

Korektor pisowni

Wykonali: Aleksandra Goryczka, Aleksander Kuś

Spis treści

1. Abstrakt	3
2. Wstęp	4
2.1. Cel	4
2.2. Zakres	4
2.3. Metodyka pracy	4
3. Część teoretyczna	5
4. Część praktyczna	6
4.1. Implementacja	6
4.2. Implementacja prostego GUI	8
4.3. Ocena działania	8
Podsumowanie	10
Bibliografia	11

1. Abstrakt

Niniejsza praca przedstawia korektor pisowni, narzędzie skonstruowane w celu poprawiania błędów ortograficznych w tekście w języku angielskim. Korektor posiada prosty interfejs użytkownika i w zależności od parametru podanego przez użytkownika będzie proponować odpowiednią liczbę korekcji.

2. Wstęp

2.1. Cel

Celem pracy było stworzenie korektora pisowni, w którym będzie mieć możliwość określenia liczby korekcji oraz stworzenie prostego interfejsu użytkownika.

2.2. Zakres

Zakres pracy obejmował implementację korektora pisowni oraz stworzenie prezentacji i raportu na temat pracy.

2.3. Metodyka pracy

W projekcie użyto Pythona 3.10.

3. Część teoretyczna

Przedstawiany program opiera się na teorii prawdopodobieństwa.

Staramy się znaleźć poprawkę c spośród wszystkich możliwych korekt, która maksymalizuje prawdopodobieństwo, że c to zamierzona poprawka, mając pierwotne słowo w :

$$\operatorname{argmax}_{c \in \text{kandydaci}} P(c|w)$$

Zgodnie z twierdzeniem Bayesa, jest to równoważne:

$$\operatorname{argmax}_{c \in \text{kandydaci}} P(c) P(w|c) / P(w)$$

Ponieważ $P(w)$ jest taka sama dla każdej możliwej poprawki c , możemy ją wyłączyć, uzyskując:

$$\operatorname{argmax}_{c \in \text{kandydaci}} P(c) P(w|c)$$

Możemy z tego wyrażenia wyodrębnić cztery części:

1. Mechanizm wyboru: argmax – wybieramy kandydata o najwyższym łącznym prawdopodobieństwie.
2. Model kandydata: $c \in \text{kandydaci}$ – to mówi nam, które korekty, c , należy rozważyć.
3. Model językowy: $P(c)$ – prawdopodobieństwo, że c występuje jako słowo w tekście angielskim. Na przykład wystąpienia "the" stanowią około 7% tekstu angielskiego, więc powinniśmy mieć $P(\text{the}) = 0,07$.
4. Model błędu: $P(w|c)$ – prawdopodobieństwo, że w zostanie wpisane w tekście, gdy autor miał na myśli c . Na przykład $P(\text{teh}|\text{the})$ jest stosunkowo wysokie, ale $P(\text{theeexyz}|\text{the})$ byłoby bardzo niskie.

4. Część praktyczna

4.1. Implementacja

Opisane w części teoretycznej mechanizmy mają odzwierciedlenie w implementacji:

1. Mechanizm selekcji

- W funkcji `correction()` sortujemy możliwe korekcje dla danego słowa.
- Kluczem sortowania jest `P`, czyli prawdopodobieństwo wystąpienia danego słowa.
- Zwracamy liczbę korekcji żadaną przez użytkownika lub domyślnie – jedną korekcji.

```
def P(self, word, N=sum(WORDS.values())):
    """Probability of `word`."""
    return SpellCorrector.WORDS[word] / N

def correction(self, word, num_corrections=1):
    """Most probable spelling correction(s) for word. Number of corrections can be specified."""
    sorted_candidates = sorted(self.candidates(word), key=self.P, reverse=True)
    return sorted_candidates[:num_corrections]
```

2. Mechanizm kandydata

- Funkcja `edits1()` zwraca słowa, które mogą być stworzone po wykonaniu tylko JEDNEJ operacji zmiany na słowie początkowym - usunięciu jednej litery, zamiana dwóch sąsiednich liter, zamiana jednej litery na inną, dodanie jednej litery.
- Funkcja `edits2()` zwraca słowa, które mogą powstać po dwukrotnym wykonaniu funkcji `edits1()` na słowie użytkownika.
- Funkcja `edits1()` zwraca ogromną liczbę słów: $54n+25$ dla słów o długości n . Dlatego w funkcji `known()` zostawiamy tylko rzeczywiste słowa, które występują w słowniku.

```
def edits1(self, word):
    """All edits that are one edit away from `word`."""
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [L + R[1:] for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
    replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
    inserts = [L + c + R for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(self, word):
    """All edits that are two edits away from `word`."""
    return {e2 for e1 in self.edits1(word) for e2 in self.edits1(e1)}

def known(self, words):
    """The subset of `words` that appear in the dictionary of WORDS."""
    return set(w for w in words if w in SpellCorrector.WORDS)
```

3. Mechanizm języka

- Plik `big.txt` zawiera pulę miliona słów z książki z Project Gutenberg, najczęstszych słów z Wiktionary oraz słów z British National Corpus.
- Funkcja pomocnicza `words()` rozбивa ten plik na pojedyncze słowa. Liczniki wystąpień są zliczane i przechowywane w zmiennej `WORDS`.

- W funkcji $P()$ szacujemy prawdopodobieństwo danego słowa na podstawie sumy zliczonych słów.

```
class WordHelper:

    @staticmethod
    def words(text):
        return re.findall(r'\w+', text.lower())

class SpellCorrector:
    WORDS = Counter(WordHelper.words(open(os.path.join(__location__, 'data/big.txt')).read()))

    def P(self, word, N=sum(WORDS.values())):
        """Probability of `word`."""
        return SpellCorrector.WORDS[word] / N
```

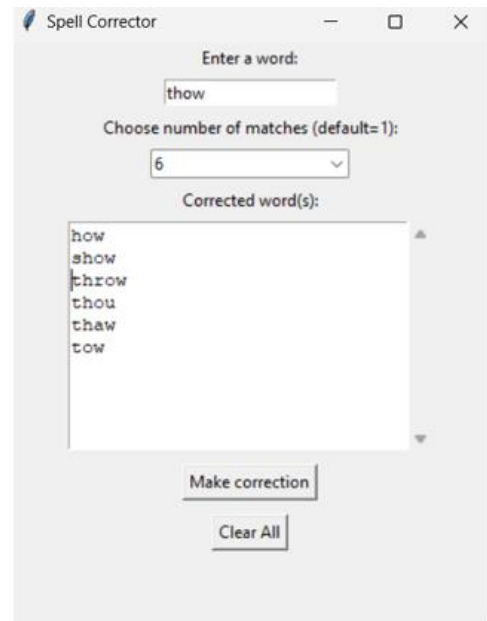
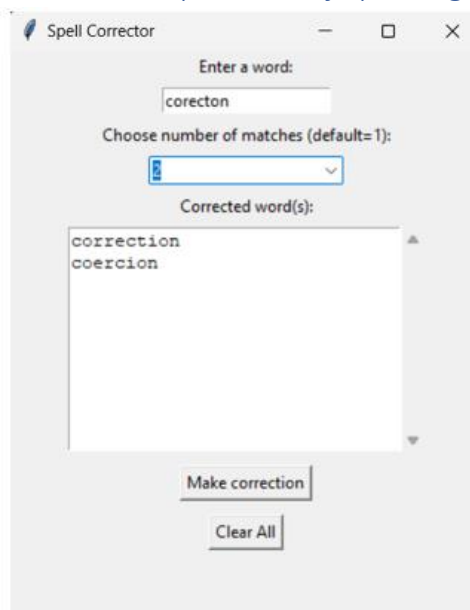
4. Model błędu

- Uproszczony model błędu generuj pierwszą niepustą listę prawdopodobnych korekcy w kolejności:
 - a. Słowo oryginalne, jeśli występuje w słowniku; w przeciwnym razie:
 - b. Lista słów powstałych po JEDNEJ edycji, jeśli występują w słowniku; w przeciwnym razie:
 - c. Lista słów powstałych po DWÓCH edycjach, jeśli występują w słowniku; w przeciwnym razie:
 - d. Oryginalne słowo, nawet jeśli nie występuje w słowniku.

```
def correction(self, word, num_corrections=1):
    """Most probable spelling correction(s) for word. Number of corrections can be specified."""
    sorted_candidates = sorted(self.candidates(word), key=self.P, reverse=True)
    return sorted_candidates[:num_corrections]

def candidates(self, word):
    """Generate possible spelling corrections for word."""
    return (
        self.known([word]) or
        self.known(self.edits1(word)) or
        self.known(self.edits2(word)) or
        [word]
    )
```

4.2. Implementacja prostego GUI



4.3. Ocena działania

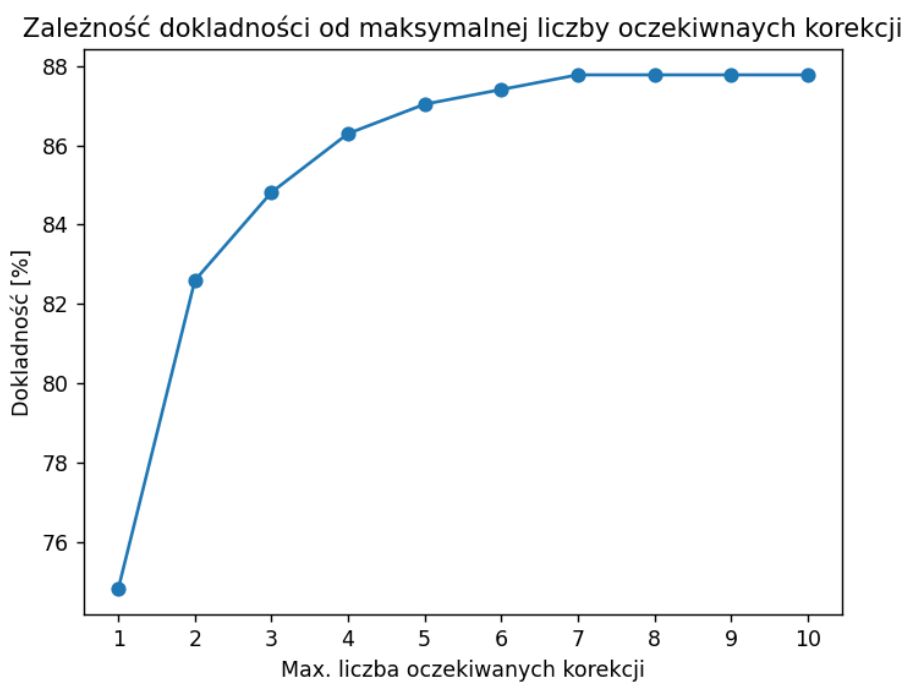
Przeprowadzono test na danych wczytanych z pliku. Fragment pliku widoczny jest poniżej.

```
contented: contented contende contended contentid  
beginning: begining  
problem: problem proble promblem proplen  
driven: dirven  
ecstasy: exstacy ecstacy  
juice: guic juce jucie juise juse
```

Pierwsze słowo w linii to poprawne, oczekiwane słowo. Testowaliśmy działanie korektora na podstawie kolejnych słów w linii, które zawierały błędy. Zbiór testowy zawierał 270 poprawnych słów (270 linii). Dla każdego z nich podane było kilka błędnych.

```
class Spelltest:  
    def __init__(self):  
        """Parse 'right: wrong1 wrong2' lines into [('right', 'wrong1'), ('right', 'wrong2')] pairs."""  
        self.spell_set = [(right, wrong) for (right, wrongs) in (line.split(':')  
                                                                for line in open('data/testset.txt'))  
                          for wrong in wrongs.split()]]  
  
    def spelltest(self):  
        """Run correction(wrong) on all (right, wrong) pairs; report results."""  
        n = len(self.spell_set)  
        sc = SpellCorrector()  
  
        x_values, y_values = [], []  
        for i in range(1, 11):  
            good = 0  
            for right, wrong in self.spell_set:  
                corrections = sc.correction(wrong, num_corrections=i)  
                good += (right in corrections)  
            print('{:.0%} of {} correct for num_correction={}'.format(good / n, n, i))  
            accuracy_percentage = good / n * 100  
            x_values.append(i)  
            y_values.append(accuracy_percentage)  
        self.plot(x_values, y_values)
```


Na podstawie zebranych wyników sporządzono poniższy wykres:



Najlepsze efekty osiągnięto, gdy parameter liczby korekcji ustawiono na ≥ 7 słów. Wtedy wynik wyniósł 88%. Stworzony korektor jest dość skuteczny nawet przy parametrze liczby korekcji = 1. Wtedy dokładność wynosi 75%.

Podsumowanie

Zrealizowano cel opisywanego projektu, stworzono korektor pisowni dla języka angielskiego. W prostym interfejsie użytkownika istnieje możliwość podania oczekiwanej liczby korekcji.

Bibliografia

1. <https://www.norvig.com/spell-correct.html>
2. https://pl.wikipedia.org/wiki/Twierdzenie_Bayesa