MM - Generiranje besedil z Markovskimi verigami

Tine Šubic, Ina Lang, Miha Novak, Zorica Ilievska

30. april 2017

1 Predstavitev problema

Za prvi projekt pri predmetu Matematično modeliranje smo si izbrali temo *Spamgenerator*, pri katerem smo s pomočjo metode markovskih verig napisali generator besedil, ki v grobem oponašajo slog vhodnih podatkov.

Projekt smo implementirali v programskih jezikih Octave in Python, osnovnemu programu pa smo dodali tudi nekaj nadgradenj. Vsa izvorna koda in navodila so na voljo v GitHub repozitoriju, katerega URL je naveden na dnu poročila.

2 Izbira vhodnih besedil

Izbrali smo si naslednja vhodna besedila za primerjanje rezultatov. Zaradi različnih slogov pisanja so bili izhodni podatki močno prepoznavni in v večini primerov precej zabavni. Besedila so dostopna v GitHub repozitoriju in oddani ZIP datoteki, z izjemo knjig (avtorske pravice) in zgodovine Facebook klepetov (zasebnost udeležencev)

1. Prozno besedilo Butalci Frana Milčinskega (15000 besed, 5300 unikatnih)
2. Zbirka filmskih citatov (2900 besed, 992 unikatnih)
3. Drama Hamlet, W. Shakespeare (33000 besed, 6500 unikatnih)
4. Združena vsebina večih Facebook klepetov (1000000 besed, 100000 unikatnih)
5. Zbirka knjig Harry Potter, J.K. Rowling (1.2 mio besed, 49000 unikatnih)

3 Matematično ozadje rešitve

Pri nalogi je bila uporabljena klasična metoda markovskih verig. Unikatne besede, ki jih izluščimo iz izvornega besedila nam predstavljajo množico stanj S (velikosti n):

$$S = set(text) = words_{unique} = \{w_1, w_2, w_3 ... w_n\}$$

S pomočjo te množice sedaj sestavimo prehodno matriko $P^{n \times n}$, katere elementi $P(i, j)$ predstavljajo verjetnost prehoda markovske verige iz stanja i v stanje j:

$$P^{n \times n} = \begin{bmatrix} P(1,1) & P(1,2) & P(1,3) & \dots & P(1,n) \\ P(2,1) & P(2,2) & P(2,3) & \dots & P(2,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P(n,1) & P(n,2) & P(n,3) & \dots & P(n,n) \end{bmatrix}$$

Pri uporabi matrike si najprej izberemo psevdo-naključno začetno stanje, nato pa z verjetnostjo $P(x0, j)$ preidemo v naslednje stanje j, ki nam v naslednji iteraciji predstavlja novo prvotno stanje sistema. To ponavljamo do željenega števila iteracij, naš rezultat pa je vektor stanj ki jih matrika zavzame po med vsemi prehodi. Tak vektor lahko kasneje uporabimo za izbiro izhodnih elementov, v našem primeru izpis besedila.

4 Osnovni program

Program smo sprva implementirali v programskem jeziku Octave, a smo zaradi hitrosti in večje fleksibilnosti pri delu z nizi besed dodali še implementacijo projekta v programskem jeziku Python z uporabo knjižnjice NumPy.

Spodnja izvorna koda predstavlja jedro programa ki izračuna prehodno matriko. Vhodne podatke transformiramo v množico, ki vsebuje vse unikatne besede in jih uredimo po abecedi. Ustvarimo prehodno matriko in zaradi učinkovitosti pred glavno zanko izračunamo indekse besed, ki jih shranimo v slovar.

Zatem iteriramo čez celotno besedilo in v matriko prištevamo pojavitve besede j za besedo i. Da iz tega dobimo prehodno matriko verjetnosti, moramo dobljene vrednosti normirati tako, da delimo elemente vsake vrstice z vsoto posamezne vrstice.

```
def generate_matrix(text):
    #Make text into set of unique words
    uniques = list(set(text))
    uniques.sort()
    num_words = len(uniques)
```

```
    #Create NumPy transition matrix
    P = np.zeros(shape=(num_words, num_words))
    previous = text[0]

    wds = {} #optimization: precalculate indexes (O(1) hashmap)
    for word in uniques:
        wds[word] = uniques.index(word)

        #count each occurence of word following another word
    for word in text[1:]:
        P[wds[previous], wds[word]] += 1
                previous = word

    #transform counts into probabilities by dividing
    #each row element by sum of counts in row
    for i in range(0, num_words):
        if sum(P[i,]) != 0.0:
            P[i,] /= sum(P[i,])

    return {'words': uniques, 'trans_matrix': P}
```

Prehodno matriko in vektor besed lahko sedaj uporabimo za generiranje stanj, za večja besedila pa jo je pametno tudi trajno shraniti, saj nova gradnja matrike P običajno traja dlje kot ponovno nalaganje z diska. Mi smo za to uporabili modul Pickle, ki omogoča de/serializacijo objektov z uporabo binarnih datotek.

Sledeča izvorna koda opisuje generiranje stanj za en prehod skripte. Najprej si pripravimo začetno stanje in vektor stanj, v našem primeru s pomočjo zanke, ki poskrbi da je beseda v začetnem stanju taka, da se začne z veliko začetnico. V slovar podatkov dodamo tudi izračun kumulativne vsote vrstic matrike P. To omogoča hitrejše izvajanje večih ponovitev celotnega postopka generiranja stanj, vendar na račun malo večje konstantne porabe pomnilnika (razlika v hitrosti je vidna pri besedilih dolžine 100000 besed in več).

Na koncu se za željeno število iteracij izračunajo stanja - generiramo naključna števila med 0 in 1, ter poiščemo naslednji element vrstice kjer je kumulativna vsota večja ali enaka številu, kar nam poda naslednje stanje.

```
def generate_text(data, length=50):
    words = data['words']
    num_words = len(words)

        #select pseudo-random initial state
    x0 = randint(0, num_words - 1)
    while words[x0].islower():
```

```
    x0 = randint(0, num_words - 1)

    #initialize vector of states
states = [x0]

#compute cumulative sum if not present, for later reuse
if 'sum' not in data:
    data['sum'] = np.cumsum(data['trans_matrix'], axis=1)

    #generate $length states
    for i in range(0, length):
    idx = np.where(data['sum'][x0,] >= random())[0][0]
    states.append(idx)
    x0 = idx

return states
```

Ko imamo vektor stanj, indekse besed pretvorimo v same besede in jih zapišemo na standardni izhod ali v datoteko.

5 Nadgradnje osnovnega programa

Ker je osnovna naloga relativno kratka, smo v skupini dodali nove izzive in si izmislili nekaj dodatkov k osnovni nalogi.

5.1 Generiranje vprasanj in odgovorov

Sprogramirali smo program, ki zgenerira vprašanje in nanj poda odgovor, ki v večini primerov sicer ni pretirano smiseln, je pa zabaven. Program je spisan v programskem jeziku Octave, logika pa se nahaja V datoteki *questiongen.m*.

Program naključno generiranemu stavku, ki ga dobimo s pomočjo skripte textgen.m, pripne vprašalnico, najde najdaljšo besedo v vprašanju ter na podlagi te besede zgenerira odgovor.
Prvi glavni del programa je generiranje vprasanja, kar naredimo s klicom funkcije textgen.m, ki generira vprašanje z naključnim začetnim stanjem.

```
textgen("../data/butalci/basic", "./question.out", questionLength, -1)
```

Drugi del programa je generiranje odgovora, kar, prav tako kot pri vprašanju, naredimo s klicom funkcije textgen.m, s to razliko, da ji za začetno stanje določimo najdaljšo besedo v vprašanju.

```
textgen("../data/butalci/basic", "./answer.out", randi(5)+5, -1, longestWord);
```

Nekaj zanimivih izhodov programa questiongen.m:

```
Vprasanje: Kam  prihranili za vrat in klešče v?
Odgovor: Prihranili za tombolo, v Butale, oh Butale!
```

```
Vprasanje: Zakaj kislo mleko bolj podobni zmaju, kakor vi?
Odgovor: Podobni zmaju, kakor nesrečni policaj oni, ki me grize.
```

Kot je razvidno iz primerov, so tako vprašanja, kot tudi odgovori nanje, popolnoma nelogični, saj je začetno stanje odgovora najdaljša beseda v vprašanju, kar pa ponavadi ni nujno glavna tema vprašanja.

5.2 Smiselnejši stavki z upoštevanjem ločil

Da bi dobili bolj smiselne stavke, smo osnovni program spremenili tako, da smo končna ločila upoštevali kot lastne besede. To je pomenilo tudi, da med generiranjem besedila končna ločila vedno prehajajo v besed z veliko začetnico, kar ustvari naravnejše stavke.

Ko so ločila ločena od besed na katerih so vezana, dobijo svoje mesto v prehodni matriki in se obravnavajo kot navadne besede.

```
        SPECIAL = '.!?'
    with open(path, "r", encoding="UTF-8") as in_file:
        #read all lines of file and all words of lines
        #separated by newlines and spaces
        for line in in_file:
            for word in line.split():
                buf = []
                #if end of word contains a punctuation mark,
                #separate it into a new 'word'
                while len(word) > 1 and word[-1] in SPECIAL:
                    buf.append(word[-1])
                    word = word[0:-1]
                text.append(word)
                for char in buf:
                    text.append(char)
```

6 Primeri generiranih besedil

```
#butalci
Nista imela vilic, zato je vsako leto prekratek, zima.
Blizala se je Butalec ostrmi: Oha, kakšna da so pričakovali in
```

```
so bili vajeni in so radi odtrgavali, ko je imelo vesoljno.

#filmski citati
Sometimes he charmed the one Lord of Udun! Everybody
knows you look retarded, not you empty handed. The first.

And you very little mini-heroin addicts. Huh. I play a
ping pong competition. Like it was that.

#Shakespeare, Hamlet
Shows itself scopes not let go. I do know, by heaven.
And Guildenstern, and there no planets strike, No reckoning
made, Unless things that body. For womens fear.

#J.K. Rowling, Harry Potter
CREATURE-INDUCED INJURIES. Jus' maybe, you such a brass instrument,
something the Weasley hesitated, then sat on the Bulgarian
Minister around them and that sometime. He put a large damp.

Being continually moving soundlessly; Seamus Finnigan asked
you ... BOOM. Using his own knees gave it be necessary to
wrestle trolls ... No, you can't help her this time as possible.
```

7 Delitev dela

- Ilievska, Zorica: Izbira besedil, matematično ozadje, poročilo, predstavitev
- Lang, Ina: Izbira besedil, poročilo, predstavitev
- Novak, Miha: Programiranje nadgradnje programa, poročilo
- Šubic, Tine: Programiranje osnovnega programa, celotna Python implementacija, poročilo, predstavitev

8 Literatura

1. Izvorna koda: `https://github.com/MikroMan/mctextgen`
2. Markovske verige: `https://en.wikipedia.org/wiki/Markov_chain`
3. Modul Pickle: `https://docs.python.org/3/library/pickle.html`
4. Python 3: `https://www.python.org/download/releases/3.0/`
5. Modul NumPy: `http://www.numpy.org/`