NSU 1 domaca naloga

April 1, 2023

avtor: Anže Mramor

UL ID: 27212071

marec 2023

1 Izbira metode in optimizacija hiperparametrov

1.1 Ročno

Po pregledu podatkov opazimo, da so podatki že v primerni obliki za strojno učenje, ni jih treba spreminjati. Morda nas lahko rahlo skrbijo nekatere spremenljivke, ki imajo visoko koreliranost z drugimi, vendar se bomo s tem po potrebi ukvarjali kasneje. Z vgrajeno funkcijo jih razdelimo na učno in testno množico.

Izberemo si 5 osnovnih klasifikacijskih modelov - **KNN(3)**, **DecisionTreeClassifier**, **GaussianNB**, **RandomForestClassifier** in **GradientBoostingClassifier**. Vse modele s 5-kratnim prečnim preverjanjem naučimo na učni množici in izračunamo ploščino pod ROC krivuljo (AUC). Izberemo tistega z najvišjim povprečnim AUC.

Vrednosti AUC za posamezen model:

- **KNeighborsClassifier(n_neighbors=3)**: 0.5993,
- **DecisionTreeClassifier():** 0.7363,
- **GaussianNB():** 0.8278,
- **RandomForestClassifier():** 0.9305,
- **GradientBoostingClassifier():** 0.9284

Vidimo, da je najboljša metoda po metriki AUC **RandomForestClassifier**. Poglejmo katere hiperparametre ima:

'bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_start'

Opazimo, da ima ta metoda veliko hiperparametrov. Izberimo si par smiselnih, ki jih bomo uporabili:

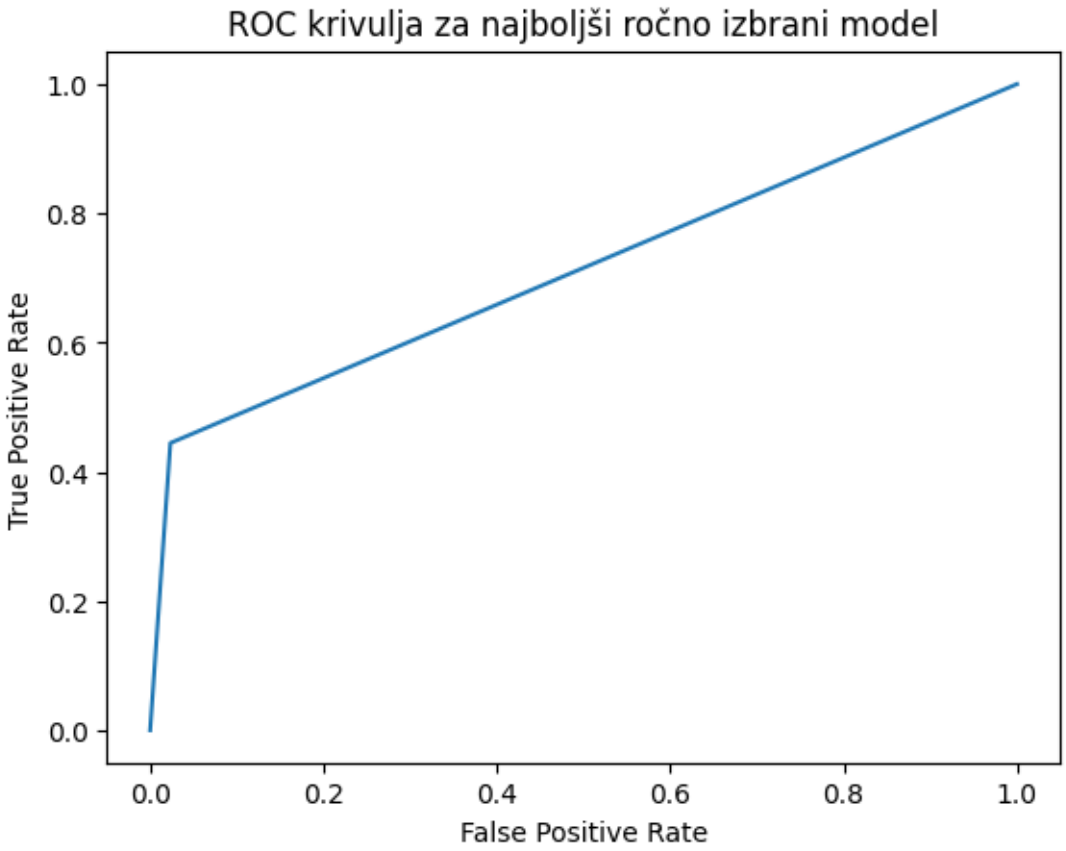
- *criterion*: s katero funkcijo bomo merili kvaliteto delitve ("gini", "entropy", "log_loss")

- *max_depth*, *max_features*, *max_leaf_nodes*, *max_samples*, *min_impurity_decrease*, *min_samples_leaf*, *min_samples_split*, *min_weight_fraction_leaf* se vse tako ali drugače navezujejo na velikost oziroma globino drevesa. Ker imamo binarno klasifikacijo, je pomembno, da je v listih dovolj primerov, da bo napoved zanesljiva. Izberemo si torej *min_samples_leaf*, prek katere bomo modelu podali najmanjše število primerov v listu (0.5%, 1%, 2%, 5%, 10%, 20% podatkov)
- *n_estimators*: najpomembnejši hiperparameter, pomeni število dreves v gozdu (10, 50, 100, 200, 300, 500)

Z vgrajeno funkcijo *GridSearchCV* poiščemo optimalne vrednosti hiperparametrov za naš izbran nabor:

- *criterion*: 'log_loss',
- *min_samples_leaf*: 6,
- *n_estimators*: 200.

Poračunajmo sedaj vrednost AUC modela na testni množici in izrišimo ROC krivuljo.



AUC = 0.7105399792315681

1.2 Avtomatizirano

Izberemo enake algoritme kot v prvi točki, le da sedaj izvajamo izbiro modela in optimizacijo hiperparametrov avtomatsko s *hyperopt*.

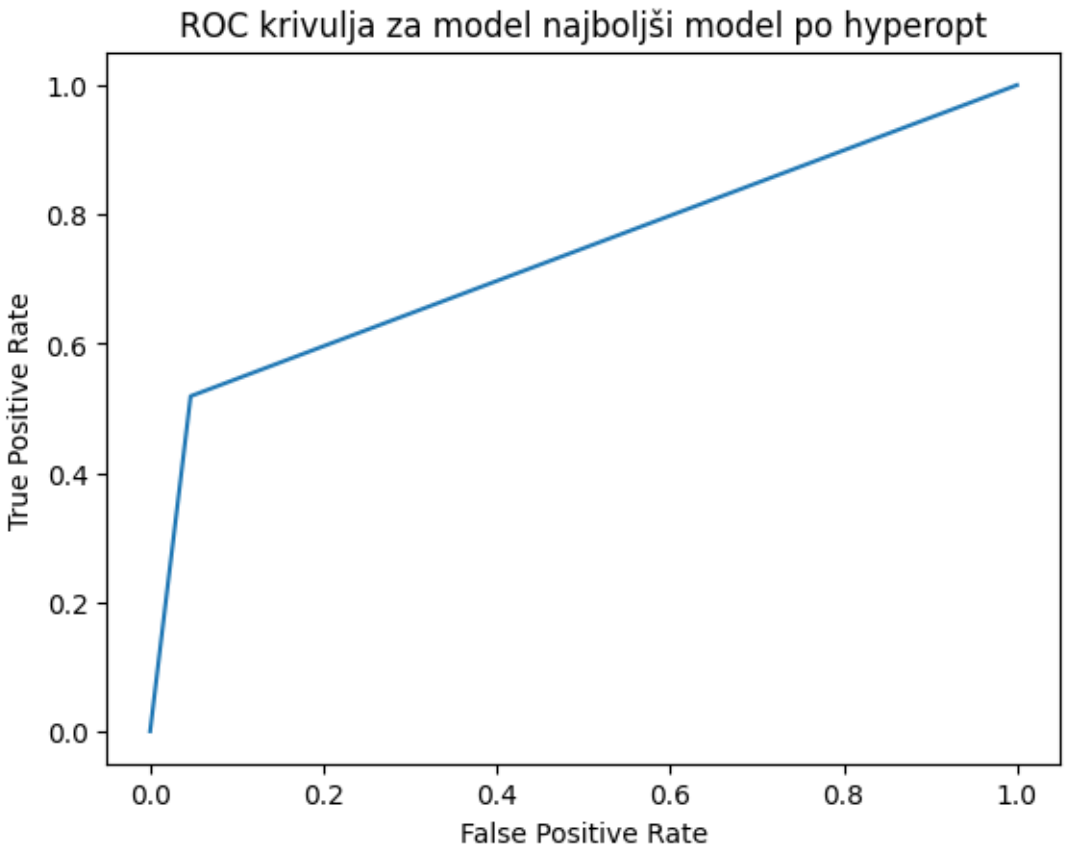
Za vsakega izmed algoritmov si izberemo hiperparametre, ki jih bomo nastavljali in navedemo vrednosti.

- Za KNN bomo izbirali sosede s quniform na intervalu med 1 in 50. Take meje smo si izbrali, ker so ponavadi števila sosedov iz takšnega intervala najbolj aktualna, poleg tega pa za metodo KNN potrebujemo celo število.
- Za model naivnega Bayesa bomo optimizirali hiperparameter *var_smoothing*, s čemer nastavljamo delež največje variance vseh spremenljivk, ki je dodana variancam za računsko stabilnost. Naivni Bayes že sam po sebi nima preveč zanimivih hiperparametrov, zato je optimiziranje tu bolj formalnost
- Za modele odločitvenega drevesa, naključnih gozdov in GradientBoostClassifier vzamemo enake hiperparametre, da jih bomo lažje medsebojno primerjali, saj predvidevamo, da bo optimalni model eden izmed njih. Izberemo *min_samples_leaf*, torej najmanjše zahtevano število vzorcev, da je vozlišče še lahko list, kjer izbiramo s quniform na intervalu med 10 in 240, s korakom 10 (glede na količino podatkov je 240 približno petina podatkov, kar je sicer že precej visoka meja, vendar se jo splača preizkusiti). Upoštevali smo tudi, da bo čas učenja precej daljši, če bomo vrednosti nastavili prenizko, zato jih nekoliko povišamo v primerjavi z ročnim iskanjem. Drugi skupni hiperparameter je *criterion*, pri vsakem izmed modelov vzamemo vse kriterijske funkcije za ocenjevanje kvalitete delitve, ki so na voljo. Zadnji hiperparameter je *n_estimators*, torej število dreves v posameznem gozdu, ki je specifičen za zadnja dva algoritma. Za iskanje vrednosti uporabimo qloguniform, saj lahko tako z ožjim intervalom dosežemo precej višje in bolj raznolike vrednosti kot pri prvem hiperparametru.

Kriterijsko funkcijo sestavimo po zgledu iz vaj. Naša metrika je AUC. Ker funkcija *fmin* minimizira končno vrednost, mora torej naša kriterijska funkcija vrniti vredno 1-AUC (bližje 0 kot smo, boljši je rezultat).

Poženemo 1000 krogov, kar sicer traja precej časa ampak se splača. Za najboljši algoritem nam *hyperopt* predlaga **GradientBoostingClassifier** s hiperparametri:

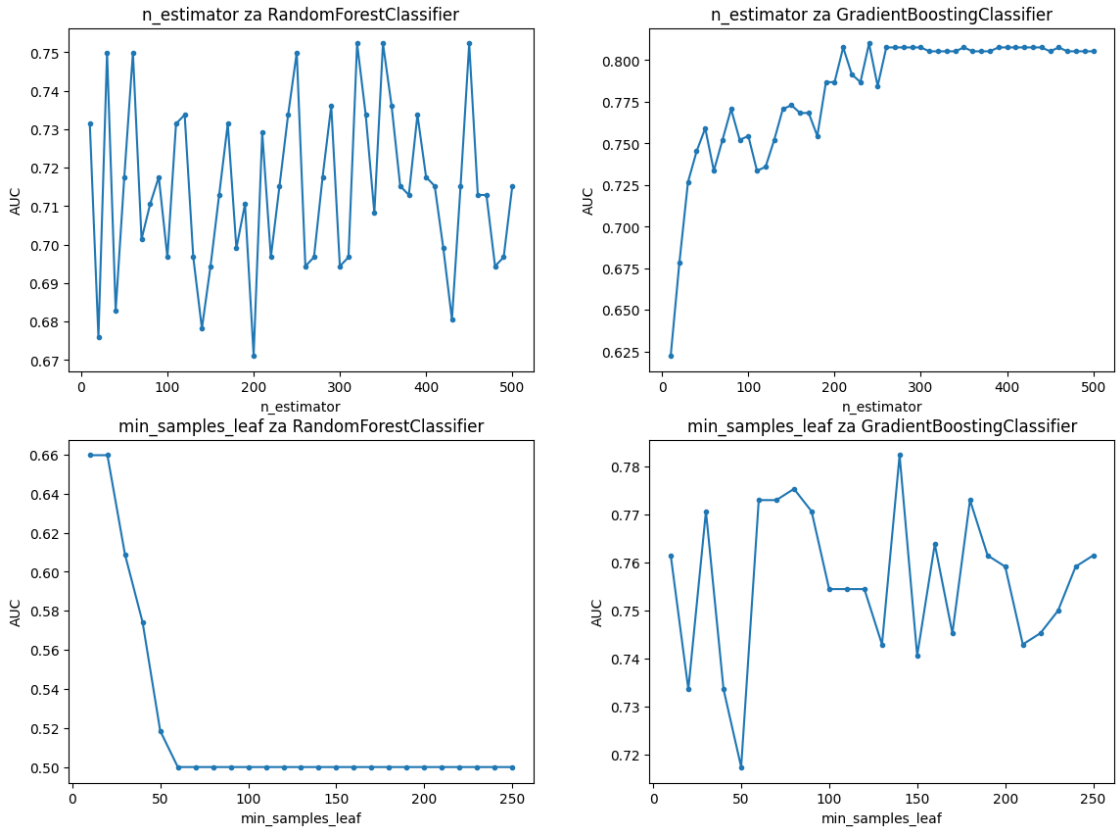
- *criterion* = 'friedman_mse',
- *min_samples_leaf* = 20,
- *n_estimators* = 50.



AUC = 0.7358947732779509

Opazimo lahko, da je AUC v primeru avtomatizirane optimizacije boljši kot pri ročnem nastavljanju, vendar se razlikujeta le za približno 0.02, kar pri vrednosti AUC ni prav veliko. To nam pove, da je naš izbrani model iz 1. točke zelo uporaben, za količino vloženega truda za njegovo pridobitev je rezultat odličen. Verjetno je razlog v tem, da smo že v točki 1.1 naredili prečno preverjanje in preizkusili več modelov, tako da smo že tam dobili precej dober model. Vsekakor pa je jasno, da lahko z avtomatiziranim iskanjem še izboljšamo naše rezultate. Seveda bi se splačalo preizkusiti še širši spekter vrednosti, morda tudi izvesti več evaluacij, saj lahko obstaja kombinacija hiperparametrov, ki je še boljša kot izbrana, vendar je algoritem ni našel.

Izrišimo še graf porazdelitev zmogljivosti za različne konfiguracije. Ker je možnosti precej, si izberemo **RandomForestClassifier** in **GradientBoostingClassifier** - najboljša po AUC pri ročnem in avtomatskem iskanju. Pri obeh spreminjajmo samo parametra *min_samples_split* in *n_estimators*, vsakega posebej, medtem ko so ostali hiperparametri konstantni, ter narišimo rezultate.



Opazimo lahko, da bi bila optimalna vrednost hiperparametra *n_estimators* 90 (ali izbrane vrednosti nad 200) za **RandomForestClassifier**, ter vsaj 200 ali celo 300 za **GradientBoostingClassifier**. Pri *min_samples_leaf* sta pripadajoči vrednosti za vsakega izmed modelov 20 in 140. To se sicer ne sklada z našimi rezultati od prej, saj smo dobili popolnoma drugačne vrednosti. Treba pa je upoštevati, da smo v tem primeru optimizirali vsakega izmed parametrov posebej, medtem ko smo prej iskali optimalno kombinacijo več parametrov. Bolj zanimivo je opazovati, kako se vrednost AUC v splošnem spremnija pri posamaznem modelu za posamezen parameter. Vidimo lahko namreč, da AUC za višje vrednosti *n_estimators* za model **GradientBoostingClassifier** raste (vse do 300, ko se bolj ali manj ustali okoli 0.8 AUC), medtem ko vrednost AUC za model **RandomForestClassifier** skače za različne vrednosti 'n', brez nekega pravila, vseskozi pa se giblje okoli 0.71. Po drugi strani je pri hiperparametru *min_samples_leaf* stvar obrnjena - za višje vrednosti v modelu **RandomForestClassifier** AUC pada (po 50 se ustali okoli 0.5), medtem ko za **GradientBoostingClassifier** vrednost AUC skače okol 0.75. Torej je celotna vrednost AUC modela res najbolje določljiva v kombinaciji več hiperparametrov, ki se medsebojno utežijo in dopolnjujejo.

Zanimivo je tudi opaziti, da je večina izračunanih vrednosti AUC tu višja, kot vrednost AUC v naših "optimalnih" primerih. To nas ne skrbi preveč, saj smo za izračun uporabili testno množico, kar pomeni, da smo testno množico porabili za odločitev o tem katere vrednosti hiperparamterov bi bile najbolj optimalne. Te vrednosti nam ne povedo nič o tem kakšna bi bila potem prava vrednost AUC, saj bi zanjo potrebovali še dodatno testno množico. Verjetno bi bilo simiselno tudi izrisati te grafe preden smo pognajali avtomatski algoritem izbiranja, vendar potem s takšnim pristopom, kot je bil uporabljen, ne bi več imeli na voljo testne množice za testiranje optimalnega modela.

2 Meta učenje

Če želimo najti približno podobno podatkovje moramo poiskati vsa taka podatkovja, ki imajo največ 100 značilk več kot naše podatkovje in največ 1000 primerov več kot naše podatkovje. Meje vzamemo nekoliko večje, za vsak slučaj (saj dodatna vrstica podatkov ali dodatna značilka nikoli zares ne škodi).

Poiskati moramo tista, ki so primerna za klasifikacijo in iz njih določiti metaznačilke. Postopamo enako kot na vajah. Uporabljene metaznačilke so:

'attr_conc.mean', 'attr_conc.sd', 'attr_ent.mean', 'attr_ent.sd', 'attr_to_inst', 'cat_to_num', 'class_conc.mean', 'class_conc.sd', 'class_ent', 'eq_num_attr', 'freq_class.mean', 'freq_class.sd', 'inst_to_attr', 'joint_ent.mean', 'joint_ent.sd', 'mut_inf.mean', 'mut_inf.sd', 'nr_attr', 'nr_bin', 'nr_cat', 'nr_class', 'nr_inst', 'nr_num', 'ns_ratio'

Za iskanje 3 najbližjih sosedov uporabimo metodo **NearestNeighbors**. Algoritem natreniramo na naši množici *meta_znacilke* pridobljenih značilk za podatkovja iz spleta, nato pa z modelom napovemo najbližje sosede glede na podatke v *meta_znacilke0*, torej glede na našo vhodno datoteko. Pri obeh smo seveda izpustili stolpca *name* in *id*. Poiščemo 5 najbližjih sosede, zato da vidimo kateri so še podobna podatkovja.

Imena podatkovij, ki so najbližji sosedi so:

- *hill-valley*
- *steel-plates-fault*
- *ibm-employee-performance*
- *ada_seed_3_nrows_2000_nclasses_10_ncols_100_st...*
- *ada_seed_4_nrows_2000_nclasses_10_ncols_100_st...*

Na roko po navodilih pogledamo, kateri algoritmi so se najbolje odrezali:

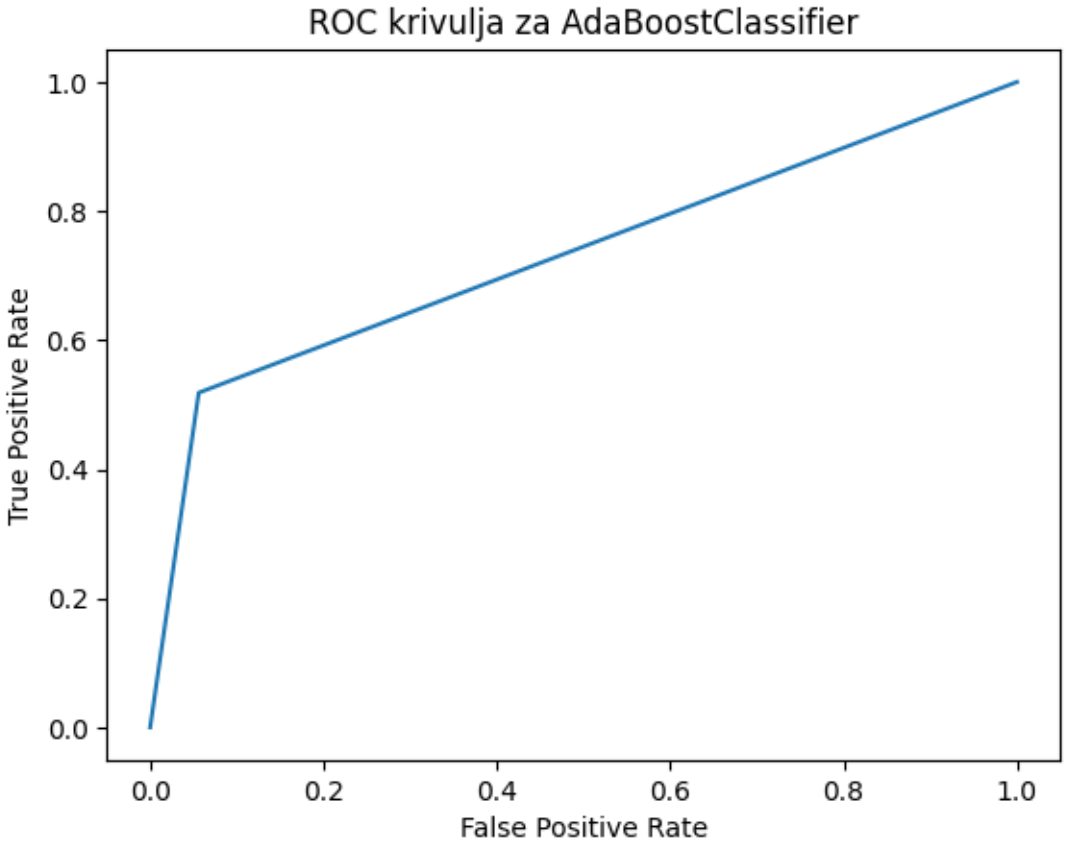
- *hill-valley*: **SVM** (V R) (1. zadetek) in **LogisticRegression** (v pythonu) (2. zadetek)
- *steel-plates-fault*: **RandomForest** (v weka) in **AdaBoostClassifier** (v pythonu)
- *ibm-employee-performance*: **ranger** (v R) in **RandomForestClassifier** (v pythonu)

Očitno so precej dobre rezultate na podobnih podatkovjih dosegali naključni gozdovi, kar je smiselno, saj smo tudi mi v 1. točki določili kot optimalni model **RandomForestClassifier**. Opazimo lahko, da se med najboljšimi algoritmi pojavlja tudi **AdaBoostClassifier**, ki je še en izmed *boosted* algoritmov. Ni presenetljivo, da sta si bila naša najboljša algoritma tako podobna, očitno so tudi modeli, ki imajo implementiran *boost* precej dobri. Poleg tega se je dobro obnesel tudi model logistične regresije, ki je še eden od izjemno popularnih in dobrih metod za klaifikacijo (predvsem binarnih) spremenljivk. Naključnih gozdov na našem podatkovju seveda ne bomo ponovno poganjali, preizkusimo torej le **AdaBoostClassifier** in **LogisticRegression**, da ju lahko primerjamo s prej izbranima algoritmoma. Za prvega si ponovno 'na roke' izberemo par vrednost iza hiperparametre - *n_estimators* (podobno, kot smo že delali) in *learning_rate*. Za drugi model bomo optimizirali samo hiperparameter *penalty* s celotnim možnim naborom vrednosti.

Hiperparametri za **AdaBoostClassifier**: *learning_rate*= 0.1, *n_estimators*= 500

Hiperparametri za **LogisticRegression**: *penalty*= None

AUC za **AdaBoostClassifier**: 0.7312218760816892



AUC za **LogisticRegression**: 0.6641571478020075

AUC modela izbranega ročno - **RandomForestClassifier**: 0.7105399792315681

AUC modela izbranega s hyperopt - **GradientBoostClassifier**: 0.7358947732779509

AUC modela izbranega z meta učenjem - **AdaBoostClassifier**: 0.7312218760816892

AUC modela izbranega z meta učenjem - **LogisticRegression**: 0.6641571478020075

Glede na rezultate bi torej izbrali **GradientBoostClassifier**, ker ima največji AUC. Opazimo pa lahko, da so vsi štirje algoritmi med seboj precej primerljivi, najbolj odstopa model logistične regresije. Očitno je, da ta model za naše podatkovje ne bi bil primeren, saj ga tudi optimizirati ne bi mogli več (razen z različno kombinacijo vhodnih spremenljivk). V nasprotju bi se z bolj natančno optimizacijo hiperparametrov dalo izboljšati ročno nastavljeni model, ter ga tako še bolj približati AUC vrednosti drugih dveh modela. Pričakovati pa je, da so vsi *Classifier* modeli med seboj primerljivi, saj vsi uporabljajo podobne pristope (algoritme) pri učenju. Meta učenje je seveda bilo koristno, saj smo najprej dobili še dodatno potrditev, da je naša izbira modela dobra, saj so tudi drugi, ki so se ukvarjali s podobnimi podatkovji izbrali podobne modele. Poleg tega, nam je dalo idejo za nove algoritme s primerljivo natančnostjo, kar je vedno uporabno. Če bi se še enkrat lotil naloge, bi verjetno naredil vse podobno. Potrudil bi se seveda izboljšati rezultate AUC - to bi dosegel s tem, da bi izvajal 10-kratno prečno preverjanje, namesto 5-kratnega in da bi bolj pazljivo izbral vrednosti za hiperparametre, ter se bolj poglobil v izbiro zares smiselnih vrednosti, saj so sedaj izbrane zelo 'po občutku'.