# 12 - STROJNO UČENJE ZA KATEGORIZACIJO PODATKOV

**Matematično-fizikalni praktikum, avgust 2024**
**Luka Skeledžija, 28201079**

---

## 1. Uvod

Dandanes 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. 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 $h_{w,b}(\mathbf{X}) = \theta\big(\mathbf{w}^T \cdot \mathbf{X} + b\big)$, kjer je $\mathbf{X}$ nabor vhodnih vrednosti, $\mathbf{w}$ vektor vrednosti uteži, s katerimi tvorimo uteženo vsoto ter $b$ dodatni konstatni premik (bias). Funkcija $\theta$ 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.

# 2. 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).

**Dotatna 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...).

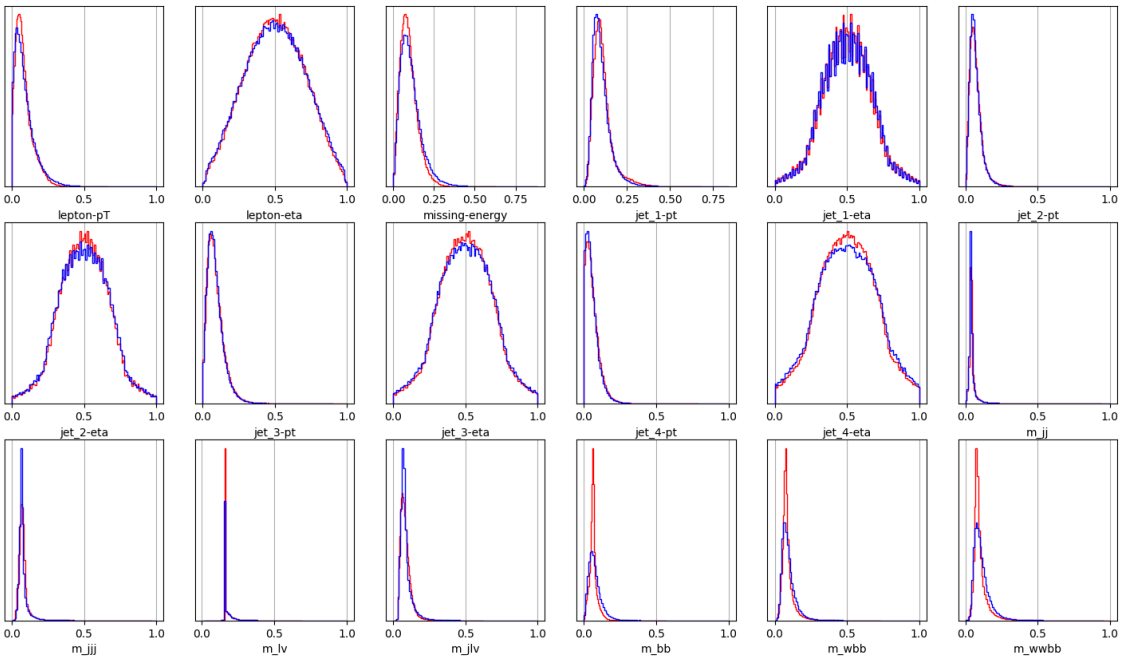
# 3. Detektiranje Higgsovega bosona

Ker imamo opravka s strojnim učenjem, velik del zgodbe predstavljajo tudi podatkovni seti. Zato se mi zdi prav, da najprej pokomentiramo še podatke, na katerih se bo stroj učil.

Na voljo imamo podmnožico 18 zveznih spremenljivk iz podatkovnega seta HIGGS. Gre za kinematične lastnosti kot jih izmerijo detektorji v trkalniku delcev LHC (Large Hadron Collider). Celoten podatkovni set je bil zgeneriran s pomočjo Monte Carlo simulacij trkov in služi za trening algoritmov, ki zmorejo klasificirati podatke dobljene iz dejanskih eksperimentov. V spodnji tabeli najdemo skromen opis posameznih spremenljivk oz. "feature"-jev.

| # | Lastnost | Opis lastnosti | Tip |
|---|---|---|---|
| 1 | lepton-pT | $\vec{p}_T$ leptona | Zvezna |
| 2 | lepton-eta | $\eta_\ell$ leptona | Zvezna |
| 3 | missing-energy | $\vec{E}_{\text{miss}}$ energija nevtrinov | Zvezna |
| 4 | jet_1-pt | $\vec{p}_T$ prvega pljuska | Zvezna |
| 5 | jet_1-eta | $\eta$ prvega pljuska | Zvezna |
| 6 | jet_2-pt | $\vec{p}_T$ drugega pljuska | Zvezna |
| 7 | jet_2-eta | $\eta$ drugega pljuska | Zvezna |
| 8 | jet_3-pt | $\vec{p}_T$ tretjega pljuska | Zvezna |

| # | Lastnost | Opis lastnosti | Tip |
|---|---|---|---|
| 9 | jet_3-eta | $\eta$ tretjega pljuska | Zvezna |
| 10 | jet_4-pt | $\vec{p}_T$ četrtega pljuska | Zvezna |
| 11 | jet_4-eta | $\eta$ četrtega pljuska | Zvezna |
| 12 | m_jj | Inv. masa $m_{jj}$ dveh pljuskov | Zvezna |
| 13 | m_jjj | Inv. masa $m_{jjj}$ treh pljuskov | Zvezna |
| 14 | m_lv | Inv. masa $m_{\ell\nu}$ leptona in nevtrina | Zvezna |
| 15 | m_jlv | Inv. masa $m_{j\ell\nu}$ pljuska, leptona in nevtrina | Zvezna |
| 16 | m_bb | Inv. masa $m_{bb}$ dveh kvarkov dno | Zvezna |
| 17 | m_wbb | Inv. masa $m_{Wbb}$ bozona $W^{\pm}$ in dveh b-kvarkov | Zvezna |
| 18 | m_wwbb | Inv. masa $m_{WWbb}$ dveh bozonov $W^{\pm}$ in dveh b-kvarkov | Zvezna |




Na podlagi teh 18 karakteristik bomo poizkusili naš model naučiti razlikovati med podatki ozadja (torej takrat ko Higgsov boson ni prisoten) in podatki signala, kjer smo Higgsov boson zaznali. Statistično gledano, bomo iz modela dobili 4 tipe rezultatov:

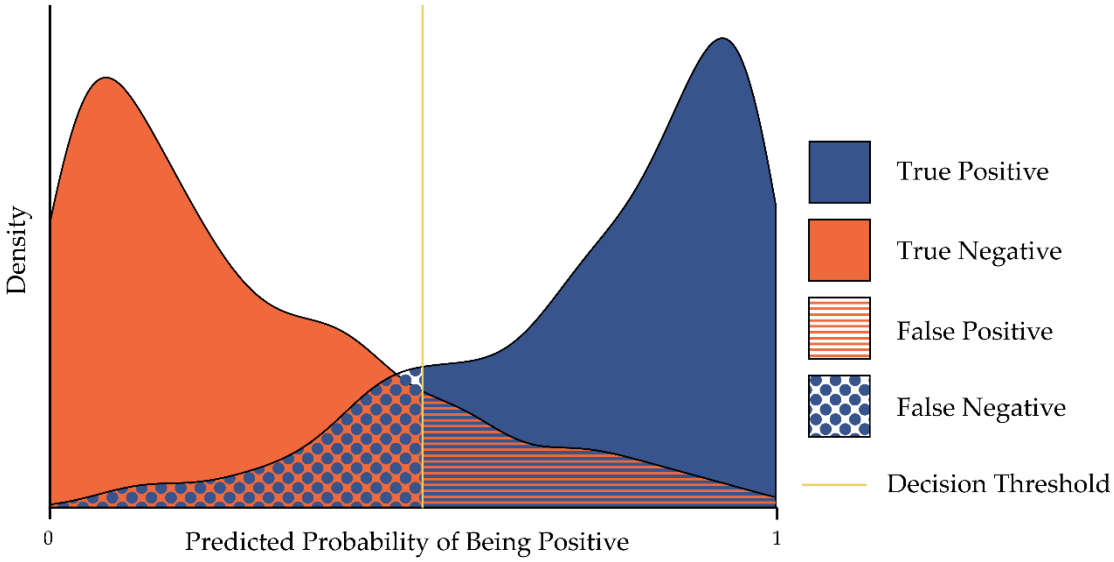
1. **True Positive (TP):** Model je pravilno napovedal prisotnost bozona.
2. **False Positive (FP):** Model je napovedal prisotnost bozona, vendar ga v resnici ni bilo.
3. **True Negative (TN):** Model je pravilno napovedal, da bozona ni.
4. **False Negative (FN):** Model je napovedal, da bozona ni, vendar je bil v resnici prisoten.

Kako torej ocenimo uspešnost modela in jih med sabo primerjamo? Uvedemo 2 novi metriki **true positive rate (TPR)** in **false positive rate (FPR)**. V bolj domačem jeziku -

kolikšen delež dejansko pozitivnih vrednosti je bil uspešno detektiran in analogno za negative. Formalno metriki definiramo kot:

$$\text{TPR} = \frac{TP}{TP + FN}$$

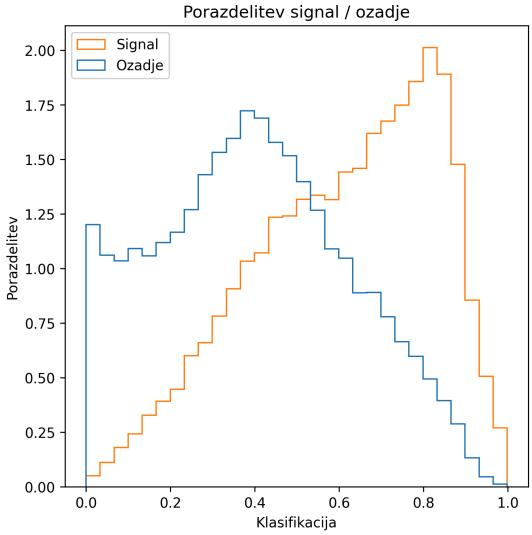
$$\text{FPR} = \frac{FP}{FP + TN}$$

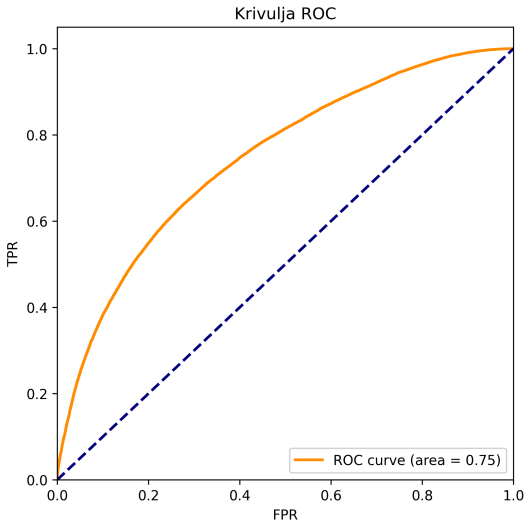



> **Vir slike**: Maxwell, A.E.; Warner, T.A.; Guillén, L.A. Accuracy Assessment in Convolutional Neural Network-Based Deep Learning Remote Sensing Studies—Part 1: Literature Review. Remote Sens. 2021, 13, 2450. https://doi.org/10.3390/rs13132450

Če narišemo odvisnost TPR od FPR za vse možne vrednosti praga (decision threshold), dobimo krivuljo ROC (receiver operating characteristic). Uspešnejši modeli se s krivuljo bolj približajo zgornjemu levemu kotu grafa. Kot celotno oceno za uspešnost modela vpeljemo metriko AUC (Area under curve) tj. vrednost površine pod ROC krivuljo. Bližje kot je vrednosti 0.5 - tem slabši je model, bližje kot je vrednosti 1 - tem boljši je model. Za ilustracijo konceptov podamo v nadaljevanju 2 zgleda iz nadaljevanja naloge.

| Klasifikacija na spektru [0, 1] | Metrika uspešnosti |
|---|---|




## 3.1. Globoke nevronske mreže (DNN)

V nadaljevanju bomo sestavili osnovno nevronsko mrežo z dvema skritima nivojema. Uporabili bomo knjižnico `TensorFlow` in učenje zagnali v okolju `Google Colab` na hardwaresko pospešenih enotah TPU enotah `T4`. Začetni hiperparametri naše mreže so:

- število nevronov: 50
- število skritih plasti: 2
- aktivacijska funkcija: ReLU
- aktivacijska funkcija na koncu: sigmoid
- optimizacijski algoritem: adam
- funkcija izgube: binary cross-entrophy
- normalizacija: na interval [0,1]
- število epoh: 10
- velikost batchov: 100
- odstotek za testiranje: 10 %

V nadaljevanju bomo spreminjali posamezne hiperparametre in opazovali, kako spreminjanje le-teh vpliva na metriko `AUC`.

> **Opazka**: Sklepam, da je v novejših verzijah TensorFlow-a prišlo do sprememb in je za uporabo našega data seta potrebno izhod iz nevronske mreže še preoblikovati z dodatnim (sicer trivialnim) nivojem: `dnn.add(Lambda(lambda x: tf.squeeze(x, axis=-1)))`

| Sprememba parametra | AUC |
|---|---|
| Osnovni parametri | 0.7591 |
| Število nevronov: 100 | 0.7573 |
| Št. skritih slojev: 5 | 0.7554 |

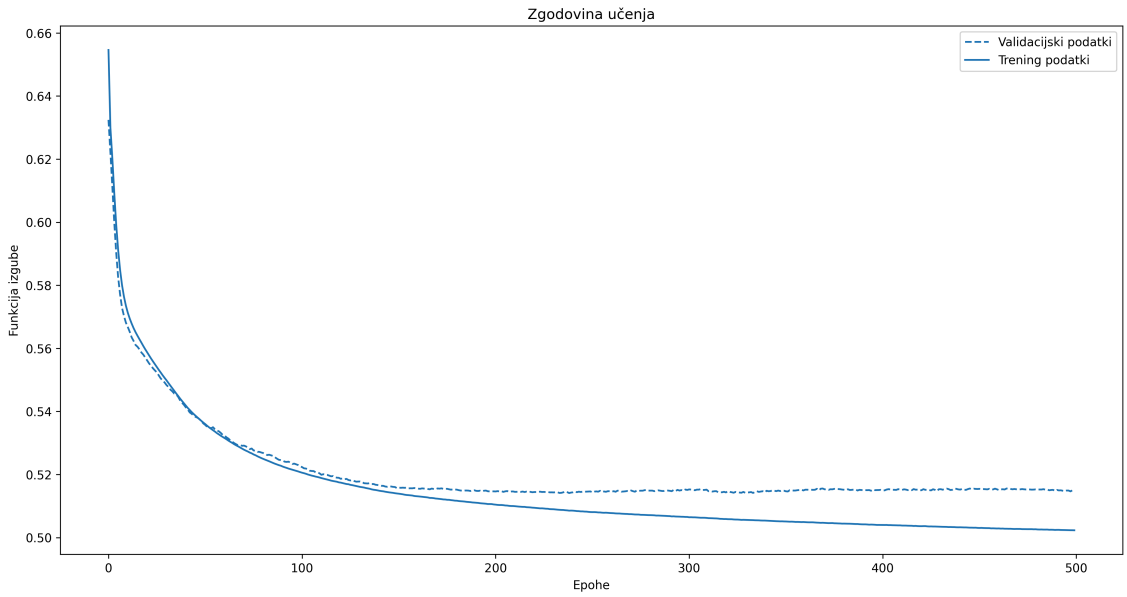
| Sprememba parametra | AUC |
|---|---|
| Št. skritih slojev: 50 | 0.5000 |
| Aktivacijska funkcija: ELU | 0.7454 |
| Funkcija izgube: MSE | 0.7480 |
| Optimizacijski algoritem: SGD | 0.6768 |
| Optimizacijski algoritem: Adamax | 0.7505 |
| Velikost batcha: 1 000 | 0.7385 |
| Velikost batcha: 10 000 | 0.6873 |
| Število epoh: 20 | 0.7532 |
| Odstotek za testiranje: 20 % | 0.7562 |
| Normalizacija: $\mu = 0$ in $\sigma = 1$ | 0.8144 |

Rezultati so relativno zanimivi. Z osnovnimi parametri smo dosegli relativno dober rezultat AUC = 0,7591, kar smo uporabili kot referenco. Povečanje skritih slojev na 50 je dramatično poslabšalo učinkovitost (AUC = 0,5000), kar kaže, da ne smemo pretiravati s kompleksnostjo topologije mreže. Tudi preprosti SGD optimizacijski algoritem je znatno znižal na AUC na 0,6768.

Največje izboljšanje smo dosegli z **normalizacijo** (μ = 0, σ = 1), kjer je AUC narasel na 0,8144. Podatke se torej splača pravilno normirati. Drugi parametri, kot so aktivacijske funkcije in arhitektura, pa so imeli manjši vpliv.

## Pretreniranje (over-training)

Pretreniranje je znan pojav v strojnem učenju in ga opazimo takrat, ko se model zaradi prekomernega učenja preveč specializira na podatke, na katerih se uči. Prikaz tega si lahko ogledamo spodaj. Funkcija izgube za učne podatke naglo pada, medtem ko za testne oz. validacijske podatke ostaja skoraj konstantna. Tedaj je smiselno z učenjem prenehati - temu namerno obstajajo tudi vgrajene `Early-Stopping` funkcije.



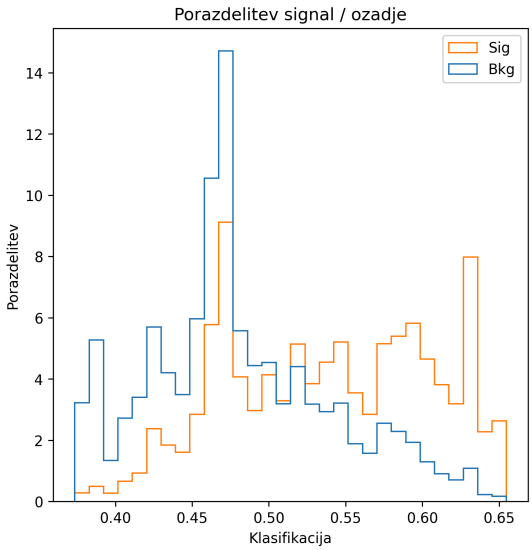
# 4. Pospešena odločitvena drevesa (BDT)

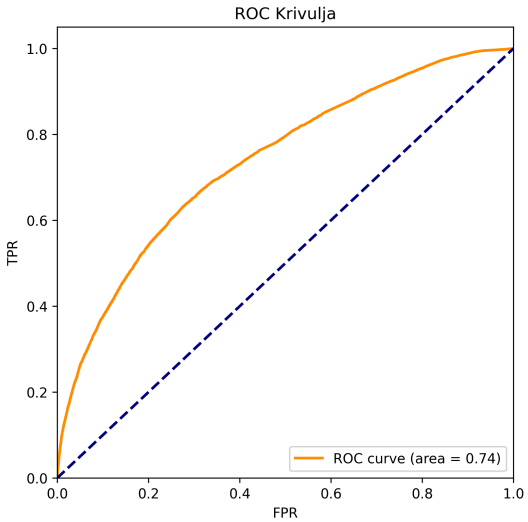
V nadaljevanju bomo preverili še kako na tem primeru delujejo pospešena odločitvena drevesa. Ideja za odločitvenimi drevesi je grajenje modela z združevanjem večih manjših odločitvenih dreves, kjer vsako novo drevo korrigira napako prejšnjega Za implementacijo bomo uporabili knjižnico `CatBoost` in model zagnali v okolju `Google Colab`. Začetni hiperparametri so:

- max. št. dreves: 50
- max. globina drevesa 6
- funkcija nečistosti: Logloss
- brez normalizacije podatkov

V nadaljevanju bomo spreminjali posamezne hiperparametre in opazovali, kako spreminjanje le-teh vpliva na metriko `AUC`.

| Klasifikacija | Metrika uspešnosti |
|---|---|



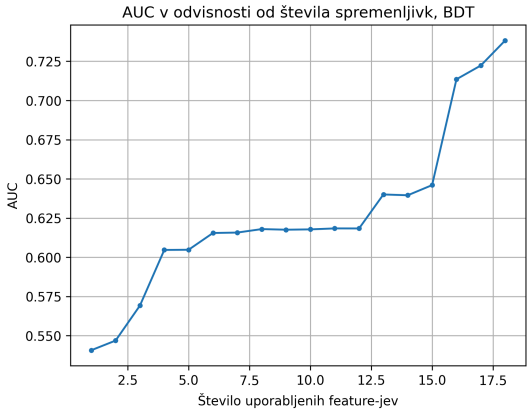
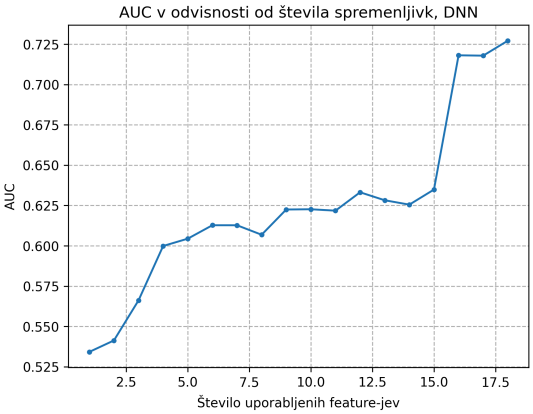

| Sprememba parametra | AUC |
|---|---|
| Osnovni parametri | 0.7380 |
| maksimalno število dreves: 25 | 0.7312 |
| maksimalno število dreves: 300 | 0.7772 |
| funkcija nečistosti: Cross-entropy | 0.7382 |
| maksimalna globina: 3 | 0.7382 |
| maksimalna globina: 15 | 0.7761 |
| Odstotek za testiranje: 20 % | 0.7376 |
| Normalizacija: $\mu = 0$ in $\sigma = 1$ | 0.7380 |

Ugotovimo, da povečanje maksimalnega števila dreves na 300 in povečanje maksimalne globine na 15 znatno izboljšata AUC. Uporaba cross-entropy funkcije

nečistosti rahlo izboljša rezultat, medtem ko sprememba števila dreves na 25, zmanjšanje globine na 3, ter normalizacija in sprememba odstotka za testiranje nimajo pomembnega vpliva.

## 4.1. Kako št. feature-jev vpliva na uspešnost modela

Primerjajmo še kako se AUC spreminja z dodajanjem števila feature-jev pri DBT in DNN. Na spodnjih grafih opazimo, da na začetku vrednost AUC s številom feature-jev narašča, nato se malo ustali in nato spet poskoči. Očitno nekateri feature-ji močneje vplivajo na vrednost AUC kot drugi.

| BDT | DNN |
|---|---|
|  |  |



# 5. Dodatek - Klasifikacija sintetičnih distribucij

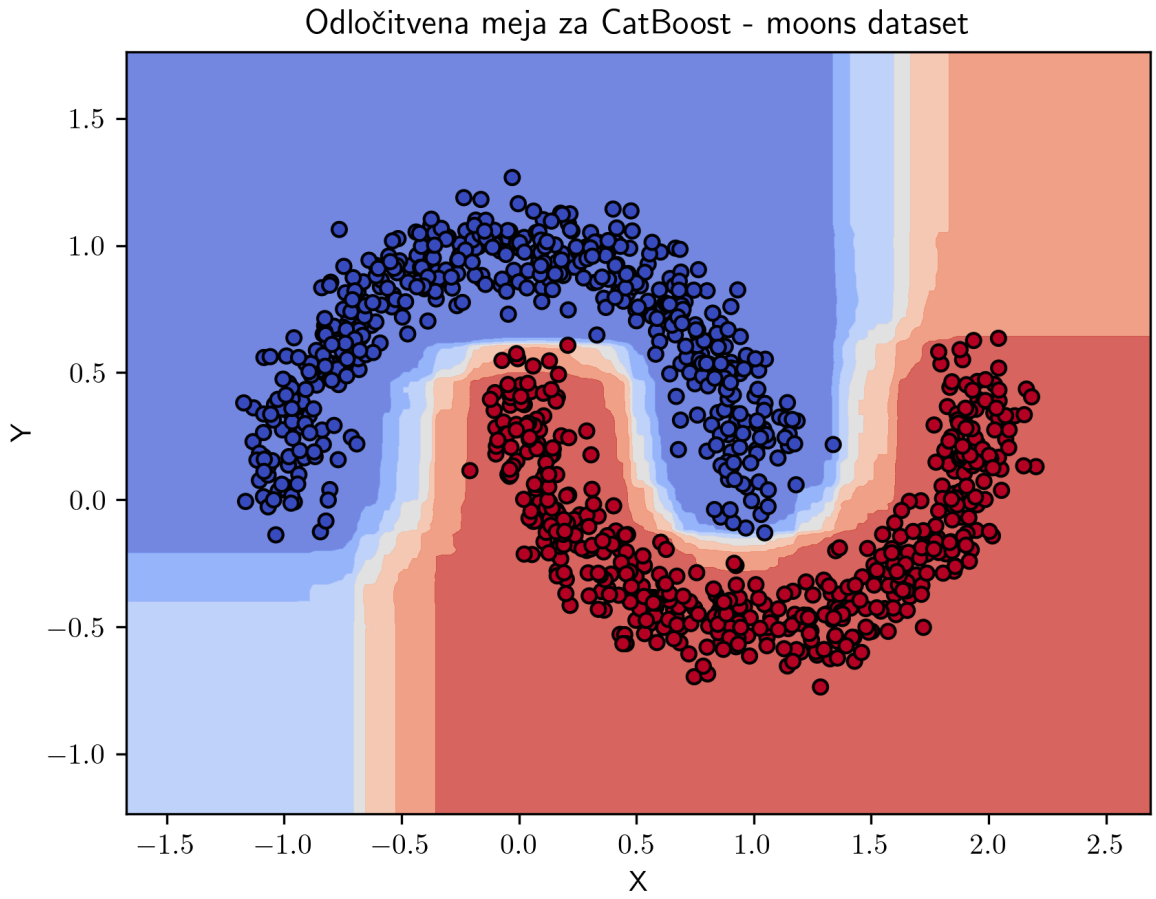
Za zaključek klasificirajmo še sintetične distribucije iz paketa `SciKit learn`. S tem početjem lahko opazimo nekaj razlik v delovanju obeh klasifikacijskih metod.

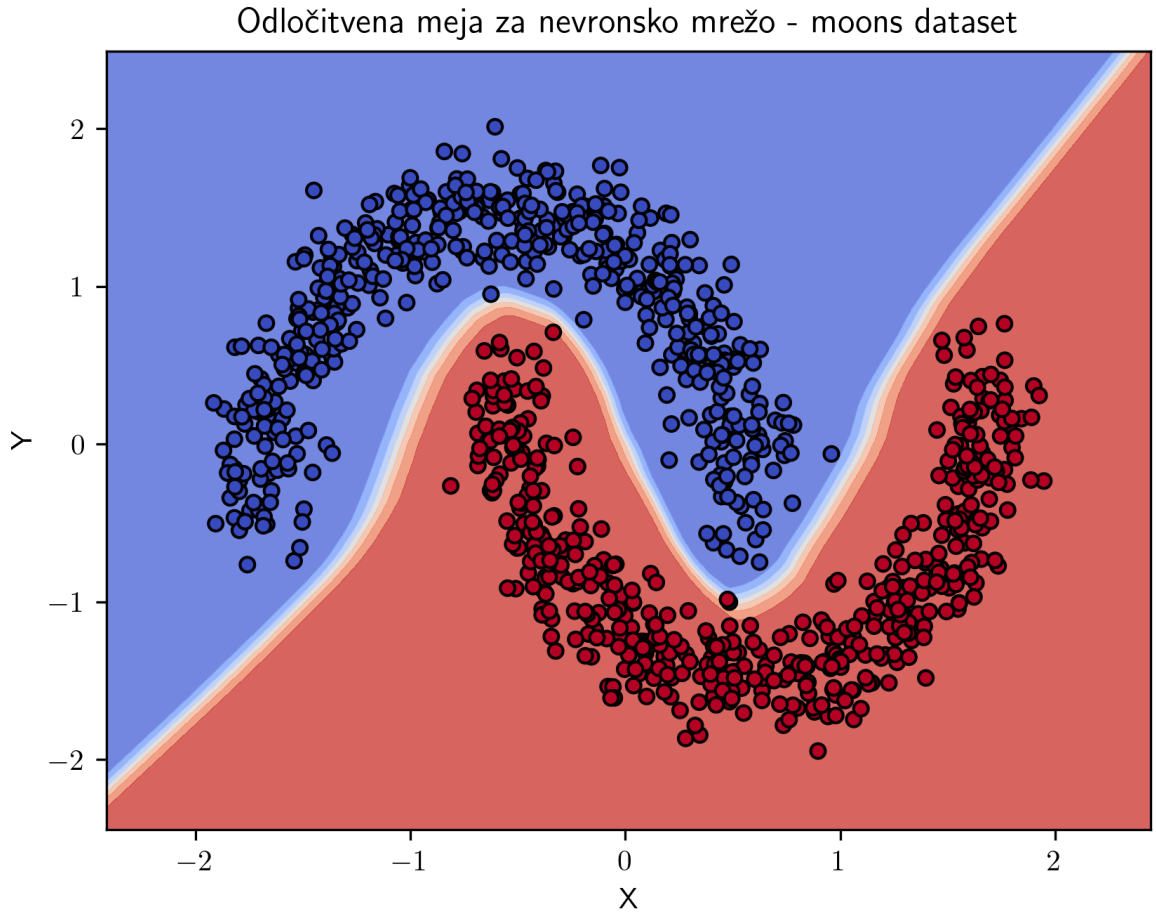
Pri metodi BDT smo za hiperparametre nastavili:

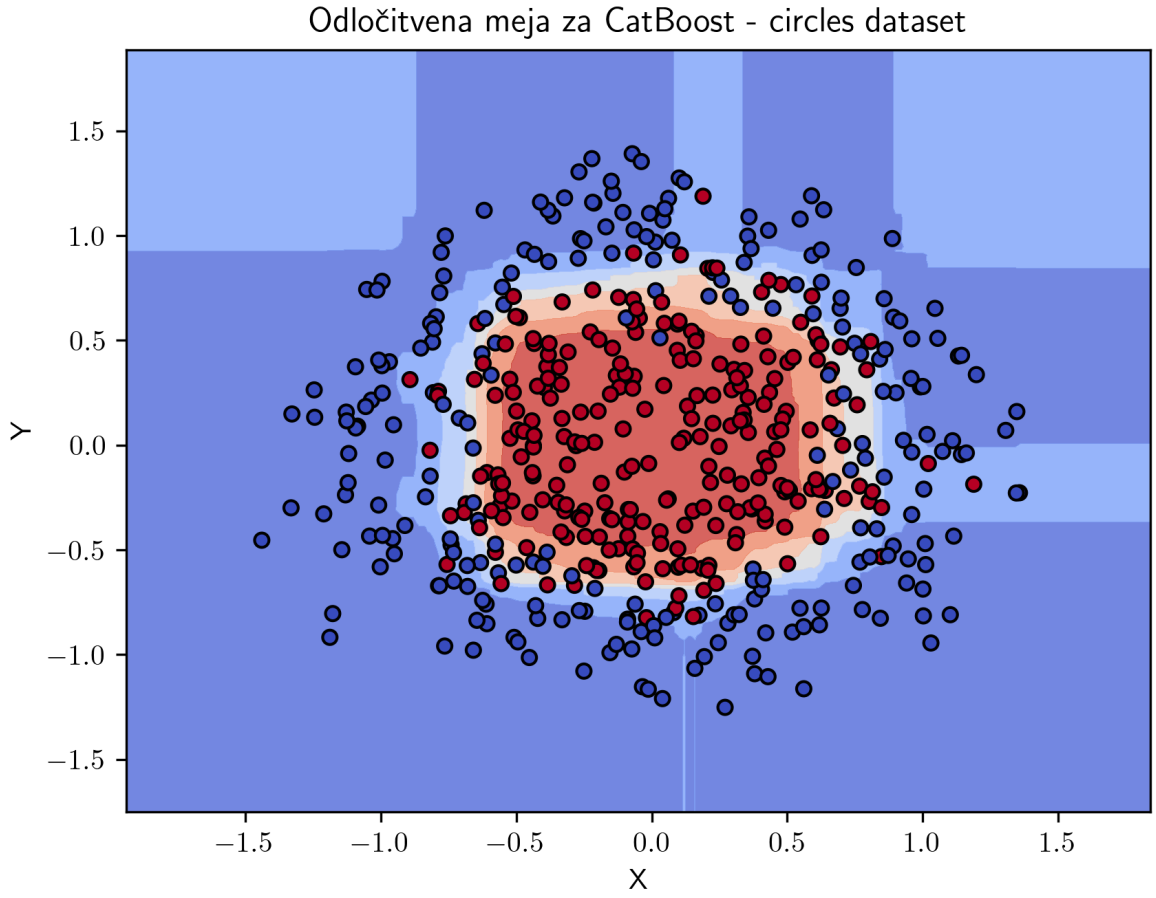
- maksinalno število dreves: 100
- maksimalna globina drevesa: 6
- funkcija nečistosti: Logloss
- normalizacija: brez

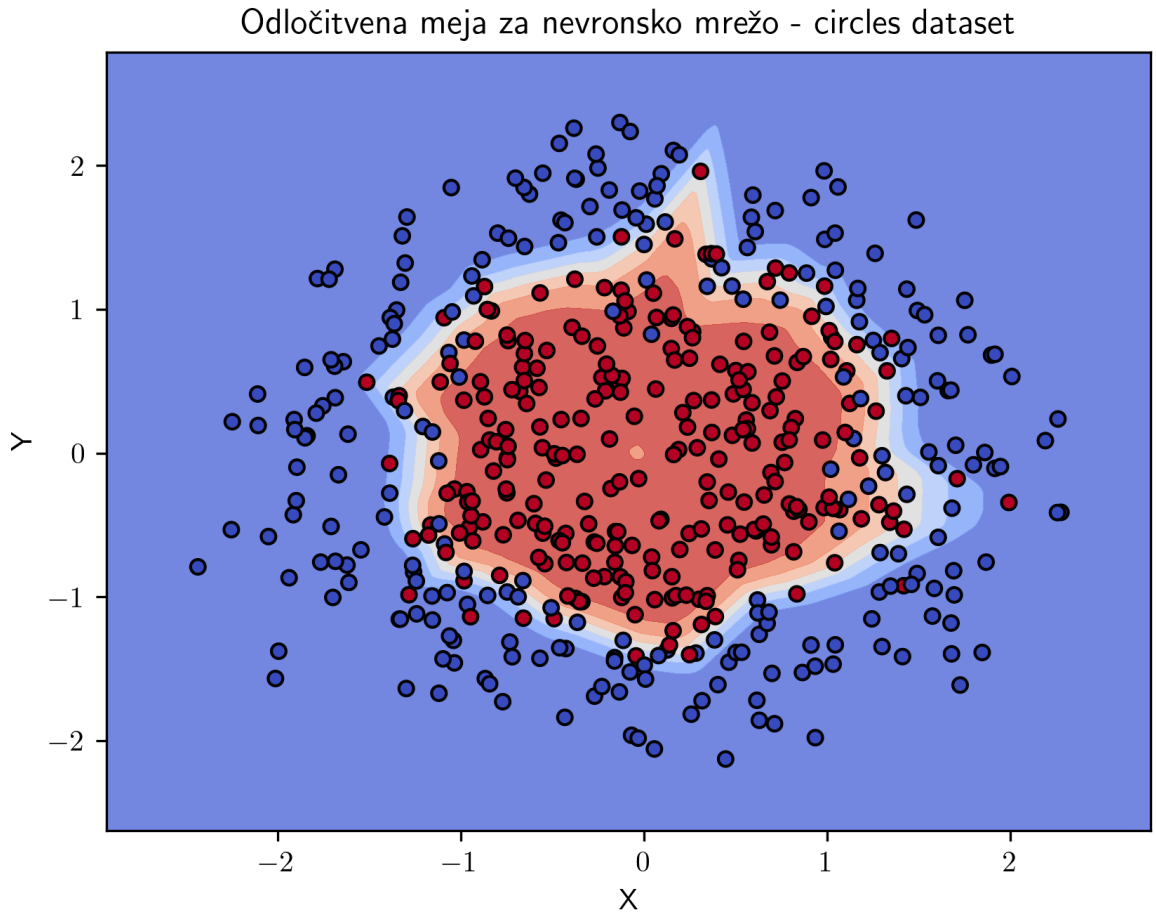
Za DNN pa smo za hiperparametre uporabili:

- število nevronov: 50
- število skritih plasti: 5
- aktivacijska funkcija: ReLU
- aktivacijska funkcija na koncu: sigmoid
- optimizacijski algoritem: adam
- funkcija izgube: binary cross-entrophy
- normalizacija $\mu = 0$ in $\sigma = 1$
- število epoh: 200
- velikost batchov: 1000
- odstotek za testiranje: 20 %


Odločitvena meja za CatBoost - moons dataset
1.5
1.0
0.5
0.0
−0.5
−1.0
Y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
X


Odločitvena meja za nevronsko mrežo - moons dataset
2
1
0
−1
−2
Y
−2
−1
0
1
2
X


Odločitvena meja za CatBoost - circles dataset
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X


Odločitvena meja za nevronsko mrežo - circles dataset
2
1
0
−1
−2
Y
−2
−1
0
1
2
X

Opazimo, da meje pri globokih nevronskih mrežah izgledajo mnogo bolj zvezno in se bolje prilagajajo podatkom. Očitna prednost algoritmov tipa BDT pa je hitrost - tako inference kot treninga.

# 6. Zaključek

V tej nalogi smo uspešno uporabili metode strojnega učenja - globoke nevronske mreže (DNN) in pospešena odločitvena drevesa (BDT) - za klasifikacijo podatkov ozadja in signala. Rezultati kažejo, da ustrezna izbira hiperparametrov in normalizacija podatkov ključno vplivata na uspešnost modelov. Povečanje kompleksnosti pa ne vodi nujno do boljših rezultatov, enako velja tudi za daljše učenje na podatkih. Metode BDT lahko smatramo za dobro alternativo, ki je v nekaterih pogledih precej hitrejša.

---

Luka Skeledžija, Github source 🔗 , 2024