Poročilo za 3. Domačo nalogo pri predmetu NSU

Matija Kerkoč (27192017)

23. maj, 2021

1 Relacijska drevesa

Za reševanje domače naloge sem izbral relacijska drevesa, oziroma njihovo implementacijo s paketom re3py. Koda je razdeljena v dva Jupyter notebooka: v `Priprava_podatkov.ipynb` imamo vse potrebno za predelavo podatkov v primerno obliko, v `Matija_Kerkoc_3DN-NSU.ipynb` pa imamo glavni del domače naloge. Pripravljene datoteke s podatki so tudi priložene, zato poganjanje prvega Jupyter notebooka ni potrebno.

1.1 Odstranitev podatkov

Iz podatkovja bomo odstranili tabelo COACHES. Tabela vsebuje atribute `coachid`, `firstname` in `lastname`. Za nas ta tabela nima nobene dodane vrednosti, saj naši modeli v imenu in priimku trenerja vidijo zgolj niz znakov in zato sta ta dva podatka ekvivalentna identifikatorju posameznega trenerja (`coachid`). Ker tabela ne vsebuje drugih atributov jo lahko odstranimo.

Ostali primeri "neustreznih" atributov (npr. `player_firstname`, `player_lastname`, ...) pa so le posamezne vrstice v tabelah in jih bomo ročno prepovedali v seznamu dovoljenih testov. Zakaj so imena neprimerni atributi? Poglejmo si primer: ekipe imajo v imenih imena mest, kjer so pozicionirane (npr. LAL = Los Angeles Lakers). Žal pa naš model iz imena Los Angeles Lakers ne zna direktno ugotoviti lokacije ekipe. Šele ko model prebere atribut `location`(LAL) = Los Angeles bo znal povezati ime ekipe z njeno lokacijo.

V tem razdelku omenimo še, da stolpcev ne bomo združevali. Zakaj taka odločitev? Prvi razlog je ta, da model deluje čisto ok tudi brez dodatnega dela, ki si ga z združevanjem naprtimo. Nekoliko bolj utemeljena razlaga pa je sledeča. Če pogledamo stolpca, ki podajata višino igralca (`h_feet` in `h_inches`) ju lahko združimo v nov stolpec `h_cm`[1]. Recimo, da imamo test: "ali obstaja igralec, ki je višji od 2 metrov" ($\exists$ igralec. višina(igralec) $\geq$ 2 m)? Tak test že znamo simulirati z agregati kot

$$\texttt{štej}\big(\{\text{ igralec} \mid \texttt{h\_cm}(\text{igralec}) \geq 200\ \}\big) > 0.$$

Vendar pa lahko enak test simulirano na nepredelanih stolpcih. Edina razlika je v tem, da bomo morali uporabili dva testa

$$\texttt{štej}\Big(\big(\{\text{ igralec} \mid \texttt{h\_feet}(\text{igralec}) \geq 6\}\big)\ \texttt{AND}\ \big(\{\text{ igralec} \mid \texttt{h\_inches}(\text{igralec}) \geq 7\}\big)\Big) > 0.$$

Kot pa bomo videli to v našem primeru ne bo veliko vplivalo na gradnjo dreves, saj bodo drevesa že z osnovnimi testi precej majhna.

1.2 Pretvorba podatkov

Paketu re3py moramo podatke podati v pravilni obliki. V našem primeru moramo konstruirati tri nove datoteke

[1] Za pretvarjanje velja 1 feet = 30,48 cm in 1 inch = 2,54 cm.

1. datoteko, kjer podamo vse možne pare identifikatorskih ter ostalih atributov (`basket_opis.txt`),
2. datoteko, kjer imamo podane ekipe (ID ekip) in njihove ciljne vrednosti (`basket_ciljna.txt`) ter
3. datoteko, kjer naštejemo dovoljene relacije, dovoljene agregate in kako v teste dodajamo nove koščke (`basket.s`).

Če smo leni, te tri datoteke poberemo kar iz datoteke `re3py-master`. Ker pa to ni namen domače naloge bomo te datoteke zgenerirali sami.

Razložimo sedaj konstrukcijo datotek. Pri datoteki `basket_opis.txt` zgolj gremo po vseh tabelah in izpišemo vse možne pare, kjer na prvem mestu podamo vrednosti iz identifikatorskega stolpca (`playerid`, `team` ali `coachid`) na drugem pa se nato pomikamo čez vse možne vrednosti ostalih atributov.

Za konstrukcijo datoteke `basket_ciljna.txt` gremo po tabeli TEAMS in poberemo vse pare vrednosti atributov `team` in `league`.

Na koncu konstruiramo še datoteko `basket.s`, kjer naštejemo vse možne relacije, vse možne agregate ter vse možne načine dodajanja novih koščkov v teste.

Opomba. Datoteko `basket.s` sem konstruiral tako, da sem najprej konstruiral datoteko, potem pa še ročno popravil nekatere vrstice. Npr. Popravil sem nekatere relacije in teste ter zakomentiral neprimerne teste. Pisanje kode za te popravke bi mi vzelo več časa, zato sem se odločil za to možnost.

Postavlja se vprašanje, zakaj uporabljamo le dvojiške relacije? Načeloma, bi lahko stvar zakomplicirali in (tako kot na vajah) uporabljali relacije, ki sprejmejo več argumentov (angl. *arity*). Vendar že s temi preprostimi relacijami (in testi) dobimo zelo dobro natančnost, zato v našem primeru nima smisla vpeljevati dodatnih relacij.

Drugo vprašanje, ki si ga zastavimo je, zakaj v testih ne uporabljamo konstant? Odgovor je zopet podoben kot zgoraj. Splošen test oblike

$\exists$ `player, position.` `playoffs`(T, `player`, `position`) (katere pozicije je imela ekipa T pokrite v končnici?)

je pravzaprav manj specifična verzija testa

$\exists$ `player.` `playoffs`(T, `player`, C) (ali je za ekipo T v končnici igral center?)

Torej lahko v drugem primeru vprašamo bolj specifična vprašanja (in upamo, da nam to pomaga pri klasifikaciji). Taki testi bi bili primerni, če bi drevesa v našem modelu rasla čez neko "smiselno mejo". Ker pa so v našem primeru drevesa plitva, se nam ni treba posluževati takih testov.

Če torej primerjamo naš model s tistim iz vaj na podatkovju YELP, nismo sploh potrebovali izkoristiti točke 1) in točke 3) v moči relacijskih dreves. Izkoristili pa smo točko 2), ko uvedemo agregate.

1.3 Metoda učenja

Za metodo učenja bomo izbrali relacijska drevesa. Med propozicionalizacijo in relacijskimi drevesi, so slednja logična izbira, saj nam prihranijo ogromno mukotrpnega premetavanja in lepljenja tabel. Delo z njimi je pravzaprav intuitivno, enkrat ko pripravimo vse potrebne datoteke, treniranje in testiranje modela poteka vedno na podoben način.

Druga smiselna izbira bi bila metoda vpetja vozlišč grafov (node2vec). Kot pa smo videli na vajah metoda ni dosegala tako visokih točnosti, hkrati pa zahteva veliko dela, saj moramo konstruirati vsa vpetja za podane tabele. Zato v tem primeru to ni smiselna izbira.

Za metodo klasifikacije smo izbrali naključni gozd, v prvem primeru s 5 drevesi, v drugem pa s 30 drevesi. Parametre smo nastavili na "smiselne" vrednsoti, kot hevristiko pa smo uporabili Gini indeks.

1.4 Točnost metode na testni množici

Kot je bilo že omenjeno, v navodilih domače naloge, ciljamo na natančnost nad 95%. Če smo pravilno konstruirali teste, bi morali že v začetku dobiti željeno natančnost. S spreminjanjem `random_seed` pa hitro najdemo takega, ki nam da točnost 100%.

Omeniti je potrebno tudi, da so drevesa, ki jih konstruira naš model, majhna (plitva/nizka). V večini primerov dobimo drevesa globine 3 ali 4, v nekaterih primerih celo 2.

1.5 Koristnosti delov podatkov

Poglejmo si koristnosti posameznih stolpcev za napovedovanje ciljne spremenljivke. Izpisi pomembnosti relacij so shranjeni v datotekah `genie3_rf.txt` in `genie3_rf30.txt`, izpisi konstruiranih dreves pa v datotekah `rel_rf.txt` in `rel_rf30.txt`. Na sliki 1 imamo izpis pomembnosti relacij, seštevka pomembnosti posameznih relacij ter pomembnost agregatov.

```
Attributes
coach_season_team[Y,X] :  4.04775e-01 ; iterations: [ 8.52238e-02 ,  7.77006e-02 ,  7.77006e-02 ,  7.71902e-02 ,  8.69599e-02 ]
team_season_o_asts[X,Y]:  8.15972e-03 ; iterations: [ 8.15972e-03 ,  0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ]
team_season_won[X,Y]   :  7.54438e-03 ; iterations: [ 0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ,  7.54438e-03 ,  0.00000e+00 ]

Attributes summed
coach_season_team      :  4.04775e-01 ; iterations: [ 8.52238e-02 ,  7.77006e-02 ,  7.77006e-02 ,  7.71902e-02 ,  8.69599e-02 ]
team_season_o_asts     :  8.15972e-03 ; iterations: [ 8.15972e-03 ,  0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ]
team_season_won        :  7.54438e-03 ; iterations: [ 0.00000e+00 ,  0.00000e+00 ,  0.00000e+00 ,  7.54438e-03 ,  0.00000e+00 ]

Aggregators
count                  :  4.04775e-01 ; iterations: [ 8.52238e-02 ,  7.77006e-02 ,  7.77006e-02 ,  7.71902e-02 ,  8.69599e-02 ]
sum                    :  1.57041e-02 ; iterations: [ 8.15972e-03 ,  0.00000e+00 ,  0.00000e+00 ,  7.54438e-03 ,  0.00000e+00 ]
```

Slika 1: Izpis pomembnosti relacij, seštevka pomembnosti ter pomembnosti agregatov za napovedovanje ciljnega stolpca `league`.

V tem primeru je najpomembnejša relacija `coach_season_team`(Y, X), sledita pa ji relaciji `team_season_o_asist`(X, Y) (število asistenc, ki jih je dosegla ekipa v sezoni) in `team_season_won`(X, Y) (število zmag ekipe v sezoni). Vendar pozor! Prvi test je oblike `coach_season_team`(new, old), če pogledamo relacije imamo zapisano `coach_season_team`(CoachId, TeamId), zato moramo to relacijo brati kot: fiksen ("old" = X) TeamId in spreminjajoč ("new" = Y) CoachId. Take teste razumemo kot vprašanje: "za izbrano ekipo X, ali jo je treniral trener Y?".

Na splošno uvedba testov oblike

`test`(new, old)

pri relacijah, kjer jih lahko uvedemo (to so tiste relacije, ki povezujejo tabele na "dveh koncih"), pozitivno vplivajo na globino dreves ter klasifikacijsko točnost. Torej je uvedba takih testov smiselna, saj le tako v polnosti izkoristimo moč, ki nam jo omogočajo gosto povezane tabele.

Kaj se zgodi v primeru povečave števila dreves v naključnem gozdu. Na sliki 2 lahko opazujemo izpise pomembnosti za metodo `random_forest` na 30 drevesih. Opazimo, da so zgoraj opisani testi oblike `test`(new, old) pomembni pri klasifikaciji, kar nam potrjuje smiselnost njihove uvedbe.

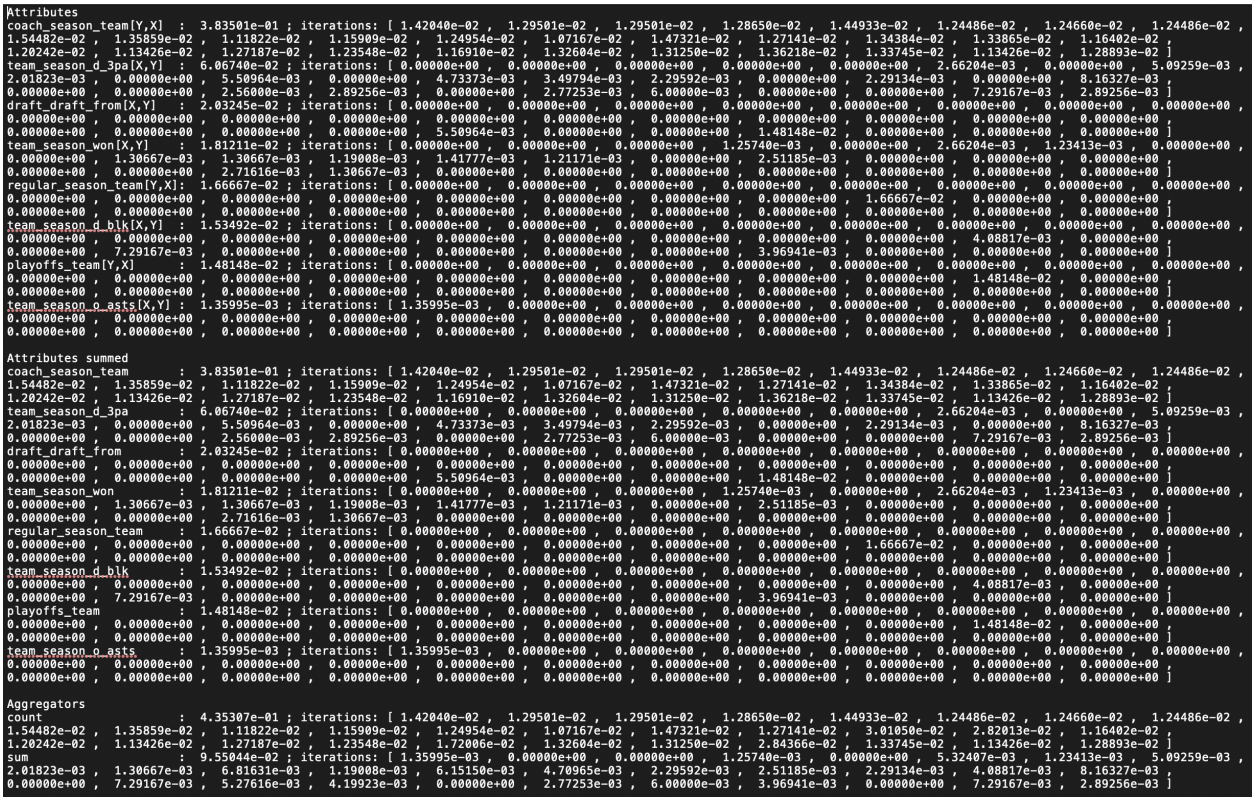
Druga stvar, ki jo opazimo je ta, da so testi v večini prisotni le v nekaj iteracijah, kar pomeni, da model vedno uporablja test `coach_season_team`(Y, X) ostale pa uporabi le v gradnji nekaterih dreves.

Zaključek

Ker tudi sam spremljam košarko, sem se nekoliko bolj poglobil v samo nalogo. Ugotovil sem, da oznaka `'A'` pomeni ligo ABA (American Basketball Association), oznaka `'N'` pa ligo NBA (National Basketball Association). Liga ABA je obstajala od 1967 do 1976, liga NBA pa od 1946 do danes (v več različicah). Zdi se, da so podatki verjetno razdeljeni tako, da bo model znal napovedovati ligo v kateri je ekipa igrala.

Če kar pogledamo prvo konstruirano drevo z metodo `random_forest` v primeru petih dreves:

Tree 1: Tree for `teams_league`, ['X0']:
IF (`'coach_season_team'`, ['Y1', 'X0'], count) > 0.5:
  YES
  IF (`'team_season_o_asts'`, ['X0', 'Y2'], sum) > 1161.5:
  YES
    return N (['A', 'N']: [0, 47])
  NO
    return A (['A', 'N']: [1, 0])
  NO



Slika 2: Izpis pomembnosti relacij, seštevka testov ter agregatov za napovedovanje ciljnega stolpca `league`, z metodo `random_forest`, kjer nastavimo število dreves na 30.

return A (['A', 'N']: [24, 0])

Najpomembnejši test je `IF ('coach_season_team', ['Y1', 'X0'], count)` > 0.5. V tem primeru je model verjetno našel povezavo med ID ekipe in trenerji, ki so trenirali to ekipo. Ker med letoma 1967 in 1976 trenerji niso hkrati trenirali ekip iz dveh različnih lig, sklepamo da bo ta podatek začrtal smernice klasifikacije tudi na ostalih časovnih intervalih (1946–1967 ter 1976–2003).

Če si pogledamo še drugi test v tem primeru:

IF (`'team_season_o_asts'`, ['X0', 'Y2'], `sum`) > 1161.5:
  YES
    return N (['A', 'N']: [0, 47])
  NO
    return A (['A', 'N']: [1, 0])

Najprej povejmo, da kratica `o_ast` pomeni *opponent assists*, torej koliko asistenc je ekipa dovolila nasprotniku.

Kako bi torej razložili dobljen rezultat? Med športnimi navdušenci vlada splošno prepričanje, da se v NBA košarki ne igra obrambe. Torej bodo ekipe dovolile več asistenc nasprotnikom. Kot je razvidno iz zgrajenega testa, je ta rezultat smiseln. V primeru, ko bo število dovoljenih asistenc preseglo $1161,5$ v celotni sezoni, bo ekipa klasificirana kot `'N'`.

Zanimivo pa je, da se model ni naučil kakšnega drugega še bolj "očitnega" (očitnega nam ljudem) testa. Tukaj mislim predvsem na število točk (če se igra slabša obramba, je tudi število točk na obeh straneh višje), število trojk (vemo, da je v NBA ligi metanje trojk pogostejše kot v drugih ligah), ... Mogoče bi model s kakšno drugo nastavitvijo dovoljenih testov našel tudi kakšno izmed teh povezav.

Vendar, če nismo pikolovski je model našel "zelo dobre" relacije. Kaj pomeni "zelo dobre"? S kombinacijo le dveh (včasih treh ali celo enega) relacij je model zadel popolnoma vse klasifikacije na testni množici. Predvidevam, da je tak rezultat bolj posledica naših podatkov in predvsem očitne separacije med podatki v `'A'` ter podatki v `'N'`, kot pa res dobrega modela. Vendar to ne pomeni, da je tak način gradnje modela slab. Tekom seminarske naloge lahko najdemo vsaj tri namige, kako model izboljšati v primeru, da bili postavljeni pred kakšno težjo klasifikacijsko nalogo.

References

[1] Knjižnjica `re3py`, dostopna na: `https://github.com/re3py/re3py`

[2] Slovar košarkaških izrazov v angleščini, dostopno na: `https://www.nba.com/stats/help/glossary/`