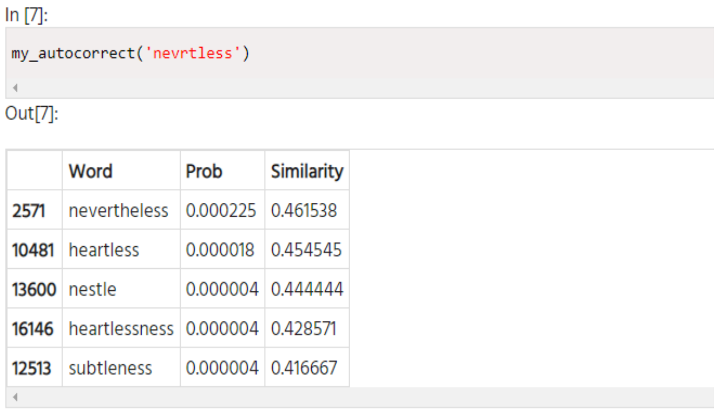


```
my_autocorrect('nevertheless')
```

Out[6]:

```
'Your word seems to be correct'
```

6.To check for the wrong spelled word 'nevrtless' and it return the 5 most similar words ordered by similarity and probability.



In [7]:

```
my_autocorrect('nevrtless')
```

Out[7]:

	Word	Prob	Similarity
2571	nevertheless	0.000225	0.461538
10481	heartless	0.000018	0.454545
13600	nestle	0.000004	0.444444
16146	heartlessness	0.000004	0.428571
12513	subtleness	0.000004	0.416667

Conclusion

In conclusion, building an autocorrect keyboard with Python is a useful project that can greatly improve the typing experience for users. However, by using machine learning techniques, such as n-gram models and deep learning, it is possible to build more sophisticated and accurate autocorrect algorithms.