Analiza besedila

|  |  |
| --- | --- |
| ***Predmet*** | |
| *Naziv predmeta:* | **Teorija informacije in kodiranja** |
| *Študijski program:* | MAG Telekomunikacije |
| *Nosilec predmeta:* | prof. dr. Mirijam Sepesy Maučec |
| *Asistent:* | mag. Uroš Berglez |
| *Študijsko leto:* | 2017/2018 |

|  |  |
| --- | --- |
| Študent(ka) | |
| *Vpisna številka:* | E5034437 |
| *Priimek in ime:* | Vučković David |
| *Datum:* | 19.1.2017 |

Osnovni potek programa

# 

# 

Diagram predstavlja osnovno delovanje programa, iz katerega je razvidna predvsem modularnost funkcij, ter povezave med posameznimi funkcijami. Kompleksnost algoritmov je prekompleksna za pregleden diagram.

## Pridobivanje besedila iz datoteke(get\_text)

V Pythonu je pridobitev besedila iz datoteke precej enostavno.

with open(path, 'r') as myFile:

data = myFile.read()

To nam enostavno pridobi besedilo v obliki enega velikega stringa. Funkcija je klicana samo enkrat za pridobitev besedila, in nikoli več.

## Izločevanje ločil in drugih znakov(split\_text)

## 

Ko pridobimo besedilo iz tekstovne datoteke je naš naslednji korak to, da izločimo nepotrebne znake, ter nato celoten string razdelimo v različne besede ter jih shranimo v podatkovno strukturo.

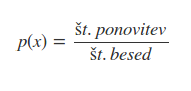
Za izločevanje nepotrebnih znakov je bil uporabljen RegEx, ki je iz teksta odstranil vse znake, ki niso v Slovenski abecedi, dovolil je le črke a-z in seveda naglasovne znake, kot so ä , saj je vhodno besedilo umetniško. Nato odstranimo morebitne prazne elemente lista. To je večinoma le varnostni ukrep. Zaradi tega, ker za kasnejše programiranje želimo obdržati čim več podatkov, tudi seznam vseh besed v podatkovni strukturni list, smo za preštetje uporabili drugo funkcijo, ki pa je po svoji velikosti tako majhna, da bi jo lahko dodali v trenutno opisano funkcijo. Ta seznam besed(vsebuje ponovitve), se kasneje uporablja za izračun vseh verjetnosti, saj potrebujemo število vseh besed. Seznam se prav tako uporabi za ustvarjanje bigramov.

**Preštetje besed(word\_count):**

* Uporabljen je le Pythonski object Counter, ki nam iz vhodnega seznama besed izpiše kolikokrat se vsaka beseda ponovi
* Funkcija nam vrne rezultat v obliki Counter{beseda:št.ponovitev besede}
* Rezultat se prav tako uporablja za izračun verjetnosti
* Pazimo, da je dodan stavek, ki prešteje znake tako, da se ne ozira na velike ali male črke!

## Izračun lastne verjetnosti(own\_probability)

Za izračun lastnih verjetnostih se uporablja isti algoritem, ki uporabi le število vseh besed ter Counter, v katerem je zapisano kolikokrat se vsaka beseda ponovi. Nato deli vsako vrednost, ki predstavlja število ponovitev s številom vseh neunikatnih besed(dolžina rezultata split\_text). Zaradi enostavnosti in nepoznavanja jezika, smo za dosego rezultatov vse vrednosti najprej shranili v dva lista(array,seznam), eden nam je služil kot list ključev(to so bile v našem primeru besede) ter list vrednosti(v našem primeru je to število ponovitev vsake besede). Seznam ponovitev se nato spremeni glede na rezultate spodnje enačbe. Sezname shrani v novi dictionary, ki ga napolnimo s seznamom besed, ki je ostal nespremenjen in seznamom vrednosti, ki smo ga spremenili.



št. ponovitev = koliko krat se iskana beseda ponovi v besedilu

št.besed = število vseh besed v besedilu

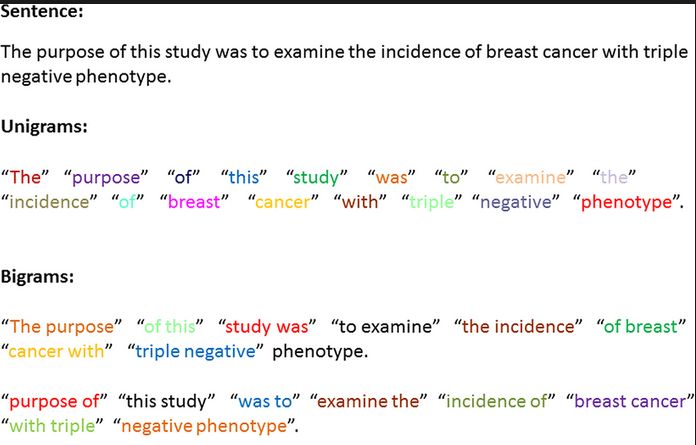
## Ustvarjanje bigramov

Bigrame(dve besedi) ustvarjamo z Pythonsko logiko. Algoritmično gledamo, le pogledamo sosednji dve besedi v seznamu besed, ter jih združimo s presledkom, da ju lahko nato še vedno ločimo. Seznam besed smo dobili iz metode split\_text. V kodi je to izvedeno tako:

* Ustvari prazni list besed, ki je velik približno toliko kot je število vseh besed
* Uporaba for zanke, ki iterira po besedilo ter združi vsako besedo s svojo naslednjo v seznamu vseh besed
* Združeni besedi shrani v prazen seznam na prosti indeks
* Seznam shrani v objekt Counter, ki ni občutljiv na velike in male črke(saj smo uporabili seznam vseh besed, ki vsebuje tako male kot velike črke!). Ta nam, kot že vemo, vrne rezultat, ki nam prešteje ponovitve bigramov v nekem besedilu.

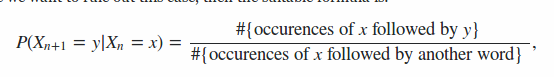
Bigrami nato služijo kot pomembna spremenljivka pri računanju pogojnih verjetnosti.

**Bigrami:**



## Računanje pogojnih verjetnosti

Tukaj pridejo v poštev tako seznam vseh besed, preštevanje besed kot bigrami. Uporabljena enačba je :



Gledamo torej kolikokrat besedi x sledi y in koliko krat se v seznamu besed pojavi beseda x kot prva beseda.

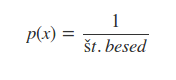
Algoritemsko razmišljanje, implementirano v kodi je sledeče:

* ustvari seznam, ki vsebuje število ponovitev besed in seznam, ki vsebuje bigrame in število njihovih ponovitev
* bigramu odstrani drugo besedo->unigram
* Če je dobljen unigram enak kot unigram znotraj slovarja, v katerem je število ponovitev unigrama deli, potem vzami iteracijski indeks, ter ga uporabi za pridobitev št. vseh ponovitev določenega bigrama v obliki verjetnosti z verjetnostjo, da se pojavi naš unigram. Te rešitvi nato deli med sabo.
* Deljene rešitve shranjuj v list, nato shrani v slovar besede iz Counterja bigramov in vrednosti v prej omenjenem listu deljenk.

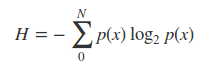
Slovar s pogojnimi verjetnostmi nato izpisujemo v datoteko Cond\_probs.txt ali v datoteko in konzolo, v primeru da imamo manj kot 100 najdenih unikatnih besed.

## Entropija, če so vse lastne verjetnosti enake(equal\_entropy)

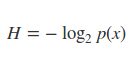
Funkcija uporabi le število vseh besed. Verjetnost nato izračunamo po naslednji enačbi:



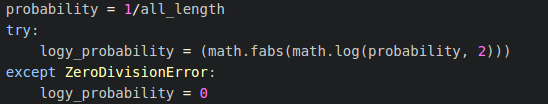
Nato uporabimo naslednjo enačbo za izračun entropije:



Pri katerem je N število vseh besed. Kot vidimo, si lahko enačbo pokrajšamo.

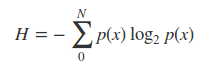


V kodi je tudi algoritmična implementacija identična matematični.



## Entropija, če upoštevamo lastne verjetnosti(own\_entropy)

Funkcija je v svoji zasnovi podobna računanju entropije, razlike je le v koraku, kjer pride do okrajšave. Uporabljeno je število besed in slovar, ki vsebuje besedo ter lastno verjetnost te besede. Kot večkrat, razdelimo slovar na dva seznama. Seznam besed in seznam lastnih verjetnosti. Uporabimo isto enačbo kot prej :



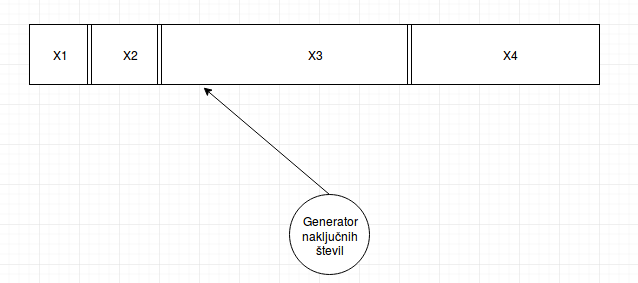
Po uporabi enačbe, lastne verjetnosti v seznamu zamenjamo z izračunanimi entropijami. V primeru, da imamo pod 100 unikatnih besed, izpišemo vse besede s svojimi besedami v konzolo, medtem ko pa v obratnem primeru besede s svojimi istoležečimi entropijami izpišemo v datoteko Own\_entropy.txt

## Generiranje besed

Sestavljena iz treh funkcij zaradi morebitnih problemov pri rekurziji in shranjevanju podatkov. Funkcija generator besed(word\_generator) služi večinoma kot matična funkcija obeh omenjenih funkcij. Nima pa druge funkcije kot klicanje spodaj opisanih besed in ločevanje slovarja bigramov na seznam verjetnosti in seznam besed. Uporablja bigrame, seznam vseh besed, pa tudi spremenljivko “limit”, ki jo vpiše uporabnik, pove pa nam, koliko besed želimo zgenerirat.

**Utežena naključnost(weighted random)**

Je princip, ki je uporabljen za generiranje besed po Markovovih verigah. V osnovi nam generator naključnih številk vse številk ne generira z enako verjetnostjo. V naši implementaciji je abstrakcija principa naslednja:



Vidimo, da ima generator naključnih števil(ki naj bi bil popolnoma naključen) večjo verjetnost, da pokaže na spremenljivko, ki je po svoji “širini” večja->ima večjo verjetnost. V naši implementaciji je to sicer opravljeno z visokonivojsko knjižnico numpy. Ta ima funkcijo numpy.random.choice(), ki nam glede na vhodni seznam in seznam verjetnosti za vsako spremenljivko v vhodnem seznamu izbere eno vrednost v vhodnem seznamu.

**Generiranje prve besede(first\_word\_generator):**

Funkcija, ki nam iz množice lastnih verjetnosti unigramov najde po principu weighted random. Od matične funkcije pridobi le seznam besed in seznam verjetnosti. Seznam verjetnosti je treba normalizirat, saj zaradi zaokroževanja pride do manjših napak. Normaliziramo na 1 tako, da vsake izmed verjetnosti delimo z vsoto vseh verjetnosti.

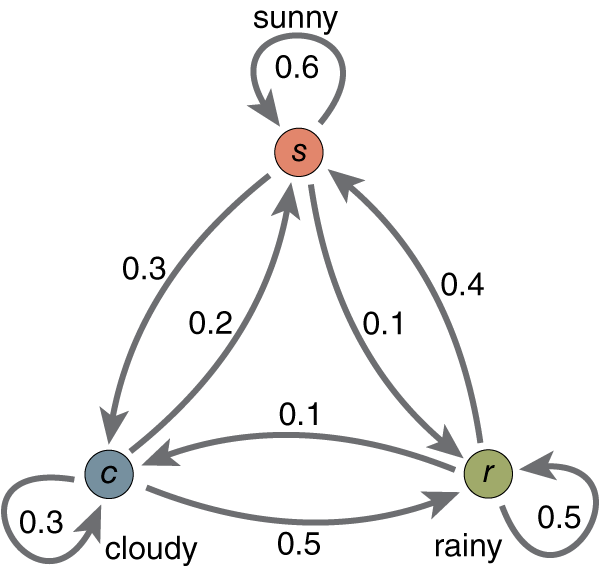
Nova prva beseda se izbere vedno, ko pri generiranju besed naletimo na zid. Ta zid je opisan v opisovanju funkcije “generiranje naslednje besede”. Večjo lastno verjetnost ima beseda, večja je verjetnost, da jo bo generator naključnih števil “izbral” torej padel v njen interval. Ko izberemo bigram seveda odstranimo drugi del bigrama, da dobimo unigram.

**Generiranje naslednje besede(next\_word\_generator)**

Deluje po principu markovovih verig, ki je v naši kodi abstraktno zamišljen kot:

[Povezava do slike zaradi preglednosti](https://imgur.com/UqiGJhy)

Uradni viri pa ga ilustrirajo kot:



Po že prej opisanem postopku weighted random preko prve besede in pogojnih verjetnosti kot uteži pridobimo naslednjo besedo in jo izpišemo. V primeru, da naslednje besede ne moremo zgenerirati(pri printanju se pojavi ~ ki označuje to), ker se druga beseda v bigramu nikjer ne pojavi kot prva beseda, smo prisiljeni, da zgeneriramo novo prvo besedo in postopek ponovimo, temu rečemo, da smo zadeli zid. Zid se zgodi le ko dosežemo zadnjo besedo v besedilo. Omejitev zanke je uporabniško definirana spremenljivka limit, ki nam pove, koliko besed zgeneriramo.

Uporabniška navodila

Uporabnik prenese projekt preko portala GitHub ali pridobi fizično kopijo mape, v kateri so tekstovni viri, uporabljeni pri testiranju programa. Nato uporabnik zažene program z vpisom naslednje komande v cmd ali terminal.

python3 \*\*\*\*pot do datoteke main.py\*\*\*\*

Uporabnika nato program vpraša, naj specificira ime tekstovne datoteke, do katere želi dostopati. Dodati je treba končnico. Primer:

samplesource.txt

**Paziti moramo, da je naša vpisana datoteka v isti mapi kot main.py**

Nato se mu izpišejo podatki, po vrstnem redu(če ima tekstovna datoteka manj kot 100 unikatnih besed):

1. “Equal entropy(equal for every word): “ ki pove, kakšna je entropija, če imamo za vsako besedo v besedilo enako verjetnost
2. “Entropy using own probabilities: Sum of all entropies: “ nam pove kakšna entropija pri pogoju, da vzamemo v obzir lastne verjetnosti besed
3. “Conditional entropy Sum of all entropies:” nam pove kakšna je vsota vseh entropij, pri pogoju, da za vsako besedo poznamo prejšnjo besedo.
4. “Amount of unique words” “Amount of non-unique words” , “Unique word pairs” nam povejo koliko unikatnih besed imamo, koliko besed imamo, če ne gledamo na unikate in koliko unikatnih bigramov imamo.
5. “Words” in “Word counter” temu izpisu sledijo vse besede v našem besedilu in besede, ki jim sledi število ponovitev
6. “Own probabilities:” nam pove, lastno verjetnost vsake izmed besed. [**se zapiše v datoteko Own\_probs.txt” če imamo več kot 100 unikatnih besed**
7. “Conditional probabilities”: pove nam pogojno verjetnost za vsak bigram, izpisana so kar v obliki “šibki ki”, kar lahko beremo tudi kot ki|šibki. [**se zapiše v datoteko “cond\_probs.txt” če imamo več kot 100 unikatnih besed]**
8. “Own entropies[bit]: izpiše entropijo pri upoštevanju lastne verjetnosti za vsako izmed besed**]se zapiše v datoteko “Own\_entropies.txt” če imamo več kot 100 unikatnih besed]**
9. “Conditional entropies[bit]:” prikazuje za vsako besedo posebej, kakšno entropijo ima, za verjetnosti so vzete pogojne verjetnosti besed!**[se zapiše v datoteko Cond\_entropies.txt” če imamo več kot 100 unikatnih besed”**

Nato nas terminal/cmd vpraša “How many words to print out?” V katerega vnesemo željeno število besed, ki jih želimo generirat. Sproti se začnejo izpisovati besede, vsaka v svojo vrstico. Ko zgenerira zadnjo besedo konča z . Takrat se tudi program zaključi in ga je treba ob ponovnem zagonu ročno preko terminala/cmd.

Vzorčni zagon

Ob vhodni datoteki, komentarji rezultatov so napisani v krepkem tekstu:

Equal entropy(equal for every word): 4.321928094887363

**Če vzamemo, da je verjetnost vsake besede 1/20(20 je število unikatnih besed) dobimo rezultat -log2(1/20), rezultat česar je 4.321928094887363...**  
Entropy using the own probabilities  
 - Sum of all entropies: 4.277613436819114

**Upoštevanje lastnih verjetnost je zelo podobna entropiji, ko so vse besede enako verjetne, saj se le besedi počivaj in v ponovita dvakrat.**   
Conditional entropy:  
 - Sum of all entropies: 2.0

**Večina bigramov ima verjetnost 1, saj se prva beseda, ki ji sledi druga prikaže le enkrat. mirno|počivaj,topli|v,popji|v,srečno|počivaj imajo pogojne verjetnosti 0.5(verjetnost za katerega koli od dveh možnih izidov je enaka), kar pomeni tudi, da so njihove entropije v bitih 0.5(log2(0.5)=0.5).**   
Amount of non-unique words:  
22  
Amount of unique words:  
20  
Unique word pairs:  
21  
Words:  
['Počivaj', 'mirno', 'angelj', 'šibki', 'Ki', 'dni', 'le', 'šteješ', 'ne', 'še', 'let', 'Počivaj', 'srečno', 'v', 'topli', 'zibki', 'Ko', 'v', 'popji', 'mlade', 'rože', 'cvet']  
Word counter:  
Counter({'v': 2, 'počivaj': 2, 'mlade': 1, 'ko': 1, 'srečno': 1, 'ki': 1, 'let': 1, 'še': 1, 'mirno': 1, 'topli': 1, 'šibki': 1, 'cvet': 1, 'zibki': 1, 'dni': 1, 'popji': 1, 'ne': 1, 'šteješ': 1, 'le': 1, 'rože': 1, 'angelj': 1})  
Own probabilities:  
{'ko': 0.045454545454545456, 'ki': 0.045454545454545456, 'let': 0.045454545454545456, 'počivaj': 0.09090909090909091, 'še': 0.045454545454545456, 'mirno': 0.045454545454545456, 'topli': 0.045454545454545456, 'šteješ': 0.045454545454545456, 'šibki': 0.045454545454545456, 'le': 0.045454545454545456, 'dni': 0.045454545454545456, 'popji': 0.045454545454545456, 'angelj': 0.045454545454545456, 'v': 0.09090909090909091, 'ne': 0.045454545454545456, 'cvet': 0.045454545454545456, 'mlade': 0.045454545454545456, 'zibki': 0.045454545454545456, 'rože': 0.045454545454545456, 'srečno': 0.045454545454545456}

**Izračun lastnih verjetnosti. Ker imamo 22 besed, je verjetnost besed, ki se pojavijo le enkrat (1/22)0.4545…, verjetnost besed ki se pojavijo 2x pa je 2/22 = 0.90...**   
Conditional probabilities:  
{'počivaj mirno': 0.5, 'ki dni': 1.0, 'popji mlade': 1.0, 'v topli': 0.5, 'ko v': 1.0, 'let počivaj': 1.0, 'v popji': 0.5, 'le šteješ': 1.0, 'ne še': 1.0, 'počivaj srečno': 0.5, 'srečno v': 1.0, 'dni le': 1.0, 'rože cvet': 1.0, 'šteješ ne': 1.0, 'še let': 1.0, 'topli zibki': 1.0, 'mlade rože': 1.0, 'angelj šibki': 1.0, 'šibki ki': 1.0, 'zibki ko': 1.0, 'mirno angelj': 1.0}

**Pogojno verjetnost a|x izračunamo tako, da preštejemo ponovitve a|x in ponovitve a kot prve besede. V našem primeru se samo beseda v in beseda počivaj ponovita dvakrat. Ostale pogojne verjetnosti so vse 1, saj če poznamo prvo besedo, lahko glede na vhodne podatke z verjetnostjo 100% sklepamo o naslednji besedi. Pri besedi v pa imamo verjetnost glede na vhodne podatke lahko naslednjo besedo popji ali besedo topli. Enako je z besedo počivaj. Enkrat sledi beseda mirno drugič beseda srečno. Tako če poznamo besedo počivaj lahko sklepamo o naslednji besedi sklepamo z verjetnostjo 50%.**   
Own entropies[bit]:  
{'mlade': 0.20270143721078623, 'ko': 0.20270143721078623, 'cvet': 0.20270143721078623, 'ki': 0.20270143721078623, 'let': 0.20270143721078623, 'srečno': 0.20270143721078623, 'še': 0.20270143721078623, 'mirno': 0.20270143721078623, 'topli': 0.20270143721078623, 'šibki': 0.20270143721078623, 'zibki': 0.20270143721078623, 'dni': 0.20270143721078623, 'popji': 0.20270143721078623, 'v': 0.31449378351248164, 'le': 0.20270143721078623, 'počivaj': 0.31449378351248164, 'šteješ': 0.20270143721078623, 'ne': 0.20270143721078623, 'rože': 0.20270143721078623, 'angelj': 0.20270143721078623}  
Conditional entropies[bit]:  
{'počivaj mirno': 0.5, 'ki dni': 0.0, 'popji mlade': 0.0, 'v topli': 0.5, 'ko v': 0.0, 'let počivaj': 0.0, 'v popji': 0.5, 'le šteješ': 0.0, 'ne še': 0.0, 'počivaj srečno': 0.5, 'srečno v': 0.0, 'dni le': 0.0, 'rože cvet': 0.0, 'šteješ ne': 0.0, 'še let': 0.0, 'topli zibki': 0.0, 'mlade rože': 0.0, 'angelj šibki': 0.0, 'šibki ki': 0.0, 'zibki ko': 0.0, 'mirno angelj': 0.0}  
Word generator  
How many words to print out?10  
Generated text:  
mlade   
rože cvet mlade   
še   
mirno   
topli   
v   
ko   
ki   
angelj   
srečno   
.