NSU 4. domača naloga poročilo

Anže Mramor

June 2023

1 Osnovna naloga

Po uvozu podatkov smo iz domenskega predznanja razbrali sledeče:

- bližje vrednosti $\frac{\pi}{2}$ kot bo θ večji bo toplotni tok, *c.p.*,
- višja razlika kot bo med T_w in T_a večji bo toplotni tok, *c.p.*. Tok je verjetno odvisen samo od njune razlike,
- višji kot je η nižji je toplotni tok, *c.p.*.

Te informacije smo v prvi fazi reševanja problema uporabili za kreiranje novih spremenljivk kot sledi:

- izračunamo sinus kota θ: $\widehat{\theta} = \sin(\theta)$ - višja kot je vrednost, večji bo toplotni tok,
- kot drugi poskus izračunamo odstopanje vrednosti θ od $\frac{\pi}{2}$: $\widehat{\theta_2} = |\frac{\pi}{2} - \theta|$ - višja kot bo njena vrednost, večji bo toplotni tok,
- izračunamo razliko med T_w in T_a (pomemben predznak zaradi smeri toka): $T_r = T_w - T_a$ - višja kot bo razlika, večji bo toplotni tok,
- ker razmerje med razliko in Q verjetno ni linearno, poračunamo še višje potence razlike (2,3,4) in njen koren,
- η "obrnemo": $\hat{\eta} = \frac{1}{\eta}$ - višji kot bo njena vrednost, nižji bo toplotni tok. Poračunamo tudi kvadrat te vrednosti, $\widehat{\eta_2}$,
- poizkusimo tudi enostavno z enostavnim kvadriranjem: $\widehat{\eta_3} = \eta^2$ - višja kot je vrednost, nižji bo toplotni tok.

Za odkrivanje povezav med spremenljivko Q in preostalimi spremenljivkami smo najprej preizkusili regresijska orodja za odkrivanje enačb, ki smo jih spoznali na vajah:

- linearna regresija,
- ridge regresija,
- lasso regresija

Funkcije za vsa 3 regresijska orodja smo definirali enako kot na vajah. Edina sprememba je bila privzeta nastavitev meje pri katerih smo morali vrednosti nastaviti na manjše kot na vajah, saj je problem delikatnejši.

Po igranju z različnimi kombinacijami, so se osnovne tri metode najbolje odrezale, ko smo večino novo generiranih spremenljivk odstranili. Med generiranimi dodatnimi spremenljivkami smo iz podatkov v tem delu odstranili T_w, T_a, θ, η, $\widehat{\theta_2}$, T_r, T_r^3, T_r^4, $\sqrt{T_r}$. Prav tako smo s funkcijo *PolynomialFeatures* vključili samo kvadratične kombinacije polinomskih členov - če smo uporabili višje kombinacije, ali vključili katere izmed zgoraj naštetih odstranjenih spremenljivk, so bili koeficienti premajhni in izbrana orodja niso našla nobene enačbe.

Spodaj so navedeni rezultati, ki so jih vrnila regresijska orodja:

- Enača, ki jo odkrije linearna regresija, pri meji 0.001:
  $Q = 0.0032 * \sin(\theta) * T_r^2$
- Enača, ki jo odkrije linearna regresija, pri meji 0.0001:
  $Q = 0.0002 * T_r^2 + 0.0032 * \sin(\theta) * T_r^2$
- Enača, ki jo odkrije linearna regresija, pri meji 0.00001:
  $Q = 0.0002 * T_r^2 + 0.0032 * \sin(\theta) * T_r^2 + 0.0000 * \sin(\theta) * \frac{1}{\eta} + 0.0000 * \sin(\theta) * \frac{1}{\eta^2} + 0.0001 * T_r^2 * \frac{1}{\eta^2}$
- Enača, ki jo odkrije ridge regresija, pri meji 0.001:
  $Q = 0.0032 * \sin(\theta) * T_r^2$
- Enača, ki jo odkrije ridge regresija, pri meji 0.0001:
  $Q = 0.0002 * T_r^2 + 0.0032 * \sin(\theta) * T_r^2$
- Enača, ki jo odkrije ridge regresija, pri meji 0.00001:
  $Q = 0.0002 * T_r^2 + 0.0032 * \sin(\theta) * T_r^2 + 0.0000 * \sin(\theta) * \frac{1}{\eta} + 0.0000 * \sin(\theta) * \frac{1}{\eta^2} + 0.0001 * T_r^2 * \frac{1}{\eta^2}$
- Enača, ki jo odkrije lasso regresija, pri vrednosti $\lambda = 1000$ in meji 0.0001:
  $Q = 0.3549 * 1 + 0.6535 * \sin(\theta) + 1.9292 * T_r^2 + 1.9953 * \frac{1}{\eta} + 0.1081 * \frac{1}{\eta^3} + 0.1913 * \sin(\theta)^2 + 0.0958 * \sin(\theta) * \frac{1}{\eta} + 0.6264 * \sin(\theta) * \frac{1}{\eta^3} + 2.8426 * T_r^2 * \frac{1}{\eta} + 0.7448 * T_r^2 * \frac{1}{\eta^3} + 0.3772 * \frac{1}{\eta} * \frac{1}{\eta^3} + 0.0863 * \frac{1}{\eta^2} * \frac{1}{\eta^3} + 0.5499 * (\frac{1}{\eta^3})^2$

Linearna in ridge regresija pri vsaki izmed mej odkrijeta enake enačbe, ki se zdijo precej enostavne in tudi smiselne. Skrb vzbuja le to, da je spremenljivka η ali kakšna izmed spremenljivk skonstruiranih iz nje vključena le v zadnji enačbi in še tam so vrednosti koeficientov precej nizke. Po drugi strani Lasso regresija vrne precej dolgo in kompleksno enačbo, ki zagotovo ni pravilna.

V nadaljevanju smo za vsako izmed odkritih enačb poračunali povprečno kvadratično napako, saj bo dober pokazatelj kako pravilna je enačba. Želimo si seveda, da bi bile vrednosti napake čim manjše. Hkrati opazimo tudi, da je povprečna vrednost spremenljivke $Q = 0.683$. Če želimo da je napaka približno 1 %, potem mora biti vrednost povprečne kvadratične napake okoli 0.01. Poračunali smo vrednosti povprečne kvadratične napake za prve tri odkrite enačbe (linearne oziroma ridge regresije). Členov v enačbah, ki so imele je vrednost koeficienta enak 0 na 4 decimalnih mestih nismo vključili, saj se je izkazalo, da ne pripomorejo k boljši natančnosti. Napake:

- Vrednost povprečne kvadratične napake za 1. enačbo je: 0.12573554309101834,
- Vrednost povprečne kvadratične napake za 2. enačbo je: 0.13360323693526674,
- Vrednost povprečne kvadratične napake za 3. enačbo je: 1.3348671054876422

Opazimo lahko, da sta enačbi, v katerih spremenljivka η ni vključena precej boljši, kot zadnja enačba z vključeno η. Hkrati lahko vidimo, da je napaka precej višja od tega kar bi si želeli, kar pomeni, da formula skoraj zagotovo ni pravilna. Verjetno manjka vključitev spremenljivke η, na še kakšen drug, alternativen način. Morda je težava tudi v tem, da regresije iz vaj niso najprimernejše za odkrivanje enačb v tem problemu, zato se lotimo odkrivanja enačb z drugimi metodami - najprej s knjižnjico *ProGED*.

V prvem poskusu iskanja enačbe na klasičen način smo vključili v iskanje vse generirane spremenljivke. Algoritem je vrnil:

ModelBox: 1 models
$\rightarrow [0.00208863155779034 * Tr2]$,
$p = 2.8954996363636375e-07$,
$error = 0.6556113356661486$,
$time = 0.044225215911865234$

torej enačbo $0.00208863155779034 * T_r^2$, z napako 0.6556113356661486.
Takoj opazimo, da je model očitno preveč enostaven, napaka pa precej višja kot pri iskanju z

regresijo, zato postopamo drugače - vključimo samo spremenljivki $\widehat{\theta}$ in T_r^2, za prva dva parametra in vse generirane za parameter η. Povišamo tudi parameter *sample_size* na 1000. Izpišemo tudi 3 najboljše modele.

ModelBox: 3 models
$\rightarrow [0.00326687336664677 * Tr2 * theta_hat]$,
$p = 3.981312000000002e-05$,
$error = 0.35370963516084986$,
$time = 0.15263843536376953$
$\rightarrow [0.00243047712091106 * Tr2 - 0.00199522747619697 * Tr2 * eta\_hat\_3/eta\_hat\_2 + 0.00763887519493214 * eta\_hat\_2 * theta\_hat * *2/eta\_hat]$,
$p = 6.9599471248387316e-40$,
$error = 0.5901130169687094$,
$time = 0.3701198101043701$
$\rightarrow [Tr2 * theta\_hat + 0.996601881774514 * Tr2 * (-theta\_hat - 0.000500370453485273) + eta_hat_3]$,
$p = 3.8955037265127066e-16$,
$error = 0.6239611175476883$,
$time = 0.21396827697753906$

V drugem poskusu je odkrita enačba $0.00326687336664677 * T_r^2 * \sin(\theta)$, z nekoliko nižjo napako kot prej 0.35370963516084986. Če na roke poračunamo še njeno povprečno kvadratno enačbo dobimo vrednost: 0.1251105060056215.

Po popravkih, je algoritem iz knjižnjice *ProGED* kot model z najnižjo napako odkril enak model kot linearna in ridge regresija prej, le da je koeficient nekoliko drugačen. Napake naslednjih najboljših predlaganih modelov so skoraj dvakrat višje, kot ta. Morda bi torej veljalo razmisliti o tem, da je to res lahko zelo dober približek pravilne enačbe. Preizkusili smo še več različnih vnosov za desno stran enačbe, vendar se z nobenim nismo mogli niti približati napaki, ki jo da zgornji model.

Kot zadnji poskus uporabimo še eno knjižnico predlagano na vajah *pysr*. Rezultati uporabljene funkcije iz paketa *pysr* so priloženi v *csv* datoteki. Funkcija očitno deluje precej drugače, saj je uspela odkriti popolnoma novo enačbo, ki pa je hkrati tudi precej boljša od vse, ki smo jih odkrili prej - ima precej nižjo napako 0.01348992911238157. Poleg tega je tudi rang napake primerljiv s tem, kar smo si želeli na začetku. Formula vključuje tudi vse 3 parametre, kar je tudi bolje, glede na prejšnje odkrite enačbe. Glavni problem prejšnjih metod je bil očitno v tem, da so parameter η lahko vključevale samo neposredno bodisi v števec bodisi v imenovalec. Funkcija iz knjižnice *pysr* pa je odkrila obliko, kjer je η vključena v imenovalcu in ji je dodana še konstanta A torej kot $A + \eta^2$, kar lahko popolnoma spremeni vrednosti, ki jih izračunamo z enačbo.

Končna enačba je torej:

$$Q = 0.0026820262 * \frac{(T_w - T_a)^2}{0.5930055 + \eta^2} * \sin(\theta)$$

Kot najbolj pravilna se mi zdi zadnja poiskana enačba, saj vključuje vse komponentne, ki smo si jih želeli in na način, da ustrezajo vsem točkam domenskega predznanja. Hkrati je tudi povprečna kvadratična napaka po tej enačbi daleč najnižja. Enačba sicer verjetno še ni popolnoma pravilna, saj bi pri pravi enači bila napaka še kakšen red velikosti nižja. Povsem možno je, da obstaja še kakšna optimizacija, morda s kakšnim drugačnim dodanim parametrom. Vsekakor pa se mi zdi, da poda enačba dovolj dober približek, da ji lahko za generiranje podatkov zaupamo.

2 Napredna naloga

2.1 Opis razvitega algoritma

Funkcija sprejme naslednje parametre:

- *gramatika*, ki je naša začetna konstruirana gramatika,
- *podatki*, ki je naša tabela podatkov generiranih z dano enačbo,
- *stevilo_iteracij* = 100, koliko iteracij naj naredi funkcija, privzeta vrednost je 100,
- *sprememba=0.01*, za koliko naj se spreminjajo verjetnosti posameznega pravila v gramatiki, privzeta vrednost je 0.01,
- *rhs_vars* = ["x1", "x2", "x3", "x4", "x5"], podane spremenljivke iz desne strani enačbe,
- *eps=10*, meja, od katere mora biti napaka manjša, privzeta vrednost je 10 (visoka, zato da se upošteva število iteracij)

Besedni opis algoritma

- najprej s pomočjo funkcije *EqDisco* generiramo eno kandidatno enačbo po standardnem postopku. S pomočjo vgrajenih funkcij ji optimiziramo konstante in pridobimo rezultate - enačbo, napako, verjetnost, čas, ... Shranimo si napako in jo nastavimo za minimalno trenutno napako,
- algoritem nato uporabi funkciji *flatten_rules* in *gramatika_v_slovar*:
  - *flatten_rules* sprejme drevesno strukturo gramatike, po kateri je bil generiran model (seznam seznamov) in ga pretvori v en seznam pravil
  - *gramatika_v_slovar* sprejme seznam pravil za generiranje modela in jih vrne v obliki slovarja

  in z njima pretvori začetno gramatiko in gramatiko (pot) s katero je bila generirana prva odkrita enačba v slovarja
- na tej točki vstopimo v *while* zanko - zanka traja toliko časa, dokler ne preteče *stevilo_iteracij* krogov in dokler ni napaka manjša ali enaka želeni meji *eps*
- najprej se v zanki izvedeta funkciji *posodobi_gramatiko* in *ustvari_gramatiko*:
  - *posodobi_gramatiko* sprejme oba generirana slovarja od prej - prvotni slovar konstruiran iz celotne vhodne gramatike in slovar enačbe, konstruiran iz poti po kateri je bila odkrita enačba. Poleg tega sprejme še logični izraz - preverja ali je nova napaka višja od neke nastavljene vrednosti - po preizkušanju se je zdelo, da je smiselna vrednost 10 in ali je nova napaka višja od stare. Če ena izmed teh vrednosti drži, potem vemo, da je nova napaka **velika** kar si funkcija zapomni. Funkcija nato za vsako pravilo posodablja verjetnosti v prvotnem slovarju (torej v slovarju v katerem je vključena začetna gramatika) kot sledi:
    - če je dano pravilo bilo uporabljeno za odkrivanje dane enačbe in je nova napaka velika, potem temu pravilu zmanjšamo verjetnost za število njegovih pojavitev v drevesu pomnoženim z neko želeno spremembo (privzeta vrednost je 0.01)
    - če je dano pravilo bilo uporabljeno za odkrivanje dane enačbe in nova napaka ni velika, potem temu pravilu verjetnost povečamo za enako vrednost kot v prejšnji točki
    - če dano pravilo za odkrivanje dane enačbe ni bilo uporabljeno ostane njegova verjetnost enaka.

Na koncu funkcija verjetnosti standardizira, tako da se seštejejo v 1. Funkcija vrne posodobljeno gramatiko v obliki slovarja.

  - *ustvari_gramatiko* sprejme slovar zgeneriran iz prejšnje funkcije in ga s pomočjo funkcije *ProbabilisticProduction* pretvori v standardno obliko za gramatike, ki ga bo funkcija *Eq.Disco* znala uporabiti za odkrivanje novih enačb.

- na tej točki ponovno poženemo funkcijo *Eq.Disco*, ki nam generira novo kandidatno enačbo iz posodobljene gramatike.

- Ponovimo še vse korake, ki smo jih izvedli že pred zanko (shranjevanje vrednosti, spreminjanje gramatike v slovar, shranjevanje napak), povečamo n za 1 in nadaljujemo na naslednjo iteracijo while zanke.

- V splošnem torej funkcija sprejme gramatiko, prek funkcije **Eq.Disco* računa kandidatne enačbe, glede na velikost napake pa posodablja verjetnosti v gramatiki. Funkcija na koncu vrne bodisi najboljši model (če je meja za *eps* nastavljena dovolj nizko vrne kar pravo enačbo), če pa nas zanima najmanjša napaka, ki jo je odkrila enačba v *stevilo_iteracij* pa vrne to (ob večji vrednosti *eps*).

2.2 Grafi

Primerjali smo najmanjšo napako, ki jo odkrijeta funkciji *Eq.Disco* in *poisci_enacbo* v 10, 20, 30, 40, 50, 60, 70, 80, 90 in 100 pregledanih enačbah (iteracijah). Poleg tega smo eksperiment ponavljali - za vsako število enačb iz nabora smo postopek ponovili 5-krat in izračunali povprečne najmanjše napake obeh algoritmov. Rezultati so prikazani na spodnjih grafih.

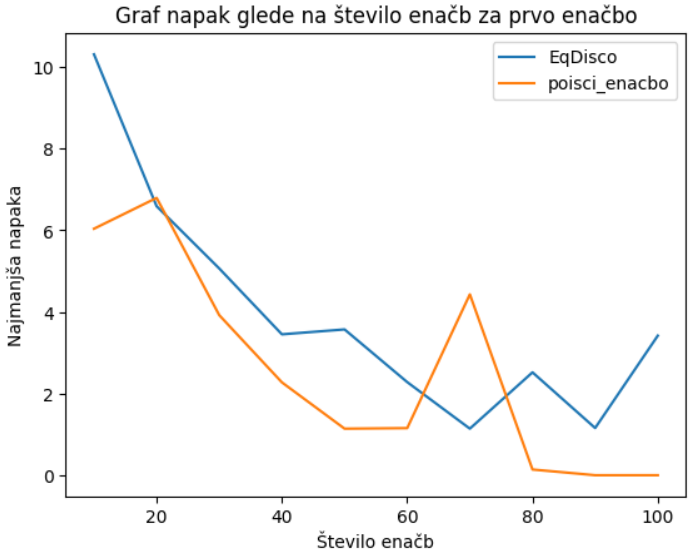

Figure 1: Graf napak glede na število enačb za prvo enačbo

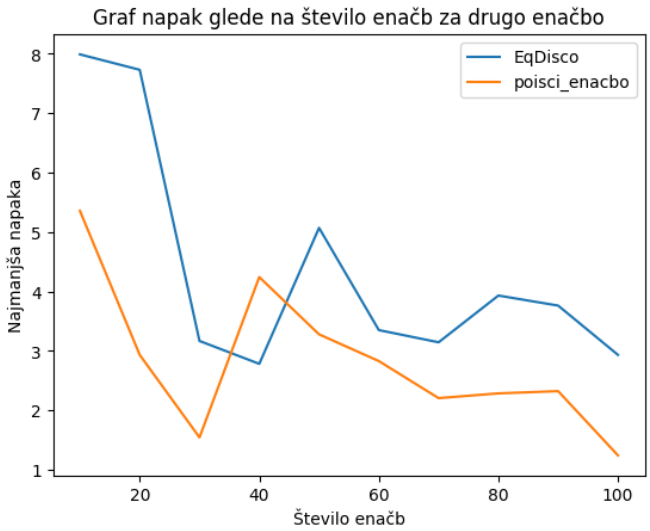

Figure 2: Graf napak glede na število enačb za drugo enačbo

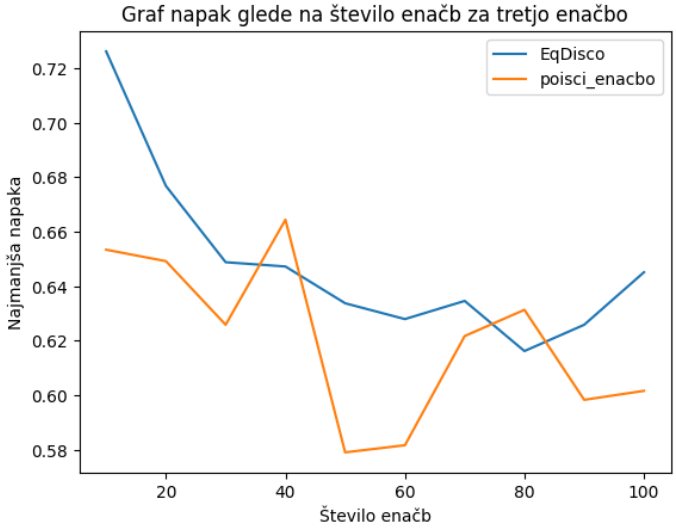

Figure 3: Graf napak glede na število enačb za tretjo enačbo

Iz grafov je jasno razvidno, da skonstruiran algoritem *poisci_enacbo* v večini primerov v povprečju odkrije enačbo z nižjo najnižjo napako kot *Eq.Disco*. V nekaterih primerih sicer še vedno odkrije enačbe z višjo napako, vendar to lahko pripišemo temu, da smo testirali absolutno premajhno število enačb, eksperiment ponavljali premalokrat in da ker je algoritem verjetnosti, se konvergenca v nekaterih primerih ob premajhnem vzorcu enostavno ni zgodila dovolj hitro. Če bi imeli večjo procesorsko moč, ter več časa bi seveda lahko enostavno pokazali, da algoritem *poisci_enacbo* k nižjim enačbam konvergira precej hitreje kot *Eq.Disco*. To se je pokazalo tudi, ko smo testirali pravilnost algoritma, saj je pravilne enačbe našel precej hitreje. Medtem, ko je *Eq.Disco* za vse tri enačbe potreboval približno 10min, je algoritem *poisci_enacbo* vse pravilne enačbe našel v manj kot minuti (običajno v 30-40s). Algoritem je tako že precej optimiziran, gotovo pa bi se ga dalo izboljšati. Opazili smo namreč, da je njegova hitrost odkrivanja enačbe še vedno precej odvisna začetne razporeditve verjetnosti. Ob neugodnih začetnih vrednostih in majhni nastavitvi spremembe, lahko še vedno precej podaljšamo čas iskanja. Če pa so začetni približki razumni in velikost spremembe nekoliko večja (recimo 0.05), pa uspe enačbe najti precej hitreje.