Univerzitet u Beogradu

Matematički fakultet

Optimizacija hiperparametara modela mašinskog učenja

Vasilije Todorović

Datum: Septembar 2023

Contents

1 Uvod

Optimizacija hiperparametara predstavlja nezaobilaznu fazu u procesu razvoja modela mašinskog učenja. Za postizanje optimalnih performansi modela, ključno je precizno podešavanje hiperparametara - parametara modela koji se ne uče direktno iz podataka, već se njihove vrednosti postavljaju pre treniranja. To uključuje parametre kao što su brzina učenja (learning rate), dubina drveta u slučaju algoritama baziranim na drvećima odlučivanja, broj sakrivenih slojeva i broj čvorova u neuronskim mrežama, i mnoge druge. Loše podešeni hiperparametri mogu dovesti do preprilagođavanja ili potprilagođavanja modela, što rezultira lošim performansama na novim podacima. Ova optimizacija je od suštinskog značaja jer može značajno uticati na sposobnost generalizacije i tačnost predviđanja modela na novim podacima.

U procesu optimizacije hiperparametara, često se koriste metaheuristički pristupi, koji omogućavaju efikasnu pretragu prostora hiperparametara kako bi se pronašli najbolji parametri modela. Dve široko korišćene klase metaheuristika su:

- **S-metaheuristike (Single-solution Metaheuristics)**: Pristupi optimizaciji koji se fokusiraju na pojedinačna rešenja tokom procesa pretrage. Ove metode iterativno unapređuju trenutno rešenje, često putem različitih mehanizama kao što su lokalna pretraga ili simulacije procesa kao što je kaljenje metala (Simulated Annealing). Glavna karakteristika S-metaheuristika je ta da se održava samo jedno trenutno rešenje i prateći skup parametara.
- **P-metaheuristike (Population-based Metaheuristics)**: Pristupi optimizaciji koji rade sa populacijom rešenja tokom procesa pretrage. Ovi algoritmi koriste populaciju rešenja i primenjuju genetske operatore kao što su ukrštanje, mutacija i selekcija kako bi evoluirali populaciju ka boljim rešenjima.

U ovom radu, fokusiraćemo se na primenu algoritama i jedne i druge klase metaheuristika. Konkretno, iz klase S-metaheuristika koristićemo simulirano kaljenje, a iz klase P-metaheuristika genetski algoritam. Ove pristupe uporedićemo sa tradicionalnom metoda koja je koristi za odabir najboljiH hiperparametara - Grid Search, koja sama po sebi predstavlja algoriam grube sile jer pretražuje unaped definisanu mrežu svih mogućih vrednosti hiperparametara kako bi pronašla najbolju kombinaciju.

Analiziraćemo performanse ovih metoda za različite veličine parametarskog prostora i različite veličine podataka (varijacije u broju kolona i redova u podacima), što će nam omogućiti da bolje razumemo kako se metode ponašaju u različitim scenarijima i kako se ponašaju sa različitim nivoima kompleksnosti.

2 Optimizacione tehnike

Jedan od najvažnijih koraka prilikom kontruisanja modela mašinskog učenja jeste odabrati hiperparametre koji obezbeđuju najbolje performanse modela izbegavajući pojavu preprilagođavanja ili potprilagođavanja.

2.1 GridSearchCV

Jedan od najčešćih pristupa prilikom pronalaženja najboljih hiperparametara jeste da ispitamo sve moguće kombinacije njihovih vrednosti i kao najbolju kombinaciju proglasimo onu koja obezbeđuje najbolje rezultate u odnosu na posmatranu metriku. U programskom jeziku Python, pomenuta tehnika implementirana je u okviru klase *GridSearchCV*.

Prostor pretrage hiperparametara konstruiše se na osnovu rečnika tako da se svaki hiperparametar slika u listu unapred zadatih vrednosti koje može uzeti. Ukupan prostor pretrage predstavlja skup svih mogućih kombinacija vrednosti hiperparametara.

GridSearchCV će proći kroz svaku moguću kombinaciju iz prostora pretrage i nad njom obučiti model i oceniti njegove peroformanse. Obučavanje i ocenjivanje performansi odvija se pomoću tehnike unakrsne validacije (eng. cross validation), zbog cega je u nazivu klase prisutno 'CV'. Ideja unakrsne validacije jeste da se skup podataka podeli na preklope (eng. folds), od kojih jedan preklop predstavlja validacioni skup na kojem će se ocenjivati performanse modela koji je obučen na trening podacima, koji zapravo predstavljaju ostale preklope. Proces se ponavlja tako da će u svakoj iteraciji sledeći preklop činiti validacioni skup, a ukupna ocena modela sa datim parametrima predstavlja srednju vrednost ocene na svakom od validacionih skupova. Kao najbolja kombinacija hiperparametra biće odabrana ona koja ima najbolju ocenu na validacionom skupu.

Ovim pristupom vršimo iscrpnu pretragu čitavog prostora, čime ćemo sigurno naći kombinaciju koja ima najbolju ocenu nad posmatranim podacima, ali ukoliko su u pitanju veliki skupovi podataka ili je prostor pretrage veliki, *GridSearchCV* se može pokazati kao neefikasan. Bolje performanse ove tehnike u slučaju velikih podataka i prostora pretrage dobijaju se paralelnim izvršavanjem.

2.2 SimulatedAnnealingCV

Simulirano kaljenje (eng. Simulated Annealing) je predstavnik klase S-metaheuristika. Pomoću ove tehnike možemo takođe pronaći hiperparametre za posmatrani model, ali za razliku od GridSearchCV, ne mora biti izvršena iscrpna pretraga celokupnog prostora pretrage, već se čitav proces nalaženja optimalnih hiperparametara može izvršiti u mnogo manjem broju iteracija.

U ovom radu, implementacija simuliranog kaljenja izvršena je u okviru klase *SimulatedAnnealingCV*. Kao i kod *GridSearchCV*, na osnovu 'CV' može se zakljuciti da je prilikom ocene performansi modela korišćena unakrsna validacija. Objasnimo ideju simuliranog kaljenja na osnovu njegove implementacije unutar ove klase:

1. **Inicijalizacija**: Početne vrednosti hiperparametara se nasumično generišu kao početna tačka za proces optimizacije. Ključni parametri za simulirano kaljenje se podešavaju prilikom inicijalizacije instance klase, uključujući broj iteracija (num_iterations), kriterijum ocene (scoring), broj preklopa za unakrsnu validaciju (cv), opcioni parametar za rano zaustavljanje (early_stopping) i vrednost k koja označava broj iteracija nakon kojeg će se primeniti rano zaustavljanje.
2. **Iterativni proces simuliranog kaljenja**:
   - Proces simuliranog kaljenja počinje generisanjem nasumičnih hiperparametara i ocenjivanjem modela sa tim parametrima.
   - Ocenjena vrednost performansi modela se koristi kao osnov za upoređivanje sa budućim rešenjima.
   - Proces se iterativno ponavlja kroz unapred definisan broj iteracija.
   - U svakoj iteraciji, novi nasumični hiperparametri se generišu, a model se ponovo ocenjuje sa tim parametrima.
   - Ako nova ocena performansi modela premaši prethodnu ocenu, nova parametari se prihvataju kao trenutno najbolje rešenje, i ta nova ocena se koristi za upoređivanje sa budućim rešenjima.
   - U suprotnom, postoji određena verovatnoća da se prihvati lošija ocena, što omogućava izlazak iz lokalnih maksimuma. Ova verovatnoća prihvatanja opada sa brojem iteracija.
   - Ako je omogućeno rano zaustavljanje i nema poboljšanja u oceni tokom određenog broja iteracija ili ukoliko je izvršen broj iteracija koji je označen kao maksimalan, proces se zaustavlja.

Na kraju procesa, vraća se najbolje pronađena kombinacija hiperparametara. Ovaj pristup može značajno ubrzati proces optimizacije u odnosu na iscrpno pretraživanje svih kombinacija hiperparametara, posebno kada je taj prostor veliki i kompleksan. Pošto prilikom ovog procesa ne mora biti izvršena pretraga čitavog prostora pretrage, za dobijeno rešenje se ne može garantovati da ima najbolju ocenu u odnosu na čitav prostor pretrage.

2.3 GeneticAlgorithmCV

Genetski algoritam (eng. genetic algorithm) je tehnika optimizacije inspirisana procesom evolucije u prirodi, i predstavnik je klase P-metaheuristika. Ovo što znači da se oslanja na populaciju rešenja i primenjuje genetske operacije kao što su selekcija, ukrštanje i mutacija kako bi se našlo optimalno rešenje u prostoru hiperparametara modela mašinskog učenja. Genetski algoritmi su takođe korisni kada je prostor hiperparametara velik i kada je potrebno istražiti različite kombinacije hiperparametara kako bi se pronašao dovoljno dobar model.

U ovom radu, pronalaženje optimalnih hiperparametara genetskim algoritmom implementirano je unutar klase *GeneticAlgorithmCV*, koja, kao i prethodni pristupi, koristi unakrsnu validaciju prilikom ocenjivanja performansi modela. Objasnimo ideju genetskog algoritma na onovu njegove implementacije u ovoj klasi:

1. **Inicijalizacija populacije**: Algoritam počinje formiranjem početne populacije rešenja koja predstavlja prvu generaciju. Svako rešenje predstavlja jednu kombinaciju hiperparametara za posmatrani model. Kombinacija hiperparametara se formira tako što svaki parametar uzme nasumičnu vrednost iz liste vrednosti koja mu je pridružena.

2. **Evaluacija populacije**: Nakon inicijalizacije, za svaku jedinku računamo faktor prilagođenosti životnoj sredini, odnosno, za svaku kombinaciju hiperparametara unutar populacije ocenjujemo performanse za zadati model mašinskog učenja. Na osnovu ovih ocena, jedinke se rangiraju unutar generacije, tako da je najprilagođenija jedinka, odnosno kombinacija hiperparametara sa najboljom ocenom, prva u generaciji.

3. **Selekcija**: U ovom koraku se iz trenutne generacije biraju jedinke koje će biti roditelji čijim se ukrštanjem nastati nove jedinke u narednoj generaciji. Odabir roditelja obavlja se po principu turnirske selekcije, gde se bira nekoliko nasumičnih jedinki iz generacije, a za roditelja se bira ona jedinka koja ima najbolju ocenu.

4. **Ukrštanje**: U ovom koraku se na osnovu dva roditelja, ukrštanjem njihovih osobina dobija nova jedinka. U našem slučaju, ukštanje se obavlja tako što se prolazi kroz skup parametra oba roditelja i s jednakom verovatnoćom nova jedinka dobija vrednost posmatranog parametra od jednog ili drugog roditelja.

5. **Mutacija**: Nakon dobijanja nove jedinke, sledi korak mutacije, u kojem je jedinka sklona manjim promenama u osobinama u odnosu na roditelje. U našem slučaju, prolazimo kroz skup parametara, i za svaki parametar računamo verovatnoću da li dolazi do mutacije. Ukoliko je verovatnoća iznad postavljenog praga, menjamo vrednost tog parametra tako što biramo nasumičnu vrednost iz liste mogućih vrednosti tog parametra.

6. **Smena generacije**: Korake selekcije, ukštanja i mutacije obavljamo sve dok se ne popuni nova generacija. Kao opciju možemo uključiti i postojanje elitizma. Ukoliko u procesu selekcije odaberemo dva dobro prilagođena roditelja, može se desiti jedinka dobijena njihovim ukrštanjem ne bude dobro prilagođena životnoj okolini. Iz ovog razloga, želimo da u svakoj generaciji čuvamo određen broj dobro prilagođenih jedinki kako bi se osigurali da ćemo na kraju algoritma imati optimalno rešenje. Ukoliko je prisutan elitizam, određen broj jedinki se prosleđuje u narednu generaciju, a ostatak nove generacije se dobija selekcijom, ukrštanjem i mutacijom jedinki iz prethodne generacije.

7. **Završetak algoritma**: Prilikom inicijalizacije klase *GeneticAlgorithmCV*, kao jedan od ključnih parametara za izvršavanje genetskog algoritma, postavljamo vrednost broja iteracija algoritma, odnosno, posle koliko generacija će se zaustaviti proces evolucije. Po završetku algoritma, jedinka koja se nalazi na prvom mestu u populaciji se vraća kao najbolje rešenje.

Važno je napomenuti da ovaj genetski algoritam ne garantuje pronalaženje apsolutno najboljih parametara u posmatranom prostoru pretragem, ali može efikasno pronaći zadovoljavajuće kombinacije hiperparametara za dati model mašinskog učenja, što ga čini korisnim alatom za optimizaciju u slučajevima gde je broj hiperparametara veliki i složen.

3 Analiza performansi različitih optimizacionih tehnika

U narednom delu ćemo uporediti performanse posmatranih tehnika optimizacije i kvalitet dobijenih hiperparametara. Važno je napomenuti da je implementacija simuliranog kaljenja i genetskog algoritma izvšena bez paralelizacije, za razliku od klase *GridSearchCV*. Iz tog razloga, prilikom analiziranja performansi *GridSearchCV*, posmatraćemo odvojeno primer sa paralelizacijom i bez paralelizacije (postavljanjem parametra n_jobs na 1).

Svaka od ovih tehnika podržava rad sa bilo kojim tipom modela mašinskog učenja, i sa bilo kojim kriterijumom ocene modela. Analiziraćemo performanse ovih tehnika za model stabla odlučivanja, nad prostorima pretrage različitih veličina i nad skupovima podataka različitih dimenzija.
Razmatraćemo prostore pretrage definisanih na osnovu sledećih rečnika:

- 1. prostor pretrage:
  'max_depth' : [4,5,6],
  'criterion' : ['gini','entropy'],
  'min_samples_leaf':[2,3,4],
  'min_samples_split':[3,4,5]
- 2. prostor pretrage:
  'max_depth' : [4,5,6],
  'criterion' : ['gini','entropy'],
  'min_samples_leaf':[2,3,4],
  'min_samples_split':[3,4,5]
- 3. prostor pretrage:
  'max_depth' : [4,5,6,7,8],
  'criterion' : ['gini','entropy'],
  'min_samples_leaf':[2,3,4,5,6],
  'min_samples_split':[3,4,5,6]
- 4. prostor pretrage:
  'max_depth' : [4,5,6,7,8,9,10],
  'criterion' : ['gini','entropy'],
  'min_samples_leaf':[2,3,4,5,6,7],
  'min_samples_split':[3,4,5,6,7]

Ukupan broj različitih kombinacija parametara koji se mogu dobiti na osnovu definisanog rečnika označava veličinu prostora pretrage za posmatrani optimizacioni problem pronalaska najoptimalnijih hiperparametara. Veličine posmatranih prostora pretrage su redom 54,120,200,420.

Posmatrajmo performanse algoritma simuliranog kaljenja u odnosu na prostor pretrage i veličinu podataka. Parametri klase SimulatedAnnealingCV su podešeni tako je može pretražiti najviše trećinu ukupnog prostora pretrage. Podaci koji su korišćeni imaju 98000 redova, ali posmatrajmo zasebno performanse na 100, 1000, 10000 i svih 98000 redova. Na sledećem grafiku možemo videti kako vreme izvršavanja algoritma zavisi od veličine podataka ali i od veličine prostora pretrage.

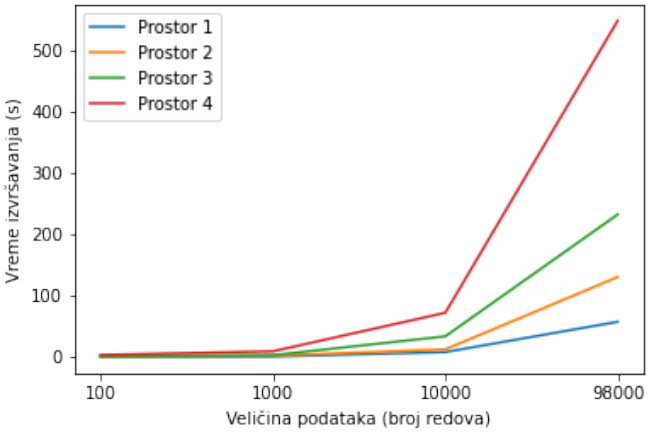

Figure 1: Vreme izvršavanja simuliranog kaljenja za različite prostore pretrage

Ono što možemo zaključiti jeste da vreme izvršavanja algoritma raste s porastom podataka, obzirom da samo treniranje modela na s porastom podataka počinje znatno da traje duže. Takođe možemo videti da vreme izvšavanja raste sa porastom prostora pretrage. Pogledajmo sledeću tabelu sa konkretnim vremenima izvršavanja.

SimulatedAnnealingCV	10^2 redova	10^3 redova	10^4 redova	$9.8 * 10^4$ redova
1. prostor pretrage	0.68s	1.31s	8.58s	58s
2. prostor pretrage	1.13s	2.73s	13s	2min 11s
3. prostor pretrage	1.81s	3.6s	34.25s	3min 53s
4. prostor pretrage	4s	9.83s	1min 13s	9min 9s

Table 1: Vreme izvršavanja simuliranog kaljenja za različite prostore pretrage.

Posmatrajmo sada performanse genetskog algoritma na isti način kao što smo kod simuliranog kaljenja. I u ovom slučaju parametri genetskog algoritma su podešeni tako da obiđe trećinu ukupnog prostora pretrage. Na sledećoj tabeli možemo videti vremena izvršavanja genetskog algoritma:

GeneticAlgorithmCV	10^2 redova	10^3 redova	10^4 redova	$9.8 * 10^4$ redova
1. prostor pretrage	0.45s	0.75s	6.12s	43.06s
2. prostor pretrage	1.19s	2.42s	11.7s	1min 58s
3. prostor pretrage	2.18s	6.24s	25.4s	4min 57s
4. prostor pretrage	4.03s	8.26s	1min 7s	9min 9s

Table 2: Vreme izvršavanja genetskog algoritma za različite prostore pretrage.

Možemo zaključiti da su vremena izvršavanja u nekim situacijama manja a u nekim situacijama veoma bliska izvšavanju simuliranog kaljenja, čime se može zaključiti da se ovaj algoritam pokazuje kao efikasniji.

Prethodne dve tehnike ne koriste paralelno izvršavanje, što je moguće kod tehnike GridSearchCV. Pogledajmo koliko je izvršavanje ove tehnike sa i bez paralelizacije, kako bismo uvideli njen značaj. Na sledećim tabelama nalaze se vremena izvršavanja ove tehnike, koja za razliku od prethodne dve pretražuje čitav prostor pretrage i nalazi apsolutno najbolje moguće rešenje na osnovu zadatog kriterijuma ocene.

GridSearchCV	10^2 redova	10^3 redova	10^4 redova	$9.8 * 10^4$ redova
1. prostor pretrage	2.02s	3.89s	6.12s	43.06s
2. prostor pretrage	4.51s	7.92s	11.7s	1min 58s
3. prostor pretrage	8.24s	16.81s	25.4s	4min 57s
4. prostor pretrage	13.60s	35.33s	1min 7s	9min 9s

Table 3: Vreme izvršavanja GridSearchCV bez paralelizacije za različite prostore pretrage.

GridSearchCV	10^2 redova	10^3 redova	10^4 redova	$9.8 * 10^4$ redova
1. prostor pretrage	0.55s	1.14s	6.3s	43.06s
2. prostor pretrage	1.14s	2.53s	16.37s	1min 58s
3. prostor pretrage	1.84s	4.57s	38.28s	4min 57s
4. prostor pretrage	4.63s	9.38s	1min 10s	9min 9s

Table 4: Vreme izvršavanja GridSearchCV sa paralelizacijom za različite prostore pretrage.

4 Zaključak

5 Literatura