

Programiranje I

izpiski s predavanj

Nace Kovačič

TO DO

- regularni izrazi - sklici na skupine
- zajem podatkov - normalizacija podatkov
- analiza podatkov - transformacija razporednic
- analiza podatkov - vektorizacija
- uvod v ocaml - parametrični polimorfizem
- funkcijsko programiranje - anonimne funkcije
- funkcijsko programiranje - delna aplikacija funkcij in curryirane funkcije
- funkcijsko programiranje - curryiranje

regularni izrazi

neko zaporedje znakov, ki opisuje vzorec v besedilu

- obstajajo boljša orodja za pridobivanje podatkov s spletne strani (*beautiful soup*)
- v VSCodeu lahko iščemo *navadno besedilo* ali pa *regularne izraze*, ki so podani v iskalniku (če vključimo možnost `.*`)

legenda osnovnih znakov

<code>.</code>	katerikoli znak
<code>\.</code>	pika
<code>\\</code>	backslash
<code>\d</code>	številka ¹
<code>\D</code>	komplement števk
<code>\w</code>	alfanumerični znak
<code>\W</code>	komplement alfanumeričnih znakov
<code>\s</code>	whitespace ²
<code>\S</code>	komplement whitespace -a
<code>.DOTALL</code>	pojdi tudi čez nove vrstice

- če želimo, da se vzorec ponovi vsaj n - krat, ga damo v *zavite oklepaje*
- v *oglate oklepaje* vstavimo znake, ki jih želimo poiskati
- če želimo, da določenih znakov ne najde, uporabimo `[^znaki]`
- če želimo poiskati več besed, jih damo v *oglate oklepaje*, mednje pa vstavimo `|`
- če želimo poiskati vse znake med dvema vrednostima uporabimo *vezaj*
- če mora biti znak na *začetku*, napišemo predenj `^`
- če mora biti znak na *koncu*, napišemo na konec `$`

¹enakovredno izrazu `[0-9]`

²presledek, tab, etc.

ponovitve elementov

? 0 ali 1
+ 1 ali več
* 0 ali več ³

skupine

nastanejo, ko damo nekaj v oklepaje

- skupin je lahko več
- lahko jih poimenujemo (*?P<naslov>*)

json

- podatke o filmih je pametno shraniti v *json* datoteko
- *standarden tekstovni format*, ki izhaja iz *java-scripta*
- podpira objekte
- manjkajočo vrednost označimo z *null*
- iz *Pythona* v *json* dobimo s funkcijo **json.dumps**
- iz *json*a v *Python* dobimo s funkcijo **json.loads**

csv

- bolje od *json*a za statistično analizo, saj so podatki podani tabelarično
- v zgornji vrstici so navadno *naslovi stolpcev*
- vsaka vrstica predstavlja svoj *podatek* ⁴

Pythonove knjižnice za obdelavo podatkov

re

re.findall(vzorec, besedilo) poišče vse ponovitve vzorca v besedilu
re.groupdict naredi slovar naših skupin
re.groupiter iterator *groupdict*
re.finditer PREVERI, KAJ NAREDI

requests

requests.get(url) dobimo odziv spletne strani⁵
requests.get(url).text vrne *vsebinsko* spletne strani

³ + in * imata *skromni* in *požrešni* način

⁴ recimo vsak film je v svoji vrstici

⁵ dobimo objekt tipa *Response*

csv

`csv.DictWriter(csv, fieldnames)` slovar zapišemo v ustrezna polja
`csv.DictWriter(csv, fieldnames).writeheader()` izpiši glavo

pandas

knjižnica za analizo podatkov

- pogosto uporabljamo skupaj z *Jupyterjem*
- *Jupyter* je vizualno okolje za delo analizo podatkov v Pythonu ⁶
- *Jupyter notebook* je sestavljen iz več celic ⁷

`pd.read_csv` prebere csv datoteko in prikaže tabelo
`pd.read.head(n)` dobimo *prvih n* elementov
`pd.read.tail(n)` dobimo *zadnjih n* elementov
`pd.read[ime_stolpca]` dostopamo do posameznega stolpca
`pd.read[seznam imen]` dostopamo do razpredelnice stolpcev
`pd.read.iloc[i]` dostopamo do vrednosti na *i* - *tem* mestu
`pd.read.loc[k]` dostopamo do ključa *k*
`pd.read.groupby` *razpredelnice*, kjer so vrstice združene glede na lastnost
`pd.read.count(pogoj)` štejemo *ponovitve*
`pd.read.size()` pove, koliko je *vnosov*, ne glede na dane podatke
`pd.read.merge(stolpci)` dve tabeli *združi* po stolpcih z istimi imeni
`pd.read.join()` PREVERI
`pd.read.mean()` izračuna *povprečje*
`pd.read.sort_values()` *uredi* po velikosti

Vrednosti v stolpcih lahko tudi *filtriramo* z *logičnimi vrednostmi*. Stolpce lahko tudi urejamo po vrednostih (*naraščajoče* ali *padajoče*).

Grafe rišemo z *%matplotlib inline*, tako, da uporabimo `.plot()`. *Razsevni diagram* dobimo z `plot.scatter()`.

⁶prednost je v tem, da lahko Python datoteko izvajamo sproti, da nam ni treba ponovno restartat vsakič, ko kaj spremenimo

⁷koda (*Python*) in besedilo (*Markdown*)

naivni Bayesov kvantifikator

enostaven primer strojnega učenja

- odločamo se, ali nekaj pripada nekemu razredu ali ne ⁸

Zanima nas verjetnost dogodka D_i , ob pogoju, da vsebuje korene K_1, \dots, K_n .

$$P(D_i|K_1 \cap \dots \cap K_n)$$

Uporabimo ***Bayesov izrek***

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Velja

$$P(D_i|K_1 \cap \dots \cap K_n) = \frac{P(K_1 \cap \dots \cap K_n|D_i) \cdot P(D_i)}{P(K_1 \cap \dots \cap K_n)}$$

Ker je klasifikator naiven, velja

$$P(K_1 \cap \dots \cap K_n|D_i) = P(K_1|D_i) \cdot \dots \cdot P(K_n|D_i)$$

oz.

$$P(D_i|K_1 \cap \dots \cap K_n) = \frac{P(K_1|D_i) \cdot \dots \cdot P(K_n|D_i) \cdot P(D_i)}{P(K_1 \cap \dots \cap K_n)}$$

⁸uporabno za *spam*

OCaml

funkcijsko programiranje

- vrednosti definiramo z *let*
- presledki niso pomembni
- komentarje pišemo v *oklepaje* in *zvezdice*
- v interaktivni konzoli damo na koncu `;;`
- ima zelo rad *tipe*
- vrstni red je vedno ime_funkcije prva_spremenljivka druga_spremenljivka⁹
- v funkciji lahko definiramo *lokalne definicije*
- *char* ima *enojne* narekovaje, *string* pa *dvojne*
- lahko definiramo več stvari hkrati z *and*¹⁰
- deljenje je *celoštevilsko*¹¹
- *logične vrednosti* so tipa *bool*
- če želimo funkcijo definirati *po kosih*, uporabimo *match*
- pri *match* je *vrstni red* pomemben
- *rekurzivne funkcije* definiramo z *let rec*
- *nabore* pišemo kot *n-terice*
- *prazen nabor* imenujemo *unit*
- *sezname* so *homogeni*¹²
- z `::` *lepimo* sezname¹³

⁹temu pravilo pravimo *aplikacija*

¹⁰DIFFERENCE BETWEEN AND AND IN

¹¹za *float* dodamo operatorju piko na konec

¹²vsi elementi imajo *enak* tip

¹³funkcija je *desno asociativna*

funkcijsko programiranje

- *tipi* v OCamlu so navadno *statični*¹⁴
- OCaml ima od Pythona *bogatejše tipe*
- klic funkcije je *repen*, če se izvede *zadnji*
- funkciji, kjer so vsi klici *repno rekurzivni* pravimo *repno rekurzivna funkcija*
- Python je *proceduralni*¹⁵ in *imperativni*¹⁶ jezik
- OCaml je *funkcijski* in *deklarativni*¹⁷

funkcije višjega reda

funkcije, ki sprejemajo druge funkcije

- če funkciji dveh elementov podamo zgolj en element, dobimo *ново* funkcijo, ki še čaka na drugi argument¹⁸
- postopek *curryiranja*¹⁹

fold

fold right

- sprejme tri spremenljivke²⁰
- če imamo *prazen seznam*, vrnemo *začetno vrednost*, če je seznam tipa $x :: xs$, funkcijo rekurzivno uporabimo na tem repu in tej vrednosti dodamo x

fold left

- repno rekurzivna

¹⁴razlika med *statičnimi* ali *dinamičnimi* tipi je v tem, da se statični tipi preverijo, *preden* se funkcija požene

¹⁵programe pišemo kot zaporedja ukazov

¹⁶z ukazi spreminjamo spremenljivke

¹⁷opišemo stanje

¹⁸ $f\ x\ y = (f\ x)\ y$

¹⁹PREVERI

²⁰naš seznam, začetno vrednost, funkcijo f

test