2. Domača naloga
UČENJE IZ PODATKOVNIH TOKOV

Študijski program Matematika (MAG)
Matija Kerkoč (27192017)

25. april, 2021

Uvod

Za izvedbo domače naloge bomo uporabljali program MOA. Za pripravo podatkov in izris grafov pa kar orodje Google Sheets, ki je prosto dostopno na spletu.

Poglejmo si najprej s kakšnimi podatki imamo opravka. Imamo stolpca `vhod1` in `vhod2`, ciljna spremenljivka pa je `izhod`. Podatkovni tok je sestavljen iz 10000 primerov. Predvidevamo, da podatki opisujejo nek sistem, pri katerem lahko uravnavamo spremenljivki `vhod1` in `vhod2`, izhodna pa je od njiju odvisna.

1 Priprava podatkovnega nabora

Poglejmo kako ustrezno pripravimo podatkovni nabor. Najprej moramo odstraniti atribut `primer`, saj je to zaporedna številka primera in vedno naraščajoč, torej kot tak, neprimeren za učenje na podatkovnih tokovih.

Ker želimo opisati dinamiko sistema, bodo verjetno nekatere vrednosti odvisne od prejšnjih. Izberemo okno velikosti 3 in namesto stolpcev `{vhod1, vhod2}` dodamo stolpce `atribut-3`, `atribut-2` in `atribut-1` za oba izmed atributov `{vhod1, vhod2}`. Za atribut `izhod` pa stolpec ohranimo ter dodamo stolpce `izhod-3`, `izhod-2` in `izhod-1`. Da je to pravilen pristop se prepričamo tako, da uporabimo enake parametre na zamaknjenem in nezamaknjenem naboru. Kot opazimo je rezultat na nezamaknjenem podatkovnem naboru veliko slabši (v smislu večje napake).

Postavlja se vprašanje kaj naredimo s prvimi in zadnjimi tremi primeri (ti niso popolni). Odločimo se, da bom prve in zadnje tri nepopolne primere pobrisali. Načeloma bi lahko prve tri tudi dopolnili z isto vrednostjo, vendar potem dobimo ogromno napako pri prvi evalvaciji modela, saj takrat model še ni naučen in vrne "neumno" vrednost. V nobenem primeru pa se celokupno gledano rezultati ne bodo kaj veliko spremenili, zato izberemo brisanje.

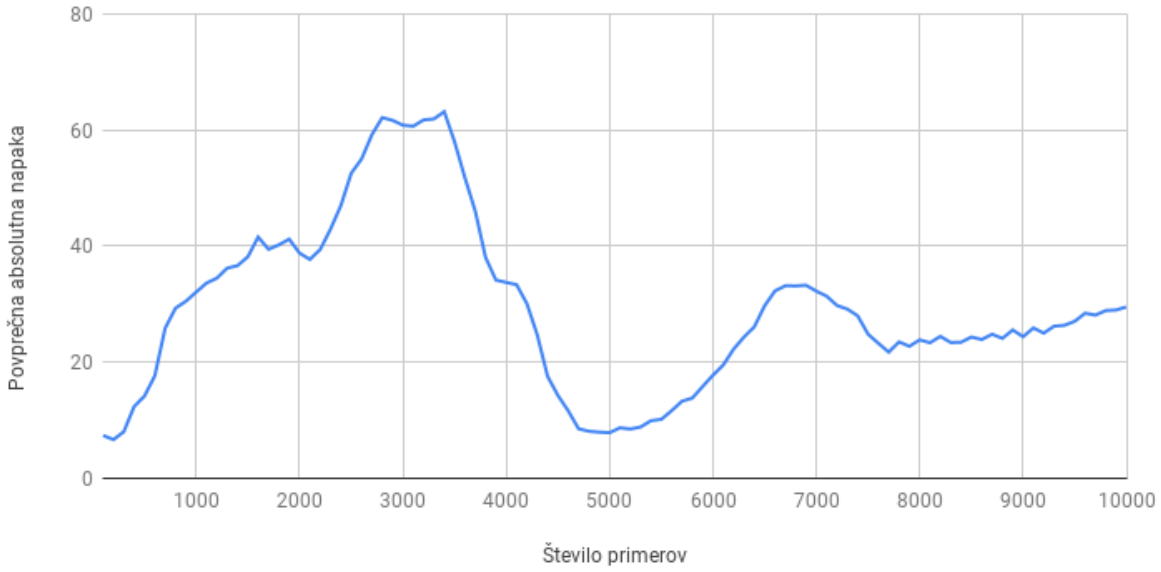
Nato podatkovni nabor pretvorimo v ARFF datoteko, tako da dodamo opis, glavo in nanizamo podatke. Shranimo s končnico `.arff`. Za delo sem v nadaljevanju uporabljal datoteko `dinamicni_lag.arff`. Pretvorbo v ARFF datoteko lahko opravimo tudi s pretvornikom tipov datotek. Primer online pretvornika je recimo [1].

2 Učenje modela

V MOA izberemo nalogo regresijse in nastavimo parametre. Izberemo `WindowRegressionPerformanceEvaluator`, kot metodo učenja izberemo FIMT-DD drevo z delitvenim kriterijem `VarianceReductionSplitCriterion`. Parametre metod zaenkrat pustimo kar privzete. Uporabimo podatkovni tok iz datoteke `dinamicni_lag.arff`, ki smo jo prej pripravili. Kot metodo evaluacije pa izberemo `WindowRegressionPerformanceEvaluator`. Spremenimo še parametra `instanceLimit=100,000`, `sampleFrequency=100`, ostale pa pustimo na privzetih vrednostih. Na kocu napovedi shranimo v `dinamicni_lag-out.pred` datoteko.

Opomba 1. Opazil sem, da se rezultati precej razlikujejo glede na to, ali imamo mogoče v ozadju odprte še kakšne druge aplikacije. Velika razlika je tudi, če recimo poženemo enako nalogo večkrat, saj bo v tem

primeru zadnji poskus najhitrejši. Razlike niso zanemarljive, saj so velike tudi par sekund, kar pomeni celo $10\% - 20\%$ celotnega časa (celotno učenje modela traja cca. 15 sekund).



Slika 1: Graf povprečne absolutne napake v odvisnosti od števila primerov.

3 Spremembe v naboru

Spremembe v naboru najlažje prepoznamo po tem, da se ob spremembi model ne prilagodi dovolj hitro in pride do zmanjšanja napovedne točnosti modela.

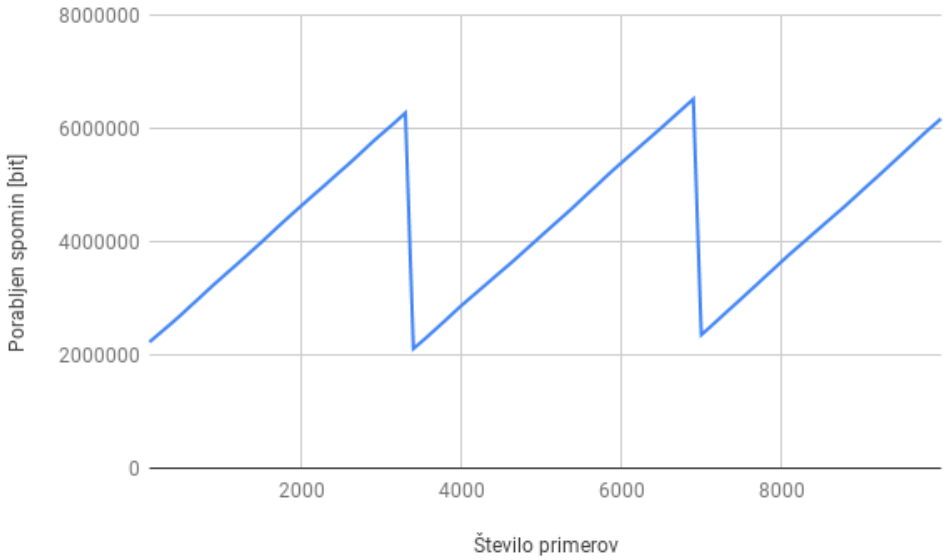
Kdaj se torej pojavijo spremembe? Najpreprosteje je na to vprašanje odgovoriti na sledeči način: ko absolutna povprečna napaka po padanju (ali stagnaciji) zopet začne naraščati. Kdaj se to zgodi v našem naboru? Iz grafa 1 lahko približno ocenimo take točke, za natančnejšo analizo pa si bomo pogledali vrednosti $|$`izhod(i)-napoved(i)`$|$, kjer je `i` zaporedna številka primera v naboru.

Prvo spremembo v naboru zaznamo okoli primera 144. Izgleda namreč, da se je model pred tem primerom že začel učiti (napaka je padala) vendar verjetno model ni še ujel točnega koncepta podatkov, zato napaka zopet naraste.

Drugo in tretjo spremembo v naboru zaznamo okoli primera 2093 in primera 5038. V teh primerih se po spremembi nabora model ne uči. Napaka namreč narašča. V resnici sta to najpomembnejši spremembi v naboru, saj model po spremembi v naboru ni več uporaben za napovedi. Tukaj se je res spremenil tok podatkov. Po teh dveh spremembah se je model zopet začel učiti šele po posodobitvi FIMT-DD drevesa.

Nazadnje si oglejmo še, kaj se dogaja po primeru 7700 Opazimo, da napaka konstantno narašča. Če pogledamo vrednosti $|$`izhod(i)-napoved(i)`$|$ po tem primeru opazimo, da absolutna razlika napak narašča in pada ter se zadržuje nekje v obsegu med 0 in 50. Zdi pa se, da v naboru konstantno prihaja do sprememb na vsakih $50 - 100$ primerov in model se ne more tako hitro prilagoditi na spremembe v naboru. Navkljub slabi točnosti, tukaj ne opazimo večjih skokov med vrednostni spremenljivk `vhod1` in `vhod2`.

Zanimivo je raziskati tudi, kaj se zgodi v primeru 3999 ("špica" ki jo opazimo na grafu 5). Če pogledamo v rezultate vidimo, da je v tem primeru izviren izhod enak -721.478394918394, napovedan izhod pa -400.278. Spremenljivka `vhod1` se je takrat spremenila iz 0.152395077307752 na -0.0052436173528479, spremenljivka `izhod2` pa iz -0.484778945865113 na -0.985013342600746, kar je ogromen preskok v primerjavi z ostalimi spremembami vhodnih spremenljivk. Predpostavli smo namreč, da se vhodne spremenljivke spreminjajo "zvezno" (kar pomeni, da so spremembe majhne in nimajo velikih skokov). Kar je zanimivo pri tem primeru



Slika 2: Graf porabe spomina. Opazimo lahko, da padec porabe spomina sovpada s posodobitvijo drevesa (pri 3400 in 7000 primerih).

je to, da je model očitno naredil le eno slabo napoved in takoj ujel naslednjo, saj po tem primeru razlike zopet padajo.

4 Drevesa

Do prve delitve posodobitve pride med 3300 in 3400 primeri, do druge pa med 6900 in 7000 primeri. Poglejmo si sedaj drevesa, ki se jih FIMT-DD nauči. V ta namen tukaj uporabimo metodo `LearnModelRegression` in nastavimo parameter `maxInstances` na 2000, 5000 in 7000.

2000 primerov: Dobimo drevo z enim samim listom, ki nam vrne:

> Leaf [class:izhod] = 0.5510 * [att 1:vhod1-3] -0.0918 * [att 2:vhod1-2] -0.4977 * [att 3:vhod1-1] -0.2348 * [att 4:vhod2-3] + 1.0552 * [att 5:vhod2-2] -0.8471 * [att 6:vhod1-1] + 0.5199 * [att 7:izhod-3] + 0.4643 * [att 8:izhod-2] + 0.4789 * [att 9:izhod-1] + -0.2164770815730056

5000 primerov: V tem primeru dobimo drevo globine dva:

- if [att 9:izhod-1] $\leq$ -410.2190892:

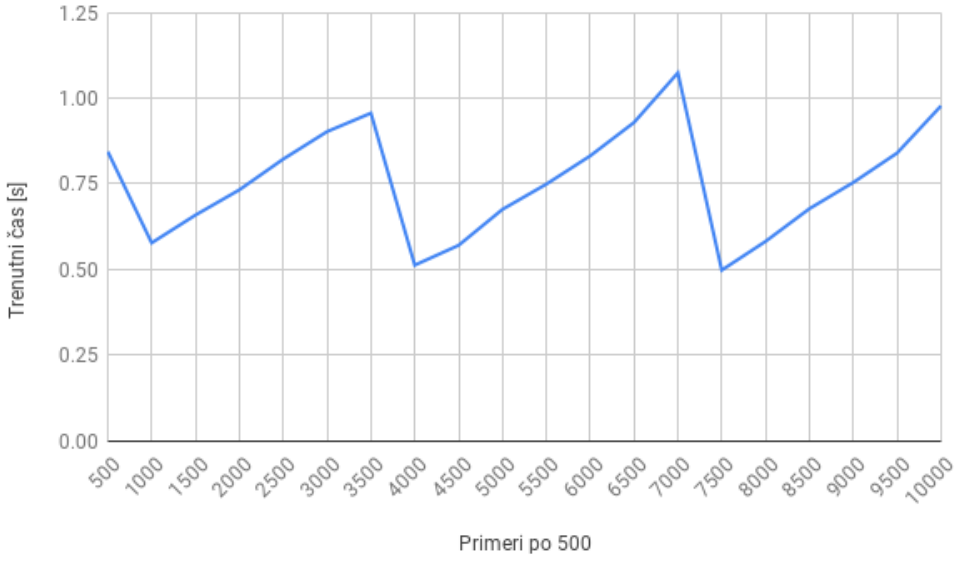
  > Leaf [class:izhod] = 0.5504 * [att 1:vhod1-3] -0.0915 * [att 2:vhod1-2] -0.4959 * [att 3:vhod1-1] -0.2309 * [att 4:vhod2-3] + 1.0618 * [att 5:vhod2-2] -0.8360 * [att 6:vhod1-1] + 0.3317 * [att 7:izhod-3] + 0.2761 * [att 8:izhod-2] + 0.2908 * [att 9:izhod-1] + -0.10727607798746513

- if [att 9:izhod-1] $>$ -410.2190892:

  > Leaf [class:izhod] = 0.5249 * [att 1:vhod1-3] -0.1168 * [att 2:vhod1-2] -0.5216 * [att 3:vhod1-1] -0.2371 * [att 4:vhod2-3] + 1.0554 * [att 5:vhod2-2] -0.8445 * [att 6:vhod1-1] + 0.2766 * [att 7:izhod-3] + 0.2209 * [att 8:izhod-2] + 0.2354 * [att 9:izhod-1] + -0.027197571200637965

7000 primerov: V zadnjem primeru pa dobimo drevo globine 3:

- if [att 9:izhod-1] $\leq$ -410.2190892:
  - if [att 9:izhod-1] $\leq$ -1369.61664:



Slika 3: Graf trenutnega časa učenja na vsakih 500 primerov. Tudi v tem primeru lahko opazimo sovpadanje porabe časa in posodobitve drevesa. Odstopanje pri prvih 500 primerih se je zgodil zaradi tega, ker MOA na začetku potrebuje nekaj časa, da se začne izvajati.

Leaf [class:izhod] = 0.5872 * [att 1:vhod1-3] -0.0546 * [att 2:vhod1-2] -0.4591 * [att 3:vhod1-1] -0.2327 * [att 4:vhod2-3] + 1.0626 * [att 5:vhod2-2] -0.8328 * [att 6:vhod1-1] + 0.3067 * [att 7:izhod-3] + 0.2511 * [att 8:izhod-2] + 0.2659 * [att 9:izhod-1] + 0.01438973530733139

- if [att 9:izhod-1] > -1369.61664:

  Leaf [class:izhod] = 0.5880 * [att 1:vhod1-3] -0.0538 * [att 2:vhod1-2] -0.4583 * [att 3:vhod1-1] -0.2327 * [att 4:vhod2-3] + 1.0625 * [att 5:vhod2-2] -0.8328 * [att 6:vhod1-1] + 0.3069 * [att 7:izhod-3] + 0.2514 * [att 8:izhod-2] + 0.2661 * [att 9:izhod-1] + 0.012191706669391572

- if [att 9:izhod-1] > -410.2190892:

  Leaf [class:izhod] = 0.5249 * [att 1:vhod1-3] -0.1168 * [att 2:vhod1-2] -0.5216 * [att 3:vhod1-1] -0.2371 * [att 4:vhod2-3] + 1.0554 * [att 5:vhod2-2] -0.8445 * [att 6:vhod1-1] + 0.2766 * [att 7:izhod-3] + 0.2209 * [att 8:izhod-2] + 0.2354 * [att 9:izhod-1] + -0.027197571200637965

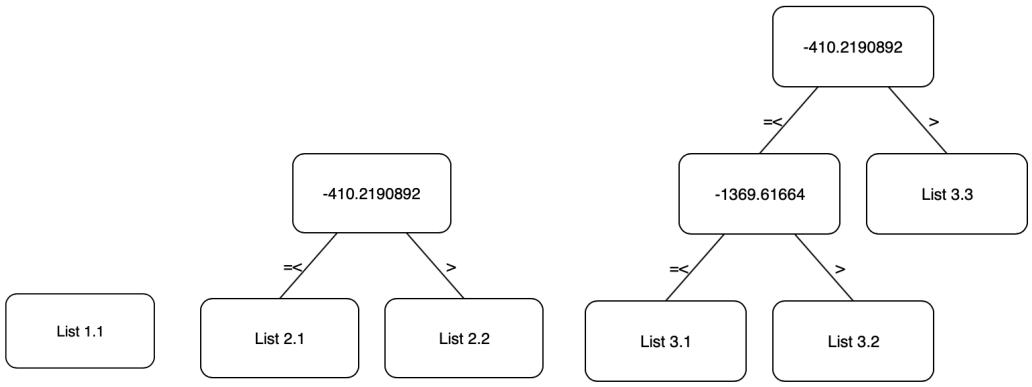
Če ta drevesa narišemo, dobimo drevesa prikazana na sliki 4. Kot je bilo pričakovati, dobimo za manjše število primerov tudi manjša oziroma bolj preprosta drevesa, saj se z učenjem modela drevo posodablja oziroma dodatno deli.

5 Ujemanje signalov

Poglejmo si sedaj ujemanje izvirnega signala ter njegove napovedi. Na sliki 5 imamo graf obeh signalov. Komentar ujemanja oziroma neujemanja signalov smo že razložili v poglavju 3, kjer obravnavali spremembe v naboru.

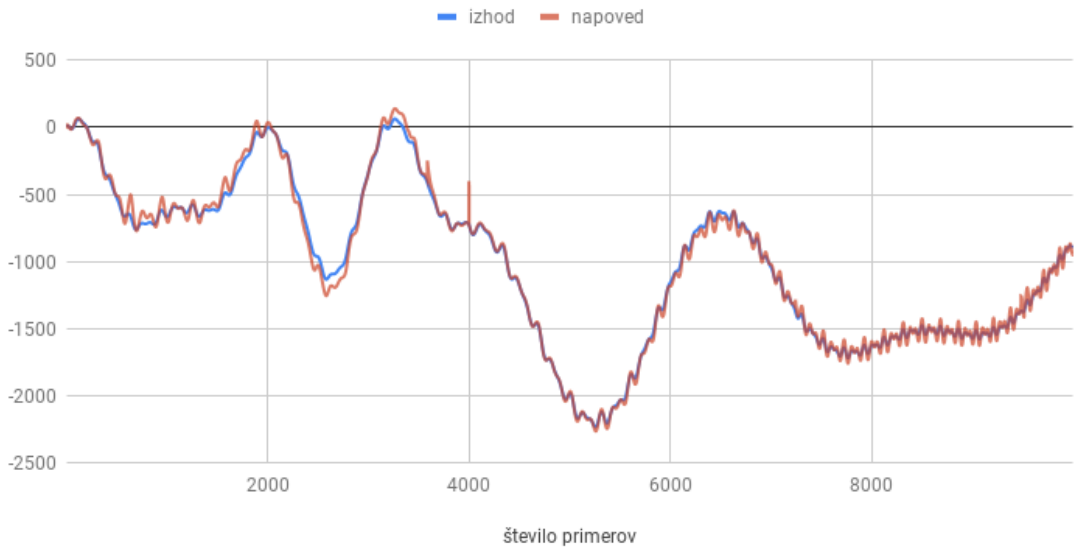
Zaključek

Za konec si poglejmo še nekaj idej za izboljšavo našega modela.



(a) (b) (c)

Slika 4: (a) Drevo, ki se ga nauči po 2000 primerih. (b) Drevo, ki se ga nauči po 5000 primerih. (c) Drevo, ki se ga nauči po 7000 primerih. Opazimo, da smo zamenjali levo poddrevo. Oznake listov so definirane na naslednji način: prva številka predstavlja zaporedno številko drevesa po 2000/5000/7000 primerih, druga pa zaporedno številko lista v drevesu.



Slika 5: Graf ujemanja napovedi modela in izvrnega signala. Na tem grafu lahko na oko lepo vidimo, kje je model "zadel" napoved in kje se je v napovedi uštel.

1. Smiselno bi bilo preveriti ali je model boljši za večji zamik podatkov. Če ima časovna odvisnost večji vpliv kot smo predvidevali (zamik za 3) potem bo naš model profitiral. V končni verziji sem se odločil, da pustim zamik 3, saj je ta največkrat uporabljen v literaturi in pri delu s podatkovnimi tokovi. Zato se taka izbira zdi smiselna (v primeru uporabe večjega časovnega zamika lahko model celo "zmedemo").

2. Opazil sem tudi, da MOA vrne napoved za izhod v točnosti na tri decimalke. Ker so originalni podatki točnosti več kot 10 decimalk bi se lahko zgodilo, da se kje pojavijo večje razlike. Vseeno to v našem modelu ne bi naredilo večje razlike, saj so napake dovolj velike in se zaokroževanje na 3 decimalke ne pozna. Vseeno pa moramo biti pri določenih modelih pozorni tudi na to.

References

[1] Pretvornik iz .csv v .arff datoteko. Dostopno na: `https://ikuz.eu/csv2arff/`