Generowanie Tekstu

Łukasz Ruzicki 271399

czwartek, 18 listopada 2021

1 Wstep

Projekt ma na celu wykonanie programu, który bedzie generował losowy tekst o określonej długości z podanego słowa. Słowo te stanowi ziarno do generowania tekstu. Program może mieć zastosowanie przy wypełnianiu pól w aplikacjach, dla przykładu w tworzeniu szablonów stron internetowych.

2 Niezbedne do programu

Program został napisany w Pythonie przy użyciu biblioteki Natural Language Toolkit, która umożliwia przetwarzanie i analize danych wykonanych przez człowieka. Ponadto dostarcza teksty na których może operować program. Oprócz NLTK użyty został moduł Random, aby generowane teksty były losowe.

3 Dane

Dane na których pracowano przy wykonaniu programu pochodziły z NLTK, która dostarcza 9 tekstów:

- text1: Moby Dick by Herman Melville 1851
- text2: Sense and Sensibility by Jane Austen 1811
- text3: The Book of Genesis
- text4: Inaugural Address Corpus
- text5: Chat Corpus
- text6: Monty Python and the Holy Grail
- text7: Wall Street Journal
- text8: Personals Corpus
- text9: The Man Who Was Thursday by G . K . Chesterton 1908

4 Teoria

Ngramy: W swojej budownie to nic innego jak tylko N kolejnych wyrazów wyjetych z jakiegoś zdania, lub tekstu.

Przykład: Dla zdania "This is red cat", n-gramy o długości 2 to:

- 'This', 'is'
- 'is', 'red'
- 'red', 'cat'

5 Założenia

Program pobiera od użytkownika przybliżona liczbe słów oraz ziarno do generowania tekstu. Z zaimportowanych tekstów tworzy bigramy i trigramy. Nastepnie sa one sortowane zgodnie z czestotliwościa ich wystepowania. Dla podanego słowa program dobiera najczestsze bigramy i trigramy w których podane ziarno wystepuje na poczatku tego wyrażenia. Nastepnie losowana jest liczba na której podstawie wybierany jest n-gram ze zbioru 10 n-gramów. Ostatnie słowo n-gramu staje sie nowym ziarnem a pozostałe słowa sa dodawane do generowanego zdania. Program działa dopóki nie przekroczy podanej długości.

6 Testy

Importowane bibliotek i utworzenie zbioru bigramów i trigramów z których generowane beda teksty.

```
import nltk
from nltk.collocations import *
from nltk.book import *
from random import randrange #, uniform

list_text = [text1, text2, text3, text4, text5, text6, text7, text8, text9]
bigramy = []
for t1 in list_text:
    k = [x.lower() for x in t1.tokens if x.isalpha()]


trigramy=[]
for t1 in list_text:
    k = [x.lower() for x in t1.tokens if x.isalpha()]
    trigramy += sorted(set(list(nltk.trigrams(k))))

f1 = FreqDist(list(nltk.bigrams(t1)))
f2 = FreqDist(list(nltk.trigrams(t1)))
```

Pierwsze wersja generatora tekstu. Dla podanego ziarna wyszukiwane sa bigramy. Jeżeli pierwsze słowo bigramu pokrywa sie z naszym ziarne to jest on wybierany. Jeżeli jest to pierwsze słowo dodane do wyniku to bedzie ono zapisane duża litera.

```
def znajdz(ziarno):
    for w in f1:
        if w[0]==ziarno and w[1].isalpha():
            if len(wynik)==0:
                wynik.append(ziarno.capitalize())
            else:
                wynik.append(ziarno)
            return w[1]
    return "I"


ziarno = "Man"
wynik = []

while(len(wynik)<30):
    ziarno = znajdz(ziarno)
s = ""
for i in wynik:
    s+=i+" "
print(s)

#generuje kilka zdań
# ziarno = "Fish"
zdania = []
wynik = []
total_words=0
word_that_cant_be_at_the_end=['DT','CC','IN','TO']
print(ziarno)
```

Wynik generatora tekstu dla długości 30:

"I am a man who had been a man who had been a man who had been a man who had been a man who had been a man who"

Można zauważyć, że generowany tekst zaczyna sie zapetlać. Ponadto nie jest on losowy. Rozwiazaniem problemu jest sprawdzenie, czy wybrany bigram nie był wcześniej użyty, np. przy ostatnich 30 słowach. Zmniejszy to liczbe zapetleń jednak nie wyeliminuje ich w pełni.

Pojawił sie też problem generowania tekstu dla słów, które nie wystepuja w zaimportowanych tekstach. Rozwiazaniem tego problemu jest utworzenie listy synonimów dla podanego słowa. W przypadku, gdz dla ziarna nie bedzie n-gramu to zostana sprawdzone jego synonimy. Jeżeli synonimy też nie pojawia sie w n-gramach to do programu zostanie wprowadzone "I" jako ziarno poczatkowe.

```
def znajdz2(ziarno):
lista_slow=[]
lista_slow.append(ziarno)
lista_slow+=get_all_word_synonyms(ziarno)
for z in lista_slow:
    for w in f1:
        if w[0]==z and w[1].isalpha():
            if w[1] not in wynik[:1]:
                if len(wynik)==0:
                    wynik.append(z.capitalize())
                else:
                    wynik.append(z)
                return w[1]
    return "I"
```

Problem jaki pojawił sie to, że należy sprawdzić powtórzenia słów we wcześniej utworzonych zdaniach a nie w tworzonym aktualnie. Przykład problemu:

"Into the Professor de Worms hobbled heavily into the Professor de. Worms hobbled heavily into the Professor de I am a man who. Had been a man who had been a man who had been a man. Who had been a man who had been a man who. Had been a man who had been a man who. Had been a man who had been a man who had been. A man who had been. Man who had been a man who had been. A man who had been a man who had been a man who. Had been a man who had been a man who had been."

Generowane teksty można uczynić bardziej czytelnymi poprzez zastosowanie trigramów zamiast bigramów. Dla podanego ziarna beda znajdowane kolejne dwa słowa a trzecie bedzie nowym ziarnem. Aby generowane teksty od tego samego ziarna różniły sie, wybierane jest 10 najczestszych n-gramów dla danego ziarna a nastepnie losowany jest n-gram z tej puli.

```
def znajdz3(ziarno):
lista_slow=[]
lista_slow.append(ziarno)
lista_slow+=get_all_word_synonyms(ziarno)

znalezione_dopasowania=[]
for z in lista_slow:
    for w in f2:
        if w[0]==z and w[1].isalpha() and len(znalezione_dopasowania)<10:
            znalezione_dopasowania.append(w)

    #losujemy ze znalezionych
    d1=len(znalezione_dopasowania)

    if d1>0:
        d2=randrange(0,d1)
```

```
            w=znalezione_dopasowania[d2]
            #dodajemy do wyniku i wybieramy ziarno
            if w[0]==z and w[1].isalpha():
                    if w[1] not in wynik[-30:]:
                        if len(wynik)==0:
                            wynik.append(z.capitalize())
                            return w[1]
                        else:
                            if len(w)==3:
                                if w[0].isalpha(): wynik.append(w[0])
                                if w[1].isalpha(): wynik.append(w[1])
                                if w[2].isalpha(): return w[2]
                                else:
                                    wynik.pop()
                                    return w[1]

                            wynik.append(z)


    return "I"
```

Ponadto tekst można podzielić na zdania o losowej długości. Nowy program sprawdza, czy dane zdanie ma miedzy 6 a 30 słów. Jeżeli tak to dołacza on je do wyniku końcowego. Ponadto sprawdza, czy zdanie nie kończy sie zaimkiem lub spójnikiem, aby nadać realność wygenerowanym zdaniom.

```
    while(len(zdania)<3 or total_words<int(max_slow)):

    if len(wynik)<randrange(6,30):
        ziarno = znajdz3(ziarno)
    else:
        while(nltk.pos_tag(wynik[-1])[0][1] in word_that_cant_be_at_the_end):
            wynik.pop()
        wynik[-1]+="."
        zdania.append(wynik)

        total_words+=len(wynik) #liczymy slowa
        wynik=[]

s=""
for z in zdania:
    for w in z:
        s+=w+" "
print(s)
print("Total words: ", total_words)
```

Dodany został prosty interfejs do pobierania danych od użytkownika, ziarna

oraz minimalnej długości generowanego tekstu.

```
print("Podaj przybliżona liczbe s␣lów generowanego tekstu: ")
max_slow = input()
int(max_slow)
print("Podaj ziarno do generowania tekstu: ")
ziarno = input()
```

Rezultat końcowy dla ziarna: "Man". Liczba słów: 503

"Man Who are these other things have been a dream or of the Marquis was a man. Who may well be in time the red hair and rather. Remarkable noble face is true early Christians were I say. That there was something bulky and then the Colonel Ducroix. Smiled brightly enough as the Christians in this respect to the world he would have. I can behave like those of Saffron Park I father expand into the landscape was. Only tell you the Professor worn I am not to tell. Nobody eats him and his hand across his seat shuddering as such truth. Can see something more or less I can be I tell you are not think that there is only. One glance over his forehead was lifted. The Colonel suggested that it is an artist had fallen upon it was. That he did not a man with one of those emptied hell only for. Me in the dark from the President or the man I tell you are a sort of which he. Cried the Marquis ate casually and conventionally of the President in a man. Of it was a kind of which he could not really. A little man with his large but indecipherable face of the end of him. I can be I am going suddenly stone I say that he is it seemed to have. A moment staggered the Professor was staring. I am a portrait of the Professor de Worms swung. I am going to know what you I can. I am not a pair of the world would. Be with a curious freedom and I I father cultivated I can see something more than the Marquis sprang back of the other. Sense as he spoke in a little vague. I am not a kind of him to the. Professor worn them as a voice that I I tell you father playing fell again into the. Secretary has a motor round and they could have sworn that he said in. I can be I am afraid we have not been wavering across the square beneath. Him like a man was a little. Vague hands rose like a woman has ever be quite suddenly took. Off a matter of his own safety. I am now going suddenly stone passage so that by this time to catch it is not think. That he said in a moment Gregory sprang back. Quite simple that is not a kind I tell you are a pair of it must. Be a matter of all the Marquis de I say that they were still. Fixed and they could almost fancy that he had been wavering across the. End of Saffron Park lay on the river and the end I say. That we are all Syme looked straight into his head out of the Marquis. Casually and at the world is full of shattered sunlight. And sat down once or twice two got into the room with a kind. Of all sorts of it seemed to be a kind of Saffron Park.."

7 Bibliografia

Wykłady "Probabilistyczny model jezyka" przygotowane przez Joanna Jedrzejowicz
Natural Language Processing with Python autorstwa Steven Bird, Ewan Klein, and Edward Loper