Spelling Corrector

Projekt PJN

Wykonał: Jan Kozieł

Cele projektu

Cele projektu

- **Ogólny cel**: utworzenie prostej aplikacji poprawiającej wyrazy wpisywane przez użytkownika

Cele projektu

- **Ogólny cel**: utworzenie prostej aplikacji poprawiającej wyrazy wpisywane przez użytkownika
  - predykcja słowa w zależności od kontekstu

Cele projektu

- **Ogólny cel**: utworzenie prostej aplikacji poprawiającej wyrazy wpisywane przez użytkownika
  - predykcja słowa w zależności od kontekstu
  - sugerowanie jak najbardziej pasujących poprawek

Cele projektu

- **Ogólny cel**: utworzenie prostej aplikacji poprawiającej wyrazy wpisywane przez użytkownika
  - predykcja słowa w zależności od kontekstu
  - sugerowanie jak najbardziej pasujących poprawek
  - danie użytkownikowi ostatecznej kontroli nad tym jak słowo zostanie poprawione

Cele projektu

- **Ogólny cel**: utworzenie prostej aplikacji poprawiającej wyrazy wpisywane przez użytkownika
  - predykcja słowa w zależności od kontekstu
  - sugerowanie jak najbardziej pasujących poprawek
  - danie użytkownikowi ostatecznej kontroli nad tym jak słowo zostanie poprawione
- **Cel dodatkowy**: dodanie funkcjonalności sugerowania kolejnych słów

1. Predykcja słów w zależności od kontekstu

1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)

1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)
- n-gram: sekwencja n kolejnych (zazwyczaj) wyrazów w zdaniu

1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)
- n-gram: sekwencja n kolejnych (zazwyczaj) wyrazów w zdaniu
- “Ala ma dwa bure koty” - “Ala ma dwa”, “ma dwa bure”, “dwa bure koty”

1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)
- n-gram: sekwencja n kolejnych (zazwyczaj) wyrazów w zdaniu
- “Ala ma dwa bure koty” - “Ala ma dwa”, “ma dwa bure”, “dwa bure koty”
    - ale także “</s> </s> Ala”, “</s> Ala ma”, “bure koty </s>”, “koty </s> </s>”

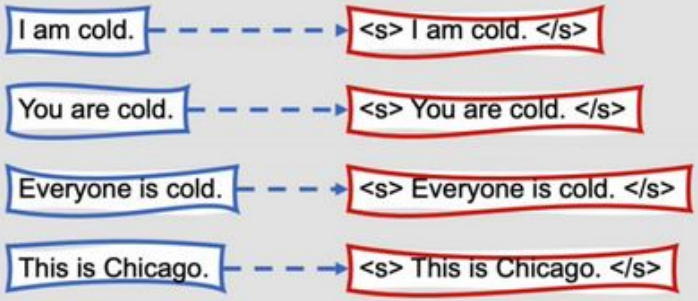
1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)
- n-gram: sekwencja n kolejnych (zazwyczaj) wyrazów w zdaniu
- “Ala ma dwa bure koty” - “Ala ma dwa”, “ma dwa bure”, “dwa bure koty”
    - ale także “</s> </s> Ala”, “</s> Ala ma”, “bure koty </s>”, “koty </s> </s>”
- Model (MLE) liczy prawdopodobieństwo w zależności od n-gramów utworzonych z korpusu

1. Predykcja słów w zależności od kontekstu

- stworzony został n-gramowy model języka (3)
- n-gram: sekwencja n kolejnych (zazwyczaj) wyrazów w zdaniu
- “Ala ma dwa bure koty” - “Ala ma dwa”, “ma dwa bure”, “dwa bure koty”
  - ale także “</s> </s> Ala”, “</s> Ala ma”, “bure koty </s>”, “koty </s> </s>”
- Model (MLE) liczy prawdopodobieństwo w zależności od n-gramów utworzonych z korpusu
  - 900 000 tweetów

Example: Maximum Likelihood Estimation



Bigram	Freq.
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Unigram	Freq.
<s>	4
I	1
am	1
cold.	3
…	…
Chicago.	1
</s>	4

P("I" | "<s>") = C("<s> I") / C("<s>") = 1 / 4 = 0.25

P("</s>" | "cold.") = C("cold. </s>") / C("cold.") = 3 / 3 = 1.00

2. Generowanie poprawek

2. Generowanie poprawek

- Generowanie ciągów znaków niekoniecznie będącymi wyrazami

2. Generowanie poprawek

- Generowanie ciągów znaków niekoniecznie będącymi wyrazami
- Odfiltrowanie niepoprawnych “poprawek”

2. Generowanie poprawek

- Generowanie ciągów znaków niekoniecznie będącymi wyrazami
- Odfiltrowanie niepoprawnych “poprawek”
    - na podstawie innego korpusu, 20 000 najpopularniejszych słów w języku angielskim

2. Generowanie poprawek

- Generowanie ciągów znaków niekoniecznie będącymi wyrazami
- Odfiltrowanie niepoprawnych “poprawek”
    - na podstawie innego korpusu, 20 000 najpopularniejszych słów w języku angielskim

```
def _generate_edits(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 list(set(deletes + transposes + replaces + inserts))
```

3. Sortowanie sugestii

3. Sortowanie sugestii

- Sprawdzenie czy model zna taką kombinację wyrazów

3. Sortowanie sugestii

- Sprawdzenie czy model zna taką kombinację wyrazów
  - Jeśli nie to ułożenie sugestii wg kolejności występowania w korpusie

3. Sortowanie sugestii

- Sprawdzenie czy model zna taką kombinację wyrazów
  - Jeśli nie to ułożenie sugestii wg kolejności występowania w korpusie
  - Jeśli tak to ułożenie sugestii wg prawdopodobieństwa

3. Sortowanie sugestii

- Sprawdzenie czy model zna taką kombinację wyrazów
  - Jeśli nie to ułożenie sugestii wg kolejności występowania w korpusie
  - Jeśli tak to ułożenie sugestii wg prawdopodobieństwa
- Skalowanie prawdopodobieństwa w zależności od jego pochodzenia

4. Przewidywanie kolejnych słów

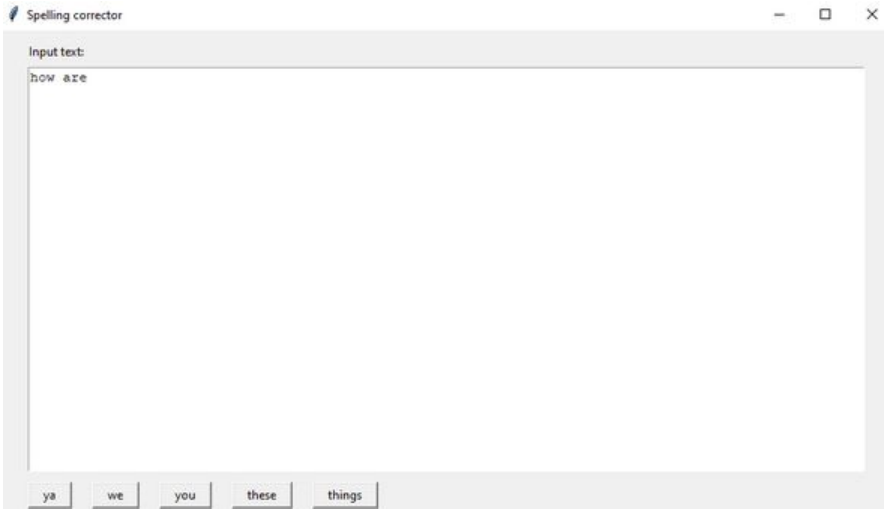
4. Przewidywanie kolejnych słów

- Wykorzystanie istniejącego modelu do wyświetlania sugerowanych słów

4. Przewidywanie kolejnych słów

- Wykorzystanie istniejącego modelu do wyświetlania sugerowanych słów
  - ta sama funkcjonalność, jednak zamiast prawdopodobieństwa kombinacji, wyciągane jest odpowiednie słowo


Spelling corrector
Input text:
how are yuo
you
your
ya
to
our
us
but
do
no
up


Spelling corrector
Input text:
how are
ya
we
you
these
things