STROJNO UČENJE (MACHINE LEARNING)

Žiga Kmecl

12. maj 2024

1 Uvod

Namen zadnje naloge je seznaniti se z osnovami strojnega učenja, saj se strojno učenje dandanes pojavlja vsepovsod na znanstvenih področjih. Ukvarjali se bomo z iskanjem signala Higgsovega bozona iz meritev, ki so bile pobrane iz članka v reviji Nature Communications.

Algoritme strojnega učenja delimo na tri vrste:

- Nadzorovano učenje
- Nenadzorovano učenje
- Stimulirano učenje

Pri nadzorovanem učenju algoritem sortira podatke v naprej pripravljene skupine, v nenadzorovanem pa jih ustvari kar sam. Stimulirano učenje je kompleksnejše, tu gre za umetno inteligenco v strožjem pomenu besede. Najosnovnejše je nadzorovano učenje, s katerim se bomo izključno ukvarjali pri tej nalogi.

Glavna prednost algoritmov strojnega učenja je pri obravnavi kompleksnih naborov podatkov. V nekem mnogodimenzionalnem prostoru, ko imam meritve z veliko parametri, je namreč zelo težko kar uganiti funkcijo, ki bi problem dobro opisala.

Princip delovanja mehanizmov strojnega učenja ni zelo kompleksen, vendar formalnih enačb tu ne bomo navajali, saj jih jaz kot avtor žal ne razumem ravno najbolje. V osnovi gre pri procesu za minimizacijo *funkcije izgube*: algoritem skozi iteracije sestavlja neko svojo funkcijo, ki na podlagi večdimenzionalnih podatkov izračuna ciljne vrednosti. Funkcija izgube nato izračunane vrednosti primerja z dejanskimi in je manjša, če se vrednosti lepše ujemajo. Vse to lahko razumemo kot nekakšen ekvivalent izračunu RMSE vrednosti, ki ga pri bolj preprostem naboru meritev delamo na roke.

Pri nalogi bomo uporabljali dva različna algoritma za strojno učenje, *boosted decision tree (BDT)* in *(deep) neural network ((D)NN).*

Pri algoritmu odločevalnega drevesa gre morda za najbolj preprosto implementacijo strojnega učenja. Algoritem po korakih deli prostor parametrov z vedno več mejami in na ta način okoli podatkov ustvarja območja, ki se skladajo s kategorijami, v katere dejansko spadajo podatki. Pri odločevalnih drevesih hitro pride do tega, da prostor postane preveč fragmentiran in so meje pretirano prilagojene učnemu setu podatkov. To kompenziramo z malenkost bolj sufisticiranimi metodami, na primer algoritmom pospešenega odločevalnega drevesa. Tu z utežmi obtežimo podatke, ki jih odločevalno drevo pušča v napačnih kategorijah. Meje se posledično nekoliko spremenijo. Na koncu različno ustvarjene meje združimo skupaj v rezultat, ki je boljši od tistega, ki ga da klasično odločevalno drevo.

Nevronske mreže delujejo približno ekvivalentno biološkim nevronskim mrežam. Kombinacije vhodnih parametrov, obteženih z utežmi, vstopajo v nevrone oz. perceptrone, to so neke vmesne funkcije, ki rezultat pošljejo naprej kot vstopni parameter za nov sloj nevronov. Postopek se nadaljuje skozi celotno globino nevronske mreže in na koncu dobimo ven končni rezultat. Ta algoritem popravlja napoved tako, da spreminja uteži vhodnih parametrov na način, ki minimizira že prej omenjeno funkcijo izgube.

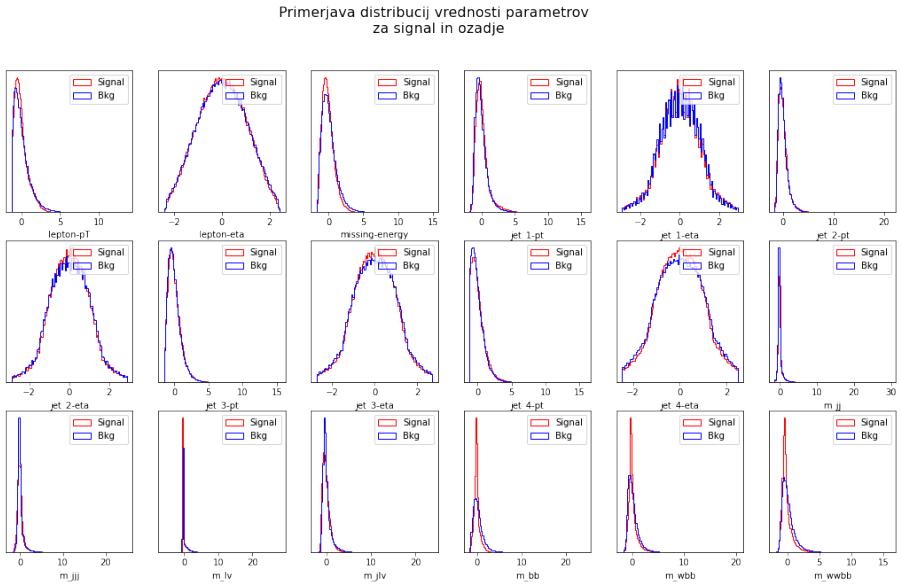
Glavni fokus naloge je primerjati oba algoritma v smislu časovne učinkovitosti ter za oba ugotoviti, kako povečevanje števila iteracij oz. epoh zmanjšuje funkcijo izgube. Izrisati želimo tudi ROC (Receiver Operating Characteristic) krivulje, to so krivulje, ki prikazujejo razmerje med pravilno identificiranimi in napačno identificiranimi pozitivnimi napovedmi.

2 Rešitev

Pri nalogi uporabljamo Pythonovi knjižnici *sklearn* in *keras*.

Ta razdelek moram nujno začeti z opozorilom, da mi je strojno učenje popolnoma novo področje. Zanašal sem se na *sklearn* paket in velik del algoritmov je zame ostal "black box". Zaradi tega in ker sem ogromno časa porabil, da sem poskušal razumeti, kaj sploh počnem, česar se v poročilu ne da dobro odraziti, je ta dokument krajši in opremljen z bolj skopimi komentarji, kot bi verjetno moral biti.

Za začetek si poglejmo distribucijo po parametrih za signal in ozadje

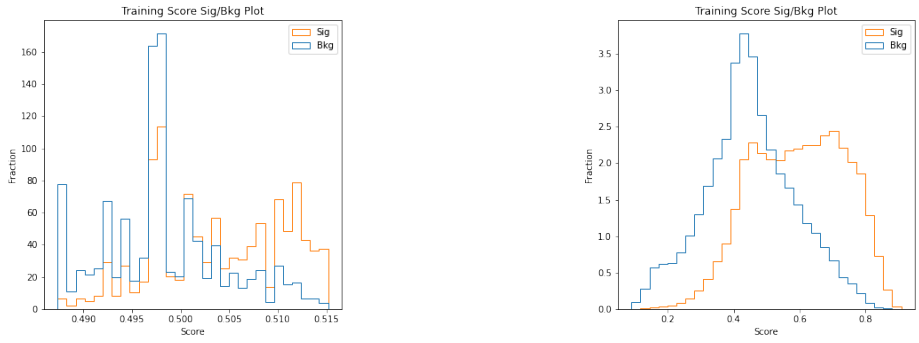


Slika 1: Primerjava signala in ozadja po parametrih.

Vizualno se zdi, da do največjih razlik prihaja predvsem v zadnji vrstici.

2.1 Pospešeno odločevalno drevo

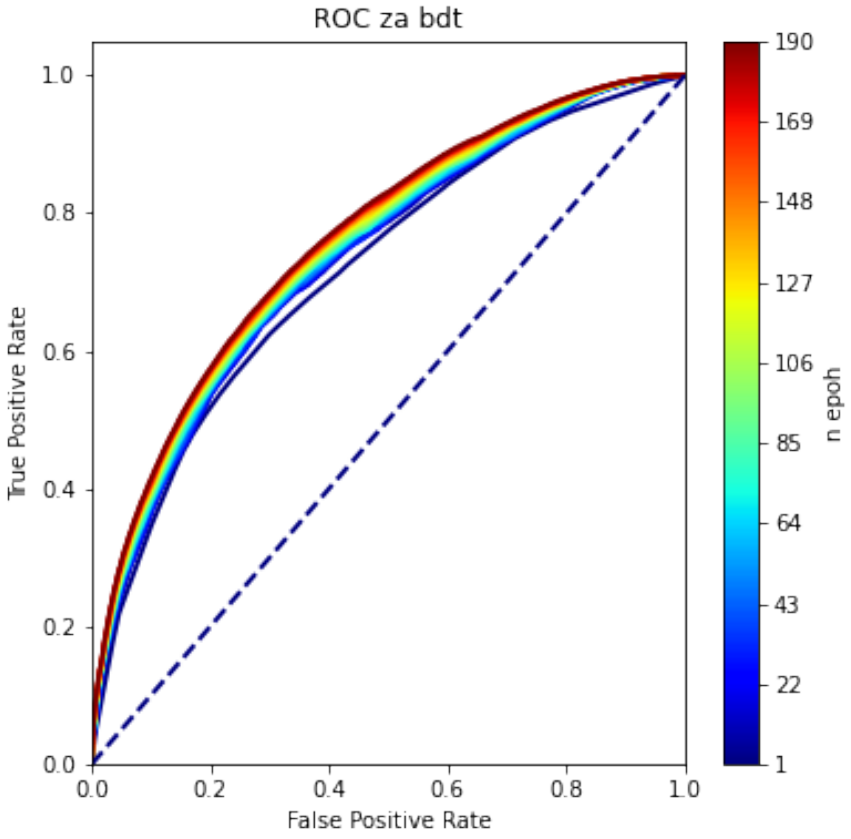
Ker se mi je zdel bdt preprostejši algoritem, sem se odločil začeti tu.



(a) 4 epohe (b) 200 epoh

Slika 2: Score po 4 in 200 epohah.

Izkaže se, da je potrebno zelo malo epoh, da algoritem v grobem loči signal od ozadja, saj je že po 4 epohah vidna razlika v porazdelitvah za signal in ozadje. Da dobimo jasno ločitev, pa je potreben kakšen red velikosti več epoh.



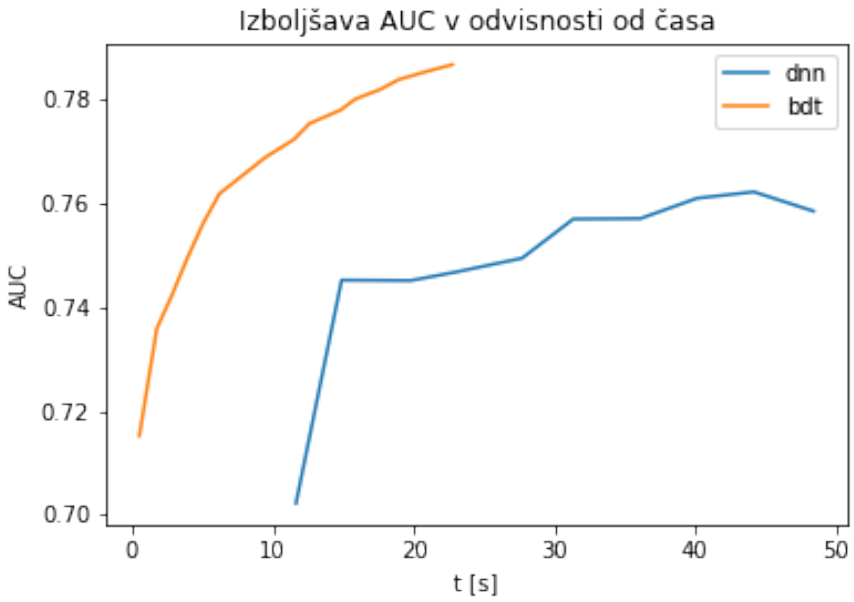
Slika 3: ROC krivulja v odvisnosti od števila epoh.

Lahko izrišemo še ROC krivulje za različno števil epoh. Zanimivo je, da že iz starta algoritem da precej dobro krivuljo, ki se potem vizualno pravzaprav ne spreminja tako bistveno.

2.2 Nevronske mreže

Ker z rezultati zgoraj nisem bil najbolj zadovoljen, saj nisem bil prepričan, ali so normalni, in se mi je zdelo, da ne dobivam dovolj dobrih modelov, sem se odločil kar premakniti na naslednji algoritem.

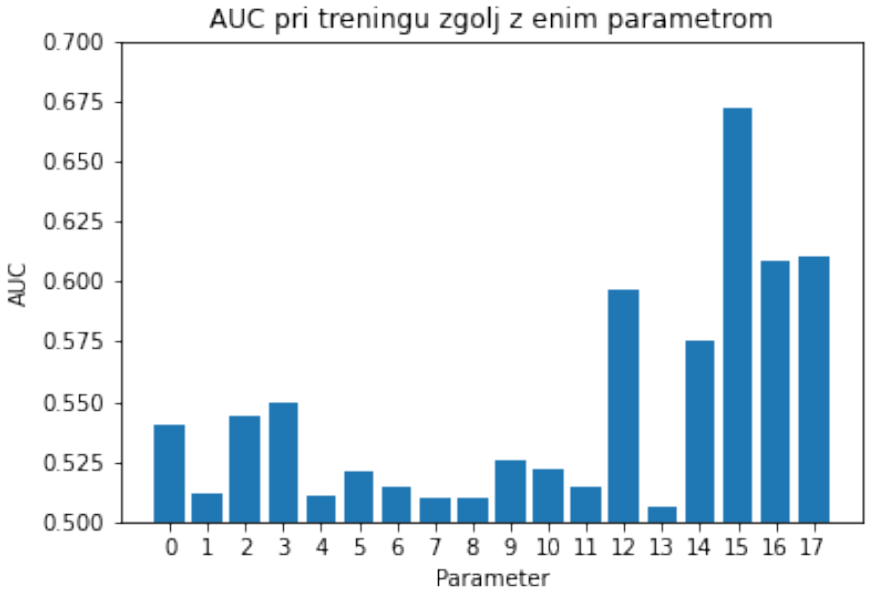
Za začetek poglejmo, kako se hitrost tega algoritma primerja s prejšnjim.



Slika 4: Primerjava AUC za obe metodi.

Verjetno nepresenetljivo se izkaže, da nevronske mreže delujejo veliko počasneje. Že za eno epoho je na primer algoritem na mojem računalniku porabil več kot 10 sekund. Nekoliko nepričakovano se pokaže, da se model pri nevronskih mrežah pravzaprav izboljšuje počasneje kot pri odločevalnih drevesih. Zdelo se mi je, da bodo nevronske mreže v tem pogledu boljše, ker gre pač za bolj sufisticirano metodo. Tu je popolnoma mogoče, da bi se na daljših časovnih skalah izkazalo, da je moja predpostavka pravilna, in zgolj nisem imel dovolj potrpljenja.

Ugotoviti sem želel še, kateri parametri vhodnih podatkov so najbolj ključni za ločevanje med signalom in ozadjem. V ta namen sem model učil na samo enem podatku naenkrat. Zaradi mojih časovnih omejitev sem se žal moral omejiti zgolj na 3 epohe, kar verjetno ni dovolj za jasne zaključke, verjetno pa vseeno lahko služi za oris.



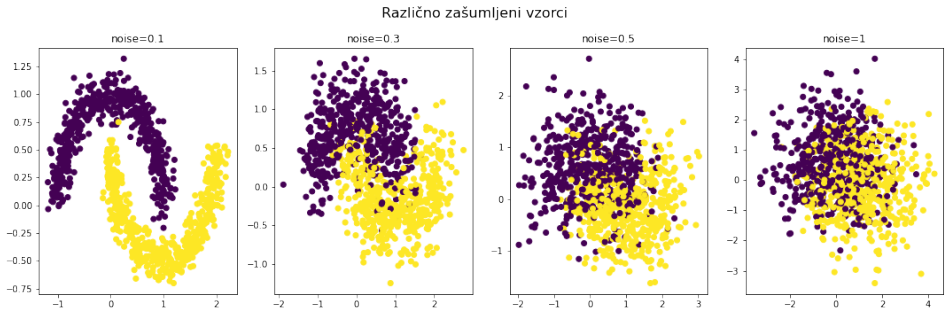
Slika 5: Učenje modela na zgolj enem parametru, 3 epohe.

Kot smo ugotavljali že v uvodu v to poglavje pri sliki 1, je najbolj relevantnih zadnjih nekaj parametrov. Pri številnih AUC pride komaj kaj nad 0,5. Pričakujem, da bi se to vseeno izboljšalo, če bi algoritmu pustil več časa.

2.3 Vgrajeni primeri

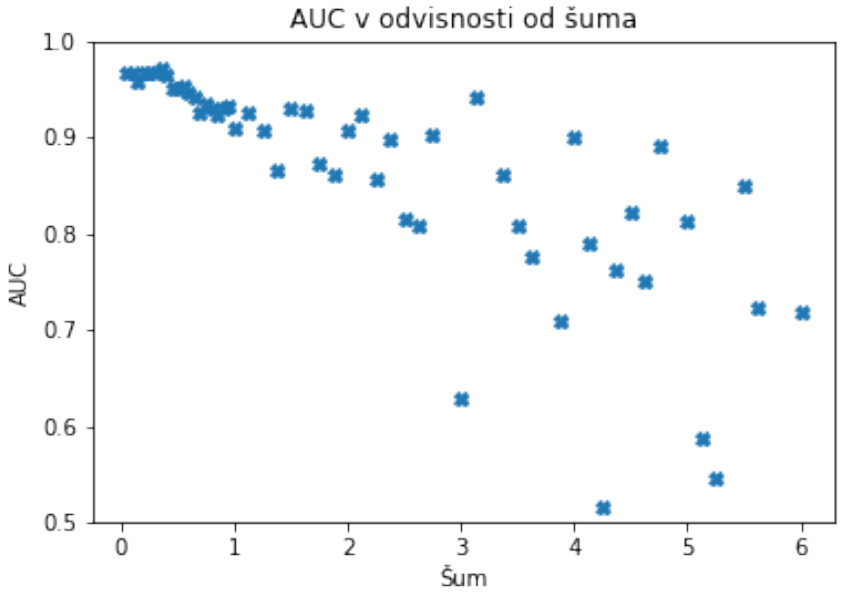
Na koncu sem se malo igral še s testnimi primeri podatkov iz uporabljenih knjižnic. Ugotoviti sem hotel, kako zašumljenost učne in testne množice vpliva na model. Še vedno sem uporabljal nevronske mreže s 50 epohami, stvari so tu potekale hitreje, ker je bilo manj podatkov.

Uporabljal sem različno zašumljene porazdelitve v obliki lunic



Slika 6: Zašumljene lunice.

Modele sem validiral na validacijski množici s šumom 0,3 (2 sličica iz leve). Za AUC v odvisnosti od zašumljenosti dobimo sledeč graf



Slika 7: AUC za zašumljeno testno porazdelitev.

Zanimivo je, kako dobro tu pravzaprav deluje algoritem. Do šuma nekaj čez ena, torej tudi za vse porazdelitve na prejšnji sliki, pri kateri zadnji dve delujeta skoraj nerazločljivo, se model zelo dobro obnese in da vrednosti AUC nad 0,9. Algoritem pri šumu približno 1,5 očitno začne izgubljati stabilnost, nekajkrat AUC pade celo pod 0,5 (kar je odrezano iz grafa).

2.4 Zaključek

To je najbrž eno mojih najslabših poročil, ker večin časa nisem ravno najbolje vedel, kaj se dogaja. Zaradi moje trenutne časovne stiske in nagnjenosti mojega računalnika k crashanju algoritmom tudi nisem namenil časa, ki bi si ga verjetno zaslužili. Precej težav sem imel tudi s kodo, primeri, ki so bili naloženi na spletno učilnico, so mi javljali neke errorje, in čeprav so bile na koncu potrebne le manjše spremembe, je troubleshooting vzel kar veliko časa, ker kode nisem najbolje razumel.

Naloga se mi zdi vsekakor med bolj zanimivimi in vesel sem, da sem se lahko seznanil z osnovami strojnega učenja. Zdi pa se mi, da bi bilo za domačo nalogo bolje, če bi se osnovna naloga ukvarjala s primeri, vgrajenimi v knjižnice, in bi bilo tole s Higgsom le dodaten del. Preprosti primeri so se vsaj meni zdeli bolj poučni, saj se mi je zdelo, da imam boljšo predstavo, kaj točno se dogaja. So tudi časovno manj zahtevni, ker so množice manjše, in to omogoča več igranja znotraj časovnih okvirjev, ki jih imamo študentje na voljo.

Ob oddaji zadnje domače naloge bi rad zapisal še, da mi je bil MFP res všeč in je bil moj najljubši del programa v 3. letniku. Druge stopnje študija najverjetneje ne bom nadaljeval na fiziki, zato je to morda zadnjič, da se srečam s predmetom, kot je ta. Koncept predmeta je zelo svež in unikaten in že vidim, da bom naslednje leto pogrešal tedenske programerske projekte. Hvala za super predmet.