

Jak napisać spell corrector?

Jakub Szpilowski, Maciej Szudy i Kamil Szpakowski

Czym jest i jak działa spell corrector wysokopoziomowo?

Spell corrector jest odpowiedzialny za automatyczne poprawianie literówek w tekstach

Działa na podstawie obliczenia prawdopodobieństwa wszystkich kandydatów do naniesienia poprawki

Po obliczeniu prawdopodobieństwa wybierany jest kandydat z największym prawdopodobieństwem

Błędnie zapisany wyraz jest zastępowany wybranym kandydatem

Prawdopodobieństwo - teoria

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

Zgodnie z twierdzeniem Bayesa jest to równoważne:

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

Ponieważ $P(w)$ jest takie samo dla każdego możliwego kandydata c , możemy je wyłączyć przed nawias, co daje:

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

c - kandydat do poprawienia błędnego wyrazu

w - analizowany wyraz

Końcowe wyrażenie

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

możemy rozbić na cztery części:

Mechanizm selekcji

- ***argmax*** - znajdź tego kandydata c spośród wszystkich kandydatów, dla którego prawdopodobieństwo znalezienia odpowiedniej poprawki jest największe

Model kandydata

- Odpowiada za informację, które poprawki kandydatów c należy wziąć pod uwagę $c \in \text{candidates}$

Model językowy

- **$P(c)$** - 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

- **$P(w|c)$** - 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.

Na końcu prawdopodobieństwo, że **c** rzeczywiście wprowadza poprawkę do **w** możemy zapisać jako:

$$\mathbf{P(c)P(w \mid c)}$$

Czyli prawdopodobieństwo, że **c** występuje w tekście angielskim (częstotliwość) i
prawdopodobieństwo, że autor miał na myśli **c** pisząc **w**

Mechanizm selekcji

Mechanizmem selekcji w kluczowym wyrażeniu dla spell correctora jest *argmax*.

argmax w Pythonie jest tożsame z użyciem funkcji **max** z argumentem *key*.

Do argumentu *key* przypisywana jest zdefiniowana przez programistę metoda.

Przykład:

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

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

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

c - kandydat do poprawienia błędnego wyrazu

w - analizowany wyraz

Generowanie kandydatów

Kolejny krok to generowanie kandydatów do zastąpienia ciągu znaków wpisanego przez użytkownika. Możemy to zrobić na przykład w ten 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)
```

Utworzenie wszystkich możliwych kombinacji wpisanego słowa po jednej modyfikacji. Na modyfikację składa się: przestawienie kolejności dwóch sąsiadujących liter, usunięcie litery, dodanie litery lub zamiana litery na inną.

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

c - kandydat do poprawienia błędnego wyrazu

w - analizowany wyraz

Generowanie kandydatów – ciąg dalszy

Następnie z wygenerowanych kombinacji wybieramy słowa dostępne w słowniku, które stają się kandydatami do poprawki.

```
def known(words):  
    return set(w for w in words if w in WORDS)  
  
>>> known(edits1('something'))  
{'something', 'soothing'}
```


Generowanie kandydatów – ciąg dalszy

Możemy wygenerować kolejną porcję kandydatów zwiększając ich zbiór. Tym razem stosujemy modyfikację podwójną. Wykonujemy to poprzez powtórzenie poprzedniej modyfikacji na wcześniej wygenerowanym zbiorze.

```
def edits2(word):  
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))  
  
>>> len(set(edits2('something')))  
90902  
  
>>> known(edits2('something'))  
{'seething', 'smoothing', 'something', 'soothing'}  
  
>>> known(edits2('somthing'))  
{'loathing', 'nothing', 'scathing', 'seething', 'smoothing', 'something', 'soothing', 'sorting'}
```

Ponieważ znacznie zwiększyliśmy liczbę kombinacji wpisanego przez użytkownika słowa zyskaliśmy również więcej kandydatów do poprawki wyrazu.

Model językowy

Prawdopodobieństwo, że dane słowo występuje w tekście, czyli $P(\text{word})$ możemy policzyć biorąc plik tekstowy z milionem słów i zliczyć ilość wystąpień interesującego nas słowa, a następnie podzielić ją przez całkowitą liczbę słów w pliku.

$$P(\text{word}) = \frac{\text{liczba_wystąpień_słowa}}{\text{liczba_wszystkich_słów_w_pliku}}$$

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

c - kandydat do poprawienia błędnego wyrazu

w - analizowany wyraz

Model językowy – ciąg dalszy

W kodzie Pythona wygląda to 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
```

WORDS - słownik
przechowujący
wyrazy i ich liczbę
wystąpień



Funkcja words dzieli
plik tekstowy na
pojedyncze słowa



Funkcja P zwraca
omawiane
prawdopodobieństwo

Model błędu

Na tym etapie programu 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.

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

c - kandydat do poprawienia błędnego wyrazu

w - analizowany wyraz

Model błędu – ciąg dalszy

```
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.

Model błędu – ciąg dalszy

Priorytet dla danego kandydata jest największy w następującej kolejności:

Słowo wpisane przez
użytkownika jest znane
(zawiera się w słowniku)

Słowo wymaga tylko
jednej modyfikacji

Słowo wymaga dwóch
modyfikacji

Oryginalne słowo bez
zmian

Ocena modelu

Ocena wydajności modelu opiera się na dwóch zestawach danych: walidacyjnym oraz testowym.

Walidacyjny zestaw służył do dopracowywania modelu podczas jego rozwijania, testowy zestaw zawiera dane na podstawie których model nie był rozwijany.

Za pomocą zestawu testowego możemy przetestować jego zdolności generalizacji.

```
def unit_tests():
    assert correction('speling') == 'spelling'           # insert
    assert correction('korrektud') == 'corrected'         # replace 2
    assert correction('bycycle') == 'bicycle'             # replace
    assert correction('inconvient') == 'inconvenient'     # insert 2
    assert correction('arrainged') == 'arranged'          # delete
    assert correction('peotry') == 'poetry'               # transpose
    assert correction('peotryy') == 'poetry'              # transpose + delete
    assert correction('word') == 'word'                   # known
    assert correction('quintessential') == 'quintessential' # unknown
    assert words('This is a TEST.') == ['this', 'is', 'a', 'test']
    assert Counter(words('This is a test. 123; A TEST this is. ')) == (
        Counter({'123': 1, 'a': 2, 'is': 2, 'test': 2, 'this': 2}))
    assert len(WORDS) == 32192
    assert sum(WORDS.values()) == 1115504
    assert WORDS.most_common(10) == [
        ('the', 79808),
        ('of', 40024),
        ('and', 38311),
        ('to', 28765),
        ('in', 22020),
        ('a', 21124),
        ('that', 12512),
        ('he', 12401),
        ('was', 11410),
        ('it', 10681)]
    assert WORDS['the'] == 79808
    assert P('quintessential') == 0
    assert 0.07 < P('the') < 0.08
    return 'unit_tests pass'
```


Ocena modelu

Ocena wydajności modelu opiera się na dwóch zestawach danych: walidacyjnym oraz testowym.

Walidacyjny zestaw służył do dopracowywania modelu podczas jego rozwijania, testowy zestaw zawiera dane na podstawie których model nie był rozwijany.

Za pomocą zestawu testowego możemy przetestować jego zdolności generalizacji.

```
def spelltest(tests, verbose=False):
    "Run correction(wrong) on all (right, wrong) pairs; report results."
    import time
    start = time.clock()
    good, unknown = 0, 0
    n = len(tests)
    for right, wrong in tests:
        w = correction(wrong)
        good += (w == right)
        if w != right:
            unknown += (right not in WORDS)
            if verbose:
                print('correction({}) => {} ({}); expected {} ({})'
                      .format(wrong, w, WORDS[w], right, WORDS[right]))
    dt = time.clock() - start
    print('{:.0%} of {} correct ({:.0%} unknown) at {:.0f} words per second '
          .format(good / n, n, unknown / n, n / dt))
```

```
def Testset(lines):
    "Parse 'right: wrong1 wrong2' lines into [('right', 'wrong1'), ('right', 'wrong2')] pairs."
    return [(right, wrong)
            for (right, wrongs) in (line.split(':') for line in lines)
            for wrong in wrongs.split()]
```

Obszary do dalszego rozwoju

- Sprawdzanie pisowni zależnie od kontekstu
- Modele do poprawiania błędów wewnątrz słów oparte o deep learning
- Algorytmy fonetyczne
- Modele sekwencja do sekwencji (seq-2-seq)
- Zaimplementowanie rozwiązania w języku kompilowanym – zwiększenie szybkości programu
- W szczególności warto rozważyć wykorzystanie rekurencyjnych sieci neuronowych oraz transformerów, których wysoka efektywność została udowodniona w praktyce.

DZIĘKUJEMY ZA UWAGĘ

Źródła:

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