

Jakub Szpilowski

Maciej Szudy

Kamil Szpakowski

Jak napisać spell corrector?

1. Wstęp

Korektor pisowni (ang. *spell corrector*) jest narzędziem do identyfikacji i korekcji błędów ortograficznych w tekście, może też służyć do sprawdzania interpunkcji. Obecnie takie oprogramowanie jest wbudowane w usługi, takie jak edytory tekstu, słowniki elektroniczny czy w wyszukiwarkach internetowych.

W poniższym raporcie opiszemy jak stworzyć podstawowy korektor pisowni oparty o techniki NLP (*Natural Language Processing*), który pozwala identyfikować literówki i sugeruje użytkownikowi możliwą poprawkę. Omawiany model działa na podstawie obliczenia prawdopodobieństwa wszystkich kandydatów do naniesienia poprawki i po jego obliczeniu wybierany jest kandydat z największym prawdopodobieństwem, który zastępuje błędnie zapisany wyraz.

2. Opis działania

Podstawą omawianego korektora jest wykorzystanie modelu statystycznego, który opiera się na obliczaniu częstotliwości występowania słów w korpusie, czyli dużym zbiorze danych. Wykorzystanie odpowiednio dużego zbioru danych pozwala ocenić, jakie słowa w danym języku mają największe prawdopodobieństwo wystąpienia.

Aby szczegółowo omówić działanie spell correctora zaczniemy od teorii prawdopodobieństwa. Na początek poniższy wzór (na razie nie musimy znać poszczególnych składowych, ponieważ wyrażenie znacznie się upraszcza):

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

zgodnie z twierdzeniem Bayesa jest równoważne:

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

gdzie $\mathbf{P(w)}$ oznacza prawdopodobieństwo wystąpienia słowa \mathbf{w} podanego przez użytkownika. $P(w)$ dla każdego słowa jest takie samo, można je pominąć co daje końcowe wyrażenie

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

gdzie \mathbf{c} to kandydat to poprawienia błędnego wyrazu.

Powyższe wyrażenie możemy rozbić na cztery części:

Mechanizm selekcji	<ul style="list-style-type: none">• argmax - znajdź tego kandydata c spośród wszystkich kandydatów, dla którego prawdopodobieństwo znalezienia odpowiedniej poprawki jest największe
Model kandydata	<ul style="list-style-type: none">• Odpowiada za informację, które poprawki kandydatów c należy wziąć pod uwagę $c \in \text{candidates}$
Model językowy	<ul style="list-style-type: none">• $P(c)$ - prawdopodobieństwo z jakim poprawka znajduje się w tekście angielskim. Na przykład $P(\text{„the”}) = 0.07$, oznacza, że słowo „the” stanowi statystycznie 7% dowolnego tekstu napisanego w języku angielskim.
Model błędu	<ul style="list-style-type: none">• $P(w c)$ - prawdopodobieństwo, że jeśli autor wpisał słowo w to miał na myśli c. Na przykład $P(\text{„teh”} \text{„the”})$ jest relatywnie wysokie, ale $P(\text{„theeabc”} \text{„the”})$ byłoby już niskie.

Omówimy teraz poszczególne części.

2.1. Mechanizm selekcji

Ze wszystkich możliwych kandydatów do poprawki szukamy takiego słówka c , którego prawdopodobieństwo występowania w słowniku jest największe.

Mechanizmem selekcji w kluczowym wyrażeniu dla spell correctora jest **argmax**, (w języku Python, *argmax* jest tożsamy z użyciem funkcji **max** z argumentem *key*, gdzie do argumentu *key* przypisywana jest zdefiniowana przez programistę metoda).

Przykładowo, w języku angielskim słowo **the** ma największą częstotliwość występowania w tekście.

```
>>> WORDS.most_common(10)
[('the', 79808),
 ('of', 40024),
 ('and', 38311),
 ('to', 28765),
 ...
]

>>> max(WORDS, key=P)
'the'
```

2.2. Model kandydata

Kolejnym krokiem jest generowanie kandydatów do poprawki, ze słów występujących w korpusie. Korzystamy tutaj z tzw. **odległości edycyjnej**, która mierzy jak bardzo dwa wyrazy różnią się od siebie ilością operacji potrzebnych, żeby jedno słowo przekształcić w drugie. Omawiany przez nas model generuje kandydatów w odległości edycyjnej jeden oraz dwa, czyli kandydaci powstałi po jednym lub dwóch modyfikacjach.

Na modyfikacje składają się:

- Przetawienie kolejności dwóch sąsiadujących liter
- Usunięcie litery
- Dodanie litery
- Zamiana litery na inną

Następnie tak wygenerowanych kandydatów należy przefiltrować, aby otrzymać słowa, które są znane to znaczy występują w korpusie. Tak otrzymuje się kandydatów do poprawki.

W Pythonie, wyrazy o odległości edycyjnej równej jeden można wygenerować na przykład w taki sposób:

```
def edits1(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)
```

Filtrowanie nieznanego słów:

```
def known(words):
    return set(w for w in words if w in WORDS)

>>> known(edits1('something'))
{'something', 'soothing'}
```

Wygenerowanie słów z dwoma modyfikacjami wykonuje się, wywołując funkcję **edits1** na każdym słowie ze zbioru po jednej modyfikacji.

2.3. Model językowy

Po wygenerowaniu listy kandydatów do poprawki, należy obliczyć prawdopodobieństwo występowania każdego z nich. Następnie model wybiera słowo z najwyższym prawdopodobieństwem jako sugerowaną poprawkę. Prawdopodobieństwo kandydata oblicza się z prostego wzoru:

$$P(word) = \frac{\text{liczba wystąpień słowa}}{\text{ilość słów w korpusie}}$$

W programie można to wykonać w ten sposób:

```
def words(text):  
    return re.findall(r'\w+', text.lower())  
  
WORDS = Counter(words(open('big.txt').read()))  
  
def P(word, N=sum(WORDS.values())):  
    return WORDS[word] / N
```

Wczytujemy plik (czyli nasz korpus), dzielimy go na wyrazy i zapisujemy w słowniku **WORDS** wyrazy i ilość ich wystąpień.

Funkcja **P** zwraca prawdopodobieństwo dla danego słowa według wzoru.

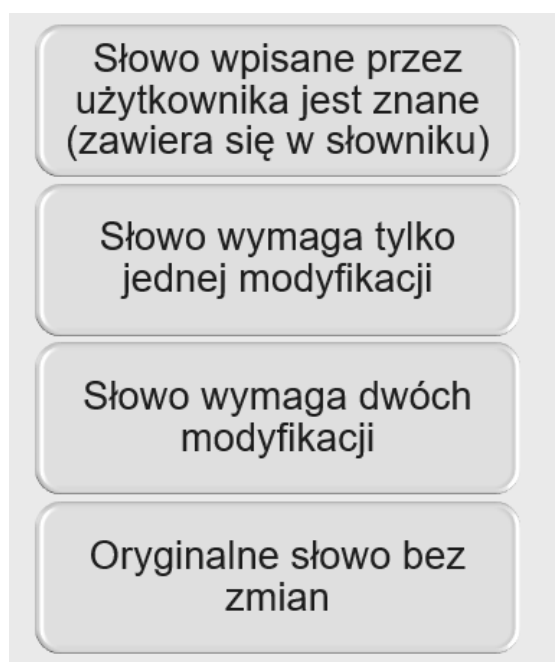
2.4. Model błędu

Na tym etapie dokonujemy wyboru najbardziej pasującego kandydata na poprawkę słowa wpisanego przez użytkownika. Zakładamy, że wszystkie znane słowa o odległości edycji 1 są nieskończenie bardziej prawdopodobne niż znane słowa o odległości edycji 2, i nieskończenie mniej prawdopodobne niż znane słowo o odległości edycji 0. Wynika z tego, że czynnik $P(w|c)$ możemy zastąpić priorytetem przypisanym do danego kandydata.

```
def correction(word):  
    return max(candidates(word), key=P)  
  
def candidates(word):  
    return known([word]) or known(edits1(word)) or known(edits2(word)) or [word]
```

Metoda **candidates(word)** tworzy więc listę ewentualnych kandydatów do poprawki.

Kandydaci są ułożeni malejąco względem priorytetu:



3. Ocena modelu i możliwe poprawki

Omawiany przez nas model jest bardzo prostym modelem korektora pisowni, który dobrze radzi sobie z podstawowymi poprawkami błędów ortograficznych w tekście. Jednak taki model jest w pewnym sensie ograniczony i wymaga poprawek.

Przykładowe obszary rozwoju modelu:

- **Sprawdzanie pisowni zależnie od kontekstu**

Model analizuje tylko pojedyncze słowa i nie bierze pod uwagę kontekstu wyrazu w zdaniu, co może prowadzić do niepoprawnie sugerowanych poprawek. Ponieważ poprawność pisowni zależy w głównej mierze od kontekstu należało by sprawdzać inne słowa w całym zdaniu, aby ustalić odpowiednią poprawkę.

- **Modele do poprawiania błędów wewnątrz słów oparte o deep learning**

Modele wykorzystujące uczenie maszynowe, w szczególności rekurencyjne sieci neuronowe (RNN) czy transformery (takie jak BERT, GPT) mogą być używane w korekcji pisowni. Takie modele są w stanie identyfikować skomplikowane wzorce i zależności w danych, co można wykorzystać do sprawdzania pisowni zarówno bez kontekstu jak i pisowni kontekstowej.

- **Algorytmy fonetyczne**

Algorytmy fonetyczne kodują słowa na podstawie ich fonetycznego podobieństwa, dlatego mogą być wykorzystane do sugerowania poprawek, które brzmią podobnie do błędnie napisanego słowa.

- **Modele oparte o sylaby lub sekwencje**

Możemy także stworzyć model modelu językowego oparty na składnikach słów, na sylabach przyrostkach, albo opierać go na sekwencjach znaków, takich jak powszechne sekwencje 2-, 3- i 4-literowe.

Modele sekwencja-do-sekwencji (seq2seq) są typem modelu, który może przekształcić jedną sekwencję (np. zdanie z błędami pisowni) w inną sekwencję (np. to samo zdanie z poprawkami).

- **Implementacja rozwiązania w języku kompilowanym**

Przedstawiony przez nas przykład korzysta z języka Python, który jest interpretowalny. Stworzenie modelu w języku kompilowanym pozwoliłoby na zwiększenie szybkości programu, ponieważ moglibyśmy buforować wyniki, aby uniknąć ich wielokrotnego przetwarzania.

- **Dodatkowo warto rozważyć wykorzystanie rekurencyjnych sieci neuronowych oraz transformerów, których wysoka efektywność została udowodniona w praktyce.**

4. Podsumowanie

Przedstawiony przez nas model spell correctora opiera się na prostym statystycznym modelu języka. Ocenia on prawdopodobieństwo słów na podstawie częstotliwości ich występowania w korpusie. Takie rozwiązanie jest proste w implementacji i dobrze radzi sobie z poprawianiem błędów ortograficznych, jednak ma pewne ograniczenia.

Główną wadą takiego modelu jest prosty model błędu oraz to, że nie uwzględnia kontekstu zdania przez co może sugerować niepoprawne (w sensie użyte w niewłaściwym kontekście) wyrazy.

Przedstawiony przez nas korektor pisowni, mimo swojej prostoty jest dobrym punktem startowym do dalszego rozwoju. Rozszerzenie modelu o kontekstualne sprawdzanie pisowni, zastosowanie technik uczenia maszynowego lub różnych algorytmów fonetycznych, mogłoby znacznie usprawnić działanie takiego korektora, sprawiając że model byłby bardziej precyzyjny i skuteczny.

5. Bibliografia

<https://www.norvig.com/spell-correct.html>

https://en.wikipedia.org/wiki/Spell_checker

https://en.wikipedia.org/wiki/Bayes%27_theorem