Matematično-fizikalni praktikum

12. naloga: Strojno učenje (machine learning)

Tadej Tomažič

July 29, 2025

Contents

List of Figures

1 Navodilo

Danes je uporaba različnih algoritmov strojnega učenja (Machine Learning, ML) v znanosti že rutinsko opravilo. Poznamo tri osnovne vrste stojnega učenja:

- Nadzorovano učenje (Supervised learning):
  - Klasifikacija (Classification): sortiranje v različne kategorije.
  - Regresija (Regression): modeliranje oz. 'fitanje' napovedi.
- Nenadzorovano učenje (npr. sam najdi kategorije).
- Stimulirano učenje (Artificial Intelligence v ožjem pomenu besede).

V fiziki (in tej nalogi), se tipično ukvarjamo s prvo kategorijo, bodisi za identifikacijo novih pojavov/delcev/... ali pa za ekstrakcijo napovedi (netrivialnih funkcijskih odvisnosti etc).

ML algoritmi imajo prednost pred klasičnim pristopom, da lahko učinkovito razdrobijo kompleksen problem na enostavne elemente in ga ustrezno opišejo:

- pomisli na primer, kako bi bilo težko kar predpostaviti/uganiti pravo analitično funkcijo v več dimenzijah (in je npr. uporaba zlepkov (spline interpolacija) mnogo lažja in boljša).
- Pri izbiri/filtriranju velike količine podatkov z mnogo lastnostmi (npr dogodki pri trkih na LHC) je zelo težko najti količine, ki optimalno ločijo signal od ozadnja, upoštevati vse korelacije in najti optimalno kombinacijo le-teh...

Če dodamo malce matematičnega formalizma strojnega učenja: Predpostavi, da imamo na voljo nabor primerov $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1.N}$, kjer je $\mathbf{x}_k = \left(x_k^1, \ldots, x_k^M\right)$ naključno izbrani vektor M lastnosti (karakteristik) in je $\mathbf{y}_k = \left(y_k^1, \ldots, y_k^Q\right)$ vektor Q ciljnih vrednosti, ki so lahko bodisi binarne ali pa realna števila[1]. Vrednosti $(\mathbf{x}_k, \mathbf{y}_k)$ so neodvisne in porazdeljene po neki neznani porazdelitvi $P(\cdot, \cdot)$. Cilj ML metode je določiti (priučiti) funkcijo $h : \mathbb{R}^Q \to \mathbb{R}$, ki minimizira pričakovano vrednost *funkcije izgube (expected loss)*

$$\mathcal{L}(h) = \mathbb{E}\, L(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \frac{1}{N} \sum_{k=1}^{N} L(\mathbf{y_k}, \mathbf{h}(\mathbf{x_k})).$$

Tu je $L(\cdot, \cdot)$ gladka funkcija, ki opisuje oceno za kvaliteto napovedi, pri čemer so vrednosti $(\mathbf{x}, \mathbf{y})$ neodvisno vzorčene iz nabora $\mathcal{D}$ po porazdelitvi P. Po koncu učenja imamo torej na voljo funkcijo $\mathbf{h}(\mathbf{x})$, ki nam za nek vhodni nabor vrednosti $\hat{\mathbf{x}}$ poda napoved $\hat{\mathbf{y}} = \mathbf{h}(\hat{\mathbf{x}})$, ki ustrezno kategorizira ta nabor vrednosti.

Funkcije $\mathbf{h}$ so v praksi sestavljene iz (množice) preprostih funkcij z (nekaj) prostimi parametri, kar na koncu seveda pomeni velik skupni nabor neznanih parametrov in zahteven postopek minimizacije funkcije izgube.

Osnovni gradnik odločitvenih dreves je tako kar stopničasta funkcija $H(x_i - t_i) = 0, 1$, ki je enaka ena za $x_i > t_i$ in nič drugače in kjer je x_i ena izmed karakteristik in t_i neznani parameter. Iz skupine takšnih funkcij, ki predstavljajo binarne odločitve lahko skonstruiramo končno uteženo funkcijo

$$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{J} \mathbf{a}_i\, H(x_i - t_i),$$

kjer so $\mathbf{a}_i$ vektorji neznanih uteži. Tako t_i kot $\mathbf{b}_i$, lahko določimo v procesu učenja. Nadgradnjo predstavljajo nato *pospešena* odločitvena drevesa (BDT), kjer nadomestimo napoved enega drevesa z uteženo množico le-teh, tipično dobljeno v ustreznih iterativnih postopkih (npr. AdaBoost, Gradient Boost ipd.).

Pri nevronskih mrežah je osnovni gradnik t.i. *perceptron*, ki ga opisuje preprosta funkcija

$$h_{w,b}(\mathbf{X}) = \vartheta\left(\mathbf{w}^T \cdot \mathbf{X} + b\right),$$

[1] ...ali pa še kaj, prevedljive na te možnosti, npr barve...

kjer je $\mathbf{X}$ nabor vhodnih vrednosti, $\mathbf{w}$ vektor vrednosti uteži, s katerimi tvorimo uteženo vsoto ter b dodatni konstatni premik (bias). Funkcija ϑ je preprosta gladka funkcija (npr. arctan), ki lahko vpelje nelinearnost v odzivu perceptrona. Nevronska mreža je nato sestavljena iz (poljubne) topologije takšnih perceptronov, ki na začetku sprejme karakteristiko dogodka $\mathbf{x}$ v končni fazi rezultirajo v napovedi $\hat{\mathbf{y}}$, ki mora seveda biti čim bližje ciljni vrednosti $\mathbf{y}$. Z uporabo ustrezne funkcije izgube (npr MSE: $\mathcal{L}(h) = \mathbb{E}\,||\mathbf{y} - \hat{\mathbf{y}}||^2$), se problem znova prevede na minimizacijo, kjer iščemo optimalne vrednosti (velikega) nabora uteži $\mathbf{w}_i$ ter b_i za vse perceptrone v mreži. Globoke nevronske mreže (DNN) niso nič drugega, kot velike nevronske mreže ali skupine le-teh.

Že namizni računalniki so dovolj močni za osnovne računske naloge, obstajajo pa tudi že zelo uporabniku prijazni vmesniki v jeziku Python, na primer:

- Scikit-Learn (*scikit-learn.org*): odprtokodni paket za strojno učenje,
- TensorFlow (*tensorflow.org*): odprtokodni Google-ov sistem za ML, s poudarkom na globokih nevronskih mrežah (Deep Neural Networks, DNN) z uporabo vmesnika Keras. Prilagojen za delo na GPU in TPU.
- Catboost: (*Catboost.ai*) : odprtokodna knjižnica za uporabo pospešenih odločitvenih dreves (Boosted Decision Trees, BDT). Prilagojena za delo na GPU.

Za potrebe naloge lahko uporabimo tudi spletni vmesnik Google Collab (*colab.research.google.com*), ki dopušča omejen dostop do večjih računskih zmogljivosti.

Naloga: Na spletni učilnici je na voljo material (koda, vzorci) za ločevanje dogodkov Higgsovega bozona od ostalih procesov ozadja. V naboru simuliranih dogodkov je 18 karakteristik (zveznih kinematičnih lastnosti), katerih vsaka posamezno zelo slabo loči 'signal' od ozadja, z uporabo BDT ali (D)NN, pa lahko tu dosežemo zelo dober uspeh. Na predavanjih smo si ogledali glavne aspekte pomembne pri implementaciji ML, kot so uporaba ustreznih spremenljivk (GIGO), učenje in prekomerno učenje (training/overtraining), vrednotenje uspeha metode kot razmerje med učinkovitostjo (efficiency) in čistostjo (precision) vzorca (Receiver Operating Characteristic, ROC). Določi uspešnost obeh metod (in nariši ROC) za nekaj tipičih konfiguracij BDT in DNN, pri čemer:

- Študiraj vpliv uporabljenih vhodnih spremenljivk - kaj, če vzamemo le nekatere?
- Študiraj BDT in NN in vrednoti uspešnost različnih nastavitev, če spreminjaš nekaj konfiguracijskih parametrov (npr. število perceptronov in plasti nevronskih mrež pri DNN in število dreves pri BDT).

Dodatna Naloga: Implementiraj distribucije iz 'playground' zgleda v BDT (lahko tudi RandomForests) in DNN, te distribucije so na voljo v vseh popularnih ML paketih (npr. Scikit...). Lahko si izbereš/izmisliš še kakšnega - npr. v vseh paketih je že na voljo standardni 'moons' dataset.

References

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Inc, Sebastopol, CA, 2019.

[2] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2017.

[3] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[4] P. Baldi, P. Sadowski and D. Whiteson, *Searching for Exotic Particles in High-Energy Physics with Deep Learning,* Nature Commun. **5** (2014) 4308 doi:10.1038/ncomms5308 [arXiv:1402.4735 [hep-ph]].

2 Rešitev

Najprej si poglejmo parametre na **??**, ki jih prejmemo.

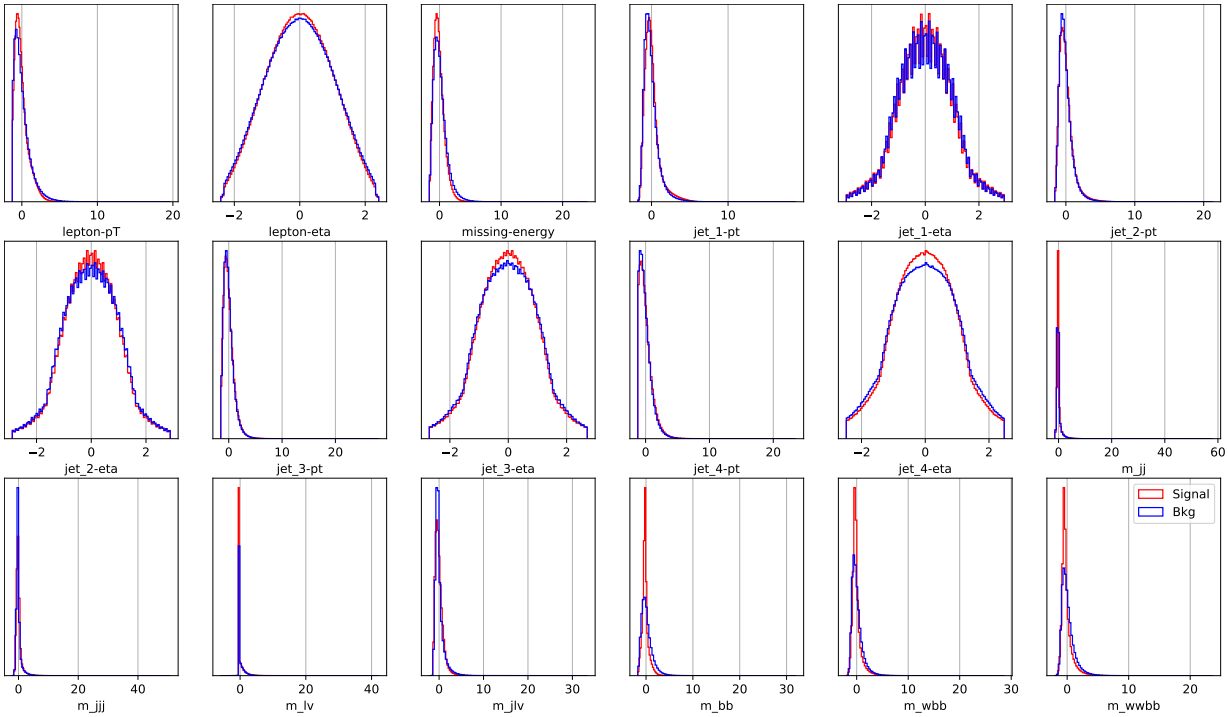

Figure 1: Porazdelitev signala in šuma glede na brezdimenzisko reskalirano x-os.

S pomočjo catboost knjižnjice si poglejmo kako se se spreminja natačnost modela glede na maksimalno število dreves in nekaterih drugih parametrov.

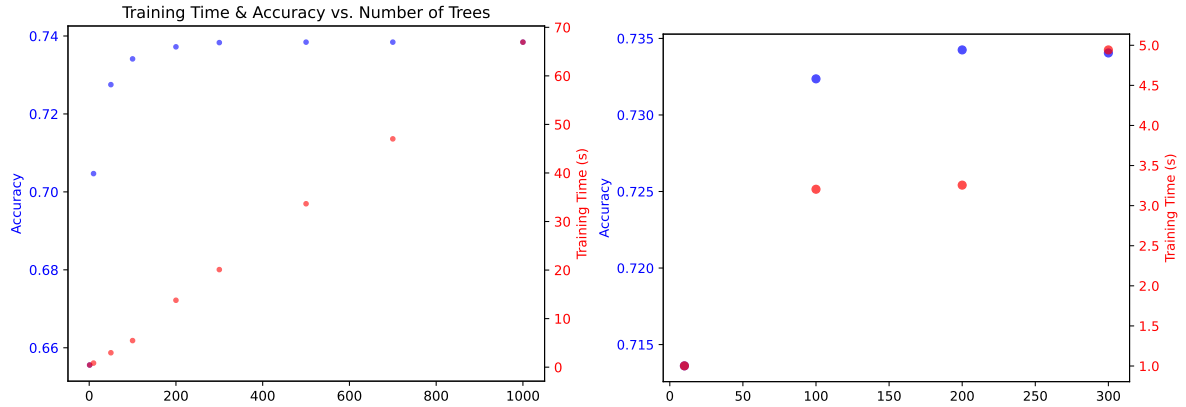

Figure 2: Natačnost modela glede na število dreves prva slika, na drugi sliki pa je y odvisna od maksimalnega števila iteracij

Poglejmo si še vpliv velikosti vhodnih podatkov na natančnost modela.
Kaj se zgodi, če učimo na parametrih, ki bolj očitno ločujejo signal in šum?

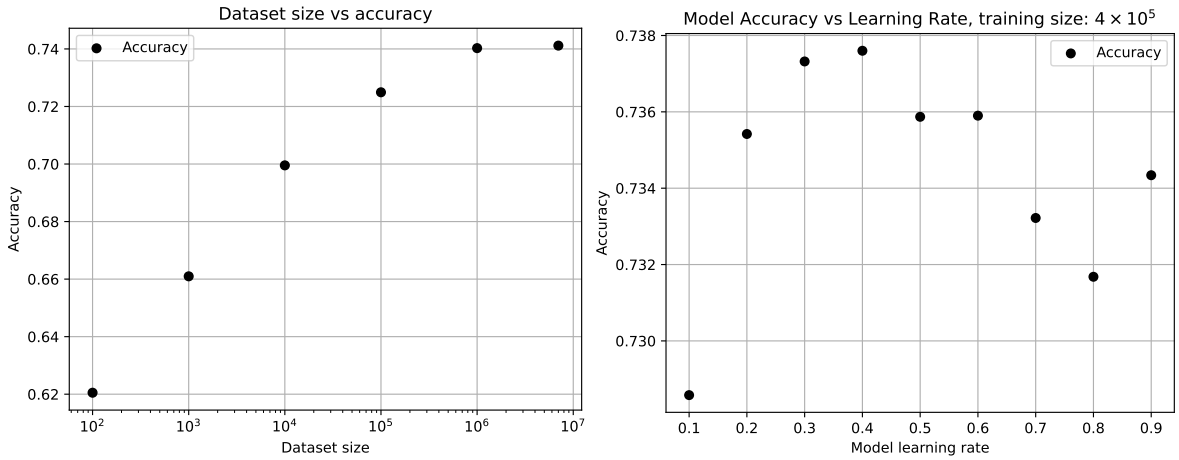

Figure 3: Natačnost modela glede na velikost vhodnih podatkov prva slika, na drugi sliki pa je y odvisna od stopnje učenja

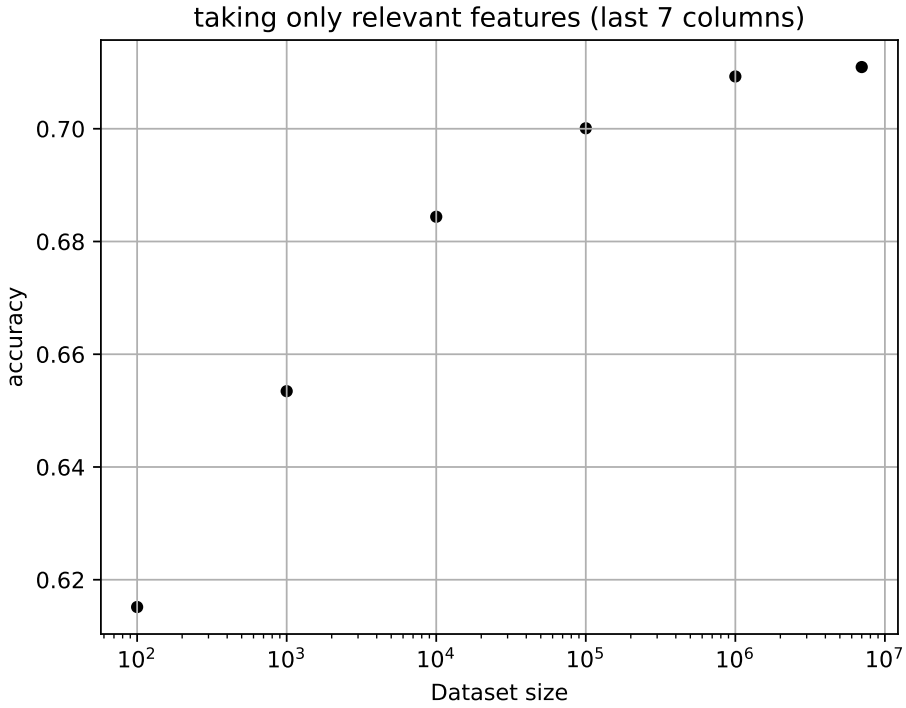

Figure 4: Uporabljenih zadnjih 7 parametrov, ki bolj očitno ločujejo signal in šum.

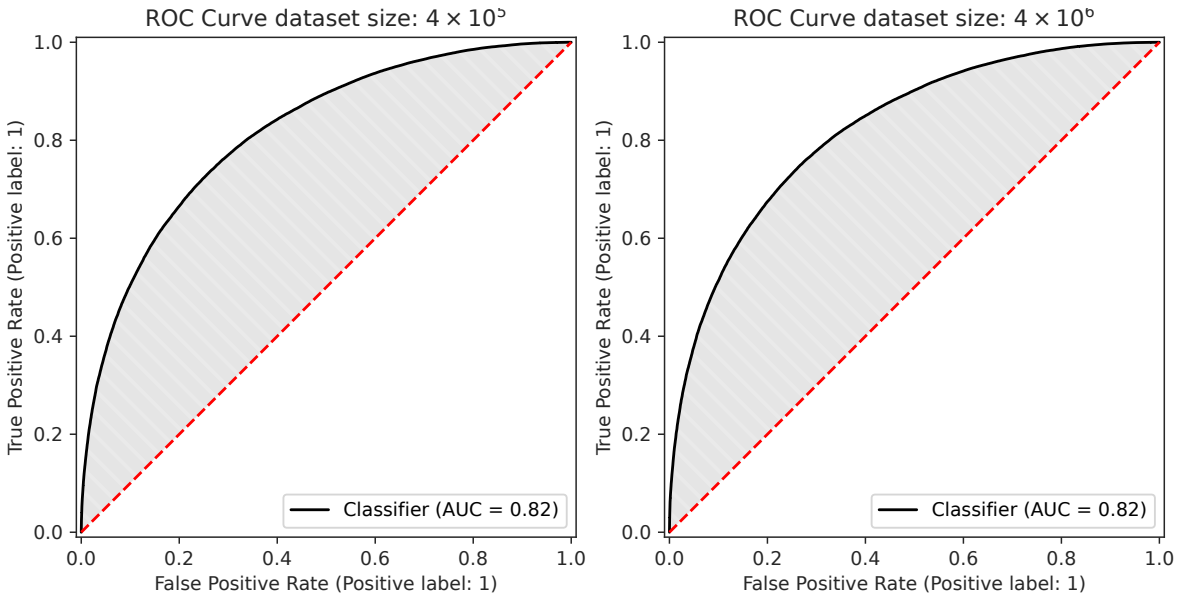

Figure 5: ROC krivulja pri 2 različnih velikostih vhodnih podatkov.

Narišemo še ROC kirvuljo pri 2 različnih velikostih vhodnih podatkov.
Poglejmo si še kako se obnašajo nevronske mreže.
Poglejmo si še ROC krivuljo in odziv na signal in na šum.
Zadnji graf pa prikazuje odvisnost števila nevronov in števila skritih plasti na natančnost modela.

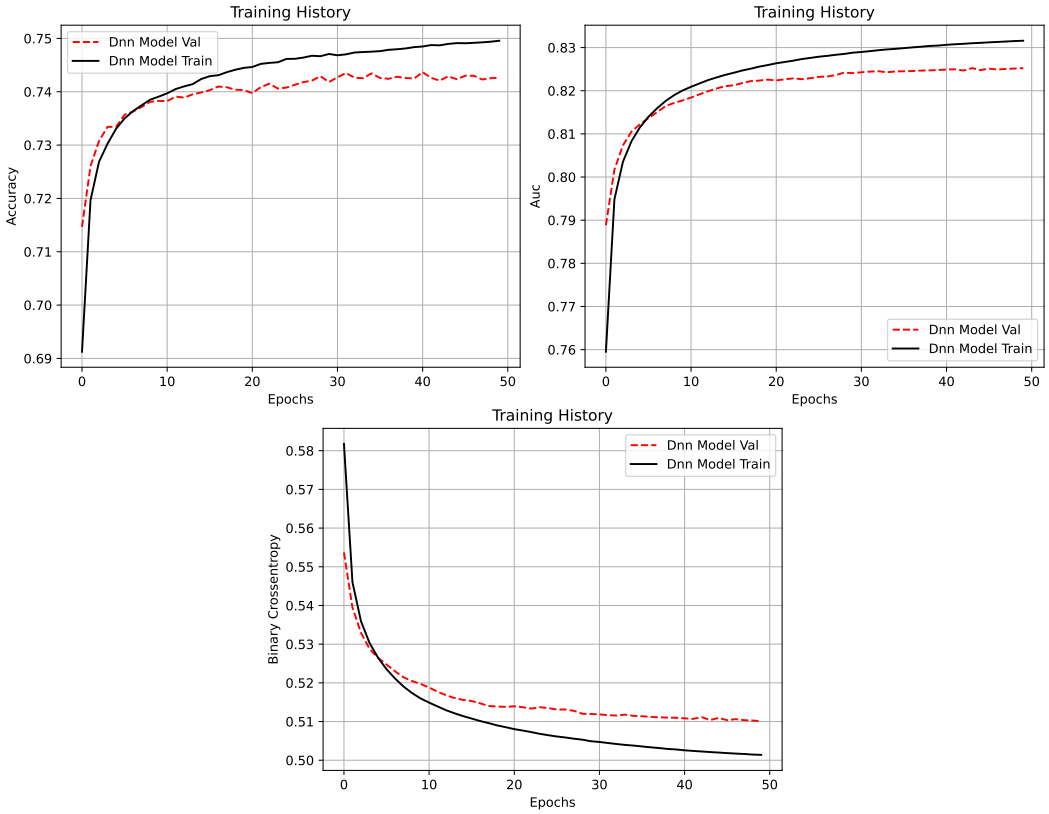

Figure 6: Natačnost in izguba nevronske mreže glede na število iteracij.

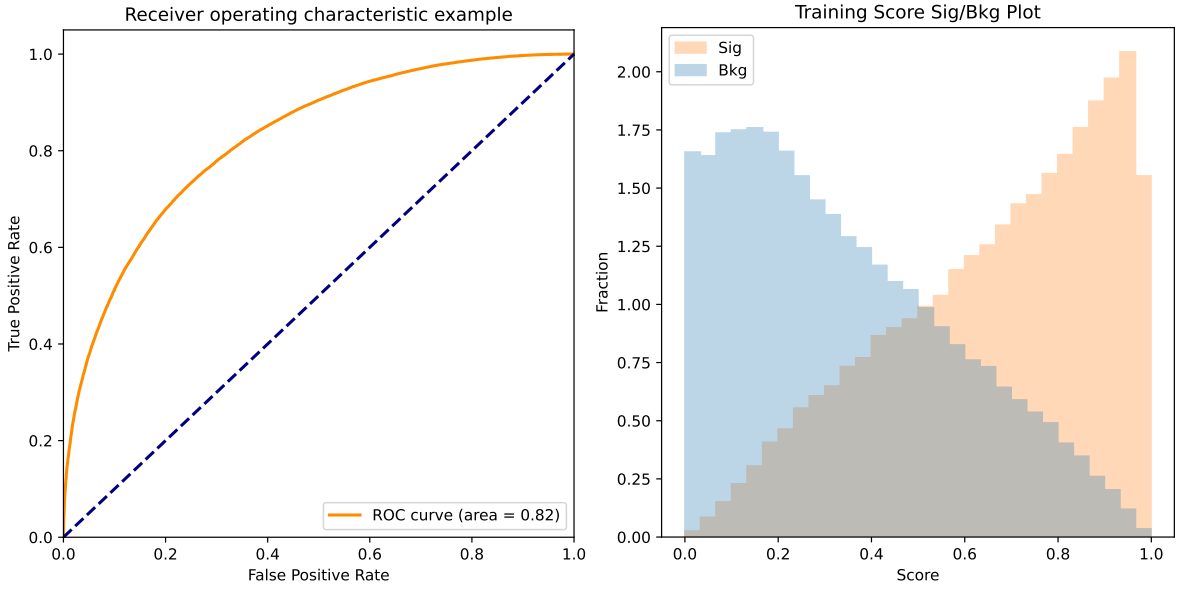

Figure 7: ROC krivulja in odziv na signal in šum.

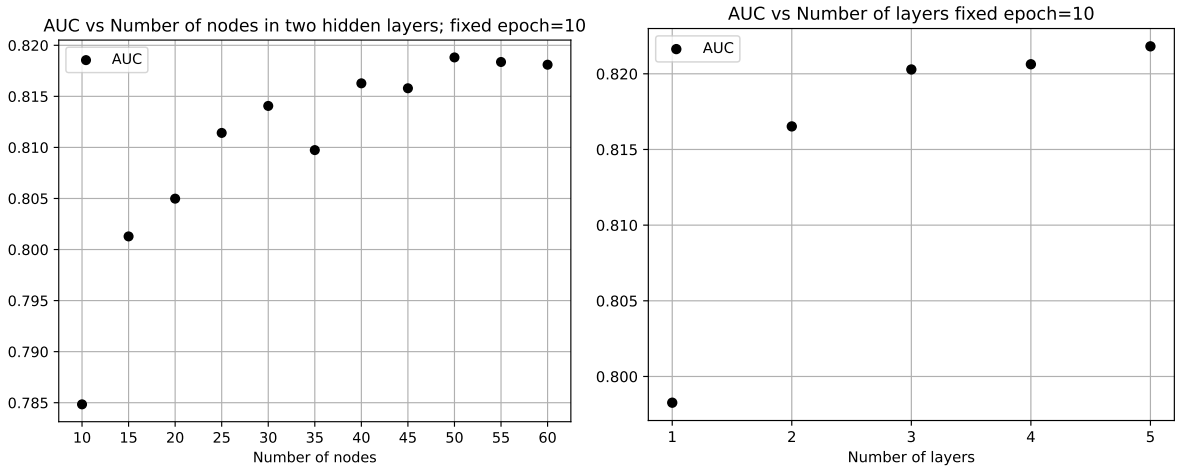

Figure 8: Odvisnost števila nevronov in števila skritih plasti na natančnost modela.

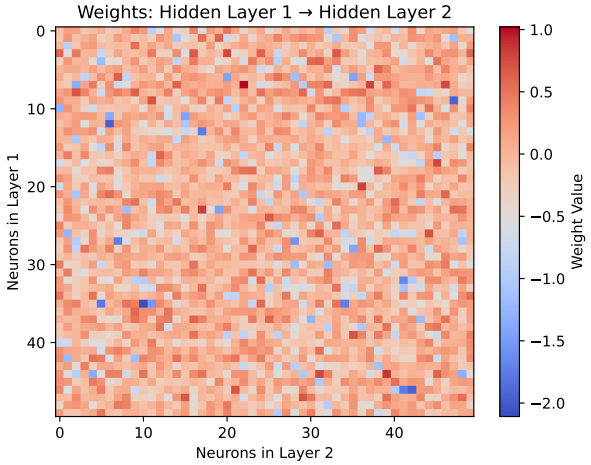

Figure 9: Moč povezav med skritimi plastmi.

Poglejmo si še kako močne so povezave med skritimi plastmi.