Univerzitet u Beogradu
Elektrotehnički fakultet

**Tamara M. Živković**

# Predviđanje defekata u softveru primenom modela mašinskog učenja optimizovanih metaheuristikama

Doktorska disertacija

Beograd, 2024.

University of Belgrade
School of electrical engineering

**Tamara M. Živković**

# Software defects prediction by machine learning models optimized by metaheuristics

Doctoral dissertation

Belgrade, 2024.

# Podaci o mentoru i članovima Komisije

**Mentor**

dr Dražen Drašković, vanredni profesor
Univerzitet u Beogradu, Elektrotehnički fakultet

**Članovi komisije**

dr Boško Nikolić, redovni profesor
Univerzitet u Beogradu, Elektrotehnički fakultet

dr Miloš Cvetanović, vanredni profesor
Univerzitet u Beogradu, Elektrotehnički fakultet

dr Dragan Pamučar, redovni profesor
Univerzitet u Beogradu, Fakultet organizacionih nauka

Datum odbrane: _____

# Predviđanje defekata u softveru primenom modela mašinskog učenja optimizovanih metaheuristikama

## Sažetak

Testiranje softvera predstavlja ključnu komponentu razvoja softvera i često je ono što pravi razliku između uspešnih i neuspešnih projekata. Iako je izuzetno važno, zbog brzog tempa i kratkih rokova savremenih projekata, često se zanemaruje ili nije dovoljno detaljno zbog nedostatka vremena, što može dovesti do potencijalnog gubitka reputacije, podataka privatnih korisnika, novca, pa čak i ljudskih života u nekim situacijama. U takvim situacijama bilo bi od vitalnog značaja imati mogućnost predviđanja koji softverski moduli su skloni defektima na osnovu skupa metrika softvera i fokusirati testiranje na njih, što je tipičan zadatak klasifikacije.

Modeli mašinskog učenja često su uspešno korišćeni za različite probleme klasifikacije, a u ovom radu se predlaže korišćenje Extreme Gradient Boosting (XGBoost) modela za izvršenje zadatka predviđanja softverskih defekata. Predložena je modifikovana varijanta dobro poznatog algoritma za optimizaciju, nazvanog algoritam pretrage reptila (engl. reptile search algorithm, skr. RSA), kako bi se izvršilo fino podešavanje hiperparametara XGBoost modela. Unapređeni algoritam nazvan je HARSA i evaluiran na kolekciji izazovnih funkcija CEC2019 za uporednu analizu (engl. benchmark), gde je pokazao izuzetne performanse. Kasnije je predstavljen XGBoost model koji koristi predloženi algoritam, i evaluiran je na dva skupa podataka za uporednu analizu za testiranje softvera. Rezultati simulacije su upoređeni sa drugim moćnim metaheurističkim algoritmima koji su korišćeni u istom eksperimentalnom okruženju, pri čemu je predloženi pristup postigao superiornu tačnost klasifikacije na oba skupa podataka. Nakon toga je izvedena SHAP (engl. Shapley Additive Explanations) analiza kako bi se otkrili uticaji različitih metrika softvera na rezultate klasifikacije. Na kraju, razmotrena je i primena ovog rešenja u nastavi, uz osvrt na druga edukaciona okruženja koja se koriste u nastavi iz oblasti testiranja softvera, i uz konkretan primer laboratorijske vežbe koja studentima ilustruje proces razvoja modela za predviđanje softverskih defekata.

**Ključne reči**: testiranje softvera, predviđanje softverskih defekata, XGBoost, reptile algoritam za pretragu, optimizacija metaheuristikama

**Naučna oblast**: Elektrotehnika i računarstvo
**Uža naučna oblast**: Računarska tehnika i informatika

# Software defects prediction by machine learning models optimized by metaheuristics

## Abstract

Software testing is a pivotal aspect of software development, often determining the success or failure of projects. However, amidst contemporary projects' rapid pace and stringent deadlines, testing is frequently overlooked or insufficiently detailed due to time constraints. This negligence can result in potential repercussions such as damage to reputation, compromise of user data, financial loss, and in extreme cases, even human casualties. In such scenarios, the ability to anticipate software modules prone to defects based on software metrics becomes crucial, constituting a typical classification task.

Machine learning models, renowned for their efficacy in addressing classification problems, offer a promising avenue for predicting software defects. In this dissertation, the utilization of the Extreme Gradient Boosting (XGBoost) model is advocated for this purpose. A modified iteration of the Reptile Search Algorithm (RSA), termed HARSA, is proposed for optimizing the hyperparameters of the XGBoost model. The efficacy of this enhanced algorithm is demonstrated through its exceptional performance on a suite of challenging benchmark functions from CEC2019. Subsequently, an XGBoost model employing HARSA is assessed on two software testing benchmark datasets, showcasing superior classification accuracy compared to other metaheuristic algorithms within the same experimental framework.

Furthermore, Shapley Additive Explanations (SHAP) analysis is conducted to elucidate the impact of various software metrics on classification outcomes. Lastly, the educational implications of this solution are explored, contemplating its integration into software testing courses. A practical example of a laboratory exercise illustrates the process of developing a predictive model for software defects to students, fostering a deeper understanding of the subject matter.

**Keywords**: software testing, software defect prediction, XGBoost, Reptile search algorithm, metaheuristics optimization

**Scientific field**: Electrical and Computer Engineering
**Scientific subfield**: Computer Science and Informatics

# Sadržaj

# Lista slika

# Lista tabela

# Poglavlje 1

# Uvod

U savremenom svetu, gde se softverski projekti razvijaju brzim tempom i gde su rokovi često kratki, proces otkrivanja defekata se ponekad zanemaruje. Ovo zanemarivanje dovodi do smanjenog broja sati posvećenih testiranju, što može da rezultira kasnim otkrivanjem defekata, a čak i otkrivanjem defekata od strane krajnjih korisnika u produkciji. Ispravka defekata koji se otkriju kasnije u razvojnom procesu ili u upotrebi od strane korisnika je skuplja. Zbog toga, testeri softvera moraju konstantno da uče i da se prilagođavaju novim situacijama kako bi efikasno testirali aplikacije iz različitih oblasti. Analiza i predviđanje defekata su ključni za procenu napretka projekta i pomažu menadžeru projekta u planiranju procesa testiranja. Takođe, ove analize pomažu u proceni kvaliteta softverskog proizvoda.

Poboljšanje pouzdanosti i funkcionalnosti proizvoda je krajnji cilj ovog procesa. Moduli skloni defektima mogu da se identifikuju pomoću specifičnih metrika, koje su uočene u prethodnim studijama. Važne informacije, poput broja defekata, mesta u kodu gde su otkriveni, ozbiljnosti posledica i distribucije, mogu da se pronađu i analiziraju kako bi se poboljšala efikasnost procesa testiranja.

Ovaj rad se bavi istraživanjem kako mašinsko učenje, optimizovano meteheurističkim algoritmima, može da unapredi proces otkrivanja defekata u softveru. Ovo poboljšanje može dovesti do povećanja kvaliteta softvera i efikasnosti razvojnog procesa. Kroz ovaj proces, može da se postigne bolje razumevanje kako se defekti javljaju, kako da se efikasno otkriju i kako da se isprave na način koji minimizira troškove i povećava kvalitet krajnjeg proizvoda.

Na kraju, ovaj rad će pružiti vredne uvide u to kako se mašinsko učenje može koristiti za poboljšanje procesa otkrivanja defekata u softveru, što će dovesti do poboljšanja kvaliteta softvera i efikasnosti razvojnog procesa. Ovi uvidi će biti korisni za istraživače, inženjere i menadžere projekata koji se bave razvojem softvera.

U literaturi su identifikovane i korišćene brojne metrike za predviđanje defekata u softveru. Ove metrike su ključne za razumevanje i predviđanje kako i gde mogu da se pojave greške u softverskom kodu. Međutim, efikasniji pristup je da se prepoznaju najvažnije metrike i da se fokusira na njihove performanse i unapređenja radi predviđanja grešaka, umesto razmatranja velikog broja metrika.

Navedene metrike mogu da se baziraju na parametrima kao što su broj linija koda, ciklomatska kompleksnost, broj linija sa komentarima, broj operatora i operanada, i mnoge druge. Ove metrike pružaju dublji uvid u strukturu i kompleksnost koda, što može da bude korisno za identifikaciju potencijalnih izvora grešaka.

Dodatne metrike koje mogu da se koriste za predviđanje grešaka su izvedene iz grafa

kontrole toka (eng. Control Flow Graph, skr. CFG) i često se koriste za procenu efektivnosti strukturnog testiranja, odnosno testiranja metodama bele kutije. Ove metrike, izvedene iz CFG, uključuju broj iskaza, grana, odluka i putanja u programskom kodu. Takođe, one mogu da se koriste za merenje pokrivenosti koda (eng. Code Coverage Techniques). Ove metrike pružaju dodatni nivo detalja o strukturi i funkcionalnosti koda, što može da bude korisno za identifikaciju potencijalnih izvora grešaka.

Klasifikacija je tehnika koja se koristi za kategorizaciju softverskih modula kao modula sklonih greškama ili modula koji nisu skloni greškama koristeći metrike zasnovane na klasifikacionim metodama. Ovaj pristup su proučavali brojni istraživači, pa su klasifikacioni algoritmi često korišćena metoda mašinskog učenja za predviđanje grešaka u softveru.

Atributi softverskog koda se klasifikuju kao defektni ili nedefektni, koristeći model klasifikacije zasnovan na podacima o softverskim metrikama iz istorijskih razvojnih projekata. Klasifikacioni algoritam je sposoban da predvidi koji moduli imaju veću verovatnoću da budu skloni greškama, omogućavajući efikasnije korišćenje resursa za testiranje.

Kada se greška otkrije tokom sistemskog testiranja, odgovarajuća kategorija greške modula se označava sa 1, ili sa 0 u suprotnom, ukoliko u modulu nije pronađena greška. Ova informacija se zatim koristi za dalje unapređenje modela klasifikacije, omogućavajući mu da postane sve precizniji u predviđanju grešaka.

Ovaj pristup omogućava razvojnim timovima da bolje razumeju i predvide gde se greške mogu pojaviti u kodu, što može da dovede do smanjenja broja grešaka, povećanja kvaliteta softvera i efikasnijeg korišćenja resursa za testiranje. Kroz ovaj proces, razvojni timovi mogu kontinuirano da unapređuju svoje metode i alate za predviđanje grešaka, što može da dovede do značajnih poboljšanja u kvalitetu i pouzdanosti softvera.

Problem predikcije defekata u softveru spada u probleme binarne klasifikacije. Ovo je izazov koji se često susreće u oblasti mašinskog učenja i predstavlja značajan problem za istraživače i inženjere. Dosadašnja istraživanja pokazala su da za takva predviđanja dobre rezultate daju modeli zasnovani na algoritmima mašinskog učenja. Ovo je veoma dinamična i živa oblast, sa stalnim napretkom i inovacijama, što se može uočiti pretragom relevantne literature.

Predmet ovog rada obuhvata i istraživanje o prethodno korišćenim pristupima mašinskog učenja kako bi se predvidelo da li je određeni modul podložan greškama ili ne. Velika količina dostupnih podataka predstavlja ključni izazov za svaki model mašinskog učenja, jer sposobnost treniranja modela sa više podataka povećava sposobnost modela da izvrši proces generalizacije.

Javno dostupni repozitorijumi podataka imaju ključnu ulogu u nepristrasnoj komparativnoj analizi različitih pristupa. Ovi repozitorijumi omogućavaju istraživačima da testiraju i upoređuju različite metode i algoritme na istim skupovima podataka, što omogućava objektivnu evaluaciju njihovih performansi. Nedavna istraživanja se fokusiraju na klasifikaciju zasnovanu na podacima o greškama i softverskim metrikama. Ove metrike mogu da uključuju različite aspekte softverskog koda, uključujući, ali ne ograničavajući se na strukturu koda, kompleksnost koda, i kvalitet koda.

Na kraju, razmatrana je i praktična primena ovakvog rešenja u nastavi iz oblasti testiranja softvera. Dat je pregled različitih okruženja koja se koriste u nastavi iz ove oblasti, a dat je i praktičan primer jedne laboratorijske vežbe gde se studentima ilustruje proces pravljenja modela koji se može koristiti za predikciju softverskih defekata.

## 1.1 Ciljevi rada i značaj istraživanja

Cilj ovog rada bila je realizacija sistema za predikciju defekata u softverskim modulima, upotrebom modela mašinskog učenja čiji su hiperparametri optimizovani pomoću metaheurističkih algoritama. Ovaj sistem treba da bude sposoban da efikasno i precizno predvidi da li je određeni modul podložan greškama ili ne, na osnovu skupa softverskih metrika za dati modul.

Za postizanje ovog cilja, bilo je neophodno odrediti odgovarajuća podešavanja hiperparametara modela, pošto se za svaki konkretni problem klasifikacije mora podesiti i model koji se koristi. Zatim su se koristili metaheuristički algoritmi za određivanje najboljeg skupa hiperparametara modela. Ovi algoritmi su sposobni da efikasno pretraže prostor mogućih vrednosti hiperparametara i pronađu one vrednosti koje daju najbolje performanse modela.

Implementirani sistem, kao rezultat, radi binarnu klasifikaciju (da li je modul defektan ili ne) na osnovu skupa softverskih metrika za dati modul. Ovaj proces klasifikacije je vođen modelom mašinskog učenja koji je optimizovan pomoću metaheurističkih algoritama. Analizirani su rezultati različitih metaheurističkih algoritama, a na kraju, prikazana je interpretacija najboljih modela generisanih metaheuristikama primenom SHAP (engl. SHapley Additive exPlanations) metoda. Ova metoda omogućava dublje razumevanje kako model mašinskog učenja donosi odluke, što može biti korisno za dalje unapređenje modela i za bolje razumevanje problema predikcije defekata. Osim toga, obavljena je i statistička analiza i dati su statistički testovi koji su pokazali koji modeli su najbolji za ovaj tip problema. Ova analiza pruža objektivnu evaluaciju performansi različitih modela i algoritama, što može biti korisno za dalje unapređenje sistema za predikciju defekata.

Značaj istraživanja ogledao se u primeni realizovanog rešenja u planiranju procesa testiranja. Moduli skloni greškama mogu biti identifikovani pomoću specifičnih metrika, a nakon toga, testiranje se može fokusirati na njih. Time se može poboljšati efikasnost procesa testiranja, kao i ukupan kvalitet softvera.

O značaju teme govori u prilog i činjenica da su oblasti predikcije defekata, kao i problem klasifikacije uopšteno i dalje veoma aktivne, imajući u vidu brojna relevantna istraživanja koja su i dalje aktuelna, a o čemu svedoči i veliki broj prikupljenih referenci iz ovih oblasti. Ova istraživanja pokazala su da je problem predikcije defekata i dalje veoma aktuelan i relevantan, i da postoji velika potreba za daljim istraživanjima i unapređenjima u ovoj oblasti.

Na kraju, dat je i sistematičan pregled okruženja koja se koriste kao nastavna sredstva u oblasti testiranja softvera, kao i konkretan primer laboratorijske vežbe gde se od studenata zahteva da razviju svoj model mašinskog učenja za predikciju softverskih defekata.

## 1.2 Polazne hipoteze

Polazne hipoteze ovog istraživanja zasnivaju se na višegodišnjem iskustvu u oblastima veštačke inteligencije i testiranja softvera. Ove hipoteze su formirane na osnovu dubokog razumevanja i analize trenutnih trendova i izazova u ovim oblastima i one su testirane u daljem radu. Osnovne hipoteze su:

1. Korišćenje modela mašinskog učenja koji do sada nisu dovoljno iskorišćeni za predikciju defekata može da rezultira boljim rešenjima u smislu brzine konvergencije i kvaliteta (performansi, MSE, $R^2$). Ovo bi omogućilo unapređenje rezultata koje su druge metode

optimizacije postigle za istu instancu problema. Ova hipoteza se zasniva na pretpostavci da postoje neiskorišćeni potencijali u primeni mašinskog učenja u ovoj oblasti.

2. Adaptacija i optimizacija postojećih implementacija modela mašinskog učenja za problem klasifikacije defekata softvera može dovesti do poboljšanja u preciznosti i efikasnosti ovih modela. Ovo bi omogućilo bolje razumevanje i predviđanje defekata u softveru, što bi moglo dovesti do smanjenja troškova i vremena potrebnog za otklanjanje defekata.

3. Adaptacija postojećih implementacija metaheuristika koje do sada nisu korišćene u ovoj oblasti, za optimizaciju mašinskog učenja za problem klasifikacije, može dovesti do novih i inovativnih pristupa u rešavanju ovog problema. Ovo bi moglo dovesti do unapređenja u efikasnosti i efektivnosti ovih modela.

4. Unapređenje postojećih implementacija metaheuristika rojeva, koje su ranije već bile korišćene u ovoj oblasti, za optimizaciju modela mašinskog učenja za problem klasifikacije, može dovesti do poboljšanja u performansama ovih modela. Ovo bi moglo dovesti do boljih rezultata u predviđanju defekata u softveru.

5. Moguće je uočiti softverske module koji su podložni defektima analizom metrika koje opisuju softver. Ova hipoteza se zasniva na ideji da postoji veza između određenih metrika softvera i verovatnoće pojave defekata. Ovo bi moglo dovesti do boljeg razumevanja i predviđanja defekata u softveru, što bi moglo dovesti do smanjenja troškova i vremena potrebnog za otklanjanje defekata.

Sve ove hipoteze će biti testirane i verifikovane tokom ovog istraživanja, sa ciljem da se unapredi trenutno stanje u oblastima veštačke inteligencije i testiranja softvera.

## 1.3 Metode istraživanja

Metode istraživanja koje su korišćene u ovom radu su brižljivo odabrane kako bi se formulisale hipoteze koje su relevantne za predmet i cilj istraživanja. Ove metode su uobičajene u praksi i naučno potvrđene, što uključuje analitičko-sintetičku metodu, deduktivno-induktivnu metodu, komparativno-kvantitativnu analizu, empirijsku metodu, statističku analizu, modeliranje i eksperimentalni deo za dokazivanje hipoteza.

Detaljna analiza relevantne naučne i stručne literature u ovoj oblasti, kao i relevantnih činjenica objavljenih putem elektronskih i ostalih medija, sprovedena je analitičkom metodom. Ova analiza omogućava dublje razumevanje trenutnog stanja u oblasti veštačke inteligencije i testiranja softvera, kao i identifikaciju ključnih izazova i pravaca za dalji razvoj.

Sintetičkom metodom su objedinjeni detaljno analizirani elementi pojave u jedinstvenu celinu kako bi se definisala određena pravila u ponašanju istraživane pojave. Drugim rečima, sve prikupljene informacije su grupisane sintetičkim metodama kako bi se izvukli pouzdani zaključci. Ova metoda omogućava da se izolovani delovi informacija kombinuju u koherentnu celinu, što omogućava bolje razumevanje kompleksnih fenomena.

Korišćenje deduktivno-induktivne metode omogućava usmeravanje istraživanja od opšteg ka specifičnom, odnosno od specifičnog ka opštem, kako bi se došlo do relevantnih zaključaka. Ova metoda omogućava da se teorijski koncepti i principi primene na konkretne situacije, kao i da se iz konkretnih situacija izvuku opšti principi.

U okviru komparativno-kvantitativne analize, u skladu s ciljevima istraživanja i fenomenom koji se istražuje, odabrana je metoda analize kao najprikladniji pristup istraživanju. Za analizu konkretnih rezultata primene metoda, tehnika i metaheuristika za rešavanje problema klasifikacije, korišćena je metoda kvantitativne analize. Metodom

komparativne analize su poređeni dobijeni rezultati optimizacije, što je omogućilo zaključak o stvarnom poboljšanju rešavanja ovog problema primenom algoritama inteligencije rojeva.

Primena empirijske metode podrazumeva prikupljanje primarnih podataka kroz praktično eksperimentisanje (izvršavanje algoritma). Prikupljeni primarni podaci su analizirani uzimajući u obzir prethodno razmatrane teorijske i empirijske nalaze, odnosno podatke iz sekundarnih izvora, kako bi se testirale osnovne istraživačke hipoteze.

Statistička analiza podrazumeva korišćenje statističkih podataka i metoda u obradi dobijenih rezultata, u skladu sa postavljenim istraživačkim ciljevima. Za određivanje pravca i stepena međuzavisnosti rezultata različitih metaheuristika, metoda i tehnika, upotrebljene su statističko-ekonometrijske metode trenda i korelacije. Ova metoda omogućava kvantitativnu analizu podataka i identifikaciju ključnih trendova i obrazaca.

Modeliranje obuhvata izradu matematičkih modela koji pružaju relevantan prikaz problema koji se istražuje i rešava. Za rešavanje optimizacionih problema posmatrani su matematički modeli i formulacije, kao i funkcije ciljeva problema istraživanja. Takođe se u eksperimentalnom delu implementacije sagledavaju tehnike metaheuristika u rešavanju problema. Ova metoda omogućava da se kompleksni problemi predstave u formi matematičkih modela, što omogućava precizniju analizu i rešavanje problema.

Sve ove metode su primenjene u ovom istraživanju sa ciljem da se postignu postavljeni istraživački ciljevi i da se testiraju postavljene hipoteze. Kroz ovaj proces, očekuje se da će se doći do novih saznanja i uvida koji će doprineti daljem razvoju oblasti veštačke inteligencije i testiranja softvera. Očekivani naučni doprinosi ovog istraživanja su brojni i značajni. Glavni podsticaj istraživanja sprovedenog za potrebe ove doktorske disertacije je realno očekivanje da će implementirani i prilagođeni algoritmi inteligencije rojeva, koji su primenjeni za optimizaciju neuronskih mreža za predikciju defekata, proizvesti unapređene modele u poređenju sa modelima mašinskog učenja dostupnim u aktuelnoj literaturi. Odnosno, očekivano je da će novi modeli postići bolje indikatore performansi od postojećih modela (MSE, RMSE, $R^2$, itd.).

## 1.4 Naučni doprinosi

Na temelju prethodno iznetog, možemo precizno identifikovati očekivane naučne doprinose:

1. Formiranje taksonomije modela mašinskog učenja i algoritama koji su dati u literaturi za problem klasifikacije i predikcije defekata. Ovo će omogućiti bolje razumevanje postojećih pristupa i tehnika, kao i identifikaciju potencijalnih oblasti za dalji razvoj i unapređenje.

2. Unapređenje rešavanja problema predviđanja defekata primenom realizovanih modela mašinskog učenja pomoću metaheurističkih metoda optimizacije. Očekuje se da će ova unapređenja dovesti do značajnih poboljšanja u preciznosti i efikasnosti predviđanja defekata.

3. Uporedna analiza rezultata različitih modela mašinskog učenja za ovu klasu problema, kao i između razvijenih unapređenih modela, optimizovanih različitim metaheuristikama koje se koriste u literaturi. Ova analiza će omogućiti objektivnu evaluaciju performansi različitih modela i tehnika, kao i identifikaciju najefikasnijih pristupa.

4. Interpretacija najboljih modela i generisanje korisnog znanja za eksperte iz domena testiranja softvera, koje se može upotrebiti za donošenje boljih upravljačkih odluka. Ovo

će omogućiti bolje razumevanje defekata u softveru i njihovih uzroka, što će doprineti efikasnijem i efektivnijem procesu testiranja.

5. Određivanje softverskih metrika koje imaju najviše uticaja na predviđanje. Ovo će omogućiti bolje razumevanje faktora koji doprinose pojavljivanju defekata, što će omogućiti razvoj efikasnijih strategija za njihovo predviđanje i prevenciju.

6. Praktičan primer upotrebe ovog rešenja u nastavi, kako bi se studentima približila primena modela mašinskog učenja i optimizacije pomoću metaheurističkih algoritama, u različitim problemima klasifikacije, na konkretnom primeru predviđanja softverskih defekata.

7. Realizovan sistematičan pregled softverskih rešenja koja se koriste u nastavi iz oblasti testiranja softvera.

8. Predlog konkretne laboratorijske vežbe gde se od studenata zahteva da razviju svoj model mašinskog učenja koji se može koristiti za predviđanje softverskih defekata.

Svaki od ovih doprinosa ima potencijal da značajno unapredi oblast veštačke inteligencije i testiranja softvera. Detaljne taksonomije modela mašinskog učenja i algoritama će omogućiti bolje razumevanje i klasifikaciju različitih pristupa i tehnika koje se koriste u ovoj oblasti. Unapređenje rešavanja problema predviđanja defekata će dovesti do razvoja efikasnijih i preciznijih modela za predviđanje defekata. Uporedna analiza rezultata različitih modela mašinskog učenja će omogućiti objektivnu evaluaciju performansi različitih modela i identifikaciju najefikasnijih pristupa. Interpretacija najboljih modela i generisanje korisnog znanja za eksperte iz domena testiranja softvera će omogućiti bolje razumevanje defekata u softveru i njihovih uzroka, što će doprineti efikasnijem i efektivnijem procesu testiranja. Konačno, određivanje softverskih metrika koje imaju najviše uticaja na predviđanje će omogućiti bolje razumevanje faktora koji doprinose pojavljivanju defekata, što će omogućiti razvoj efikasnijih strategija za njihovo predviđanje i prevenciju. Sve ove aktivnosti će zajedno doprineti ostvarenju glavnog cilja ovog istraživanja, a to je unapređenje oblasti veštačke inteligencije i testiranja softvera.

## 1.5 Struktura disertacije

Ova doktorska disertacija se sastoji od osam poglavlja, skupa neophodnih priloga, kao i pregleda upotrebljene literature.

Prvo poglavlje predstavlja uvod u disertaciju, gde su dati ciljevi rada i značaj istraživanja, polazne hipoteze, opisane metode istraživanja, i predstavljeni naučni doprinosi.

Drugo poglavlje daje kratak uvod u testiranje softvera, kao i opis problema koji se rešava. U ovom poglavlju su ukratko opisane i tehnologije koje su upotrebljene u istraživanju.

Treće poglavlje sadrži opise algoritama koji su korišćeni u disertaciji. Prvo je opisan osnovni algoritam pretrage reptila, a nakon toga i predložena hibridna verzija ovog algoritma.

Četvrto poglavlje predstavlja rezultate testiranja predloženog algoritma na standardnom skupu funkcija za uporednu analizu, što je uspostavljena praksa u modernoj literaturi.

Peto poglavlje daje opis eksperimentalnog okruženja, skupova podataka koji su upotrebljeni u eksperimentima za predviđanje defekata u softveru, kao i metrika koje su upotrebljene za evaluaciju rezultata simulacija.

Šesto poglavlje pruža uvid u rezultate glavnih eksperimenata na dva skupa podataka za predviđanje defekata u softveru, gde je predloženi algoritam upotrebljen za optimizaciju hiperparametara XGBoost modela za klasifikaciju. Data su i poređenja sa rezultatima

drugih modernih metaheurističkih algoritama koji su upotrebljeni u istom eksperimentalnom okruženju, i poređenja sa rezultatima koje su ostvarili tradicionalni modeli mašinskog učenja. Osim toga, data je i statistička analiza rezultata i interpretacija najboljih modela upotrebom SHAP analize.

Poglavlje sedam daje detaljan pregled primene simulatora u edukaciji, sa akcentom na postojeća rešenja koja se mogu koristiti u nastavi iz predmeta Testiranje softvera. U okviru ovog poglavlja, dat je i primer laboratorijske vežbe u okviru koje studenti ispituju modele mašinskog učenja na skupovima podataka za predviđanje defekata u softveru.

U poglavlju osam su data zaključna razmatranja ove disertacije. Na kraju, priloženi su svi neophodni prilozi i pregled upotrebljene literature.

# Poglavlje 2

# Opis problema

Testiranje softvera predstavlja jedan od ključnih segmenata svih projekata razvoja softvera i ako se obavlja ispravno, može biti ključno za uspeh ili neuspeh projekta. Nažalost, testiranje softvera obično trpi zbog brzog tempa savremenih projekata, što često dovodi do nedostatka vremena i smanjenja radnih sati posvećenih testiranju. Ovakav loš pristup testiranju često dovodi do otkrivanja grešaka suviše kasno, često od strane korisnika u produkciji. Generalno je skuplje ispraviti grešku koja se otkrije kasnije u procesu razvoja. U oblastima gde mogu biti ugroženi životi, nedovoljno testiranje može biti opasno. Da bi efikasno testirali aplikacije iz različitih oblasti, testeri softvera moraju stalno učiti i prilagođavati se novim situacijama [1].

Postoje dve osnovne kategorije metoda testiranja softvera: testiranje metodom crne kutije (engl. black-box) i testiranje metodom bele kutija (engl. white-box). Testiranje metodom crne kutije tretira softver koji se testira kao zatvorenu jedinicu, razmatrajući samo ulaze i izlaze sistema bez razmatranja unutrašnjeg funkcionisanja. Ovo testiranje (takođe poznato kao funkcionalno testiranje) obično se koristi da bi se potvrdilo da softver radi kao što je opisano u zahtevima dokumenta, jer se oslanja na korisničke specifikacije. Testiranje metodom crne kutije se obično izvodi u okviru sistemskog testiranja i testiranja od strane krajnjih korisnika.

Testiranje metodom bele kutije (takođe poznato kao strukturalno testiranje) fokusira se na analizu programskog koda i podatke unutar softvera. Koristi se pre svega kako bi se osiguralo da su svi segmenti koda detaljno testirani i da je postignut određeni nivo pokrivenosti koda. Ovaj pristup se obično primenjuje u okviru jediničnog testiranja (engl. unit testing) i integracionog testiranja (engl. integration testing). Da bi se adekvatno testirala aplikacija, važno je koristiti i funkcionalne i strukturalne tehnike testiranja. Proces testiranja softvera ima za cilj otkrivanje defekata u softveru, ali i pronalaženje mera za podizanje kvaliteta softvera.

Softverski defekt predstavlja nedostatak u softveru koji može prouzrokovati netačne ili neočekivane rezultate ili učiniti da se softver ponaša na nepredvidive načine. On predstavlja problem sa softverskim proizvodom koji ometa njegovo očekivano funkcionisanje [2]. Standardne definicije IEEE (Institute of Electrical and Electronics Engineers) za grešku, defekt i softverski otkaz takođe se mogu koristiti za definisanje softverskog defekta [3]. Ljudska greška u kodu dovodi do defekta. Kada sistem izvršava defektni kod, to može dovesti do otkaza tog sistema. Kako su savremena softverska rešenja sve veća i kompleksnija, pregled i testiranje postaju neophodan deo procesa razvoja softvera, naročito kada je reč o identifikaciji i otklanjanju nedostataka. Povećanje veličine i kompleksnosti savremenog softvera čini pregled i testiranje neophodnim delom procesa razvoja, posebno za

identifikaciju i otklanjanje softverskih defekata. Međutim, ispravljanje ovih grešaka može biti skupo. Trošak otkrivanja i ispravljanja defekta predstavlja značajan izdatak u razvoju softvera [4]. Trošak rešavanja softverskog defekta se povećava što se greška otkrije kasnije tokom procesa razvoja softvera. Dosta je skuplje ispraviti defekt u produkciji nego ga identifikovati i rešiti tokom faze kodiranja (trošak može biti i 10 puta veći) [5].

Klasifikacija je tehnika koja se koristi kako bi se softverski moduli kategorizovali kao skloni defektima ili ne, koristeći metode klasifikacije zasnovane na metrikama. Ovaj pristup je proučavan od strane više istraživača, uključujući [6, 7, 8, 9, 10, 11]. Klasifikacioni algoritam je često korišćena metoda mašinskog učenja za predviđanje defekata u softveru [12]. Atributi softverskog koda se klasifikuju kao defektni ili nedefektni koristeći model klasifikacije zasnovan na podacima o metrikama softvera iz prethodno završenih projekata [13]. Klasifikacioni algoritam je sposoban da predvidi koji moduli imaju veću verovatnoću da budu skloni greškama, omogućavajući efikasnije korišćenje resursa za testiranje. Kada se greška otkrije tokom izvršavanja sistemskog testiranja ili terenskog testiranja, relevantna kategorija grešaka modula označava se kao 1, ili 0 u suprotnom. U modeliranju predviđanja, softverske metrike su nezavisne promenljive, a podaci o greškama predstavljaju zavisne promenljive [14]. Hiperparametri klasifikatora se izračunavaju korišćenjem prethodnih softverskih metrika i podataka o greškama. Nekoliko metoda klasifikacije je korišćeno za predviđanje softverskih defekata, a neki od njih su logistička regresija [15], stabla odlučivanja [16, 17], neuronske mreže [18, 19, 20] i naivni Bajes [9].

Brojne metrike za primenu na softveru su identifikovane i korišćene u literaturi za predviđanje defekata. Međutim, bilo bi praktičnije i efikasnije identifikovati najvažnije metrike i fokusirati se na njih kako bi se napravile predikcije o defektima, umesto razmatranja velikog broja metrika. Neke od najvažnijih metrika za predviđanje softverskih defekata razvili su Thomas McCabe [21] i Maurice Halstead [22, 23], a one uključuju broj linija koda, ciklomatsku kompleksnost, broj linija sa komentarima, broj operatora i operanada, i broj praznih linija. Dodatne metrike koje se mogu koristiti za predviđanje defekata izvedene su iz grafa kontrole toka (CFG), i često se koriste za evaluaciju efikasnosti metoda bele kutije. Ove metrike, izvedene iz CFG, uključuju broj izjava, grana i puteva kroz određeni segment koda. One se mogu koristiti za merenje pokrivenosti testiranja skupom testova, i uključuju pokrivenost izjava, grana/odluka i puteva.

Analiziranje i predviđanje defekata su ključni za ispunjenje tri ključna zahteva [24]. Prvo, to pomaže u proceni napretka projekta i pomaže menadžeru projekta u planiranju procesa testiranja. Drugo, pomaže u proceni opšteg kvaliteta proizvoda. I poslednje, ali ne manje važno, poboljšava pouzdanost i funkcionalnost proizvoda. Moguće je identifikovati komponente sklonije greškama koristeći specifične metrike, identifikovane u ranijim studijama predviđanja grešaka. Važne informacije, kao što su broj defekata, zajedno sa njihovom lokacijom, ozbiljnošću i distribucijom, mogu se izvući kako bi se poboljšala efikasnost procesa testiranja. To zauzvrat može poboljšati opšti kvalitet naredne softverske verzije. Predviđanje softverskih defekata nudi dve glavne prednosti. Prvo, poboljšava se ukupni proces testiranja fokusiranjem na module sklonije greškama. Drugo, omogućava identifikaciju potencijalnih kandidata za refaktorisanje koji najverovatnije sadrže defekte [25].

Istraživanje opisano u ovom radu ispituje popularni model mašinskog učenja, eXtreme Gradient Boosting (XGBoost) [26], podešen metaheurističkim algoritmom, i njegovu efikasnost u predviđanju defekata, koristeći nekoliko softverskih metrika. Kako bi postigli visoke performanse, modeli mašinskog učenja zahtevaju podešavanje svojih hiperparametara za svaki pojedinačni problem klasifikacije, jer nažalost, univerzalno rešenje koje će

dobro funkcionisati na svakom mogućem problemu ne postoji. Ako se podešavanje izvršava ručno, izuzetno je vremenski zahtevno, jer bi uključivalo ručno podešavanje hiperparametara modela, obuku modela, a zatim testiranje, u brojnim iteracijama dok model ne počne da daje željeni rezultat. Ovaj rad predlaže rešenje za ovaj problem preporučujući hibridnu varijaciju algoritma pretrage reptila (RSA) [27]. Sa ciljem da proizvede najbolje moguće XGBoost modele, modifikovani RSA je razvijen hibridizacijom sa poznatim algoritmom svitaca [28] kao alat za ovo istraživanje. Performanse osmišljenog algoritma su prvo procenjene na skupu izazovnih CEC2019 funkcija za uporednu analizu, gde je testiran protiv poznatih osnovnih metaheuristika i dve nedavno modifikovane metaheuristike, tj. COLSHADE i SASS. Rezultati CEC2019 funkcija za uporednu analizu su pokazali da je predložena metoda nadmašila druge algoritme korišćene u nedavnim poređenjima. Zatim je predložena metoda integrisana u okvir za klasifikaciju mašinskog učenja sa ciljem optimizacije hiperparametara posmatranog modela za zadatak predviđanja softverskih defekata. Budući da je ovo prva implementacija ovog tipa za ovaj problem, predložena metoda je validirana na paru poznatih setova podataka za uporednu analizu za predviđanje softverskih defekata [29, 30], a eksperimentalni rezultati su upoređeni sa rezultatima postignutim od strane običnog XGBoost klasifikatora, kao i rezultatima dobijenim od ovih modela podešenih drugim modernim metaheurističkim algoritmima koji su implementirani u identičnom eksperimentalnom okviru.

Ključni doprinosi ovog rada su predstavljeni sledećim redosledom:

- Razvijena je poboljšana varijanta RSA metaheuristike da bi se precizno obratila pažnja na poznate nedostatke koje pokazuje osnovni RSA algoritam;
- Predloženi modifikovani algoritam je dalje korišćen za utvrđivanje odgovarajućih vrednosti hiperparametara i poboljšanje tačnosti klasifikacije XGBoost modela kao integralnog dela razvijenog okvira za predviđanje softverskih defekata;
- Eksperimentalni rezultati dobijeni predloženom strukturom su upoređeni sa drugim uspostavljenim metaheuristikama rojeva, koje su korišćene u identičnom simulacionom okviru za predviđanje defekata u softveru u odnosu na skup važnih softverskih metrika. Cilj je bio ispitivanje potencijala drugih metaheuristika koje postižu dobre performanse u procesu podešavanja XGBoost-a za ovaj konkretni problem. Preciznije, širi spektar metaheurističkih algoritama je evaluiran kao dodatni doprinos.
- Procedura Shapley Additive Explanations (SHAP) je sprovedena za najbolje proizvedeni XGBoost model kako bi se bolje razumeli rezultati predviđanja i utvrdila važnost pojedinih karakteristika. Na taj način je stvoren dragocen alat za testere softvera koji će im pomoći da se fokusiraju na module sklonije greškama tokom testiranja.

Ostatak ove disertacije je organizovan na sledeći način. U narednom delu pruža se pregled prethodnih istraživanja koja se bave predviđanjem softverskih defekata, a zatim sledi kratak uvod u osnove razmatranog modela neuronske mreže XGBoost, kao i suštinske aspekte optimizacije metaheuristikama i pristupa baziranih na inteligenciji rojeva. Potom je opisan osnovni RSA pristup, sa akcentom na njegove poznate slabosti, a zatim je predstavljen modifikovani hibridni RSA, uz eksperimente na skupu funkcija za uporednu procenu CEC2019. Sledeći deo objašnjava eksperimentalnu postavku, predstavlja rezultate eksperimenta i diskusiju, a zatim sledi statistička analiza i interpretacija modela. Zaključak sumira istraživanje, ukazuje na moguće pravce budućeg istraživanja i donosi konačne zaključke disertacije.

## 2.1 Pregled literature

### 2.1.1 Testiranje softvera i predviđanje defekata

Primena metoda za predviđanje defekata u softveru pokazala se kao ekonomičnija u odnosu na oslanjanje isključivo na testiranje i preglede. Nedavne studije su pokazale da ovi modeli mogu biti potencijalno efikasniji u otkrivanju defekata u poređenju sa procesima pregleda koji se tradicionalno koriste u industriji [31]. Precizno predviđanje koja komponenta softvera koji se testira ima veću verovatnoću da sadrži defekte može pomoći u prioritizaciji testiranja, smanjenju troškova i poboljšanju ukupnog kvaliteta softvera ciljanjem ovih potencijalno problematičnih oblasti [32]. Stoga je predviđanje defekata u softveru postalo značajno polje istraživanja u softverskom inženjerstvu [11, 33, 34].

Softverska industrija širom sveta brzo raste, a softverske aplikacije postaju sve složenije, dok se, s druge strane, vreme potrebno za razvoj softvera generalno smanjuje. Drugim rečima, danas se prave vrlo složene i sofisticirane aplikacije sa rokovima od samo nekoliko meseci, a sve zbog konkurencije na tržištu. Međutim, ako se aplikacija sa kritičnim defektima objavi na tržištu, to bi moglo potencijalno izazvati frustraciju kod krajnjih korisnika, gubitak privatnih podataka, novca, pa čak i ljudskih života. Svi ti problemi će nesumnjivo dovesti do gubitka zadovoljstva kupaca, smanjenja broja korisnika i uništavanja reputacije kompanije koja je objavila neispravnu aplikaciju. Stoga, sposobnost predviđanja pouzdanosti softvera može biti ključna tokom razvoja softvera, prema istraživanjima [34, 35].

Softverski defekt označava neispravno ponašanje aplikacije tokom izvršavanja, kada aplikacija nije u mogućnosti da ispuni određeni korisnički zahtev. Ova pojava često proizilazi iz grešaka programera tokom faze razvoja ili netačnih specifikacija aplikacije koja se razvija [36]. Mogućnost preciznog predviđanja modula s povećanim rizikom od defekata može značajno doprineti fokusiranju testiranja na ove module, što je pravi put ka stabilnijem softveru. Osim toga, ovo predviđanje može imati šire implikacije na druge aspekte procesa razvoja softvera, uključujući [34]:

- Budžet: Iako nije jedini cilj, precizno predviđanje defekata može pozitivno uticati na budžet, jer je generalno jeftinije ispraviti grešku koja je ranije otkrivena.
- Planiranje projekta: Značajno bi olakšalo procenu troškova i vremena razvoja, posebno kada se radi o modulima i iteracijama razvoja.
- Analiza rizika: Ovaj aspekt se odnosi na sposobnost fokusiranja razvojnog procesa i testiranja na rizične (kompleksne) module koji su verovatno skloni defektima. Takođe utiče na proces donošenja odluka u vezi s troškovima, rasporedom, osobljem, korišćenim alatima, itd.
- Unapređenje procesa razvoja softvera: Precizno predviđanje grešaka može doprineti raznim aktivnostima, uključujući primenu naučenih lekcija, recikliranje koda, čuvanje posebnog podskupa softvera za testiranje (engl. testware), upotrebu alata, ponovnu upotrebu i druge metode koje poboljšavaju zrelost razvojnog procesa.

Nekoliko pristupa mašinskog učenja koristilo se u prošlosti za predviđanje da li je određeni modul pod većim rizikom da bude sklon defektima. Ogromna količina dostupnih podataka predstavlja ključni izazov za bilo koji model mašinskog učenja, jer sposobnost obučavanja modela sa više podataka generalno povećava sposobnost modela da generalizuje. Javno dostupni repozitorijumi podataka, uključujući NASA-MDP i PROMISE [29],

imaju ključnu ulogu u pravljenju nepristrasne komparativne analize različitih pristupa i korišćeni su u ovom radu. Nedavna istraživanja se fokusiraju na pravljenje klasifikacija na osnovu defekata i metrika softvera. Autori u [30] koristili su učenje grafovske reprezentacije na skupu podataka GitHub pull request (GHPR) (koji je korišćen i u ovom radu). Koristili su neuronske mreže na grafu (engl. Graph neural networks, skr. GNN) i uporedili rezultate sa metodama AdaBoost, K-najbližih suseda (engl. K-nearest neighbor, skr. KNN), stablima odlučivanja (engl. decision tree, skr. DT), logističkom regresijom (engl. logistic regression, skr. LR) i metodom slučajnih šuma (engl. random forest, skr. RF). Njihov predloženi model postigao je tačnost predviđanja od oko 82,8%.

Rad [37] raspravlja o nesrazmernoj prirodi skupova podataka za predviđanje defekata i kako to čini učenje mnogo težim zadatkom. Autori u [38] istražili su performanse mašine za vektore podrške (engl. Support vector machine, skr. SVM) za zadatak predviđanja defekata softvera na NASA skupovima podataka. Metoda HVSM-based GRU (engl. Historical Version Sequence of Metrics - Gate Recurrent Unit) korišćena je u [39] za predviđanje modula sklonih defektima u softveru primenom pravila defekata naučenih iz istorijskih verzija istog softvera. Konačno, rad [40] razmatra problem neravnoteže klasa i koristi unapređene autoenkodere sa uklanjanjem šuma za klasifikaciju defekata.

### 2.1.2 XGBoost model

Koristeći pristup aditivnog obučavanja, XGBoost algoritam optimizuje ciljnu funkciju oslanjajući se na rezultat prethodnog koraka na svakoj etapi procesa optimizacije. Jednačina za $t$-tu ciljnu funkciju XGBoost modela je data ispod:

$$F_o{}^i = \sum_{k=1}^{n} l\left(y_k, \hat{y}_k^{i-1} + f_i\left(x_k\right)\right) + R(f_i) + C, \tag{2.1}$$

gde je gubitak $t$-te iteracije predstavljen sa $l$, $C$ je konstantni član, dok $R$ predstavlja regularizacioni parametar modela, definisan kao:

$$R(f_i) = \gamma T_i + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2 \tag{2.2}$$

U suštini, veće vrednosti prilagođavajućih parametara $\gamma$ i $\lambda$ vode ka manje kompleksnoj strukturi stabla. Prvi $g$ i drugi $h$ derivati modela mogu se izraziti pomoću sledećeg para jednačina:

$$g_j = \partial_{\hat{y}_k^{i-1}} l\left(y_j, \hat{y}_k^{i-1}\right) \tag{2.3}$$

$$h_j = \partial^2_{\hat{y}_k^{i-1}} l\left(y_j, \hat{y}_k^{i-1}\right) \tag{2.4}$$

Rezultat se može naći koristeći sledeće jednačine:

$$w_j^* = -\frac{\sum g_t}{\sum h_t + \lambda} \tag{2.5}$$

$$F_o{}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum g)^2}{\sum h + \lambda} + \gamma T, \tag{2.6}$$

gde je $F_o{}^*$ rezultat funkcije gubitka, a $w_j^*$ predstavlja rešenje težina.

### 2.1.3 Metaheuristička optimizacija

Metaheuristički algoritmi su vrsta stohastičkih algoritama koji se često koriste za rešavanje NP-teških problema (nedeterministički u polinomijalnom vremenu teški problemi). Značaj NP-teških problema leži u činjenici da su brojni problemi u računarstvu po prirodi NP-teški. Stoga, deterministički pristupi nisu izvodljivi jer zahtevaju prekomerne računarske resurse.

Metaheuristički algoritmi mogu se klasifikovati u različite kategorije, uzimajući u obzir prirodne fenomene koje primenjuju za modelovanje procesa pretrage [41, 42, 43]. Koristeći ovu šemu klasifikacije, najvažnije kategorije metaheurističkih algoritama su one koje su inspirisane prirodom (na primer genetski algoritmi i inteligencija rojeva), fizičkim događajima (kao što su talasi, oluje i gravitacija), ljudskim ponašanjem (kao što su grupno osmišljavanje ideja i ponašanje na društvenim mrežama) i matematičkim principima (kao što su fluktuacije sinusa i kosinusa).

Genetski algoritmi koriste koncept prirodne evolucije za sprovođenje procesa pretrage, odražavajući prirodnu selekciju i opstanak najjačih pojedinaca. Najjači će biti izabrani za reprodukciju i proizvodnju potomstva koje će izgraditi sledeću generaciju. Koncept genetskih algoritama nije nov, jer je uveden krajem 20. veka [44, 45]. Nove varijante evolucijskih algoritama (engl. evolutionary algorithms, skr. EA) [46] su vrlo popularne u istraživačkim zajednicama, a najpoznatiji primeri su diferencijalna evolucija (engl. differential evolution, skr. DE) [47, 48, 49], genetski algoritam (engl. genetic algorithm, skr. GA) [50, 51, 52] i bioinspirisana optimizacija [53].

Inteligencija rojeva može se klasifikovati kao oblik veštačke inteligencije inspirisan načinom na koji se ponašaju velike grupe životinja. Ove životinje, koje se često nazivaju "rojevima", sastoje se od mnogih pojedinačnih članova koji su relativno jednostavni sami po sebi. Međutim, kada se nalaze zajedno u velikim grupama, mogu pokazati visoko koordinisane i složene obrasce ponašanja, kao što su oni viđeni tokom lova, traženja hrane, parenja ili migracije [54, 55]. Metode inteligencije rojeva poznate su po svojoj sposobnosti da efikasno reše širok spektar složenih problema, uključujući one koji su klasifikovani kao NP-teški i gde je teško otkriti optimalno rešenje u prihvatljivom vremenskom okviru koristeći tradicionalne tehnike optimizacije. Algoritmi inteligencije rojeva su posebno efikasni u pronalaženju dobrih rešenja za ove vrste izazova u relativno kratkom vremenskom intervalu. Klasa inteligencije rojeva obuhvata više algoritama. Neki od najpoznatijih primera uključuju: optimizaciju rojem čestica (engl. particle swarm optimization, skr. PSO) [56], veštačku koloniju pčela (engl. artificial bee colony, skr. ABC) [57], optimizaciju kolonijom mrava (engl. ant colony optimization, skr. ACO) [58], algoritam slepih miševa (engl. bat algorithm, skr. BA) [59, 60] i algoritam svitaca (engl. firefly algorithm, skr. FA) [61]. Neki algoritmi optimizacije su zasnovani na matematičkim funkcijama. Algoritam sinus-kosinus (engl. sine cosine algorithm, skr. SCA) [62] i aritmetički algoritam optimizacije (engl. arithmetic optimization algorithm, skr. AOA) su među najpoznatijim primerima takvih algoritama [63].

Teorema "Nema besplatnog ručka"(engl. - „No free lunch“, skr. NFL) otkriva da ne postoji jedan algoritam optimizacije koji daje garantovano najbolje rešenje za sve probleme. To znači da su različiti algoritmi optimizacije često bolje prilagođeni za rešavanje različitih vrsta problema. Kao rezultat, veliki broj metoda zasnovanih na populaciji implementiran je za rešavanje širokog spektra problema optimizacije [64]. Kada se suoče sa novim problemom optimizacije, u mnogim slučajevima, potrebno je eksperimentisati sa nekoliko metoda kako bi se utvrdila najefikasnija za taj određeni problem. Zbog toga postoji takva raznolikost metaheurističkih metoda, i zašto je važno pažljivo razmotriti koji

je algoritam najprikladniji za određeni zadatak optimizacije.

U poslednjim godinama, algoritmi zasnovani na populaciji uspešno su primenjeni na značajan broj stvarnih problema koji obuhvataju predviđanja infekcija COVID-19 [65, 66], optimizaciju računarstva u oblaku [67, 68, 69], IoT i senzorske mreže [70, 71, 72, 73], problem selekcije atributa [74], klasifikaciju slika za upotrebu u medicini [75, 76, 77], globalne izazove optimizacije [78], sigurnost kreditnih kartica i identifikaciju njihove zloupotrebe [79, 80], analizu proizvodnje i potrošnje električne energije [81, 82, 83], analizu kvaliteta vazduha [84, 85], rešenja za detekciju upada u mrežu [86, 87, 88], i optimizaciju različitih modela mašinskog učenja [89, 90, 91, 92, 93].

# Poglavlje 3

# Algoritmi

Ovaj odeljak prvo opisuje izvornu varijantu algoritma za pretragu reptila (RSA). Nakon toga, diskutuje se o nedostacima osnovnog RSA, a zatim sledi opis predloženog hibridnog algoritma. Ovaj odeljak objašnjava simulacioni okvir gde je predstavljeni algoritam korišćen za rešavanje problema predviđanja defekata softvera.

## 3.1 Osnovni algoritam pretrage reptila

Ovaj segment opisuje RSA [27], modeliran prema ponašanju krokodila u divljini, sa posebnim osvrtom na opkoljavanje, lov i socijalno ponašanje. RSA se sastoji od dve ključne faze: istraživanje (globalna pretraga) i eksploatacija (lokalna pretraga). Krokodili primenjuju taktike opkoljavanja i lova kako bi efikasno uhvatili plen. Ponašanja krokodila su precizno matematički modelirana kako bi se kreirao RSA, koristan za optimizaciju različitih procesa. RSA predstavlja tehniku optimizacije zasnovanu na populaciji i bez upotrebe gradijenata, što ga čini primenljivim za rešavanje širokog spektra problema, od jednostavnih do kompleksnih, uz poštovanje određenih ograničenja. U sklopu RSA, izvršavanje počinje s kolekcijom kandidatskih rešenja (X) generisanih slučajnim redosledom prema jednačini (3.1). Rešenje koje se najbolje pokaže u svakom ciklusu smatra se približno optimalnim:

$$x_{ij} = rand \times (UB - LB) + LB, j = 1, 2, ..., n \tag{3.1}$$

gde $x_{ij}$ predstavlja $j$-tu komponentu $i$-tog rešenja, $n$ je dimenzija veličine povezanog problema, $rand$ je proizvoljan broj, $UB$ označava gornju granicu problema, a $LB$ prikazuje donju granicu. Proces pretrage je podeljen na dve glavne faze: istraživanje i eksploataciju, gde svaka ima svoj skup strategija. Metoda istraživanja uključuje strategije visokog hodanja i hodanja na trbuhu, dok metoda eksploatacije uključuje strategije koordinacije lova i saradnje.

### 3.1.1 Faza opkoljavanja

Tokom opkoljavanja, krokodili deluju na dva načina - koriste visoko hodanje ili hodanje na trbuhu. Pokreti su specifično prilagođeni za istraživanje i pretragu plena na širokom području, a ne za prikradajući pristup i hvatanje određenog plena kao u fazi lova. Što su pokreti skloniji uznemiravanju plena, to je teže krokodilu da ga uhvati. Istraživačka pretraga omogućava krokodilima da pokriju veliko područje kako bi locirali oblasti visoke gustine plena. To može zahtevati više pokušaja. Štaviše, pokreti istraživanja poput visokog

hodanja i hodanja na trbuhu se koriste tokom ove faze procesa pretrage kako bi pomogli fazi lova omogućavajući šire i opsežnije pretraživanje. Algoritam RSA je sposoban da prelazi između opkoljavanja (istraživanja) i lova (eksploatacije), a ove tranzicije se pokreću deljenjem ukupnog broja iteracija na četiri dela. Procedura istraživanja RSA ispituje domen pretrage i identifikuje potencijalna rešenja, koristeći dve primarne tehnike pretrage: strategiju visokog hodanja i strategiju hodanja na trbuhu.

Ova faza pretrage se odvija prema dva zahteva: pristup visokog hoda se koristi kada je $t <= \frac{T}{4}$, a hodanje trbuhom se koristi kada je $t <= 2\frac{T}{4}$ i $t > \frac{T}{4}$. Stoga, tokom faze istraživanja, strategija visokog hoda će se koristiti otprilike za polovinu ukupnog broja iteracija, a strategija hodanja na trbuhu će se koristiti za preostalu polovinu. Ovo su dve različite metode istraživačke pretrage. Pored toga, primenjuje se stohastički koeficijent skaliranja kako bi se proizvela raznovrsnija rešenja i istražio širi spektar regiona. U ovom istraživanju, koristi se jednostavno pravilo koje modelira ponašanje krokodila tokom opkoljavanja. Jednačine koje ažuriraju pozicije jedinki tokom faze istraživanja predstavljene su u jednačini (3.2):

$$x_{(ij)}(t+1) = \begin{cases} Best_j(t) \times -\eta_{(i,j)}(t) \times \beta - R_{(i,j)}(t) \times rand & t \leq \frac{T}{4} \\ Best_j(t) \times x_{(r_1,j)} \times ES(t) \times rand & t \leq 2\frac{T}{4} \text{ and } t > \frac{T}{4} \end{cases} \tag{3.2}$$

gde $Best_j(t)$ predstavlja $j$-tu poziciju trenutno najbolje jedinke, $rand$ označava proizvoljnu vrednost unutar $[0, 1]$, $t$ predstavlja trenutnu iteraciju, dok $T$ označava maksimalni broj iteracija. Pored toga, izraz $\eta_{(i,j)}(t)$ predstavlja operator lova $j$-te pozicije $i$-te jedinke, koji se može dobiti korišćenjem jednačine (3.3). Varijabla $\beta$ predstavlja parametar osetljivosti, koji se koristi za podešavanje tačnosti istraživanja u odnosu na proces opkoljavanja tokom iteracija, i originalno je fiksiran na vrednost 0.1. $R_{(i,j)}(t)$ predstavlja funkciju smanjenja, koja se koristi za sužavanje prostora pretrage i može se odrediti korišćenjem jednačine (3.4). Vrednost $r_1$ označava proizvoljan broj unutar $[1, N]$, dok $x_{(r_1,j)}$ označava nasumičnu poziciju $i$-te jedinke, gde je $N$ broj rešenja kandidata. Konačno, $ES(t)$ se odnosi na evolutivni smisao, implementiran kao verovatnoća koja uzima nasumično opadajuće brojeve u opsegu $[-2, 2]$ tokom iteracija, i može se dobiti pomoću jednačine (3.5).

$$\eta_{(i,j)}(t) = Best_j(t) \times P_{(i,j)} \tag{3.3}$$

$$R_{(i,j)}(t) = \frac{Best_j(t) - x_{(r_2,j)}}{Best_j(t) + \epsilon} \tag{3.4}$$

$$ES(t) = 2 \times r_3 \times (1 - \frac{1}{T}) \tag{3.5}$$

gde $\epsilon$ predstavlja malu vrednost, $r_2$ označava proizvoljnu vrednost uzetu iz opsega $[1, N]$, $r_3$ je proizvoljna celobrojna vrednost unutar $[-1, 1]$. $P_{(i,j)}$ označava procentualnu razliku između $j$-te pozicije trenutno najbolje jedinke i trenutnog rešenja, izračunatu kao:

$$P_{(i,j)} = \alpha + \frac{x_{(i,j)} - M(x_i)}{Best_j(t) \times (UB_{(j)} - LB_{(j)}) + \epsilon} \tag{3.6}$$

U jednačini (3.6), $M(x_i)$ predstavlja prosečnu poziciju jedinke $i$, dobijenu jednačinom (3.7). $UB_{(j)}$ i $LB_{(j)}$ odgovaraju gornjim i donjim granicama $j$-te pozicije. Varijabla $\alpha$ je osetljivi kontrolni parametar, koji kontroliše tačnost faze istraživanja (razlika među

kandidovanim jedinkama), i samim tim kontroliše saradnju u lovu između krokodila tokom iteracija, koja je postavljena na fiksnu vrednost od 0.1.

$$M(x_i) = \frac{1}{n}\sum_{j=1}^{n} x(i,j) \tag{3.7}$$

### 3.1.2 Faza lova (eksploatacija)

U ovom odeljku predstavljena je eksploatacija, odnosno faza lova algoritma RSA. Krokodili imaju dve strategije tokom lova, koordinaciju i saradnju. Ove strategije su intenzivne tehnike koje se koriste za pretragu tokom eksploatacije, odnosno lova, i dizajnirane su da se fokusiraju na određeni ciljani plen. To se razlikuje od faze opkoljavanja, gde su pokreti manje intenzivni i manje je verovatno da će uznemiriti plen. Na taj način se krokodilu omogućava da lako pristupi plenu tokom faze lova koristeći koordinaciju i saradnju u lovu. Pretraga tokom faze eksploatacije fokusira se na pronalaženje skoro optimalnog rešenja, za šta može biti potrebno više pokušaja. Tokom ove faze procesa optimizacije koriste se mehanizmi eksploatacije kako bi se sprovelo intenzivno pretraživanje u blizini optimalnog rešenja, sa naglaskom na komunikaciju između agenata. Mehanizmi eksploatacije RSA koriste prednost prostora pretrage i fokusiraju se na pronalaženje optimalnog rešenja koristeći dve glavne strategije pretrage, koordinaciju lova i saradnju u lovu, koje su modelirane u jednačini (3.8). Pretraga tokom ove faze se odvija prema uslovima: strategija koordinacije lova se koristi kada je $t <= 3\frac{T}{4}$ i $t > 2\frac{T}{4}$ , a strategija saradnje u lovu se koristi kada je $t <= T$ i $t > 3\frac{T}{4}$.

$$x_{(ij)}(t+1) = \begin{cases} Best_j(t) \times P_{(i,j)}(t) \times rand & t \leq 3\frac{T}{4} \text{ and } t > 2\frac{T}{4} \\ Best_j(t) - \eta_{(i,j)}(t) \times \epsilon - R_{(i,j)}(t) \times rand & t \leq T \text{ and } t > 3\frac{T}{4} \end{cases} \tag{3.8}$$

ovde, $Best_j(t)$ označava $j$-tu poziciju trenutno najbolje jedinke, $\eta_{(i,j)}(t)$ definiše operator lova koji pripada $j$-toj poziciji $i$-te jedinke, u skladu sa jednačinom (3.3). $P_{(i,j)}(t)$ ponovo označava procentualnu razliku između posmatrane i najbolje jedinke, kako je navedeno u jednačini (3.6), dok $R_{(i,j)}(t)$ sužava prostor pretrage kako je definisano jednačinom (3.4).

Procedure pretrage tokom faze eksploatacije, kao što su koordinacija i saradnja u lovu, dizajnirane su da spreče zadržavanje u lokalnim optimumima. Ove tehnike pomažu istraživačkoj pretrazi u identifikovanju optimalnog rešenja, dok takođe održavaju raznolikost kandidovanih rešenja. Par kontrolnih promenljivih ($\beta$ i $\alpha$) se koristi za generisanje nasumične vrednosti u svakom krugu, omogućavajući da se istraživanje nastavi tokom celog procesa pretrage, ne samo u početnim fazama, već i tokom kasnijih faza. Ovaj aspekt pretrage je posebno koristan kada se susreće sa stagniranjem u lokalnim optimuma, posebno tokom kasnijih faza pretrage. Pseudokod osnovne implementacije RSA je prikazan u Algoritmu 1.

## 3.2 Predloženi novi hibridni RSA algoritam

Originalni RSA je visoko cenjen kao superiorna tehnika optimizacije, ali kao i druge metaheuristike, ima neka ograničenja. Studije na funkcijama za testiranje pokazale su da je RSA dobar u istraživanju prostora pretrage, ali mu nedostaje sposobnost da eksploatiše

**Algorithm 1** Pseudokod osnovnog RSA algoritma

Produce a random starting population of $N$ solutions
Initialize RSA control variables $\alpha$, $\beta$, etc.
**while** $(t < T)$ **do**
  evaluate individuals in terms of the fitness function
  determine the best individual attained until now
  update $ES$ with respect to the Eq. (3.5)
  **for** $(i = 1$ to $N)$ **do**
    **for** $(j = 1$ to $n)$ **do**
      Update $\eta$, $R$ and $P$ according to the Eq. (3.3), Eq. (3.4) and Eq. (3.6)
      **if** $(t \leq \frac{T}{4})$ **then**
        $x_{(ij)}(t+1) = Best_j(t) \times -\eta_{(i,j)}(t) \times \beta - R_{(i,j)}(t) \times rand$
        **High walking phase**
      **else if** $(t \leq 2\frac{T}{4}$ and $t > \frac{T}{4})$ **then**
        $x_{(ij)}(t+1) = Best_j(t) \times x_{(r_1,j)} \times ES(t) \times rand$
        **Belly walking phase**
      **else if** $(t \leq 3\frac{T}{4}$ and $t > \frac{2}{T}4)$ **then**
        $x_{(ij)}(t+1) = Best_j(t) \times P_{(i,j)}(t) \times rand$
        **Hunt coordination phase**
      **else**
        $x_{(ij)}(t+1) = Best_j(t) - \eta_{(i,j)}(t) \times \epsilon - R_{(i,j)}(t) \times rand$
        **Cooperated hunt phase**
      **end if**
    **end for**
  **end for**
  $t = t + 1$
**end while**
**return** Best determined individual

prostor pretrage tokom kasnijih iteracija kada bi trebalo da sužava pretragu. Pored toga, algoritam svitaca je široko poznat po svojim snažnim sposobnostima eksploatacije ([28]).

Pristup opisan u ovom radu rešava slabost osnovnog RSA algoritma uključivanjem mehanizma koji kombinuje sposobnosti istraživanja RSA sa snagama eksploatacije algoritma svitaca. Inicijalno, rešenja se ažuriraju koristeći jednačinu pretrage RSA (jednačina (3.2)). Kako algoritam napreduje i identifikuju se povoljni regioni domena pretrage, eksploatacija se pojačava uključivanjem jednačine pretrage svitaca (jednačina (3.9)):

$$X_i^{t+1} = X_i^t + \beta_0 \cdot e^{-\gamma r_{i,j}^2}(X_j^t - X_i^t) + \alpha^t(\kappa - 0.5) \tag{3.9}$$

gde je faktor randomizacije u jednačini predstavljen sa $\alpha$, $\kappa$ predstavlja vrednost izabranu iz Gausove raspodele. Rastojanje između rešenja $i$ i $j$ je predstavljeno kao $r_{i,j}$.

U ovom radu, rešavamo osnovni problem u algoritmu RSA uvođenjem dva nova kontrolna parametra, od kojih je jedan varijabilni parametar pretrage $vs$. Ovaj parametar nam omogućava da pređemo na drugačiju metodu pretrage tokom kasnijih faza izvršavanja, omogućavajući kombinovani režim pretrage kada broj iteracija, $t$, premaši vrednost $vs$ parametra. Ova vrednost vs parametra je otkrivena eksperimentalno i postavljena na vrednost $maxIter/5$ (3 u eksperimentima, koji su imali maksimum od 15 iteracija).

Druga kontrolna promenljiva, režim pretrage $sm$, odlučuje o metodi pretrage za svaku jedinku, bilo RSA ili FA. Za svaku jedinku generiše se proizvoljan broj $rnd$ unutar $[0, 1]$ i ako je $rnd$ manje od $sm$, izvodi se RSA pretraga, a ako je $rnd$ veće od $sm$, izvodi se

FA pretraga. Parametar *sm* se postepeno smanjuje tokom iteracija, premeštajući fokus ka preciznijoj FA pretrazi kako algoritam konvergira. Početna vrednost *sm* je unapred postavljena na 0.8 i smanjuje se upotrebom jednačine (3.10) kako je utvrđeno empirijskom analizom:

$$sm_t = sm_{t-1} - (sm_{t-1}/10) \tag{3.10}$$

Ovaj unapređeni algoritam je nazvan hibridni RSA (HARSA). Kratak pseudokod HARSA je dat u Algoritmu 2.

**Algorithm 2** Pseudokod predloženog HARSA algoritma

```
Produce a random starting population of N solutions
Initialize RSA control variables α, β, etc.
Initialize sm = 0.8 and vs = 3
while (t < T) do
  for Each solution X in the population do
    if t < vs then
      Execute RSA search, according to the procedure given in Algorithm (2).
    else
      Generate random value rnd.
      if rnd > sm then
        Execute FA search, according to the Eq. (3.9).
      else
        Execute RSA search, according to the procedure given in Algorithm (2).
      end if
    end if
  end for
  Update parameters' of algorithm.
  Update candidate solutions' positions.
  Update sm according to 3.10
  t = t + 1
end while
return Best discovered solution
```

# Poglavlje 4

# Simulacije sa standardnim funkcijama za uporednu analizu

Prema uspostavljenim praksama u modernoj literaturi, predloženi metod je inicijalno testiran na standardnim ograničenim (ili neograničenim) funkcijama za uporednu analizu pre procene njegovih performansi u praktičnom podešavanju XGBoost-a za problem klasifikacije softverskih defekata koristeći HARSA metaheuristike.

Razlog za izbor CEC2019 skupa test funkcija bio je dvostruk: on je složeniji i teži u poređenju sa drugim skupovima za uporednu procenu, kao što su standardne test funkcije i CEC2017, i ranije je korišćen za evaluaciju osnovnog RSA kada je pristup inicijalno uveden [27]. Ovaj skup sadrži deset funkcija, a njihovi odgovarajući detalji, uključujući ime, dimenziju, granične vrednosti pretraživačkog prostora i globalni optimum, predstavljeni su u Tabeli 4.1.

Da bi se sprovela sveobuhvatna komparativna analiza, simulacije CEC2019 izvedene su koristeći i originalni RSA algoritam i novouvedeni HARSA algoritam. Pored toga, da bi se proširio opseg analize, razmatrane su i druge savremene metaheuristike, uključujući optimizaciju rojem čestica (PSO)[56], veštačku koloniju pčela (ABC)[57], algoritam svitaca (FA)[28], i optimizaciju grupom šimpanzi (engl. chimp optimization algorithm, skr. ChOA)[94]. Ovaj izbor algoritama imao je za cilj da postigne ravnotežu između tradicionalnih metoda poput PSO i novijih poput ChOA. Konačno, u poređenja su uključena i dva od najnovijih najbolje ocenjenih algoritama, a to su COLSHADE [95] i samoadaptivni sferni algoritam pretrage (engl. self-adaptive spherical search algorithm, skr. SASS) [96].

Eksperimentalni uslovi korišćeni u ovom istraživanju bili su uporedivi sa onima u istraživanju [27], gde su maksimalni broj iteracija ($T$) i veličina populacije ($N$) bili postavljeni na 500 i 30, respektivno. Pored toga, zbog stohastičke prirode korišćenih metoda, simulacije su nezavisno replicirane 30 puta ($runtime = 30$), a beležene su metrike kao što su najbolje, najgore, srednje, prosečne vrednosti i standardna devijacija.

**Tabela 4.1:** Pregled upotrebljenih CEC2019 funkcija za uporednu analizu

| Br. | Funkcija | $F_i^* = F_i(x^*)$ | Dim | prostor pretrage |
|---|---|---|---|---|
| 1 | Storn's Chebyshev Polynomial Fitting Problem | 1 | 9 | [-8192, 8192] |
| 2 | Inverse Hilbert Matrix Problem | 1 | 16 | [-16384, 16384] |
| 3 | Lennard-Jones Minimum Energy Cluster | 1 | 18 | [-4,4] |
| 4 | Rastrigin's Function | 1 | 10 | [-100,100] |
| 5 | Griewank's Function | 1 | 10 | [-100,100] |
| 6 | Weierstrass Function | 1 | 10 | [-100,100] |
| 7 | Modified Schwefel's Function | 1 | 10 | [-100,100] |
| 8 | Expanded Schaffer's F6 Function | 1 | 10 | [-100,100] |
| 9 | Happy Cat Function | 1 | 10 | [-100,100] |
| 10 | Ackley Function | 1 | 10 | [-100,100] |

Upoređivanje performansi metrika konkurentnih algoritama je dato u Tabeli 4.2.

**Tabela 4.2:** Rezultati izvršavanja HARSA algoritma nad CEC2019 test funkcijama

| Fun | Measure | HARSA | RSA | PSO | ABC | FA | ChOA | SASS | COLSHADE |
|---|---|---|---|---|---|---|---|---|---|
| CEC-2019-F1 | Worst | **4.25E+04** | 6.33E+04 | 2.23E+13 | 4.66E+10 | 3.15E+08 | 1.01E+09 | 4.64E+04 | 7.13E+04 |
| | Average | **3.62E+04** | 5.65E+04 | 8.12E+12 | 2.41E+10 | 1.01E+08 | 5.12E+08 | 4.05E+04 | 5.42E+04 |
| | Best | **3.15E+04** | 5.06E+04 | 8.26E+11 | 4.33E+09 | 5.03E+05 | 2.57E+08 | 3.52E+04 | 3.79E+04 |
| | STD | 4.79E+03 | 4.93E+03 | 8.60E+12 | 2.16E+10 | 1.45E+08 | 2.76E+08 | **2.72E+03** | 3.52E+03 |
| | Rank | 1 | 4 | 8 | 7 | 5 | 6 | 2 | 3 |
| CEC-2019-F2 | Worst | 1.73E+01 | 1.73E+01 | 1.63E+04 | 1.73E+01 | 1.73E+01 | 1.35E+02 | 1.73E+01 | 1.73E+01 |
| | Average | 1.73E+01 | 1.73E+01 | 1.27E+04 | 1.73E+01 | 1.73E+01 | 8.34E+01 | 1.73E+01 | 1.73E+01 |
| | Best | 1.73E+01 | 1.73E+01 | 8.32E+03 | 1.73E+01 | 1.73E+01 | 4.35E+01 | 1.73E+01 | 1.73E+01 |
| | STD | **5.66E-09** | 1.26E-07 | 3.13E+03 | 2.41E-04 | 3.45E-05 | 3.06E+01 | 8.72E-09 | 2.13E-08 |
| | Rank | 1 | 1 | 3 | 1 | 1 | 2 | 1 | 1 |
| CEC-2019-F3 | Worst | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 |
| | Average | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 |
| | Best | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 | 1.27E+01 |
| | STD | 0.00E+00 | 5.43E-05 | 0.00E+00 | 5.64E-09 | 4.29E-06 | 5.23E-05 | 0.00E+00 | 7.14E-09 |
| | Rank | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CEC-2019-F4 | Worst | 1.25E+01 | 2.71E+01 | 2.36E+01 | 1.15E+02 | 1.65E+02 | 2.72E+01 | **1.19E+01** | 1.75E+01 |
| | Average | 9.05E+00 | 2.45E+01 | 1.73E+01 | 5.63E+01 | 7.54E+01 | 2.43E+01 | **8.42E+00** | 9.85E+00 |
| | Best | 5.02E+00 | 2.13E+01 | **4.96E+00** | 1.11E+01 | 4.88E+01 | 2.07E+01 | 4.98E+00 | 5.48E+00 |
| | STD | 3.38E+00 | 2.44E+00 | 6.53E+00 | 3.73E+01 | 5.13E+01 | **2.40E+00** | 4.29E+00 | 2.66E+00 |
| | Rank | 2 | 6 | 4 | 7 | 8 | 5 | 1 | 3 |
| CEC-2019-F5 | Worst | 1.11E+00 | 1.09E+00 | 1.66E+00 | 1.09E+00 | 2.68E+00 | 1.71E+00 | 1.13E+00 | 1.42E+00 |
| | Average | 1.06E+00 | 1.05E+00 | 1.45E+00 | 1.05E+00 | 2.41E+00 | 1.54E+00 | 1.08E+00 | 1.23E+00 |
| | Best | 1.00E+00 | 1.00E+00 | 1.25E+00 | 1.00E+00 | 2.17E+00 | 1.31E+00 | 1.00E+00 | 1.06E+00 |
| | STD | 5.79E-02 | 4.66E-02 | 1.63E-01 | **4.23E-02** | 2.57E-01 | 1.66E-01 | 6.23E-02 | 5.49E-02 |
| | Rank | 2 | 1 | 5 | 1 | 7 | 6 | 3 | 4 |
| CEC-2019-F6 | Worst | **8.43E+00** | 8.54E+00 | 1.25E+01 | 1.21E+01 | 1.21E+01 | 1.14E+01 | 8.49E+00 | 9.32E+00 |
| | Average | **5.19E+00** | 5.32E+00 | 9.85E+00 | 1.14E+01 | 1.16E+01 | 1.06E+01 | 5.76E+00 | 7.33E+00 |
| | Best | **2.94E+00** | 3.05E+00 | 8.96E+00 | 1.05E+01 | 1.14E+01 | 9.15E+00 | 3.41E+00 | 4.72E+00 |
| | STD | 2.85E-01 | 2.31E-01 | 9.48E-01 | 4.72E-01 | 2.38E-01 | 8.05E-01 | **2.15E-01** | 3.19E-01 |
| | Rank | 1 | 2 | 5 | 7 | 8 | 6 | 3 | 4 |
| CEC-2019-F7 | Worst | **1.53E+02** | 6.51E+02 | 4.13E+02 | 1.41E+03 | 6.63E+02 | 6.06E+02 | 1.69E+02 | 1.88E+02 |
| | Average | 8.88E+01 | 4.48E+02 | 2.88E+02 | 6.55E+02 | 3.55E+02 | 2.57E+02 | **7.66E+01** | 9.85E+01 |
| | Best | **4.32E+01** | 2.79E+02 | 3.43E+02 | 1.85E+02 | 1.26E+02 | 4.78E+01 | 6.92E+01 | 7.43E+01 |
| | STD | 6.71E+01 | 1.53E+02 | 1.73E+02 | 4.45E+02 | 2.31E+02 | 2.93E+02 | **6.43E+01** | 7.21E+01 |
| | Rank | 2 | 7 | 5 | 8 | 6 | 4 | 1 | 3 |
| CEC-2019-F8 | Worst | 4.90E+00 | 6.55E+00 | 5.65E+00 | 6.11E+00 | 6.85E+00 | 6.63E+00 | **4.79E+00** | 5.52E+00 |
| | Average | **4.15E+00** | 5.62E+00 | 4.96E+00 | 5.71E+00 | 5.49E+00 | 5.87E+00 | 4.39E+00 | 4.89E+00 |
| | Best | 2.59E+00 | 4.66E+00 | 4.39E+00 | 5.13E+00 | 2.92E+00 | 4.61E+00 | **2.55E+00** | 3.75E+00 |
| | STD | 8.29E-01 | 7.41E-01 | 6.37E-01 | **4.76E-01** | 1.76E+00 | 8.13E-01 | 8.35E-01 | 6.83E-01 |
| | Rank | 1 | 6 | 4 | 7 | 5 | 8 | 2 | 3 |
| CEC-2019-F9 | Worst | 2.45E+00 | 2.45E+00 | 2.44E+00 | 2.86E+00 | 7.16E+00 | **2.43E+00** | 2.45E+00 | 2.44E+00 |
| | Average | 2.43E+00 | 2.41E+00 | 2.42E+00 | 2.62E+00 | 4.95E+00 | **2.36E+00** | 2.44E+00 | 2.42E+00 |
| | Best | 2.42E+00 | 2.36E+00 | 2.38E+00 | 2.43E+00 | 3.33E+00 | **2.35E+00** | 2.43E+00 | .2.41E+00 |
| | STD | **6.36E-06** | 3.17E-02 | 3.41E-02 | 1.76E-01 | 1.52E+00 | 1.09E-02 | 4.19E-04 | 7.03E-01 |
| | Rank | 4 | 2 | 3 | 6 | 7 | 1 | 5 | 3 |
| CEC-2019-F10 | Worst | 2.03E+01 | 2.04E+01 | 2.07E+01 | **2.01E+01** | 2.07E+01 | 2.05E+01 | 2.02E+01 | 2.03E+01 |
| | Average | **1.98E+01** | 2.03E+01 | 2.04E+01 | 2.00E+01 | 2.06E+01 | 2.04E+01 | 1.99E+01 | 2.00E+01 |
| | Best | 1.96E+01 | 2.02E+01 | 2.01E+01 | 2.00E+01 | 2.05E+01 | 2.04E+01 | 1.97E+01 | 1.99E+01 |
| | STD | 3.29E-01 | 1.36E-01 | 1.85E-01 | **3.97E-02** | 4.72E-02 | 5.43E-02 | 1.17E-01 | 8.53E-02 |
| | Rank | 1 | 4 | 5 | 3 | 6 | 5 | 2 | 3 |
| Mean | rank | **1.6** | 3.4 | 4.3 | 4.8 | 5.4 | 4.4 | 2.1 | 2.8 |
| Final | Ranking | **1** | 4 | 5 | 7 | 8 | 6 | 2 | 3 |

Prema rezultatima testiranja funkcija za uporednu procenu CEC2019 datih u Tabeli 4.2, algoritam HARSA je generalno postigao najbolje rezultate. Međutim, treba napomenuti da to nije konstantno bio slučaj za sve test funkcije. To nije iznenađujuće, jer teorema NFL optimizacije sugeriše da nijedan algoritam ne može podjednako dobro da se ponaša u svim aplikacijama. Stoga je potrebno sprovesti eksperimente da bi se identifikovali algoritmi koji su najprikladniji za određeni problem.

Dalja analiza rezultata svake funkcije za uporednu analizu može pomoći u razumevanju snage i slabosti svakog algoritma. Rezultati na funkciji F1 pokazuju da su HARSA, RSA, COLSHADE, i SASS postigli superiornije rezultate, gde je predloženi HARSA postigao najbolje rezultate osim standardne devijacije, gde je SASS postigao najbolji rezultat, pokazujući da je stabilniji u ovom scenariju. U slučaju F2, ista četiri algoritma su postigla odlične performanse, sa malim razlikama u standardnoj devijaciji, gde je HARSA završio ispred algoritma SASS, a zatim su usledili COLSHADE i RSA. Ostali algoritmi su znatno zaostajali u ovom konkretnom slučaju.

U slučaju F3, svi algoritmi su postigli odlične rezultate, ali kada je u pitanju standardna devijacija, predloženi HARSA je imao najbolji rezultat, a sledili su ga SASS i PSO. SASS algoritam je imao bolje performanse od predloženog HARSA na drugom mestu u slučaju F4, gde je COLSHADE postigao treće mesto. SASS je bolji od HARSA u pogledu

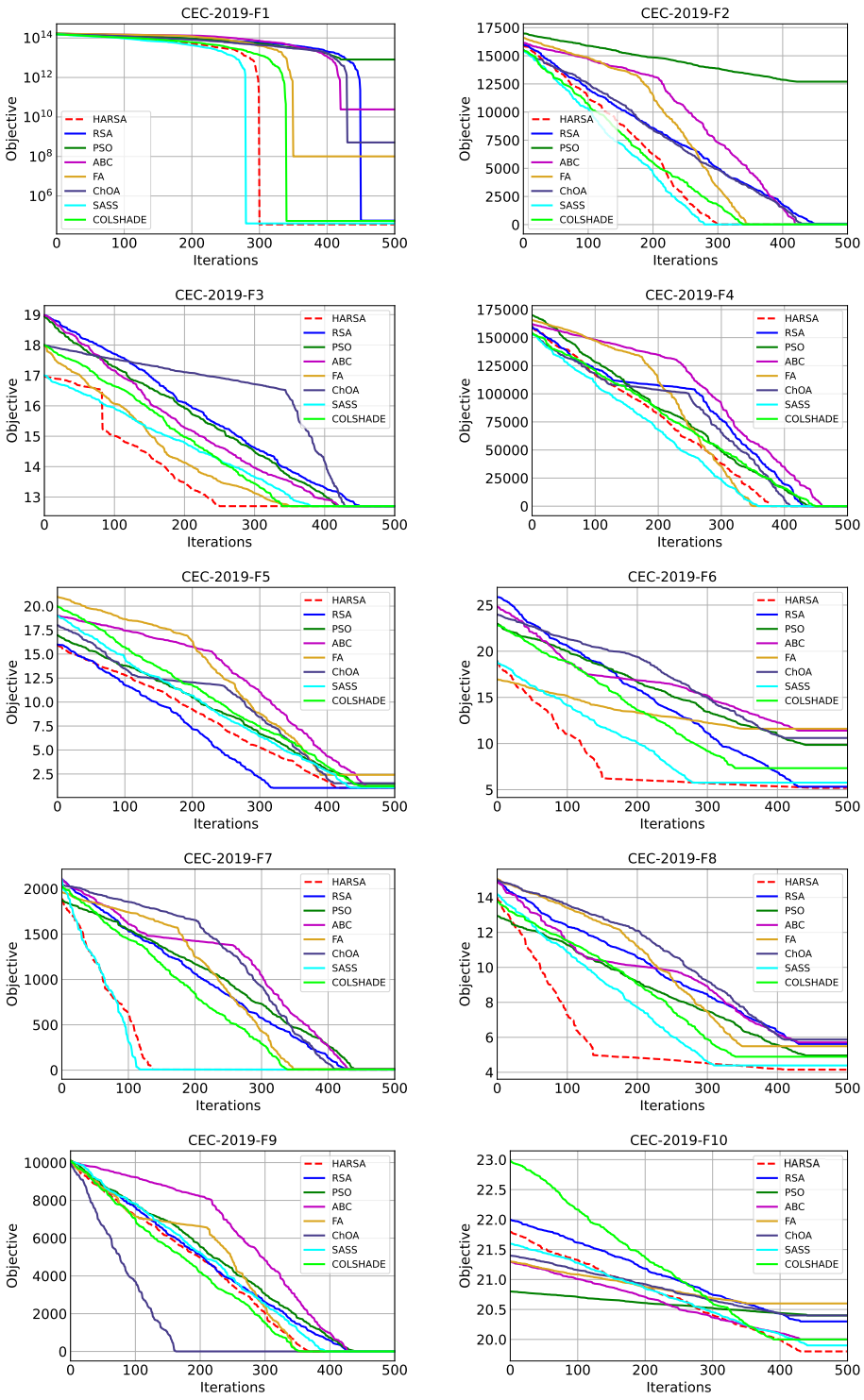
prosečnih i najgorih rezultata, dok je najbolji rezultat u jednom izvršavanju postigao PSO. Predstavljeni HARSA je bolji od SASS-a samo u pogledu standardne devijacije, što znači da je malo stabilniji. Kada se pogledaju rezultati na F5, optimalnu vrednost su postigli HARSA, RSA, i SASS, međutim, RSA ima najbolji prosek, što ukazuje da je najbolji s obzirom na stohastičku prirodu ovih algoritama. ABC algoritam je bio najstabilniji, sa najmanjom standardnom devijacijom.

Rezultati na funkciji F6 pokazuju da je predloženi HARSA najbolji u ovom konkretnom slučaju, postižući najbolje rezultate za najbolje, najgore, i prosečne metrike, dok je SASS postigao najbolju standardnu devijaciju. Osnovni RSA je takođe veoma konkurentan, jer je postigao drugi najbolji prosek u ovom scenariju, ispred algoritama SASS i COLSHADE. U slučaju funkcije F7, predloženi HARSA je postigao najbolje rezultate u najboljim i najgorim metrikama, međutim, najbolji prosek je postigao SASS, koji je takođe postigao najbolju standardnu devijaciju. Iza algoritama SASS i HARSA, treće mesto je postigao COLSHADE, dok je ChOA bio četvrti. Suprotno ponašanje se primećuje u slučaju F8, gde je predloženi HARSA postigao najbolji prosek rezultata, ali je SASS postigao najbolje vrednosti za najbolje i najgore metrike.

S druge strane, u slučaju F8, najstabilniji rezultati (najmanja standardna devijacija) su postignuti osnovnim metaheuristikama, a to su ABC i PSO. ChOA metaheuristika je postigla najbolje rezultate za najbolje, najgore, i medijalne metrike u vezi sa F9. Osnovni RSA, COLSHADE i predloženi HARSA su bili vrlo blizu u pogledu prosečnih vrednosti, dok je HARSA bio u stanju da postavi najmanju standardnu devijaciju i najstabilnije rezultate. Konačno, kada se pogledaju rezultati na F10, predloženi HARSA je pokazao nešto veću disperziju tokom pokretanja od drugih metoda, na primer, najgori rezultat je iza algoritma SASS. Međutim, HARSA je postigao najbolje rezultate za prosečne i najbolje rezultate, dok je imao veću standardnu devijaciju. SASS i COLSHADE su bili blizu na drugom i trećem mestu u pogledu prosečnog rezultata, ali treba napomenuti da su oba postigla manje standardne devijacije.

Nakon izvođenja Shapiro-Wilk testa [97], primećeno je da rezultati svih posmatranih algoritama dolaze iz normalne distribucije, što znači da je bilo sigurno koristiti srednje vrednosti kao reprezentativne. Stoga, uzete su srednje vrednosti svih metoda na svim funkcijama za uporednu analizu i algoritmi su rangirani za svaku funkciju uporedne procene, gde metode koje su imale najbolje performanse postigle niže rangove, dok su metode sa lošijim performansama postigle više rangove (ako je bilo nerešenih, algoritmi su dobili isti rang). Na kraju, prosečne vrednosti rangova preko svih 10 funkcija uporedne analize su uzete kao konačne i prikazane su u poslednja dva reda Tabele 4.2. Može se primetiti da je predloženi HARSA postigao najbolje rangove u većini funkcija za uporednu analizu, ispred algoritma SASS i COLSHADE metoda. Prosečan rang preko svih deset funkcija pokazuje da je HARSA postigao prosečan rang od 1.6, a sledili su ga SASS i COLSHADE koji su postigli prosečne rangove od 2.1 i 2.8, respektivno. Osnovna verzija RSA algoritma je završila četvrta sa prosečnim rangom od 3.4. Prema ovim rangovima, može se zaključiti da je nivo performansi HARSA bio superioran u odnosu na druge algoritme posmatrane u komparativnoj analizi, i on je pokazao poboljšanje u odnosu na osnovni RSA koji je takođe postigao respektabilan nivo performansi. Konačno, treba napomenuti da su sve metode bile nezavisno razvijene i testirane od strane autora od početka, bez korišćenja ugrađenih algoritama iz eksternih biblioteka. Dobijeni rezultati su u skladu sa prethodnim istraživanjem [27], gde je osnovni RSA algoritam slično validiran u odnosu na nekoliko najnovijih metoda.

Konačno, grafovi konvergencije na svih 10 CEC2019 funkcija za uporednu procenu za




**Slika 4.1:** Grafovi konvergencije svih posmatranih algoritama nad CEC2019 funkcijama

sve algoritme su dati na slici 4.1. Grafikoni su generisani na osnovu prosečnih rezultata, a vizualizacija je u potpunosti u skladu sa rezultatima prikazanim u Tabeli 4.2. Treba napomenuti da je y-osa logaritamska za funkciju F1, pošto su brojevi veliki i postoje velike razlike između algoritama.

U slučaju F1, SASS algoritam je pokazao najbržu konvergenciju na početku, međutim, u kasnijim iteracijama, HARSA je bio u stanju da ga blago nadmaši. COLSHADE je

takođe pokazao odličnu konvergenciju jer je završio na trećem mestu, dok je osnovni RSA pokazao relativno sporu konvergenciju, ali na kraju, bio je u stanju da postigne četvrto mesto. Dalje, gledajući prosečne grafikone konvergencije posmatranih metoda na funkciji F2, moguće je primetiti slično ponašanje. SASS je konvergirao brže u ranim rundama izvršavanja, ali na kraju, bio je sustignut od strane HARSA. Brzina konvergencije algoritma COLSHADE bila je treća najbolja, a sledio je FA. Osnovni RSA je pokazao prosečnu srednju konvergenciju u ovom konkretnom slučaju.

HARSA pokazuje najbržu prosečnu konvergenciju i u slučaju F3, a slede ga FA, COLSHADE i SASS. Na kraju, svi algoritmi su se dobro ponašali na ovoj funkciji uporedne procene. SASS je imao najbržu konvergenciju na funkciji F4, a sledili su ga HARSA i COLSHADE. Prosečna konvergencija osnovnog RSA u ovom slučaju je bila ispod proseka. S druge strane, osnovni RSA je pokazao odličnu prosečnu konvergenciju i ukupne rezultate u slučaju funkcije F5, a sledili su ga HARSA, SASS, i COLSHADE. HARSA je imao najbržu prosečnu konvergenciju na funkciji F6, ispred algoritama SASS, COLSHADE i RSA. Treba napomenuti da, iako je konvergirao sporije od algoritama SASS i COLSHADE, osnovni RSA je na kraju završio drugi na ovom testu. SASS je imao najbržu prosečnu konvergenciju na funkciji F7, a blisko ga je sledio HARSA. COLSHADE je konvergirao sporije, međutim, sva tri algoritma su završila sa odličnim i vrlo bliskim prosečnim rezultatima. Gledajući prosečnu konvergenciju na F8, može se primetiti da je HARSA pokazao najbržu konvergenciju u ovom slučaju, i takođe najbolje rezultate na kraju. SASS je imao drugu najbolju brzinu prosečne konvergencije, dok je COLSHADE završio treći.

U ovom slučaju, osnovni RSA je imao ispodprosečnu konvergenciju i ukupne rezultate. ChOA algoritam je imao najbolje performanse na funkciji F9, pokazujući i najbržu prosečnu konvergenciju i najbolje ukupne rezultate. HARSA, SASS, i COLSHADE su imali slične brzine konvergencije u ovom scenariju, dok je osnovni RSA imao ispodprosečnu brzinu konvergencije. Na kraju, RSA je uspeo da završi drugi u ukupnim rezultatima uprkos relativno sporoj konvergenciji u ovom slučaju. Konačno, u slučaju funkcije F10, ABC je pokazao brzu konvergenciju tokom ranih rundi, međutim, na kraju je bio pretekao od strane algoritama ERSA i SASS, dok je COLSHADE uspeo da sustigne ABC i postigne iste prosečne rezultate na trećem mestu.

# Poglavlje 5

# Eksperimenti

Ovaj odeljak prvo opisuje detaljno skupove podataka koji su korišćeni u eksperimentima. Nakon toga, dat je opis korišćenih metrika za evaluaciju dobijenih rezultata, kao i postavka okruženja za simulaciju.

## 5.1 Skupovi podataka

Ova podsekcija donosi dva skupa podataka koji su korišćeni kao osnova za eksperimente. Svaki skup podataka je detaljno objašnjen, zajedno sa mapama toplote korelacije karakteristika i opisom najrelevantnijih karakteristika.

### 5.1.1 NASA skup podataka

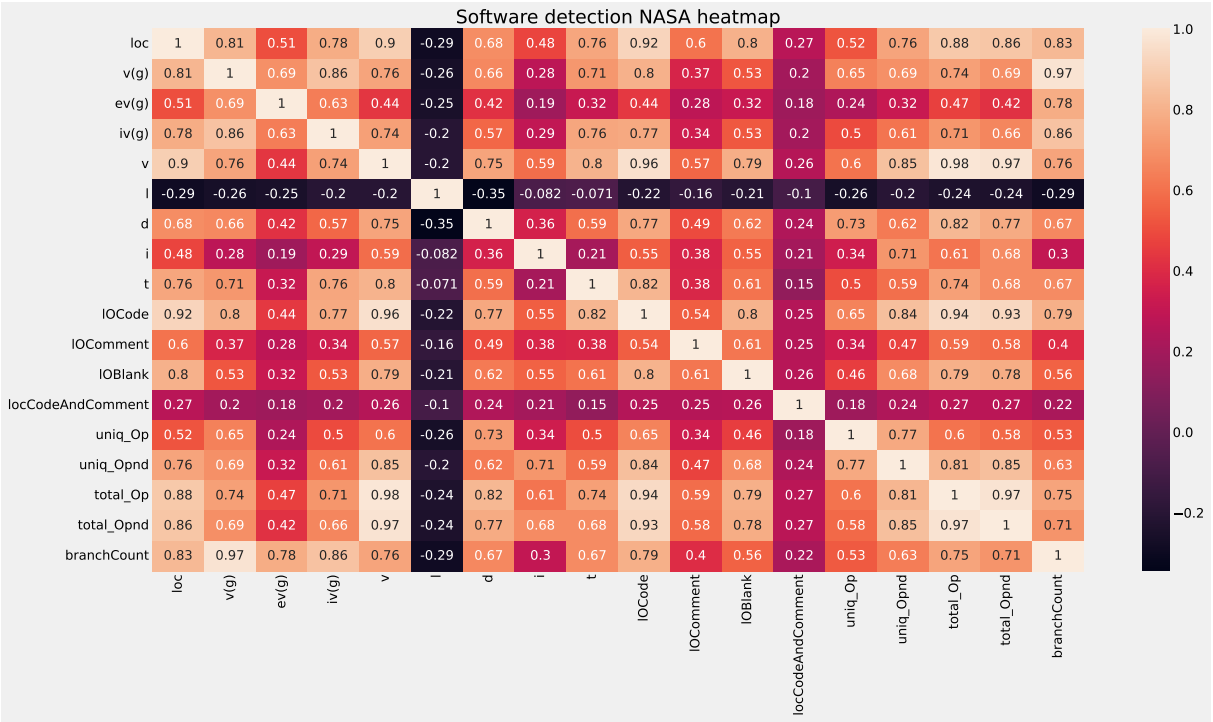
Eksperimenti izvedeni u okviru ovog istraživanja koristili su skupove podataka prikupljene od strane PROMISE Software Engineering Repository [29], uglavnom vezane za softversko inženjerstvo, softverske defekte, kvalitet usluge i procene troškova. Ovo istraživanje se fokusira na zadatak predviđanja defekata, jer je ključan za disciplinu kontrole kvaliteta i ukupni kvalitet softvera. Posmatrani skup podataka naziva se JM1, a prikupljen je od strane NASA Metrics Data Program. Podaci su prikupljeni uz pomoć McCabe i Halstead ekstraktora karakteristika iz izvornog koda brojnih modula. Skup podataka za predviđanje defekata sadrži brojne važne karakteristike i metrike o povezanom softveru, koje bi mogle biti iskorišćene za uspešno predviđanje da li određeni softverski modul ima defekte. Ovaj skup karakteristika je uspostavljen još sedamdesetih godina sa ciljem da objektivno karakteriše svojstva koda relevantna za kvalitet softvera i danas se još uvek koristi.

Karakteristike uključuju nekoliko metrika koje je predstavio Thomas McCabe [21], kao što su broj linija koda (engl. lines of code, skr. LOC), ciklomatska složenost (tj. CC ili v(g)), esencijalna složenost (ev(g)), i dizajnerska složenost (iv(g)). Pored toga, posmatrane su i nekoliko metrika koje je predložio Maurice Halstead [22, 23], uključujući četiri osnovne i osam izvedenih metrika, kao što su ukupni operatori + operandi, dužina programa, procena vremena, broj linija, broj linija koje sadrže komentare i prazne linije.

Konačno, skup podataka takođe razmatra jedinstvene operatore, jedinstvene operande, ukupne operatore, ukupne operande, i broj grana (neophodno za testiranje pokrivenosti grana/odluke, izvedeno iz kontrolnog toka grafa). Ciljna promenljiva je defekt, sa mogućim vrednostima tačno ili netačno, što ukazuje na to da li posmatrani modul ima ili nema jedan ili više prijavljenih defekata. Ovaj problem pripada klasičnim izazovima binarne

klasifikacije. Toplotna mapa koja pokazuje korelaciju između karakteristika je prikazana na slici 5.1, dok su neki od najrelevantnijih indikatora prikazani na slici 5.2. Konačno, slika 5.3 prikazuje dijagram koji prikazuje distribuciju klasa posmatranog skupa podataka.

Takođe se može primetiti da je ovaj konkretni skup podataka nebalansiran, jer 77,5% skupa podataka predstavlja unose bez defekata, a preostalih 22,5% označava instance sa defektima.

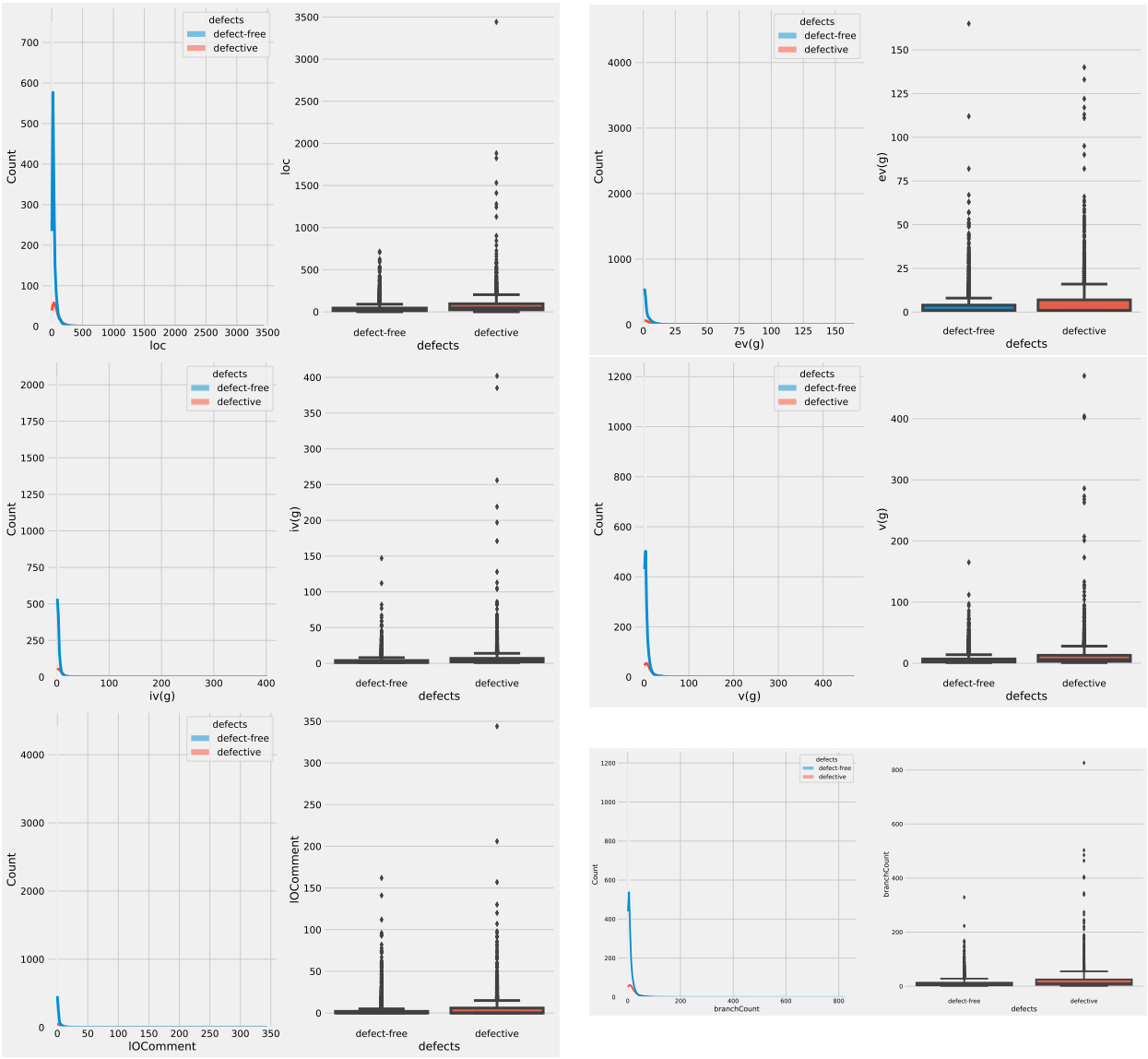



**Slika 5.1:** Toplotna karta PROMISE NASA Metrics Data Program JM1 skupa podataka

### 5.1.2 Github skup podataka

Drugi korišćeni skup podataka je kreiran od strane Jiaxi Xu i drugih autora [30], na osnovu zahteva za povlačenje koda (engl. pull requests) vezanih za ispravke grešaka na platformi Github. Osnovni skup podataka se sastoji od 6052 unosa, gde je 50% defektno, dok drugih 50% nije defektno. Svaki unos sadrži 21 karakteristiku, svaka predstavlja određene softverske metrike, kao što su broj zavisnosti klase, McCabe-ova ciklomatska složenost na nivou metode, dubina stabla nasleđivanja, broj metoda i polja, i linije koda. Skup podataka takođe uzima u obzir broj povratnih izjava, broj petlji, maksimalni nivo ugnježđivanja, broj try/catch blokova. Ovo je noviji skup podataka, jer je kreiran 2021. godine, stoga su u njega uključene brojne metrike koje opisuju objektno-orijentisani softver.

Slično prvom skupu podataka, ciljna promenljiva je defekt, sa mogućim vrednostima 1 ili 0, što ukazuje da li su defekti (jedan ili više) prijavljeni za određeni modul. Ovaj problem pripada klasičnim izazovima binarne klasifikacije. Mapa toplote koja pokazuje korelaciju između karakteristika je prikazana na slici 5.4, dok su neki od najrelevantnijih indikatora prikazani na slici 5.5. Konačno, slika 5.6 prikazuje dijagram koji prikazuje distribuciju klasa posmatranog skupa podataka. Kao što je ranije rečeno, može se primetiti da je ovaj konkretni skup podataka savršeno balansiran, jer defektni i instance bez defekata svaka predstavljaju 50% skupa podataka.

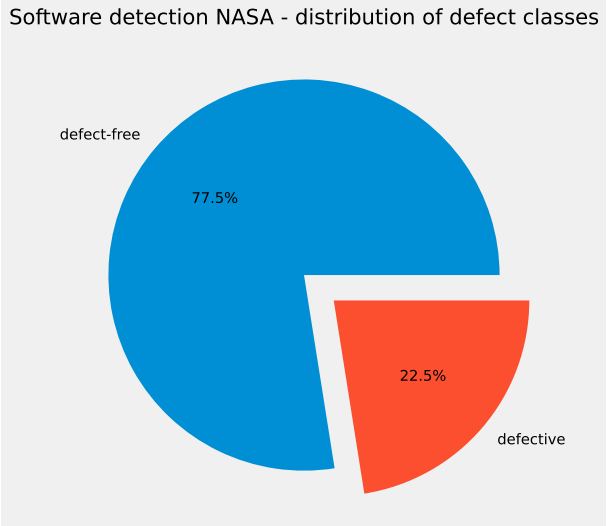



**Slika 5.2:** Vizuelizacija nekih od relevantnih indikatora, poput broja linija koda, esencijalne kompleksnosti, kompleksnosti dizajna, ciklomatske kompleksnosti, broja linija sa komentarima i broja odluka

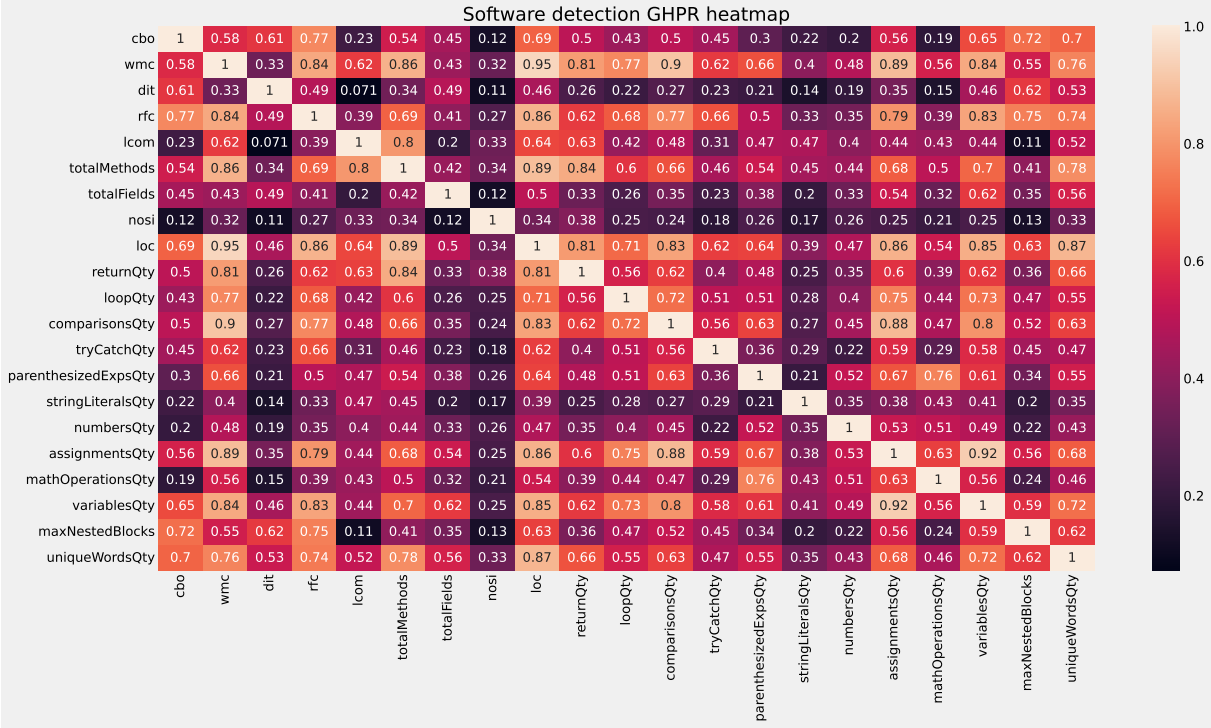
## 5.2 Metrike

Predloženi model je evaluiran koristeći tradicionalne metrike mašinskog učenja bazirane na tačno pozitivnim (TP), tačno negativnim (TN), lažno pozitivnim (FP), i lažno negativnim (FN) predikcijama. Ovaj skup metrika omogućava izračunavanje važnih indikatora mašinskog učenja kao što su tačnost (engl. accuracy), preciznost (engl. precision), osetljivost (engl. sensitivity) i F-skor (engl. F-score). U eksperimentima, u slučaju balansiranog skupa podataka (GHPR), funkcija cilja koja je minimizirana je greška klasifikacije. Za NASA skup podataka, koji je nebalansiran, primenjena su dva pristupa. Prvo, stopa greške klasifikacije je posmatrana kao funkcija cilja koja treba da bude minimizirana, međutim, kada postoji značajan nesklad između klasa, stopa greške možda neće otkriti pravu tačnost modela. Drugi pristup se fokusirao na optimizaciju Koenovog kapa koeficijenta, definisanog kao što sledi.

Funkcija cilja koja je optimizovana (maksimizirana) je bio Koenov kapa koeficijent $\kappa$



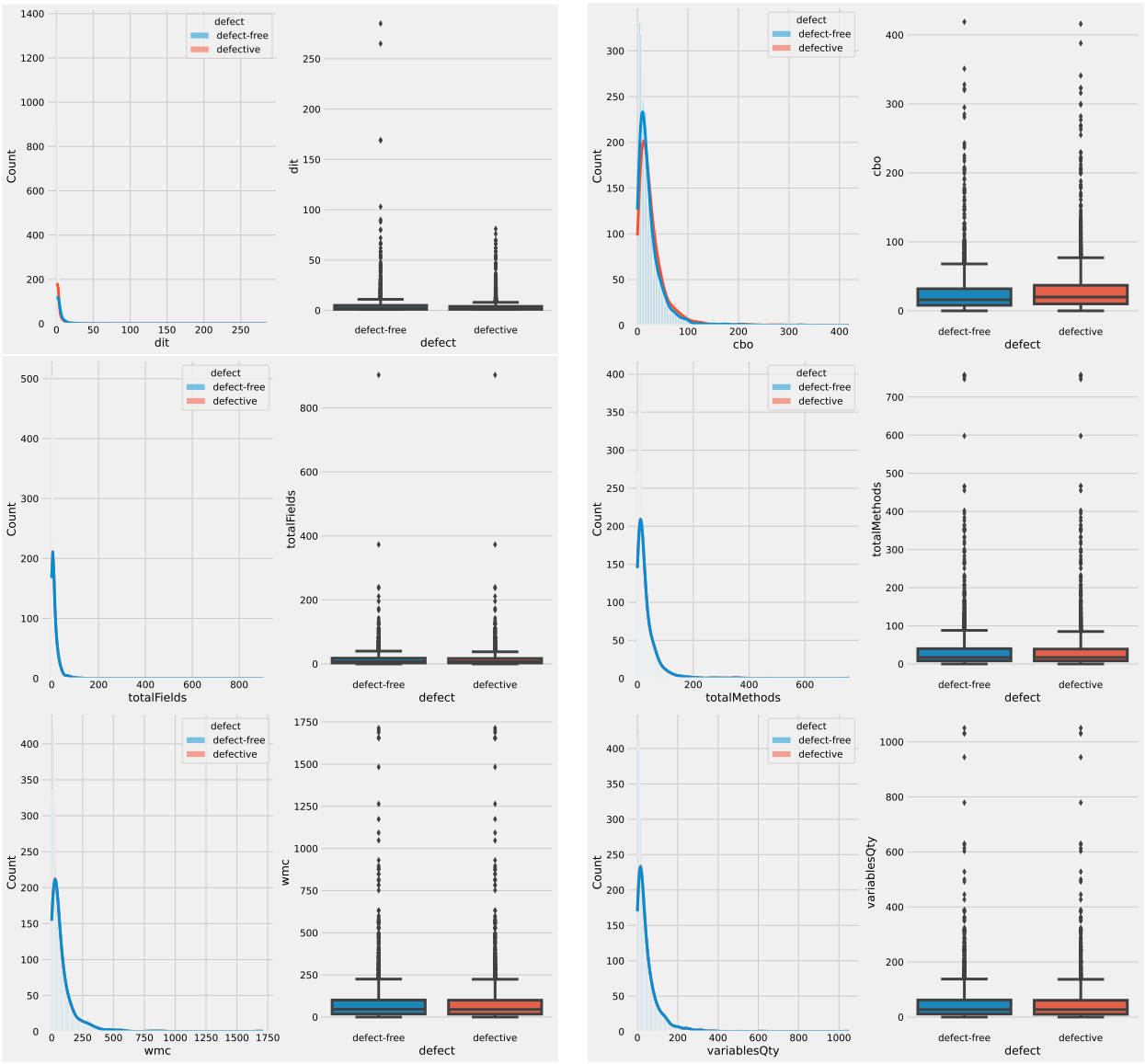

**Slika 5.3:** Distribucija klasa




**Slika 5.4:** Toplotna karta atributa GHPR skupa podataka

[98]. Ovaj koeficijent meri pouzdanost između ocenjivača [99] i takođe se može koristiti za evaluaciju performansi određenog modela klasifikacije. Koenov kapa koeficijent se izvodi iz matrice konfuzije koja se koristi u evaluaciji binarnih i višeklasnih klasifikacija u modelima mašinskog učenja. Za razliku od ukupne tačnosti modela, koja može biti obmanjujuća u prisustvu nebalansiranih skupova podataka, Koenov kapa uzima u obzir nesklad u distribuciji klasa kako bi pružio pouzdanije rezultate. Izračunava se kako je opisano u jednačini (5.1):

$$\kappa = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e} \tag{5.1}$$



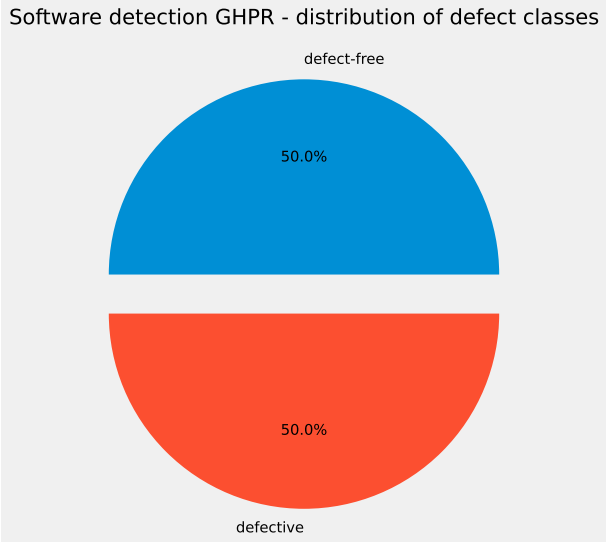

**Slika 5.5:** Vizuelizacija nekih od najrelevantnijih indikatora, uključujući dubinu stabla nasleđivanja, spregu između objekata, broj polja i metoda, težinu metoda klasa (ciklomatska kompleksnost) i broj promenljivih

gde $p_o$ prikazuje posmatrane vrednosti, dok $p_e$ obuhvata očekivane vrednosti.

## 5.3 Postavka simulacije

Predložena HARSA metoda je upotrebljena za optimizaciju XGBoost modela za dva opisana skupa podataka o defektima softvera. Kolekcija hiperparametara modela XGBoost koji su prošli kroz proces optimizacije, zajedno sa njihovim odgovarajućim ograničenjima prostora pretrage i tipovima promenljivih su navedeni u Tabeli 5.1.

Broj parametara softprob ciljne funkcije ('num class':self.no classes) je prosleđen kao parametar modela XGBoost. Ostali brojni hiperparametri modela XGBoost su fiksirani na podrazumevane vrednosti modela XGBoost tokom eksperimenata. Opisani okvir je kreiran koristeći Python programski jezik, i oslanja se na često korišćene Python biblioteke za mašinsko učenje, kao što su numpy, scipy, i pandas. XGBoost model je implementiran




**Slika 5.6:** Distribucija klasa

**Tabela 5.1:** Skup XGBoost hiperparametara optimizovanih u ovoj disertaciji

| XGBoost hiperparametar | donja i gornja granica | tip promenljive |
|---|---|---|
| stopa učenja ($\eta$) | $[0.1, 0.9]$ | kontinualna |
| min_child_weight | $[1, 10]$ | kontinualna |
| subsample | $[0.01, 1]$ | kontinualna |
| collsample_bytree | $[0.01, 1]$ | kontinualna |
| max_depth | $[3, 10]$ | celobrojna |
| *gamma* | $[0, 0.8]$ | kontinualna |

koristeći Python biblioteku scikit-learn. Šema kodiranja koju koristi ovaj okvir je da se svaka jednika posmatra kao vektor dužine $l$, gde $l$ prikazuje broj optimizovanih hiperparametara. Kao što je ranije rečeno, šest XGBoost hiperparametara je prošlo kroz proces optimizacije, stoga je u ovom radu, $l$ bio fiksiran na 6.

Nivo performansi XGBoost-a optimizovanog HARSA algoritmom je evaluiran kroz komparativnu analizu naspram rezultata postignutih od strane osam drugih moćnih metaheuristika. Skup metaheuristika koje su poređene je obuhvatao sledeće algoritme optimizacije: originalni RSA, optimizaciju rojem čestica (PSO), algoritam veštačke kolonije pčela (ABC), algoritam svitaca (FA), algoritam slepih miševa (BA) [60], algoritam kitova (engl. whale optimization algorithm, skr. WOA) [100], optimizator zavisan od fitnesa (engl. fitness dependent optimizer, skr. FDO) [101], i algoritam šimpanza (ChOA). Svaki od konkurenata je implementiran nezavisno za ovo istraživanje, koristeći preporučena podešavanja kontrolnih parametara kako je predloženo od strane njihovih respektivnih kreatora. Takođe, svaka metaheuristika je dobila identičan zadatak, da optimizuje identičan skup XGBoost hiperparametara. Zbog jednostavnosti, svakoj metodi je dat akronim za lakše razumevanje tabela sa rezultatima (XG-HARSA označava XGBoost model podešen HARSA algoritmom, i tako dalje). Svaka metaheuristička metoda je prošla kroz nezavisnih 30 izvršavanja, sa populacijom od 20 jedinki ($N = 20$) i maksimumom od 20 iteracija po pokretanju ($max_iter = 20$). Kao što je opisano gore, stopa greške i Koenov kapa koeficijent su korišćeni kao funkcije cilja u eksperimentima.

# Poglavlje 6

# Rezultati simulacija

Ovaj odeljak donosi eksperimentalne rezultate oba posmatrana skupa podataka. Simulacije su izvršene sa XGBoost za svaki skup podataka, prateći stopu grešaka i Koenov kapa koeficijent u svim slučajevima, ali sa različitim funkcijama cilja. Za NASA skup podataka, koji je neuravnotežen, prvo je stopa grešaka posmatrana kao funkcija cilja koja treba da se optimizuje, a zatim Koenov kapa kao funkcija cilja u drugom delu. Zbog neuravnoteženih klasa, stopa grešaka može biti obmanjujuća, a Koenov kapa koeficijent može pružiti značajnije informacije. Uvek postoji kompromis između Koenovog kapa i greške, što znači da nije moguće istovremeno optimizovati oba. S druge strane, za GHPR skup podataka, koji je savršeno uravnotežen, samo je stopa grešaka postavljena kao funkcija cilja, jer nema potrebe za optimizacijom Koenovog kapa koeficijenta za uravnotežene skupove podataka.

## 6.1 Eksperimenti sa NASA skupom podataka

NASA skup podataka, kako je prethodno navedeno, je neuravnotežen, stoga minimizacija stope grešaka može imati ograničene rezultate. Stoga su izvršena dva eksperimenta, sa XGBoost modelima podešenim metaheuristikama. Prvo, stopa grešaka je definisana kao funkcija cilja, koja zahteva minimizaciju. Kasnije, Koenov kapa indikator je postavljen kao funkcija cilja, koju je potrebno maksimizovati, jer takođe uzima u obzir distribuciju klasa i može pružiti pouzdanije rezultate. Kao što je već pomenuto, nije moguće istovremeno podešavati stopu grešaka i Koenov kapa, i uvek postoji kompromis između ova dva indikatora.

### 6.1.1 XGBoost - stopa grešaka kao funkcija cilja

Ovaj odeljak sumira rezultate simulacija nad NASA skupom podataka sa XGBoost podešenim algoritmima metaheuristike, gde je stopa grešaka u klasifikaciji postavljena kao funkcija cilja koja se minimizuje. U svakoj tabeli koja sadrži rezultate simulacija, najbolji rezultat u svakoj od posmatranih kategorija je istaknut podebljanim tekstom.

Tabela 6.1 donosi rezultate svih algoritama metaheuristike u pogledu najboljih, najgorih, srednjih i medijalnih vrednosti preko 30 odvojenih pokretanja. Pored toga, pruženi su i standardna devijacija i varijansa. Ova tabela sadrži rezultate za objektivnu funkciju (stopa grešaka) i indikator Koenovog kapa. Predloženi pristup XG-HARSA postigao je vrhunske rezultate, sa najboljim rezultatima za najbolje, najgore, srednje i medijalne

vrednosti objektivne funkcije. XG-ABC i XG-ChOA završili su na drugom mestu za najbolju metriku, dok je XG-ChOA završio na drugom mestu za srednju metriku. XG-RSA je postigao najstabilnije rezultate, sa najboljim rezultatima za standardnu devijaciju i varijansu. Gledajući vrednosti indikatora (Koenov kapa), ponovo, predloženi XG-HASCA je postigao najbolje vrednosti za najbolje, srednje i medijalne metrike, dok je XG-FA postigao najbolji rezultat za najgoru metriku. Tabela 6.2 predstavlja sveobuhvatne metrike za najbolje izvođenje svakog ispitivanog algoritma.

Može se primetiti da je predloženi XG-HARSA postigao superiorni nivo tačnosti klasifikacije od 79,39%, ispred XG-ABC i XG-ChOA koji su postigli tačnost od 79,16%. XG-HARSA je takođe postigao najbolju ukupnu preciznost, ukupni recall i ukupni F1 skor (prosečne vrednosti za obe klase 0 i 1). Takođe se može primetiti da, pošto je skup podataka neuravnotežen, vrednosti po klasi pokazuju znatno niže vrednosti za preciznost, recall i F1 skor za klasu 1 koja je manjinska klasa.

Na kraju, najbolji skup hiperparametara XGBoost (generisan najboljim izvršavanjem svakog pojedinačnog posmatranog algoritma) je predstavljen u Tabeli 6.3. Najbolji model proizveden predloženim HARSA algoritmom imao je stopu učenja od 0,348330, maksimalnu težinu deteta od 2,683932, vrednost subsample od 0,869931, vrednost collsample bytree od 0,845813, maksimalnu dubinu jednaku 9, i vrednost gama jednaku 0,750711.

**Tabela 6.1:** Ukupni rezultati koji prikazuju vrednosti funkcije cilja (stopa grešaka) XGBoost modela na NASA skupu podataka

| | OBJECTIVE ERROR | | | | | COHEN KAPPA | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | Best | Worst | Mean | Median | Std | Var | Best | Worst | Mean | Median | Std | Var |
| XG-HARSA | **0.206135** | **0.210625** | **0.208804** | **0.209128** | 1.03E-03 | 1.07E-06 | **0.204163** | 0.177680 | **0.195813** | **0.199550** | 1.31E-02 | 1.70E-04 |
| XG-RSA | 0.208754 | 0.212495 | 0.210824 | 0.210999 | **8.73E-04** | **7.63E-07** | 0.184327 | 0.149176 | 0.175116 | 0.172890 | 1.20E-02 | 1.44E-04 |
| XG-PSO | 0.208754 | 0.213618 | 0.211061 | 0.211186 | 1.32E-03 | 1.74E-06 | 0.183013 | 0.155130 | 0.174294 | 0.173829 | 1.28E-02 | 1.64E-04 |
| XG-ABC | 0.208380 | 0.212121 | 0.210837 | 0.210999 | 9.20E-04 | 8.46E-07 | 0.177184 | 0.184940 | 0.180138 | 0.181085 | 1.34E-02 | 1.80E-04 |
| XG-FA | 0.209128 | 0.213992 | 0.212146 | 0.212308 | 1.10E-03 | 1.20E-06 | 0.201494 | **0.204803** | 0.175050 | 0.172897 | 1.65E-02 | 2.72E-04 |
| XG-BA | 0.208754 | 0.214366 | 0.212121 | 0.212495 | 1.34E-03 | 1.80E-06 | 0.177713 | 0.139559 | 0.169402 | 0.170106 | 1.48E-02 | 2.18E-04 |
| XG-WOA | 0.209502 | 0.214366 | 0.211647 | 0.211747 | 1.02E-03 | 1.04E-06 | 0.180085 | 0.146611 | 0.175376 | 0.173167 | 1.47E-02 | 2.17E-04 |
| XG-FDO | 0.209128 | 0.213992 | 0.212071 | 0.212495 | 1.34E-03 | 1.81E-06 | 0.175570 | 0.152960 | 0.171202 | 0.173032 | 1.58E-02 | 2.49E-04 |
| XG-ChOA | 0.208380 | 0.212869 | 0.210687 | 0.210625 | 1.13E-03 | 1.27E-06 | 0.189051 | 0.166236 | 0.176717 | 0.177557 | **9.50E-03** | **9.02E-05** |

**Tabela 6.2:** Detaljni rezultati najboljeg izvršavanja svakog algoritma u odnosu na fukciju cilja (stopa greške) XGBoost modela na NASA skupu podataka

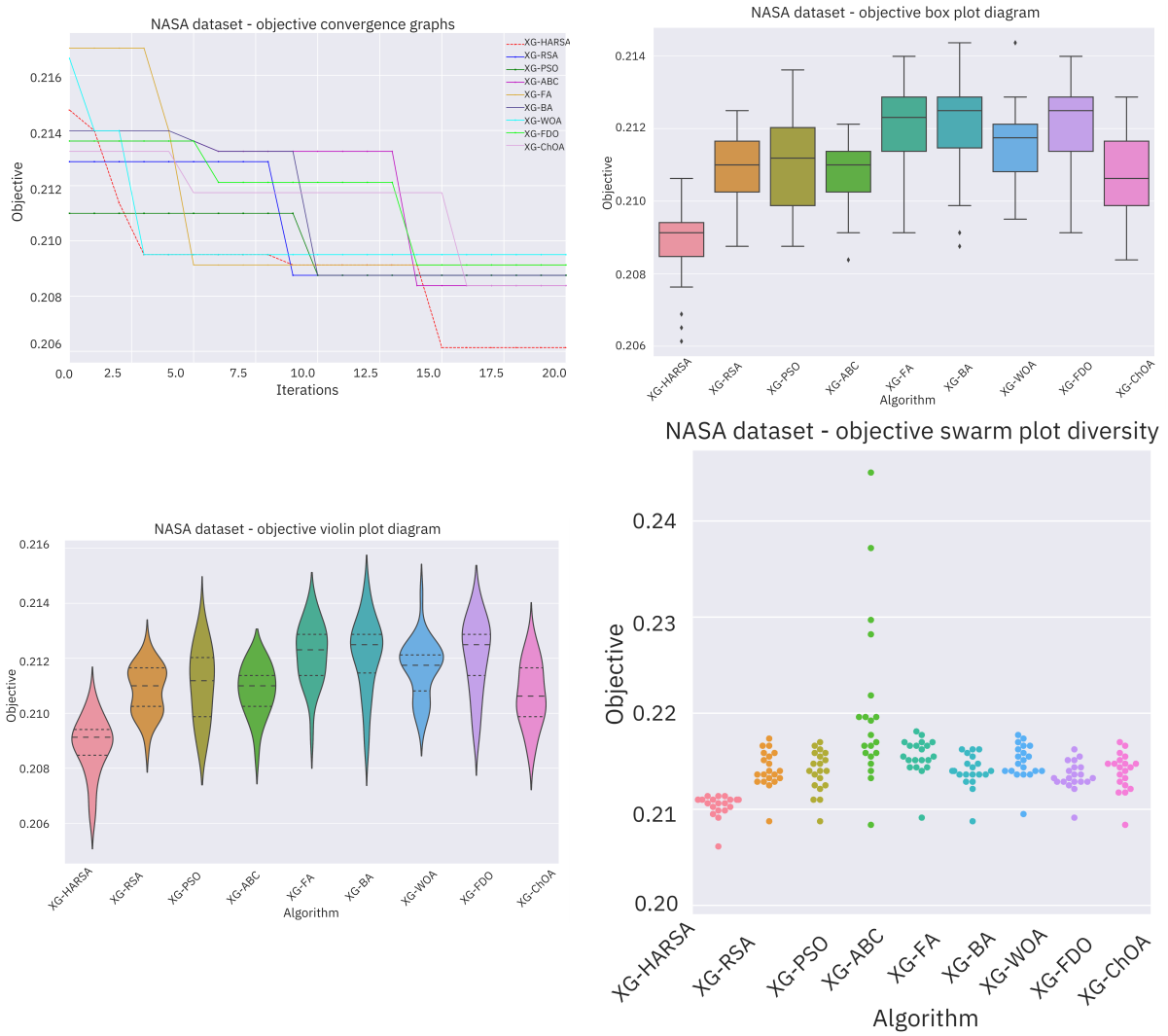
| | XG-HARSA | XG-RSA | XG-PSO | XG-ABC | XG-FA | XG-BA | XG-WOA | XG-FDO | XG-ChOA |
|---|---|---|---|---|---|---|---|---|---|
| Accuracy (%) | **79.3865** | 79.1246 | 79.1246 | 79.1620 | 79.0872 | 79.1246 | 79.0498 | 79.0872 | 79.1620 |
| Precision 0 | 0.803351 | 0.800159 | 0.799921 | 0.798817 | **0.803450** | 0.798973 | 0.799525 | 0.798658 | 0.800954 |
| Precision 1 | 0.650602 | 0.642384 | 0.644295 | **0.659420** | 0.616667 | 0.652482 | 0.637584 | 0.650000 | 0.641026 |
| W.Avg. Precision | **0.769007** | 0.764684 | 0.764930 | 0.767475 | 0.761453 | 0.766036 | 0.763114 | 0.765233 | 0.764995 |
| Recall 0 | 0.972008 | 0.973938 | 0.974421 | **0.977317** | 0.966699 | 0.976351 | 0.973938 | 0.976351 | 0.972973 |
| Recall 1 | 0.179700 | 0.161398 | 0.159734 | 0.151414 | **0.184692** | 0.153078 | 0.158070 | 0.151414 | 0.166389 |
| W.Avg. Recall | **0.793865** | 0.791246 | 0.791246 | 0.791620 | 0.790872 | 0.791246 | 0.790498 | 0.790872 | 0.791620 |
| F1 score 0 | **0.879668** | 0.878537 | 0.878590 | 0.879097 | 0.877547 | 0.878801 | 0.878155 | 0.878610 | 0.878623 |
| F1 score 1 | 0.281617 | 0.257979 | 0.256000 | 0.246279 | **0.284251** | 0.247978 | 0.253333 | 0.245614 | 0.264201 |
| W.Avg. F1 score | **0.745202** | 0.739010 | 0.738606 | 0.736814 | 0.744149 | 0.736966 | 0.737669 | 0.736287 | 0.740476 |

**Tabela 6.3:** Najbolje vrednosti skupa XGBoost parametara određene od strane svakog posmatranog algoritma

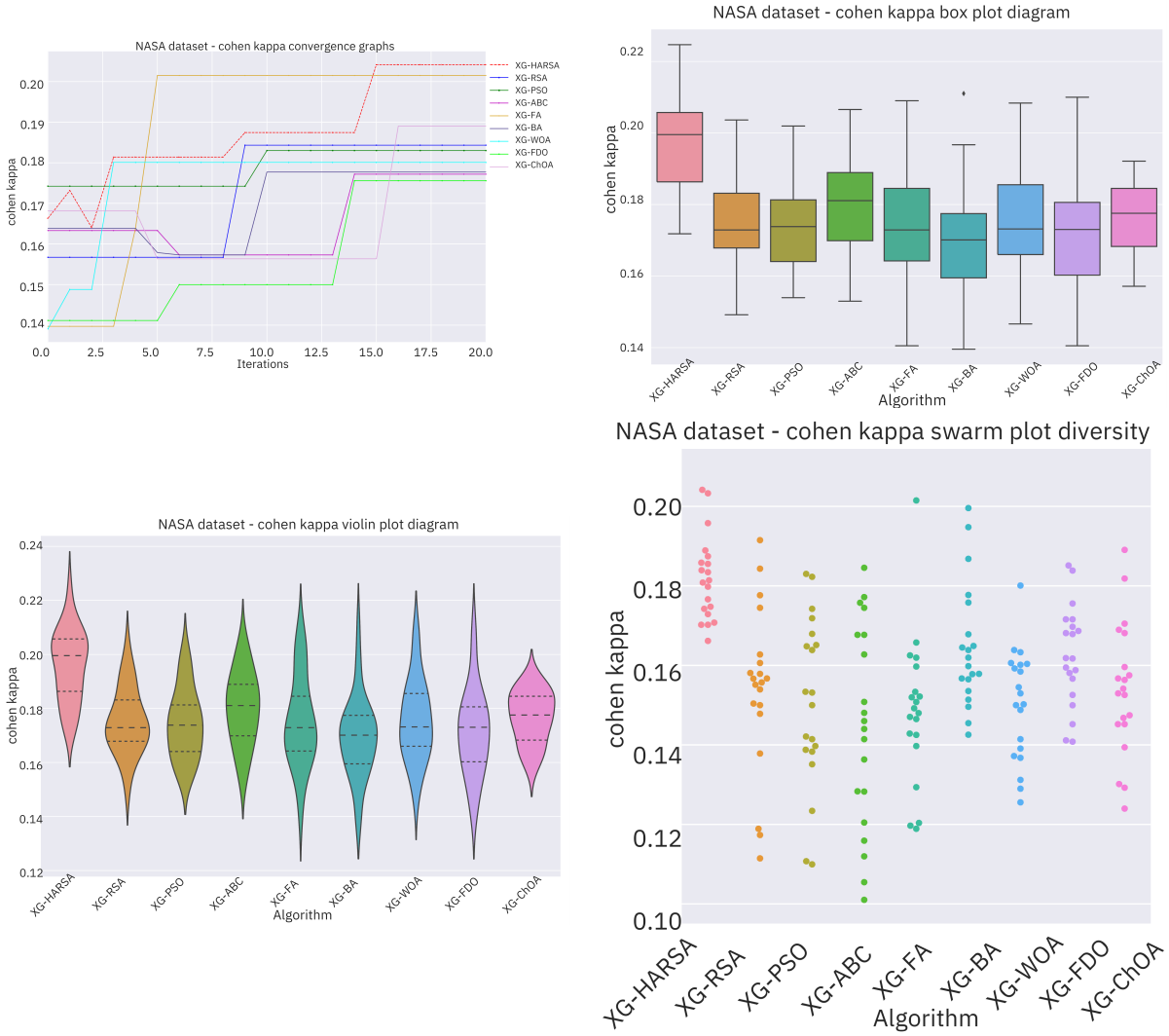
| Method | **l.r. ($\mu$)** | **max_child_weight** | **subsample** | **collsample_bytree** | **max_depth** | **gamma** |
|---|---|---|---|---|---|---|
| XG-HARSA | 0.348330 | 2.683932 | 0.869931 | 0.845813 | 9.000000 | 0.750711 |
| XG-RSA | 0.163150 | 6.950769 | 1.000000 | 0.733043 | 10.000000 | 0.186304 |
| XG-PSO | 0.223638 | 1.729505 | 0.996653 | 0.430342 | 10.000000 | 0.000000 |
| XG-ABC | 0.100000 | 1.000000 | 0.573462 | 0.849393 | 10.000000 | 0.621795 |
| XG-FA | 0.897411 | 2.877706 | 0.846435 | 0.977122 | 4.000000 | 0.565312 |
| XG-BA | 0.100000 | 5.986113 | 1.000000 | 0.629801 | 10.000000 | 0.800000 |
| XG-WOA | 0.900000 | 10.000000 | 0.562943 | 0.703360 | 3.000000 | 0.800000 |
| XG-FDO | 0.100000 | 1.382818 | 1.000000 | 0.485929 | 10.000000 | 0.439494 |
| XG-ChOA | 0.168585 | 1.000000 | 1.000000 | 0.473235 | 10.000000 | 0.800000 |

Slike 6.1 i 6.2 prikazuju rezultate simulacije u sledećim grafikonima: grafikoni konvergencije za funkciju cilja (stopa greške) i Koenov kapa indikator postignut od strane najboljeg izvršavanja svakog algoritma, boks dijagrami (engl. box-plot) i violinski dijagrami koji prikazuju distribuciju funkcije cilja na 30 nezavisnih izvršavanja, i dijagram roja koji ilustruje raznolikost populacije tokom finalne runde najboljeg izvšravanja svakog algoritma. Posmatrajući dijagrame roja za stopu greške, očigledno je da ABC pokazuje najveću raznolikost u finalnoj iteraciji, za razliku od predloženog HARSA pristupa, gde su svi pojedinci koncentrisani blizu optimalne podregije prostora pretrage. Ovo ponašanje ABC-a je očekivano, jer ABC koristi *limit* parametar, obezbeđujući visoku raznolikost populacije u celini. S druge strane, HARSA algoritam pokazuje snažnu intenzifikaciju prema trenutno najboljem pojedincu.

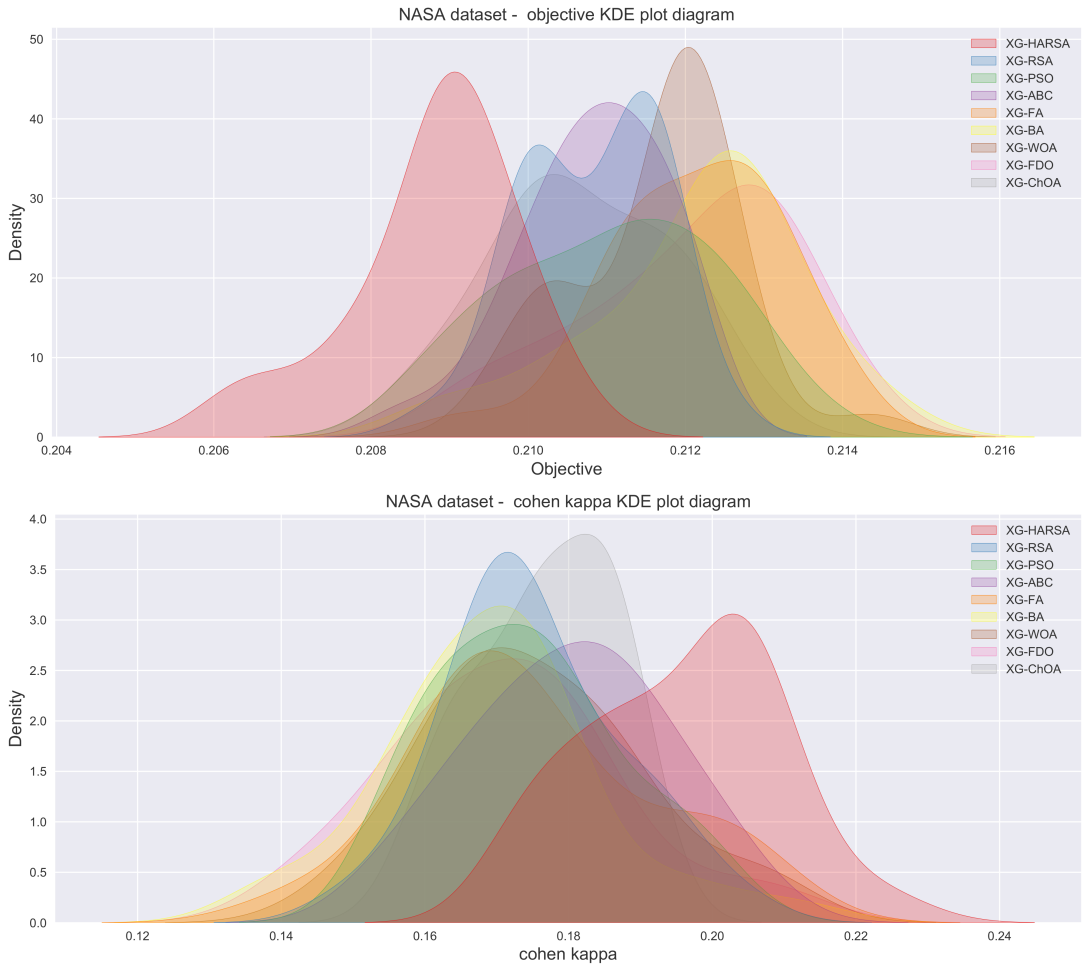
Slika 6.3 donosi Kernel Distribution Estimation (KDE) dijagrame za funkciju cilja (stopa greške) i Koenov kapa indikator, prikazujući funkciju gustine verovatnoće. Konačno, Slika 6.4 prikazuje matrice konfuzije, PR (preciznost-poziv) i ROC (karakteristike prijemnika) krive predloženog HARSA i osnovnog RSA jedan pored drugog, za NASA skup podataka sa stopom greške klasifikacije kao objektivnom funkcijom. PR kriva pokazuje preciznost i pozivne vrednosti na grafikonu, pružajući vizuelni prikaz kompromisa između ove dve metrike. ROC kriva, s druge strane, pokazuje performanse modela za sve pragove klasifikacije.



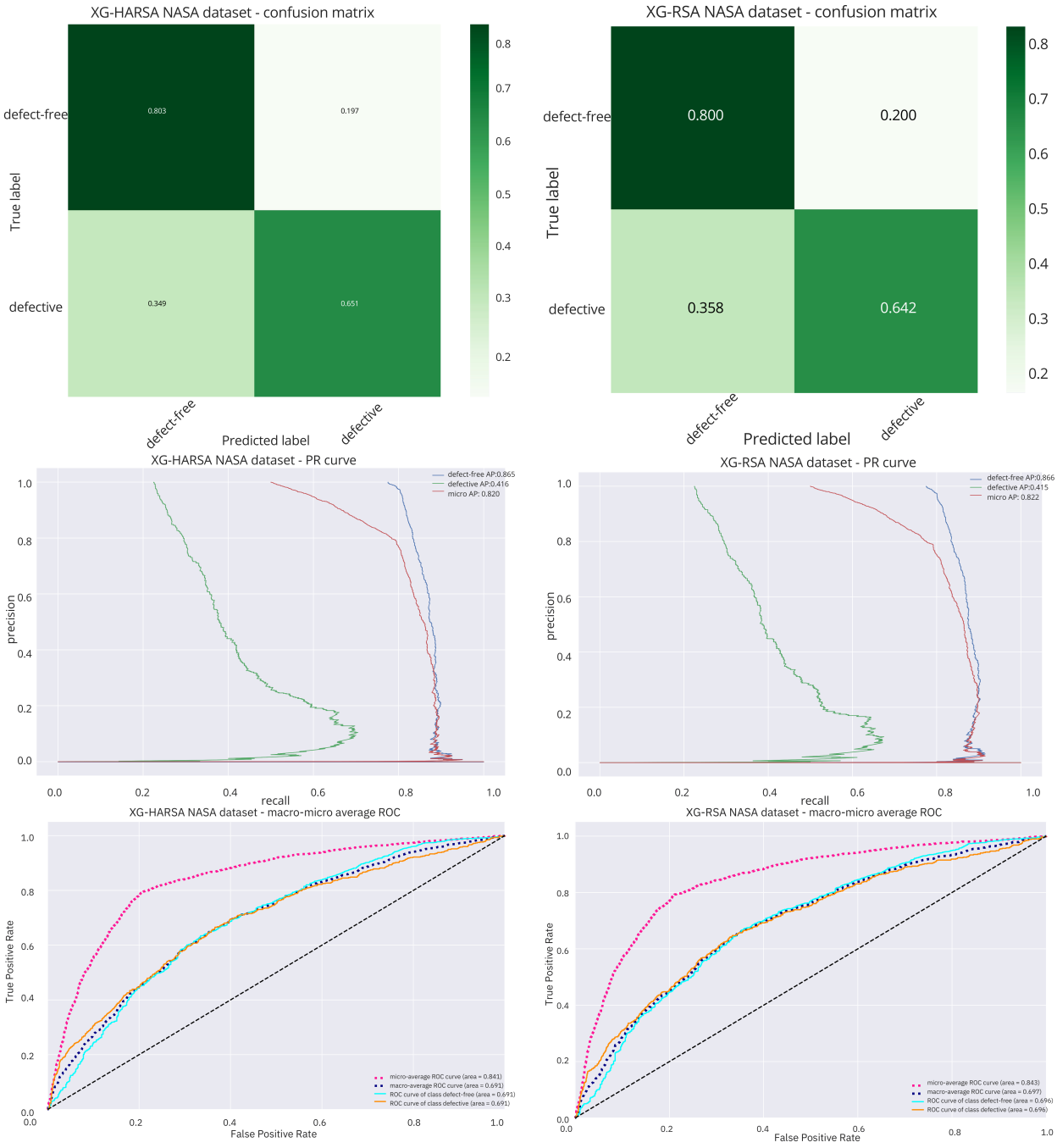

**Slika 6.1:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergeciju, boks dijagrame, violinske dijagrame, i dijagrame roja za funckiju cilja (stopu greške) na NASA skupu podataka.




**Slika 6.2:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergenciju, boks dijagrame, violinske dijagrame, i dijagrame roja za indikator (Koenov kapa koeficijent) na NASA skupu podataka.




**Slika 6.3:** Vizuelizacije KDE dijagrama za funkciju cilja (stopa greške - gore) i Koenov kapa indikator (dole) za NASA skup podataka.




**Slika 6.4:** Vizuelizacije matrica konfuzije, PR i ROC krivih za predloženi XG-HARSA (levo) i XG-RSA (desno) algoritme na NASA skupu podataka sa stopom greške kao funkcijom cilja.

### 6.1.2 XGBoost - Koenov kapa kao funkcija cilja

Ovaj deo sumira rezultate simulacije preko NASA skupa podataka sa XGBoost podešenim metaheurističkim algoritmima, gde je Koenov kapa vrednost postavljena kao funkcija cilja koja treba da bude maksimizirana. U svakoj tabeli koja sadrži rezultate simulacije, najbolji rezultat u svakoj od posmatranih kategorija je istaknut podebljanim tekstom.

Tabela 6.4 donosi rezultate svih metaheurističkih algoritama u vezi sa najboljim, najgorim, srednjim i medijalnim vrednostima preko 30 odvojenih pokretanja. Dodatno, pružene su standardna devijacija i varijansa. Ova tabela sadrži rezultate za funkciju cilja (Koenov kapa) i stopu greške klasifikacije. Predloženi XG-HARSA pristup još jednom je postigao vrhunske rezultate, sa najboljim rezultatima za najbolje, najgore, srednje i medijalne vrednosti objektivne funkcije (Koenov kapa). XG-FDE i XG-RSA završili su drugi i treći za najbolju metriku, dok je XG-WOA završio drugi za srednju metriku. Predloženi XG-HARSA pristup takođe je uspostavio najstabilnije rezultate u ovom eksperimentu, sa najboljim rezultatima za standardnu devijaciju i varijansu. Gledajući vrednosti stope greške klasifikacije kada se optimizuje Koenov kapa, predloženi XG-HASCA je postavio najbolju vrednost za srednju metriku, dok je XG-WOA postigao najbolji rezultat za najbolju metriku, XG-FA je postigao najbolju najgoru metriku, i na kraju, XG-ABC je postigao najbolju medijalnu vrednost. Tabela 6.5 predstavlja sveobuhvatne metrike za najbolje izvršavanje svakog algoritma evaluiranog u ovoj studiji. Pošto je funkcija cilja, u ovom slučaju, bila Koenov kapa, XG-WOA je postigao najbolju tačnost klasifikacije od 78,53%. XG-ChOA je završio drugi, sa tačnošću od 78,34%, a predloženi XG-HARSA je završio treći sa poštovanom tačnošću od 78,26%. Iako je malo iza sa ukupnom tačnošću, XG-HARSA je postigao najbolji prosečni F1 rezultat, metriku koja uzima u obzir i preciznost i poziv i pokazuje sposobnosti modela preko celog skupa podataka (svih klasa). Pošto je Koenov kapa bila podešena u ovom eksperimentu, ovo je očekivani rezultat, jer je NASA skup podataka nebalansiran. Prosečni F1 rezultat u ovom eksperimentu je veći u poređenju sa prethodnim eksperimentom gde je minimizirana stopa greške klasifikacije.

Konačno, najbolje proizvedeni skup hiperparametara XGBoost-a (generisan najboljim izvršavanjem svakog algoritma) prikazan je u Tabeli 6.6. Najbolji model proizveden predloženim HARSA algoritmom imao je stopu učenja od 0.732441, maksimalnu vrednost težine deteta od 1, vrednost poduzorka od 0.664398, vrednost collsample bytree od 0.788768, maksimalnu dubinu jednaku 9, i vrednost gama jednaku 0.413284.

**Tabela 6.4:** Ukupni rezultati koji prikazuju vrednosti funkcije cilja (Koenov kapa koeficijent) XGBoost modela na NASA skupu podataka

| | ERROR | | | | | OBJECTIVE COHEN KAPPA | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | Best | Worst | Mean | Median | Std | Var | Best | Worst | Mean | Median | Std | Var |
| XG-HARSA | 0.217359 | 0.220726 | **0.220564** | 0.220726 | **4.26E-03** | **1.82E-05** | **0.237225** | **0.222601** | **0.229187** | **0.228095** | **4.36E-03** | **1.90E-05** |
| XG-RSA | 0.222596 | 0.218855 | 0.227385 | 0.225589 | 1.16E-02 | 1.34E-04 | 0.231067 | 0.205571 | 0.215355 | 0.213456 | 7.12E-03 | 5.07E-05 |
| XG-PSO | 0.227086 | 0.219978 | 0.224068 | 0.222409 | 6.68E-03 | 4.46E-05 | 0.230875 | 0.202094 | 0.215660 | 0.215821 | 7.39E-03 | 5.46E-05 |
| XG-ABC | 0.227834 | 0.217733 | 0.222272 | **0.220539** | 8.43E-03 | 7.11E-05 | 0.219635 | 0.199475 | 0.208328 | 0.208748 | 5.14E-03 | 2.64E-05 |
| XG-FA | 0.217359 | **0.216236** | 0.225776 | 0.225215 | 7.10E-03 | 5.04E-05 | 0.224905 | 0.178872 | 0.212883 | 0.212892 | 8.54E-03 | 7.30E-05 |
| XG-BA | 0.220352 | 0.259633 | 0.225265 | 0.224467 | 9.66E-03 | 9.33E-05 | 0.227838 | 0.195270 | 0.213333 | 0.212985 | 7.25E-03 | 5.25E-05 |
| XG-WOA | **0.214740** | 0.243172 | 0.225477 | 0.224280 | 9.99E-03 | 9.98E-05 | 0.230288 | 0.201448 | 0.216246 | 0.218532 | 7.20E-03 | 5.18E-05 |
| XG-FDO | 0.218855 | 0.221474 | 0.225016 | 0.223532 | 7.88E-03 | 6.22E-05 | 0.235289 | 0.197864 | 0.212069 | 0.211257 | 7.33E-03 | 5.37E-05 |
| XG-ChOA | 0.216611 | 0.239431 | 0.223295 | 0.223158 | 7.07E-03 | 5.00E-05 | 0.228706 | 0.205135 | 0.214546 | 0.212672 | 6.15E-03 | 3.78E-05 |

**Tabela 6.5:** Detaljni rezultati najboljeg izvršavanja svakog algoritma u odnosu na fukciju cilja (Koenov kapa koeficijent) XGBoost modela na NASA skupu podataka

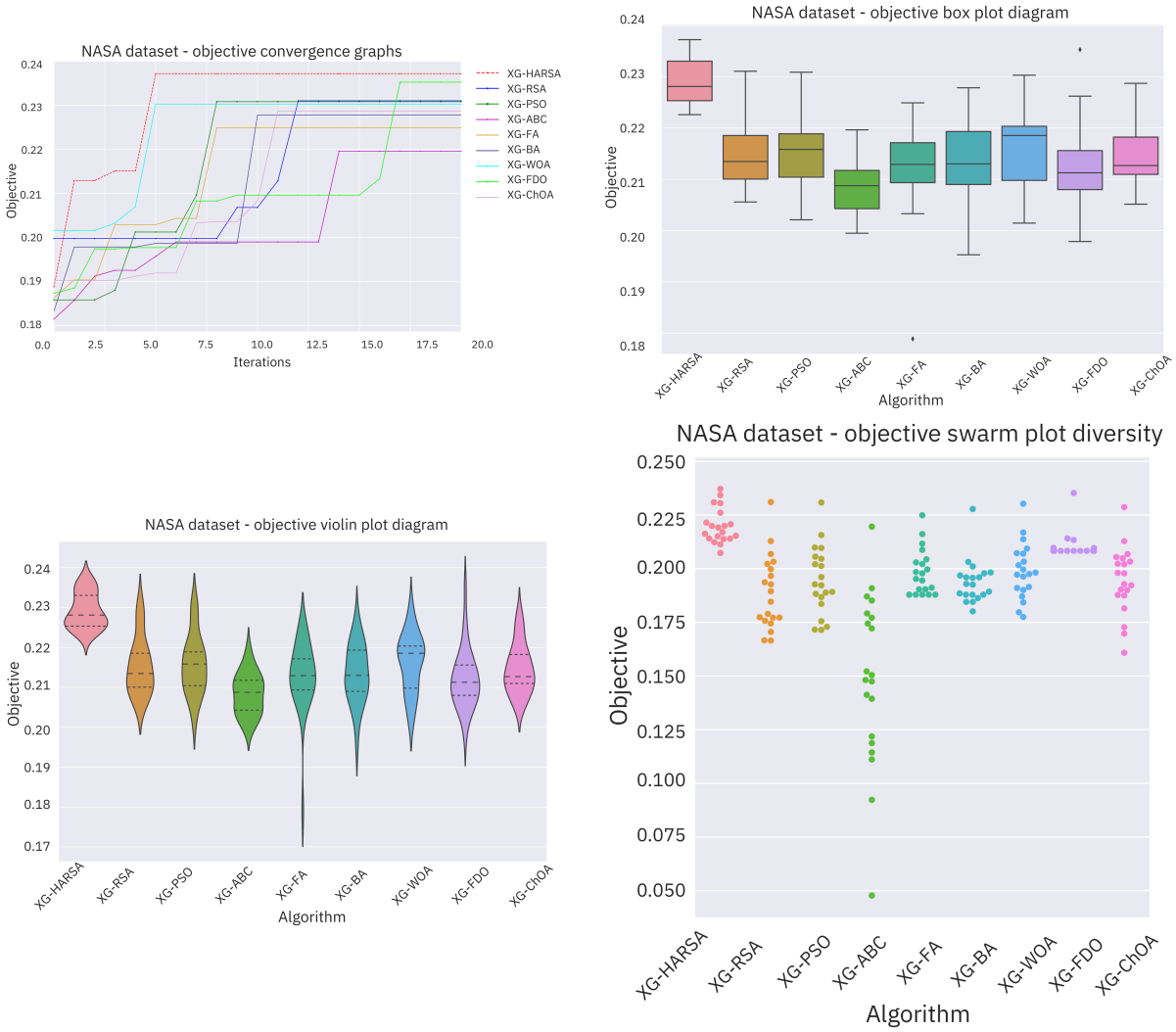
| | XG-HARSA | XG-RSA | XG-PSO | XG-ABC | XG-FA | XG-BA | XG-WOA | XG-FDO | XG-ChOA |
|---|---|---|---|---|---|---|---|---|---|
| Accuracy (%) | 78.2641 | 77.7404 | 77.2914 | 77.2166 | 78.2641 | 77.9648 | **78.5260** | 78.1145 | 78.3389 |
| Precision 0 | 0.813104 | 0.813056 | **0.814243** | 0.811674 | 0.810237 | 0.811686 | 0.810788 | 0.813053 | 0.810912 |
| Precision 1 | 0.534247 | 0.509554 | 0.491228 | 0.487730 | 0.537037 | 0.520408 | **0.551331** | 0.526846 | 0.540441 |
| W.Avg. Precision | 0.750405 | 0.744817 | 0.741616 | 0.738838 | 0.748811 | 0.746194 | **0.752452** | 0.748702 | 0.750099 |
| Recall 0 | 0.934363 | 0.925676 | 0.916023 | 0.919402 | 0.939672 | 0.931950 | **0.943050** | 0.931950 | 0.939672 |
| Recall 1 | 0.259567 | 0.266223 | **0.279534** | 0.264559 | 0.241265 | 0.254576 | 0.241265 | 0.261231 | 0.244592 |
| W.Avg. Recall | 0.782641 | 0.777404 | 0.772914 | 0.772166 | 0.782641 | 0.779648 | **0.785260** | 0.781145 | 0.783389 |
| F1 score 0 | 0.869526 | 0.865719 | 0.862139 | 0.862186 | 0.870168 | 0.867670 | **0.871932** | 0.868451 | 0.870557 |
| F1 score 1 | 0.349384 | 0.349727 | **0.356310** | 0.343042 | 0.332951 | 0.341899 | 0.335648 | 0.349277 | 0.336770 |
| W.Avg. F1 score | **0.752577** | 0.749703 | 0.748408 | 0.745461 | 0.749379 | 0.749455 | 0.751354 | 0.751719 | 0.750539 |

**Tabela 6.6:** Najbolje vrednosti skupa XGBoost parametara određene od strane svakog posmatranog algoritma

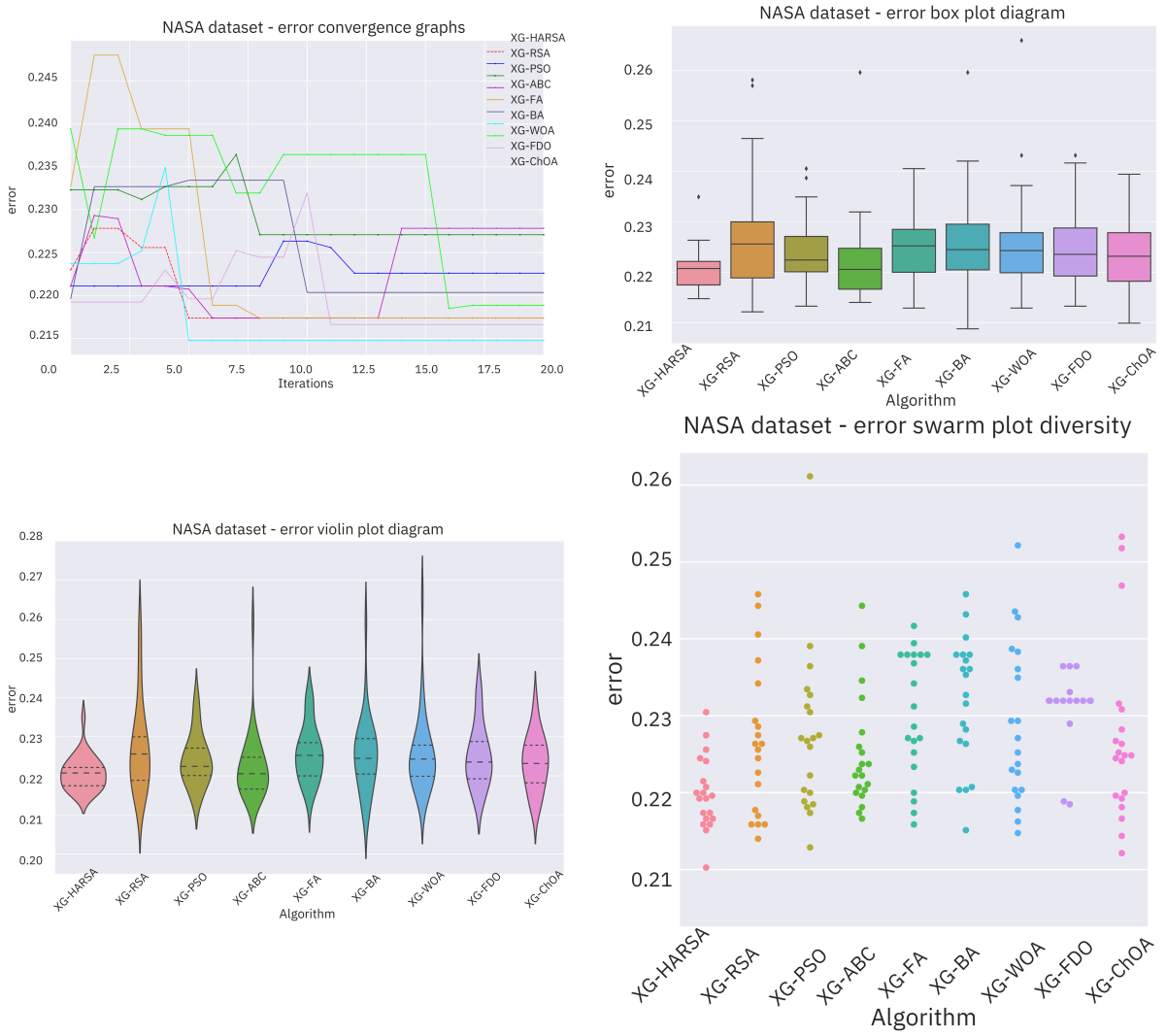
| Method | **l.r. ($\mu$)** | **max_child_weight** | **subsample** | **collsample_bytree** | **max_depth** | **gamma** |
|---|---|---|---|---|---|---|
| XG-HARSA | 0.732441 | 1.000000 | 0.664398 | 0.788768 | 9.000000 | 0.413284 |
| XG-RSA | 0.826363 | 10.000000 | 0.480902 | 1.000000 | 10.000000 | 0.564550 |
| XG-PSO | 0.695729 | 3.462237 | 0.472967 | 0.980642 | 10.000000 | 0.000000 |
| XG-ABC | 0.899881 | 5.891881 | 0.339652 | 0.637970 | 8.000000 | 0.531654 |
| XG-FA | 0.900000 | 8.829254 | 1.000000 | 0.752832 | 10.000000 | 0.800000 |
| XG-BA | 0.900000 | 6.201581 | 0.603000 | 0.924486 | 9.000000 | 0.491992 |
| XG-WOA | 0.523674 | 1.000000 | 1.000000 | 0.778599 | 10.000000 | 0.040756 |
| XG-FDO | 0.900000 | 9.539775 | 1.000000 | 1.000000 | 10.000000 | 0.297808 |
| XG-ChOA | 0.563379 | 10.000000 | 1.000000 | 1.000000 | 10.000000 | 0.000767 |

Izvedene simulacije su prikazane na Slici 6.5 i Slici 6.6 Ove slike ilustruju različite dijagrame, uključujući grafikone konvergencije dobijene najboljim pokretanjem svakog algoritma, boks dijagrame i violinske dijagrame za distribuciju funkcije cilja (Koenov kapa) na 30 nezavisnih izvršavanja, i dijagrame roja koji pokazuju raznolikost populacije tokom finalne runde najboljeg izvršavanja za funkciju cilja (Koenov kapa) i indikator stope greške. Pregledajući dijagrame roja za objektivnu funkciju (Koenov kapa), postaje očigledno da ABC pokazuje najveću raznolikost u finalnoj iteraciji, za razliku od HARSA algoritma, gde su svi pojedinci koncentrisani blizu najbolje podregije domena pretrage. Ovo ponašanje ABC-a je očekivano, jer ABC koristi *limit* parametar, obezbeđujući visoku raznolikost populacije u celini. Nasuprot tome, HARSA algoritam pokazuje snažnu intenzifikaciju prema trenutno najboljem pojedincu.

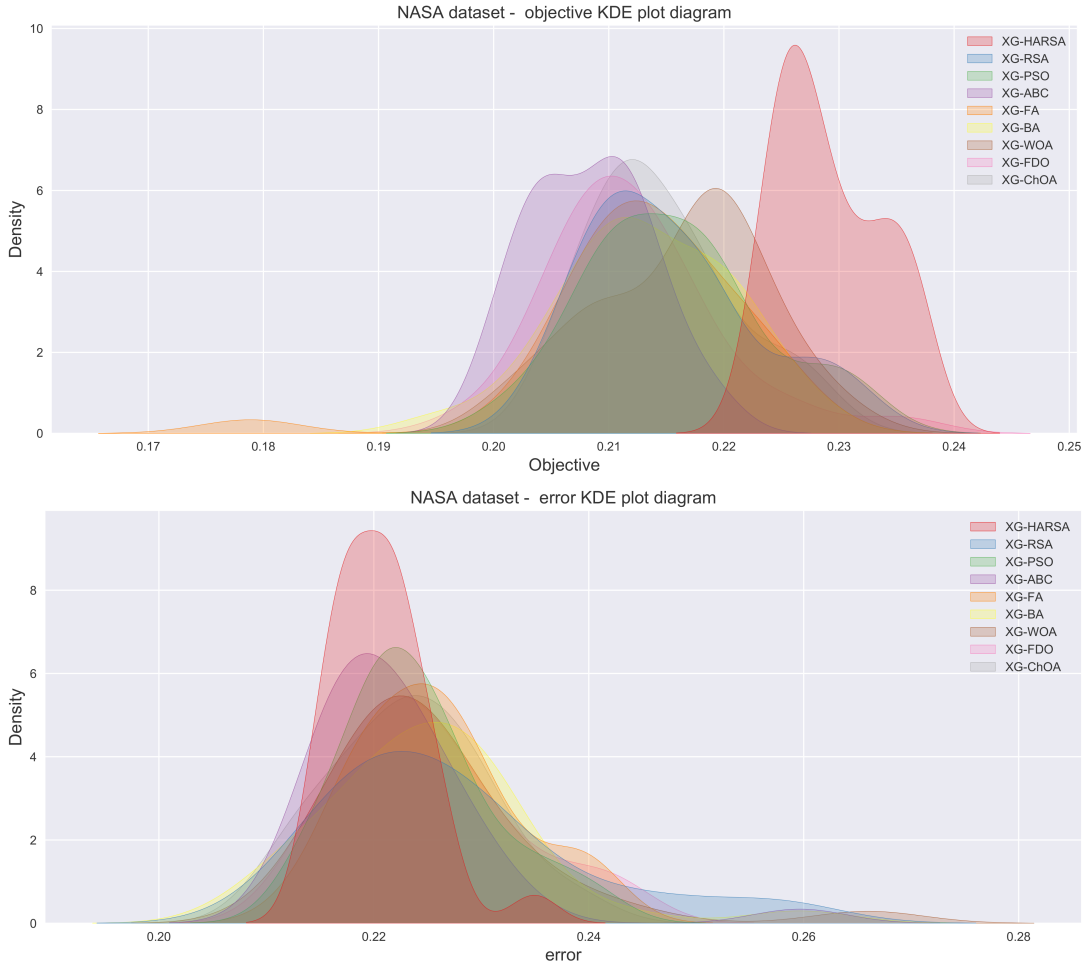
Slika 6.7 prikazuje Kernel Distribution Estimation (KDE) dijagrame koji pokazuju funkciju gustine verovatnoće za funkciju cilja (Koenov kapa) i indikator stope greške. Konačno, Slika 6.8 prikazuje matrice konfuzije, PR i ROC krive predloženog HARSA i osnovnog RSA jedan pored drugog, za NASA skup podataka sa Koenovim kapom kao objektivnom funkcijom. PR kriva pokazuje preciznost i pozivne vrednosti na grafikonu, pružajući vizuelni prikaz kompromisa između ove dve metrike. ROC kriva, s druge strane, pokazuje performanse modela za sve pragove klasifikacije.



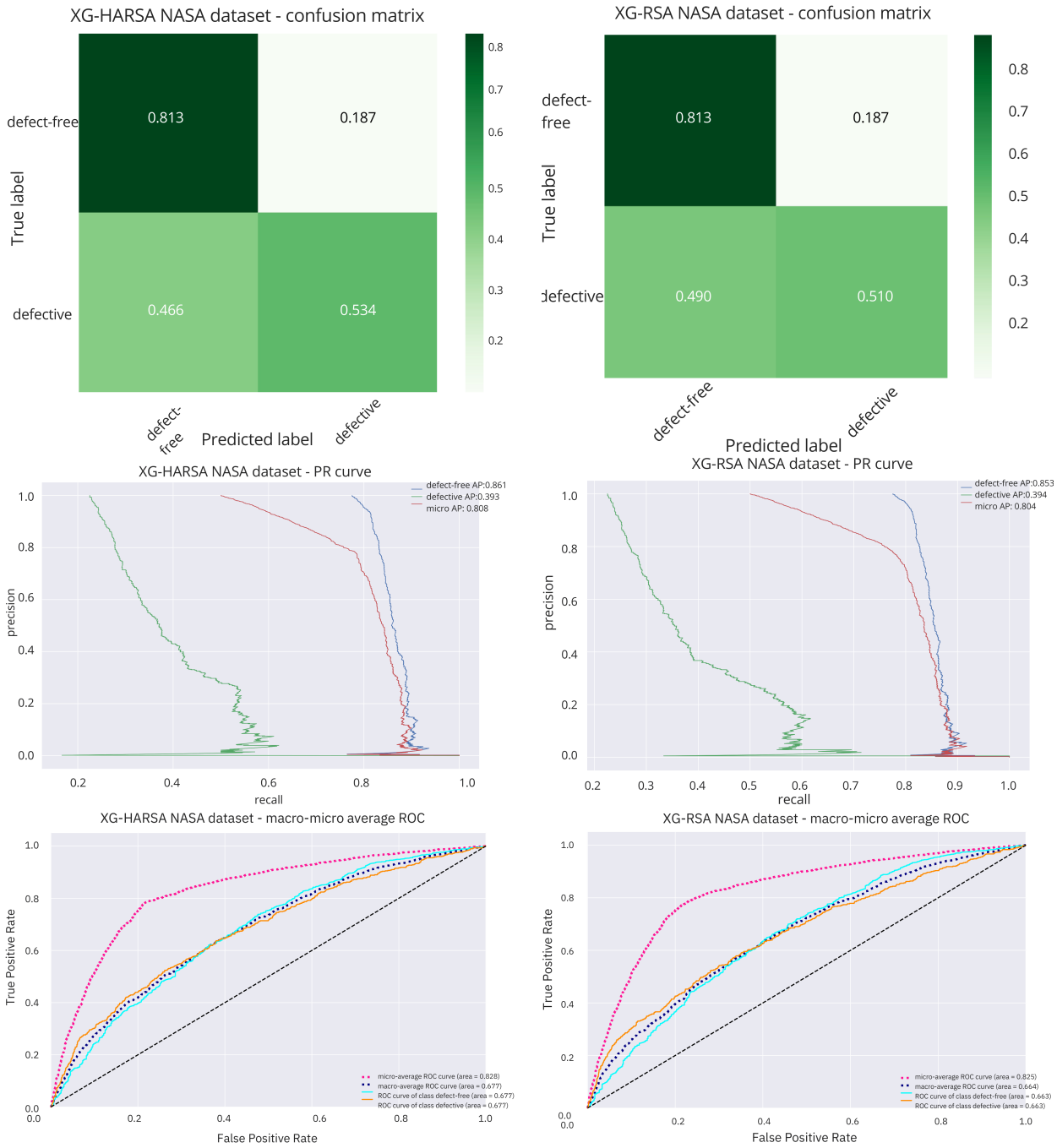

**Slika 6.5:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergenciju, boks dijagrame, violinske dijagrame, i dijagrame roja za funckiju cilja (Koenov kapa koeficijent) na NASA skupu podataka.




**Slika 6.6:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergenciju, boks dijagrame, violinske dijagrame, i dijagrame roja za indikator (stopa greške) na NASA skupu podataka.




**Slika 6.7:** Vizuelizacije KDE dijagrama za funkciju cilja (Koenov kapa koeficijent - gore) i indikator stopu greške (dole) za NASA skup podataka.




**Slika 6.8:** Vizuelizacije matrica konfuzije, PR i ROC krivih za predloženi XG-HARSA (levo) i XG-RSA (desno) algoritme na NASA skupu podataka sa Koenovim kapa koeficijentom kao funkcijom cilja.

## 6.2 Eksperimenti sa GHPR skupom podataka

GHPR skup podataka, s druge strane, je savršeno balansiran, stoga nema potrebe da se postavi Koenov kapa kao funkcija prilagođenosti. Shodno tome, izveden je jedan eksperiment, sa XGBoost modelima podešenim metaheuristikama. Stopa greške je definisana kao funkcija cilja koja je trebala da se minimizira. Takođe je praćena vrednost Koenove kape kao indikator.

### 6.2.1 XGBoost - stopa greške kao funkcija cilja

Ovaj deo sumira rezultate simulacije preko GHPR skupa podataka sa XGBoost podešenim metaheurističkim algoritmima, gde je stopa greške klasifikacije postavljena kao funkcija cilja koja treba da se minimizira. U svakoj tabeli koja sadrži rezultate simulacije, najbolji rezultat u svakoj od posmatranih kategorija je istaknut podebljanim tekstom.

Tabela 6.7 prikazuje rezultate svih metaheurističkih algoritama u vezi sa najboljim, najgorim, srednjim i medijalnim vrednostima preko 30 pojedinačnih pokretanja. Dodatno, pružene su standardna devijacija i varijansa. Ova tabela sadrži rezultate za funkciju cilja (stopa greške) i indikator Koenov kapa. Predloženi XG-HARSA pristup postigao je vrhunske rezultate, sa najboljim rezultatima za najbolje, najgore, srednje i medijalne vrednosti objektivne funkcije. XG-ChOA i XG-RSA završili su drugi i treći za najbolju metriku, dok je XG-ChOA završio drugi za najgore i srednje metrike. XG-PSO je postigao najstabilnije rezultate, sa najboljim rezultatima za standardnu devijaciju i varijansu. Pregledajući vrednosti indikatora (Koenov kapa), može se primetiti da je XG-HASCA algoritam postigao najviše vrednosti za najbolje, najgore, srednje i medijalne metrike. S druge strane, XG-PSO algoritam postigao je najbolji rezultat u pogledu standardne devijacije i varijanse. Tabela 6.8 prikazuje detaljne metrike za najbolje izvršavanje svakog ispitivanog algoritma. Može se primetiti da je predloženi XG-HARSA postigao superiorni nivo tačnosti klasifikacije od 81,33%, ispred XG-ChOA sa 81,28%, a XG-RSA je postigao tačnost od 81,22%. XG-HARSA je takođe postigao najbolji ukupni poziv i ukupni F1 rezultat (prosečne vrednosti za obe klase 0 i 1). Najbolju prosečnu preciznost postigao je XG-ChOA metod.

Na kraju, najbolje proizvedeni skup hiperparametara XGBoost-a (generisan najboljim izvršavanjem svakog algoritma) prikazan je u Tabeli 6.9. Najbolji model proizveden predloženim HARSA algoritmom imao je stopu učenja od 0.541359, maksimalnu vrednost težine deteta od 6.409800, vrednost poduzorka od 0.529058, vrednost collsample bytree od 0.930035, maksimalnu dubinu jednaku 3, i vrednost gama jednaku 0.318682.

**Tabela 6.7:** Ukupni rezultati koji prikazuju vrednosti funkcije cilja (stopa grešaka) XGBoost modela na GHPR skupu podataka

| | OBJECTIVE ERROR | | | | | COHEN KAPPA | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | Best | Worst | Mean | Median | Std | Var | Best | Worst | Mean | Median | Std | Var |
| XG-HARSA | **0.186674** | **0.190529** | **0.189042** | **0.189152** | 8.55E-04 | 7.31E-07 | **0.626652** | **0.618943** | **0.621916** | **0.621696** | 1.71E-03 | 2.92E-06 |
| XG-RSA | 0.187775 | 0.192731 | 0.190419 | 0.190253 | 9.89E-04 | 9.78E-07 | 0.624449 | 0.614537 | 0.619163 | 0.619493 | 1.98E-03 | 3.91E-06 |
| XG-PSO | 0.189427 | 0.192181 | 0.190529 | 0.190529 | **7.52E-04** | **5.66E-07** | 0.621145 | 0.615639 | 0.618943 | 0.618943 | **1.50E-03** | **2.26E-06** |
| XG-ABC | 0.188326 | 0.193282 | 0.191171 | 0.191630 | 1.37E-03 | 1.88E-06 | 0.623348 | 0.613436 | 0.617658 | 0.616740 | 2.74E-03 | 7.53E-06 |
| XG-FA | 0.189427 | 0.192731 | 0.191465 | 0.191630 | 9.34E-04 | 8.72E-07 | 0.621145 | 0.614537 | 0.617070 | 0.616740 | 1.87E-03 | 3.49E-06 |
| XG-BA | 0.189978 | 0.193833 | 0.191391 | 0.191079 | 8.84E-04 | 7.82E-07 | 0.620044 | 0.612335 | 0.617217 | 0.617841 | 1.77E-03 | 3.13E-06 |
| XG-WOA | 0.189427 | 0.194934 | 0.191538 | 0.191630 | 1.22E-03 | 1.50E-06 | 0.621145 | 0.610132 | 0.616924 | 0.616740 | 2.45E-03 | 5.99E-06 |
| XG-FDO | 0.188877 | 0.192731 | 0.191281 | 0.191630 | 1.01E-03 | 1.02E-06 | 0.622247 | 0.614537 | 0.617438 | 0.616740 | 2.02E-03 | 4.08E-06 |
| XG-ChOA | 0.187225 | 0.191630 | 0.190272 | 0.190529 | 1.02E-03 | 1.05E-06 | 0.625551 | 0.616740 | 0.619457 | 0.618943 | 2.05E-03 | 4.18E-06 |

**Tabela 6.8:** Detaljni rezultati najboljeg izvršavanja svakog algoritma u odnosu na fukciju cilja (stopa greške) XGBoost modela na GHPR skupu podataka

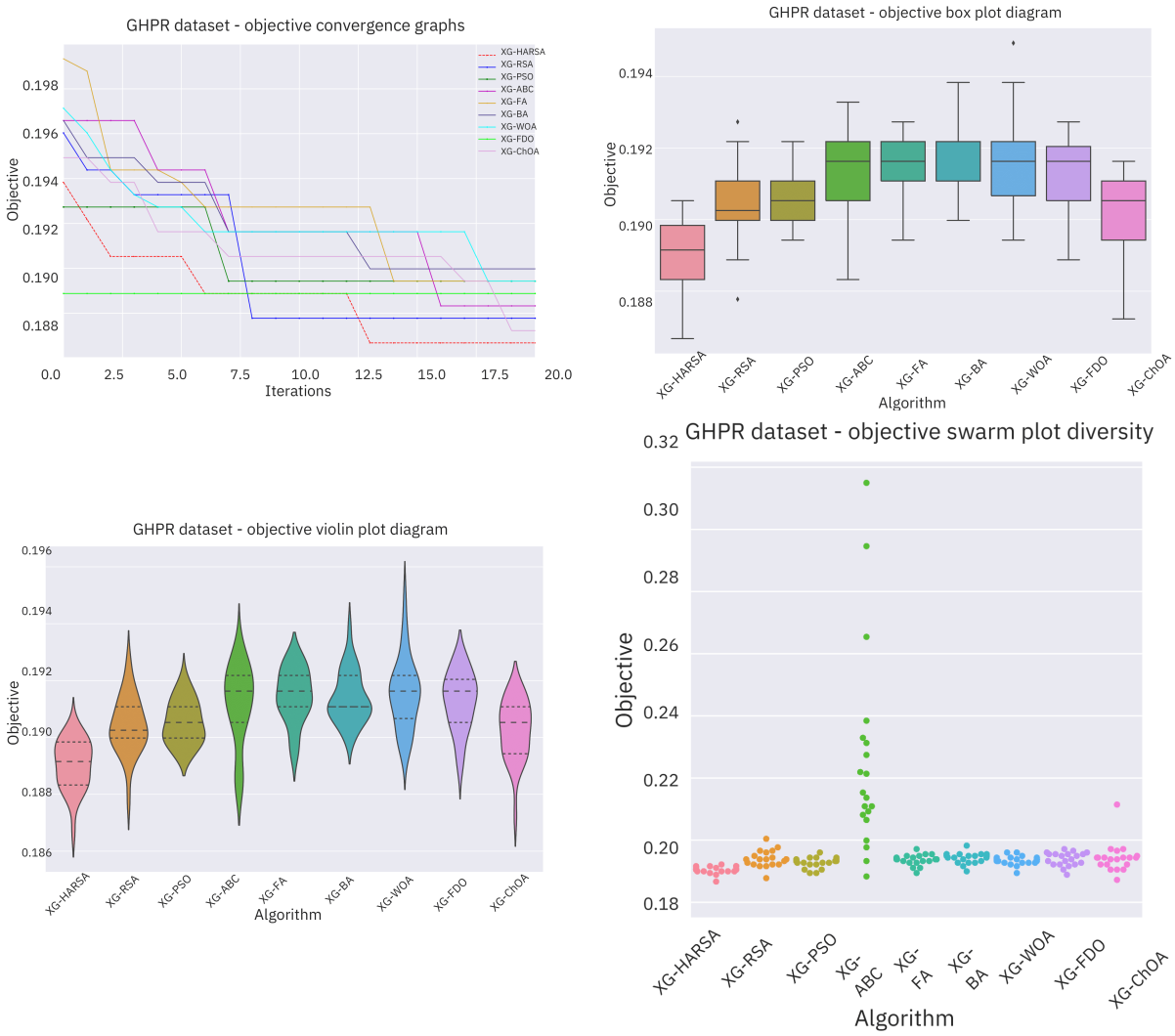
| | XG-HARSA | XG-RSA | XG-PSO | XG-ABC | XG-FA | XG-BA | XG-WOA | XG-FDO | XG-ChOA |
|---|---|---|---|---|---|---|---|---|---|
| Accuracy (%) | **81.3326** | 81.2225 | 81.0573 | 81.1674 | 81.0573 | 81.0022 | 81.0573 | 48.6233 | 81.2775 |
| Precision 0 | 0.845687 | 0.848708 | 0.841404 | 0.838517 | 0.849010 | 0.845399 | 0.849010 | 0.490566 | 0.848894 |
| Precision 1 | 0.786506 | 0.782652 | 0.784848 | **0.788776** | 0.779762 | 0.781219 | 0.779762 | 0.474542 | 0.783433 |
| W.Avg. Precision | 0.816096 | 0.815680 | 0.813126 | 0.813646 | 0.814386 | 0.813309 | 0.814386 | 0.482554 | **0.816164** |
| Recall 0 | 0.766520 | 0.759912 | 0.765419 | **0.772026** | 0.755507 | 0.758811 | 0.755507 | 0.715859 | 0.761013 |
| Recall 1 | 0.860132 | 0.864537 | 0.855727 | 0.851322 | 0.865639 | 0.861233 | 0.865639 | 0.256608 | 0.864537 |
| W.Avg. Recall | **0.813326** | 0.812225 | 0.810573 | 0.811674 | 0.810573 | 0.810022 | 0.810573 | 0.486233 | 0.812775 |
| F1 score 0 | **0.804159** | 0.801859 | 0.801615 | 0.803899 | 0.799534 | 0.799768 | 0.799534 | 0.582176 | 0.802555 |
| F1 score 1 | 0.821673 | 0.821559 | 0.818757 | 0.818856 | 0.820459 | 0.819277 | 0.820459 | 0.333095 | **0.821990** |
| W.Avg. F1 score | **0.812916** | 0.811709 | 0.810186 | 0.811378 | 0.809997 | 0.809522 | 0.809997 | 0.457636 | 0.812272 |

**Tabela 6.9:** Najbolje vrednosti skupa XGBoost parametara određene od strane svakog posmatranog algoritma

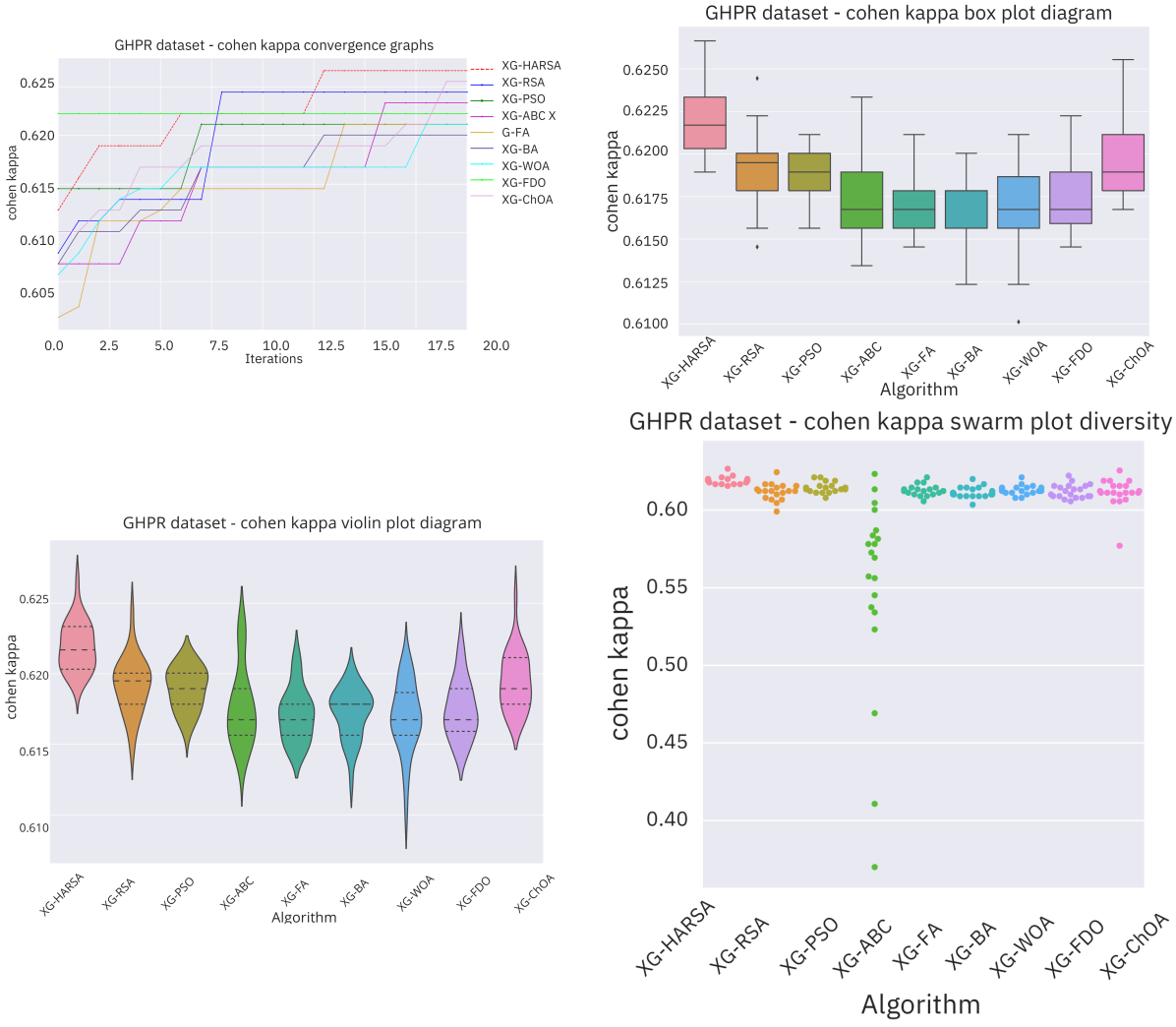
| Method | **l.r. ($\mu$)** | **max_child_weight** | **subsample** | **collsample_bytree** | **max_depth** | **gamma** |
|---|---|---|---|---|---|---|
| XG-HARSA | 0.541359 | 6.409800 | 0.529058 | 0.930035 | 3.000000 | 0.318682 |
| XG-RSA | 0.404205 | 10.000000 | 0.547269 | 1.000000 | 3.000000 | 0.258175 |
| XG-PSO | 0.143699 | 7.414493 | 1.000000 | 0.518893 | 3.000000 | 0.031470 |
| XG-ABC | 0.397667 | 5.106322 | 0.333215 | 1.000000 | 4.000000 | 0.628889 |
| XG-FA | 0.100000 | 2.560790 | 0.297314 | 0.956494 | 4.000000 | 0.800000 |
| XG-BA | 0.277573 | 9.738516 | 1.000000 | 0.509089 | 3.000000 | 0.790795 |
| XG-WOA | 0.100000 | 10.000000 | 0.296498 | 1.000000 | 4.000000 | 0.800000 |
| XG-FDO | 0.229929 | 7.283359 | 0.367354 | 0.979733 | 4.000000 | 0.750163 |
| XG-ChOA | 0.524442 | 1.396984 | 0.548038 | 1.000000 | 3.000000 | 0.754814 |

Funkcija cilja (stopa greške) i Koenov kapa indikator su vizualizovani u izvedenim simulacijama prikazanim na Slici 6.9 i Slici 6.10. Figure uključuju grafikone konvergencije postignute najboljim izvršavanjem svakog algoritma, boks dijagrame i violinske dijagrame za distribuciju funkcije cilja na 30 nezavisnih izvršavanja, i dijagrame roja koji predstavljaju raznolikost populacije tokom finalne runde najboljeg pokretanja. Gledajući dijagrame roja, očigledno je da je ABC imao najveću raznolikost u finalnoj iteraciji, dok je predloženi HARSA metod imao sve pojedince koncentrisane u blizini najbolje pod-regije domena pretrage. Ovo ponašanje ABC-a je očekivano, jer ABC koristi limitni parametar, obezbeđujući visoku raznolikost populacije u celini. Nasuprot tome, HARSA algoritam pokazuje snažnu intenzifikaciju prema trenutno najboljem pojedincu.

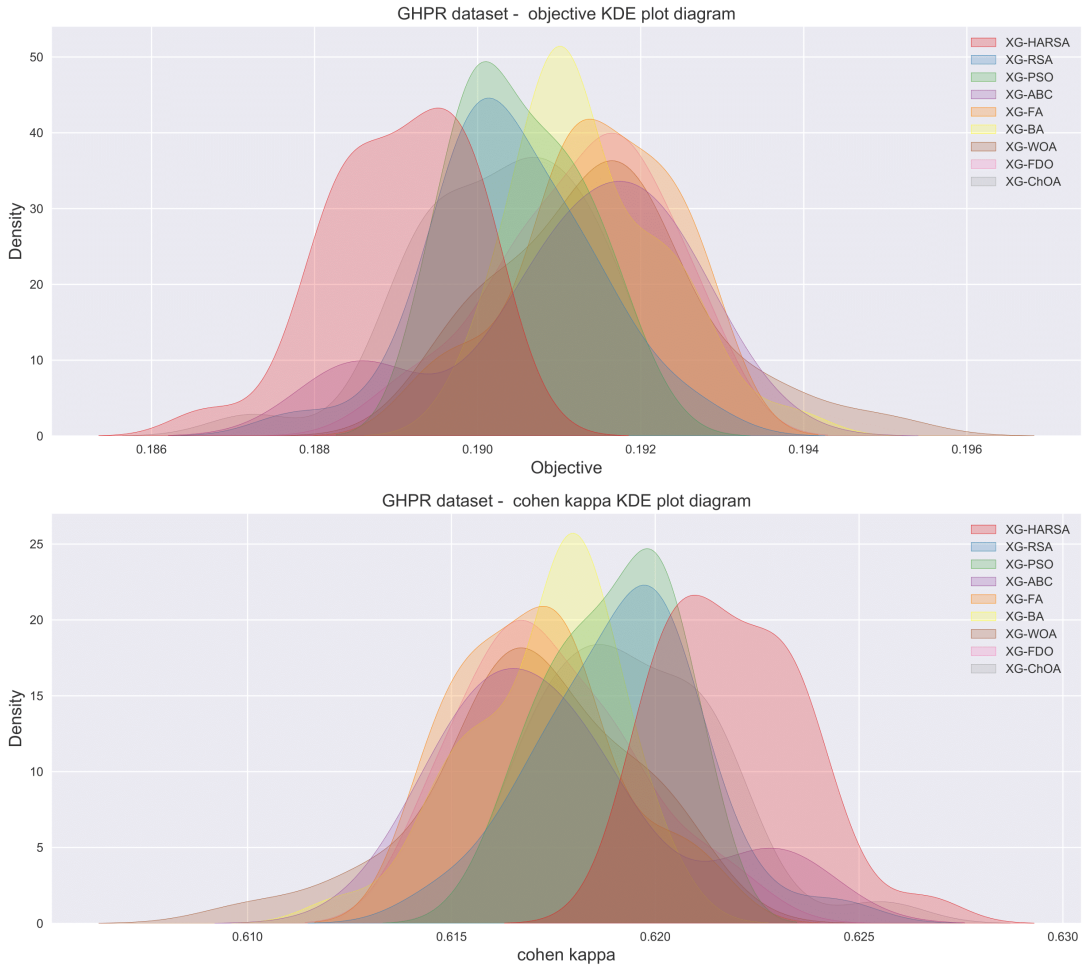
Kernel Distribution Estimation (KDE) dijagrami za funkciju prilagođenosti (stopa greške) i Koenov kapa indikator, koji prikazuju funkciju gustine verovatnoće, predstavljeni su na Slici 6.11. Konačno, Slika 6.12 prikazuje matrice konfuzije, PR i ROC krive predloženog HARSA i osnovnog RSA jedan pored drugog. PR kriva pokazuje preciznost i pozivne vrednosti na grafikonu, pružajući vizuelni prikaz kompromisa između ove dve metrike. ROC kriva, s druge strane, pokazuje performanse modela za sve pragove klasifikacije.



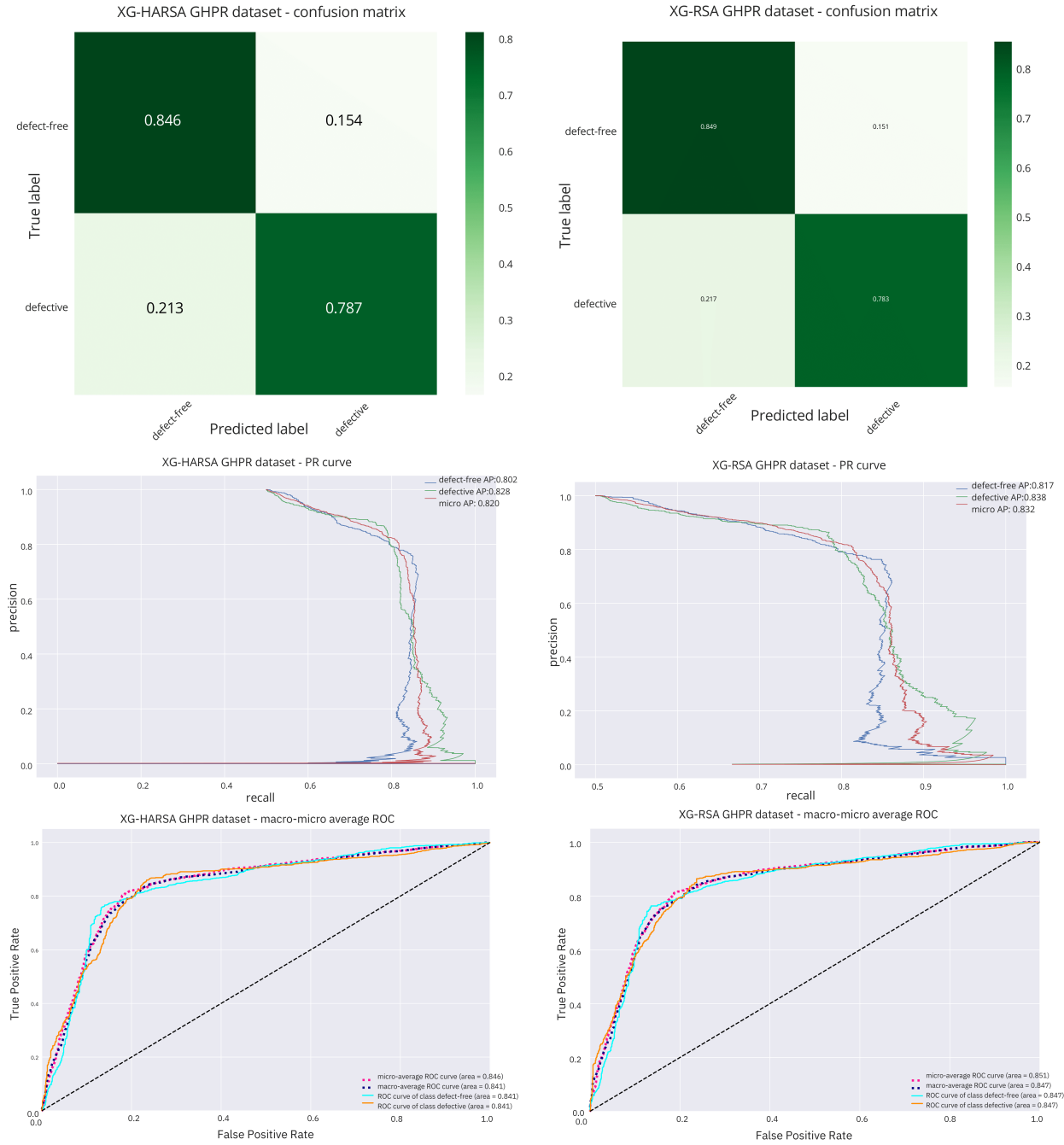

**Slika 6.9:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergenciju, boks dijagrame, violinske dijagrame, i dijagrame roja za funkciju cilja (stopu greške) na GHPR skupu podataka.




**Slika 6.10:** Vizuelizacije izvršenih XGBoost simulacija za svih 9 algoritama u odnosu na konvergenciju, boks dijagrame, violinske dijagrame, i dijagrame roja za indikator (Koenov kapa koeficijent) na GHPR skupu podataka.




**Slika 6.11:** Vizuelizacije KDE dijagrama za funkciju cilja (stopa greške - gore) i Koenov kapa indikator (dole) za GHPR skup podataka.




**Slika 6.12:** Vizuelizacije matrica konfuzije, PR i ROC krivih za predloženi XG-HARSA (levo) i XG-RSA (desno) algoritme na GHPR skupu podataka sa stopom greške kao funkcijom cilja.

## 6.3 Poređenja sa tradicionalnim modelima

Da bi se ilustrovao nivo performansi predstavljenog XG-HARSA modela, ovaj deo donosi poređenja sa tradicionalnim modelima (u čistom obliku, bez optimizacije metaheuristikama), na oba skupa podataka. Tradicionalni modeli koji su posmatrani bili su klasifikator slučajna šuma - RF [102](podrazumevano podešavanje parametara), SVM [103](sa podrazumevanim parametrima), i dva modela dubokih neuronskih mreža (DNN), sa jednim i dva skrivena sloja, respektivno, i sa brojem neurona jednakim broju karakteristika u skupu podataka.

Za NASA skup podataka, oba XG-HARSA modela (sa stopom greške i Koenovom kapom kao funkcijom cilja) su uzeta i korišćena u poređenjima, a rezultati su prikazani u Tabeli 6.10. Još jednom, treba naglasiti da je NASA skup podataka nebalansiran, stoga postignuta tačnost modela može biti obmanjujuća, jer model može imati visoku tačnost, a da i dalje ima loše performanse za manjinsku klasu. Manjinska klasa u ovom skupu podataka je zapravo defektni softver, i loša tačnost na manjinskoj klasi će nesumnjivo dovesti do značajnog broja propuštenih defekata. Kao što je ranije diskutovano, vrednost Koenove kape je daleko bolji indikator za nebalansirane skupove podataka. Imajući sve ovo na umu, može se primetiti da je najviša tačnost od 79,68% postignuta modelom DNN sa dva skrivena sloja, međutim sa vrlo niskom vrednošću Koenove kape (0,09175), što ukazuje na lošu tačnost na manjinskoj klasi. Oba XG-HARSA modela (sa greškom i Koenovom kapom kao funkcijom cilja) su se vrlo dobro pokazala u pogledu tačnosti, gde je model sa greškom kao funkcijom cilja postigao drugu najbolju tačnost od 79,39% i drugu najbolju vrednost Koenovog kapa indikatora od 0,204163. XG-HARSA model gde je optimizovana Koenova kapa postigao je treću najbolju tačnost od 78,26%, ali i superiorni nivo Koenove kape od 0,237225, što ukazuje da je ovaj model najbolji u rukovanju manjinskom klasom.

Gledajući performanse drugih tradicionalnih modela, RF, SVM i DNN sa jednim skrivenim slojem postigli su pristojnu tačnost od 78,11%, 77,52% i 75,77%, međutim, posmatrane vrednosti Koenove kape bile su vrlo niske - na primer, SVM je imao najgori rezultat Koenove kape od 0,012657. Običan, neoptimizovani XGBoost postigao je tačnost od 74,34%, što je 4-5% ispod tačnosti postignute predloženim XG-HARSA modelima, direktno pokazujući važnost podešavanja modela za specifični problem. Običan XGBoost je postigao vrednost Koenove kape od 0,144685, što je znatno niže od XG-HARSA modela, ali ipak bolje od drugih tradicionalnih modela korišćenih u poređenjima.

**Tabela 6.10:** Poređenje performansi XG-HARSA modela sa drugim tradicionalnim modelima na NASA skupu podataka.

| **Methods** | accuracy | error rate | Cohen's kappa |
|---|---|---|---|
| XG-HARSA (error) | 79.39% | 0.206135 | 0.204163 |
| XG-HARSA (Cohen's kappa) | 78.26% | 0.217359 | **0.237225** |
| XGBoost | 74.34% | 0.256610 | 0.144685 |
| RF | 78.11% | 0.218856 | 0.089953 |
| SVM | 77.52% | 0.224841 | 0.012657 |
| DNN 1 hidden layer | 75.77% | 0.242277 | 0.0872 |
| DNN 2 hidden layers | **79.68**% | **0.20324** | 0.09175 |

Za GHPR skup podataka, XG-HARSA sa stopom greške kao funkcijom cilja je uzet i korišćen u poređenjima, a rezime je prikazan u Tabeli 6.11. Ovo je primer savršeno balansiranog skupa, tako da Koenova kapa nije ključni indikator koji treba pratiti, ali fokus

bi trebalo da bude na postignutoj tačnosti modela. Predloženi XG-HARSA (sa stopom greške klasifikacije postavljenom kao funkcijom cilja) se znatno bolje pokazao od tradicionalnih modela, postižući tačnost od 81,33%, i takođe postižući najbolju vrednost Koenove kape od 0,626652. Osnovni XGBoost postigao je drugu najbolju tačnost od 78,63%, što znači da je poboljšanje tačnosti primenom optimizacije metaheuristikama hiperparametara modela oko 2,7%, ponovo očigledno ilustrujući važnost pravilne optimizacije modela za bilo koji specifični zadatak klasifikacije. Ostali tradicionalni modeli su imali znatno nižu tačnost, gde je DNN sa dva skrivena sloja postigao tačnost od 74,18% (više od 7% niže od predloženog modela), a klasifikator RF postigao je tačnost od 72,26% (oko 9% niže od predloženog modela). SVM model je imao najgoru tačnost na ovom skupu podataka, postižući samo 60,21%, što je više od 20% niže od predloženog modela.

**Tabela 6.11:** Poređenje performansi XG-HARSA modela sa drugim tradicionalnim modelima na GHPR skupu podataka.

| **Methods** | accuracy | error rate | Cohen's kappa |
|---|---|---|---|
| XG-HARSA | **81.33%** | **0.186674** | **0.626652** |
| XGBoost | 78.63% | 0.213656 | 0.572687 |
| RF | 72.26% | 0.277423 | 0.54515 |
| SVM | 60.21% | 0.39786 | 0.45429 |
| DNN 1 hidden layer | 70.65% | 0.29349 | 0.533039 |
| DNN 2 hidden layers | 74.18% | 0.2582 | 0.55969 |

## 6.4 Validacija rezultata simulacije i interpretacija najboljeg postignutog modela

### 6.4.1 Statistički testovi

Nakon završetka eksperimenata, dobijene rezultate treba validirati da bi se utvrdilo da li su statistički značajni. Sa ciljem da se to postigne, najbolji rezultati svakog od 30 pojedinačnih izvršenja svakog posmatranog metoda, i sva tri posmatrana problema (NASA skup podataka - stopa greške klasifikacije i Koenova kapa, i GHPR skup podataka) su sakupljeni i ispitani kao serije podataka. Zatim, potrebno je utvrditi da li se može sigurno nastaviti sa parametričkim testom, inače se mora koristiti neparametrički test. Da bi se procenilo da li je prikladno nastaviti sa parametričkim testom, potrebno je ispitati zahteve nezavisnosti, normalnosti i homoscedastičnosti [104]. Uslov nezavisnosti je jasno zadovoljen jer svako pojedinačno pokretanje metaheurističkog algoritma počinje generisanjem kolekcije slučajnih rešenja. Međutim, uslov normalnosti nije zadovoljen, jer je već bilo moguće uočiti iz KDE dijagrama prikazanih na Slikama 6.3, 6.7 i 6.11 da rezultati ne izgledaju kao da dolaze iz normalne distribucije. Kao sporedna napomena ovde, do neke mere se očekuje da rezultati ne dolaze iz normalne distribucije, jer su metaheuristički algoritmi stohastički, i generalno je potreban veliki broj pokretanja da bi se jasno otkrila njihova priroda. Rezultati bi mogli poticati iz normalne distribucije ako bi algoritmi bili izvršeni u više od 30 nezavisnih pokretanja, međutim, pošto su simulacije izuzetno zahtevne u pogledu računarskih resursa i vremena izvršenja, nažalost, to nije bilo primenljivo.

**Tabela 6.12:** Shapiro-Wilk $p-vrednosti$ za XGBoost modele izračunate za verifikaciju uslova normalnosti

| **Metode** | HARSA | RSA | PSO | ABC | FA | BA | WOA | FDO | ChOA |
|---|---|---|---|---|---|---|---|---|---|
| NASA - error | 0.037 | 0.019 | 0.029 | 0.045 | 0.039 | 0.042 | 0.043 | 0.039 | 0.036 |
| NASA - Cohen | 0.034 | 0.039 | 0.038 | 0.028 | 0.042 | 0.034 | 0.025 | 0.031 | 0.033 |
| GHPR - error | 0.041 | 0.038 | 0.039 | 0.024 | 0.016 | 0.032 | 0.027 | 0.037 | 0.036 |

Da bi se potvrdila ova pretpostavka, uslov normalnosti je dodatno proveren korišćenjem Shapiro-Wilkovog testa jednostrukog problema [97]. U ovom testu, Shapiro-Wilkove $p-vrednosti$ su izračunate odvojeno za svaki posmatrani algoritam. Pošto su sve određene $p-vrednosti$ ispod praga od 0,05, H0 hipoteza može biti odbačena na nivou $alpha = 0,05$. Kao posledica, može se sigurno zaključiti da eksperimentalni rezultati ne pripadaju normalnoj distribuciji, potvrđujući posmatranje iz KDE dijagrama. Shapiro-Wilkovi rezultati za sve 3 instance problema su dati u Tabeli 6.12.

Zaključak sprovedenog Shapiro-Wilkovog testa je da nije moguće bezbedno nastaviti sa parametričkim testovima jer uslov normalnosti nije zadovoljen, stoga je potrebno nastaviti sa neparametričkom alternativom. U ovom radu, izvršen je Wilcoxonov signed-rank test [105] koristeći istu kolekciju serija podataka sastavljenih od najboljih rezultata svakog izvršenja svih algoritama. Predloženi HARSA je izabran kao kontrolni metod, a dobijeni rezultati za sve tri instance problema su dati u Tabeli 6.13.

**Tabela 6.13:** $p-vrednosti$ izračunate pomoću Wilcoxon signed-rank testa za sva tri eksperimenta

| **Metode** | HARSA | RSA | PSO | ABC | FA | BA | WOA | FDO | ChOA |
|---|---|---|---|---|---|---|---|---|---|
| NASA - error | N/A | 0.040 | 0.035 | 0.039 | 0.027 | 0.029 | 0.033 | 0.029 | 0.042 |
| NASA - Cohen | N/A | 0.039 | 0.042 | 0.021 | 0.030 | 0.032 | 0.046 | 0.027 | 0.035 |
| GHPR - error | N/A | 0.048 | 0.046 | 0.042 | 0.031 | 0.034 | 0.027 | 0.038 | 0.057 |

Za eksperimente NASA sa stopom greške kao funkcijom cilja, dobijene $p-vrednosti$ su u svakom slučaju ispod praga od 0,05 (izračunate $p-vrednosti$ su date u Tabeli 6.13). Stoga, moguće je sigurno pretpostaviti da je predloženi HARSA algoritam statistički značajno bolji od ostalih konkurenata za oba praga $alpha = 0,1$ i $alpha = 0,05$.

Za eksperimente NASA sa Koenovom kapom kao funkcijom cilja, dobijene $p-vrednosti$ su ponovo u svakom slučaju ispod praga od 0,05 (izračunate $p-vrednosti$ u ovom scenariju su date u Tabeli 6.13). Stoga, dozvoljeno je sigurno pretpostaviti da je predloženi HARSA algoritam statistički superiorniji od ostalih konkurenata za oba praga $alpha = 0,1$ i $alpha = 0,05$.

U slučaju eksperimenata GHPR, određene $p-vrednosti$ su ispod 0,05 za svaki metod osim ChOA osim ChOA čija je vrednost bila $0,057 > 0,05$, dok su ostale $p-vrednosti$ u ovom scenariju date u Tabeli 6.13. U ovom scenariju, može se izjaviti da je uvedeni HARSA znatno bolji u poređenju sa drugim konkurentima osim ChOA za prag $alpha = 0,05$, i izuzetno superiorniji od svih posmatranih algoritama za prag $alpha = 0,1$.

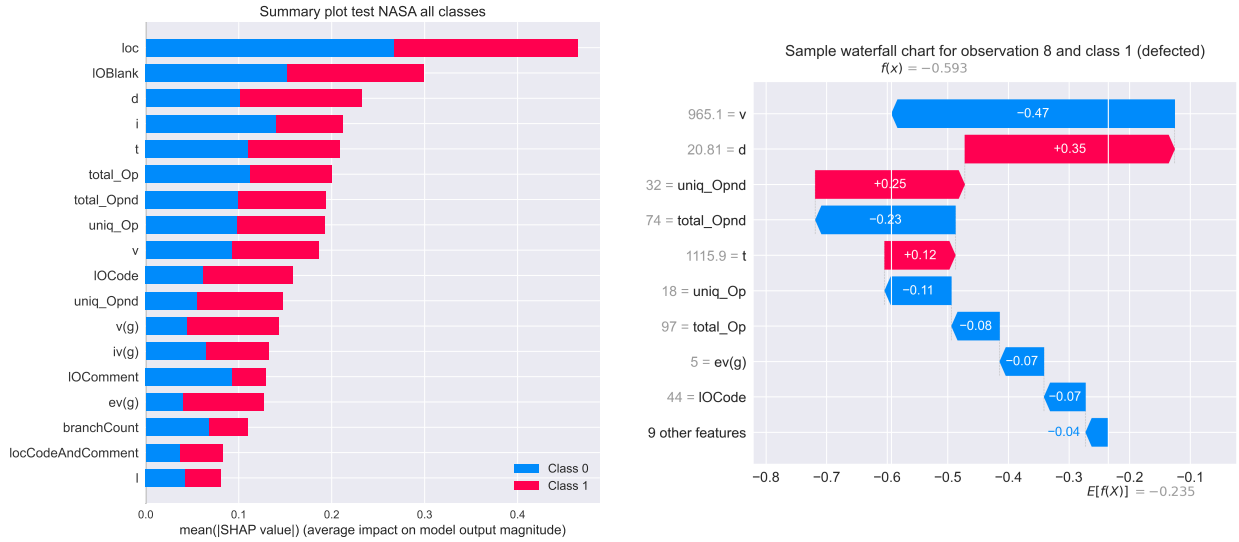
### 6.4.2 Interpretacija rezultata

Modeli mašinskog učenja mogu se posmatrati kao crna kutija, što čini interpretaciju neophodnom kako bi se utvrdilo koje karakteristike imaju najveći uticaj na ciljnu promen-

ljivu. Ovo razumevanje je ključno u domenu mrežne sigurnosti i može pomoći donosiocima odluka da donesu bolje odluke. Ponašanje modela može se razumeti kroz upotrebu napredne tehnike objašnjive veštačke inteligencije poznate kao Shapley Additive Explanations (SHAP), koja omogućava dublje razumevanje i modela i rezultata eksperimenata. SHAP mehanizam omogućava izbegavanje kompromisa između tačnosti i interpretabilnosti rezultata. Pruža jednostavnu i relevantnu interpretaciju predviđanja modela merenjem važnosti karakteristika. SHAP postupak se oslanja na Shapley vrednosti iz teorije igara, koje pružaju vredne informacije o karakteristikama koje imaju najveći uticaj na predviđanje [106].

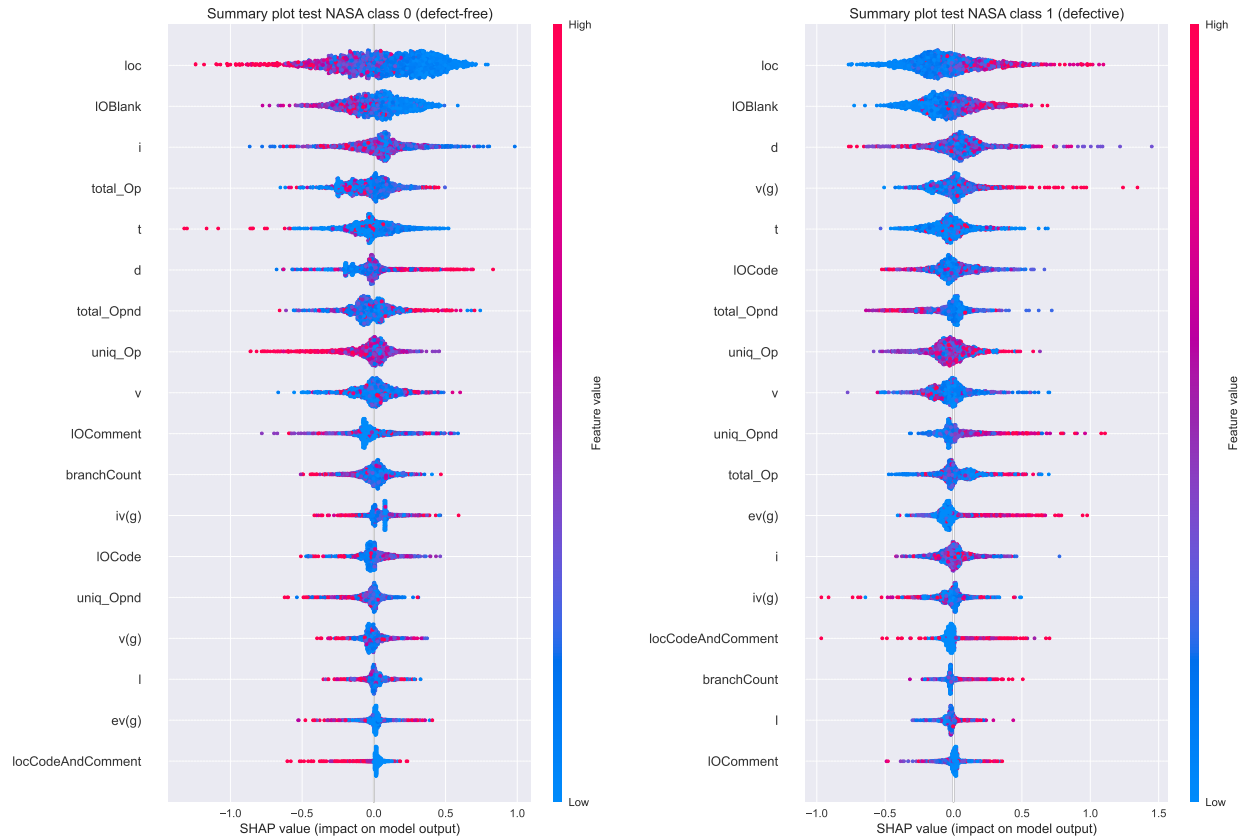
Jednostavno rečeno, Shapley skup vrednosti je distribucija isplata među karakteristikama, odražavajući doprinos svake karakteristike sveobuhvatnoj isplati (koja predstavlja predviđanje). Kao rezultat, SHAP postupak dodeljuje meru važnosti svim karakteristikama, predstavljajući koliko svaka od njih doprinosi specifičnom predviđanju.

U slučaju NASA skupa podataka, uzet je najbolji model, tačnije XG-HARSA model, sa Koenovom kapom kao objektivnom funkcijom, jer u proseku postiže najbolje rezultate. Slika 6.13 prikazuje sažeti dijagram svih klasa i vodopadni dijagram za klasu 1 (defektni modul), dok Slika 6.14 pokazuje sažete dijagrame za klasu 0 (bez defekata) i klasu 1 (defektna).




**Slika 6.13:** SHAP sažeti dijagram (levo) i vodopadni dijagram za klasu 1 (defektni modul) za NASA skup podataka (desno).

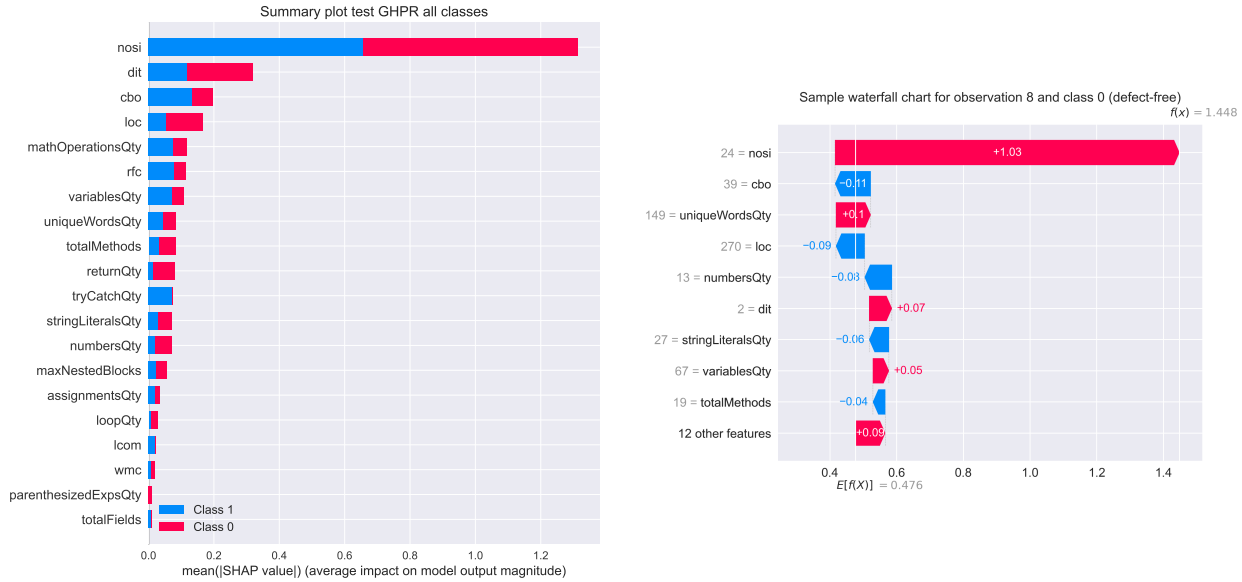
Analiza SHAP-a za NASA skup podataka pokazuje da su najvažnije karakteristike v(g), koje označavaju ciklomatsku složenost, zatim d (Halsteadova težina), jedinstveni operandi i ukupan broj operanada. Gledajući sažeti dijagram za klasu 1 sa Slike 6.14, očigledno je da će modul znatno verovatnije imati defekte ako je ciklomatska složenost veća (v(g)). Ovo se očekuje jer McCabeova ciklomatska složenost pokazuje složenost koda (broj nezavisnih putanja kroz kod). Broj linija koda (karakteristika loc) je takođe u direktnoj korelaciji sa verovatnoćom defekata jer ako modul ima veliki loc, verovatnije je da će takođe imati defekte. Ponovo, kako se očekuje, povećanje grana / odluka u kodu (karakteristika branchCount), McCabeova esencijalna složenost (ev(g)), broj jedinstvenih operanada i broj ukupnih operanada će sve drastično povećati verovatnoću da će modul imati defekte. Zanimljivo je da broj linija koje sadrže komentare (Halsteadova metrika lOComment) nema značajan uticaj. Svi ovi zaključci su takođe podržani posmatranjima u praksi.



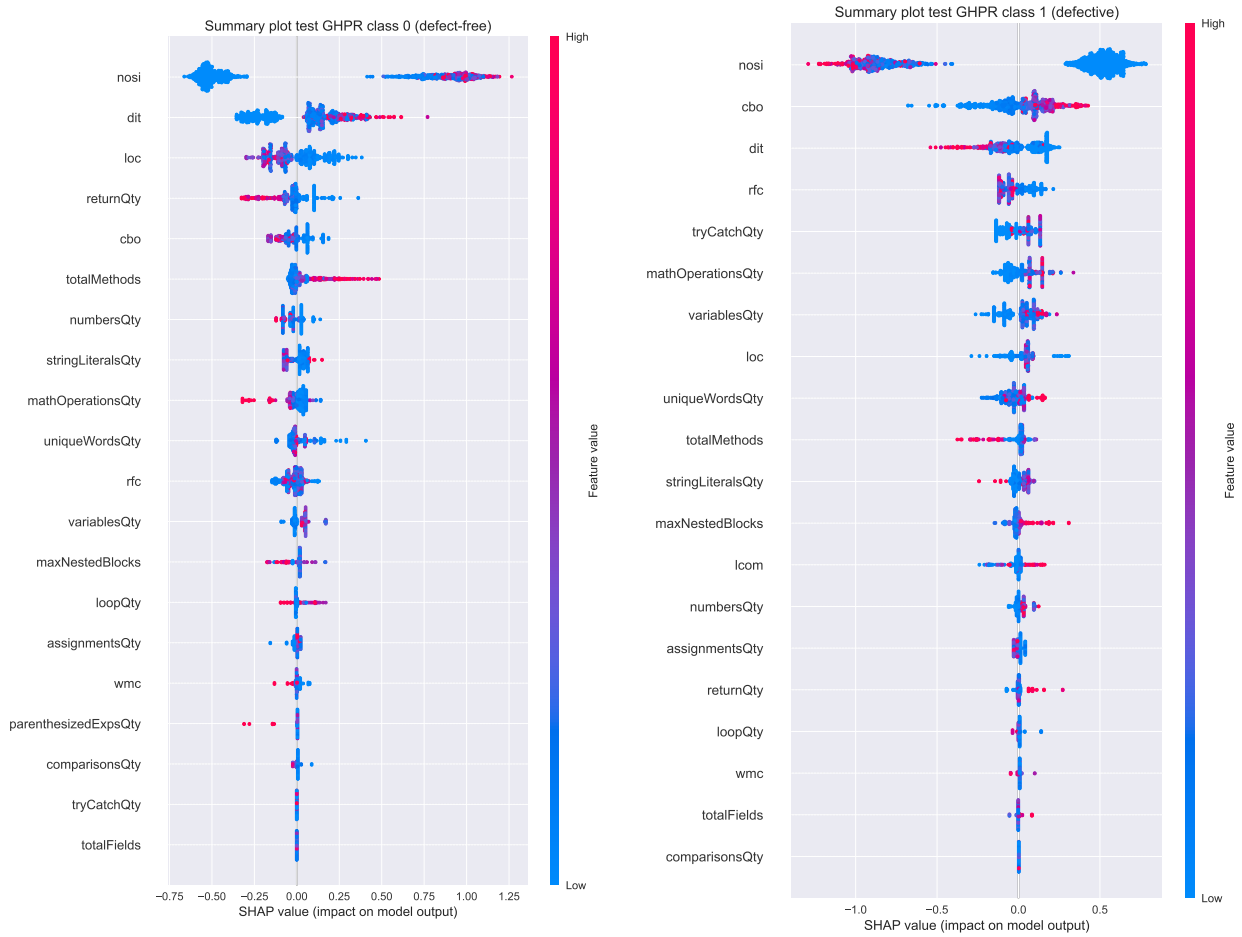

**Slika 6.14:** SHAP sažeti dijagram za klasu 0 (levo) i klasu 1 (defektni modul) za NASA skup podataka (desno).

U slučaju GHPR skupa podataka, uzet je najbolji generisani model, tačnije XGHARSA model sa stopom greške kao objektivnom funkcijom. Slika 6.15 prikazuje sažeti dijagram svih klasa i vodopadni dijagram za klasu 0 (modul bez defekata), dok Slika 6.16 pokazuje sažete dijagrame za klasu 0 (bez defekata) i klasu 1 (defektna).

Analiza SHAP-a za GHPR skup podataka pokazuje da su najvažnije karakteristike broj statičkih poziva (NOSI karakteristika), praćen brojem zavisnosti koje klasa sadrži (CBO - sprega između objekata), jedinstvena količina reči u izvornom fajlu, linije koda (LOC karakteristika), količina brojeva (numbersQty - pokazuje broj numeričkih literala, kao što su int, long, double i float) i dubina stabla nasleđivanja (DIT karakteristika). Kada se posmatra sažeti dijagram za klasu 1 (koji ukazuje na defektni modul), očigledno je da će posmatrani modul verovatnije imati defekte ako je broj statičkih poziva nizak, i ako je dubina stabla nasleđivanja niska. Karakteristike koje su u direktnoj korelaciji sa verovatnoćom defekata su broj zavisnosti, maksimalni broj ugnežđenih blokova, i metrika nedostatka kohezije metoda (lcom), kao i količina povratnih izjava. Sve ove metrike takođe ukazuju na to da je modul vrlo složen i da bi verovatno trebalo da bude refaktorisan, kako bi se smanjio rizik od defekata. Zanimljivo je da metrika ukupnog broja metoda ima negativnu korelaciju sa verovatnoćom da će modul imati defekte. Na prvi pogled, može zvučati čudno da ako modul ima više metoda, manje je verovatno da će imati defekte, ali zapravo, ovo se takođe očekuje. U ovom slučaju, veći broj metoda u modulu u većini scenarija takođe znači da su te metode jednostavnije i kraće (stoga su lakše za testiranje, i manje verovatno da će imati defekte), dok će, s druge strane, moduli sa malim brojem složenih metoda verovatnije imati greške. Direktan zaključak koji se može izvući iz ovog




**Slika 6.15:** SHAP sažeti dijagram (levo) i vodopadni dijagram za klasu 0 (bez defekata) za GHPR skup podataka (desno).




**Slika 6.16:** SHAP sažeti dijagrami za klasu 0 (levo) i klasu 1 (defektni modul) za GHPR skup podataka (desno).

posmatranja je da je najbolje da metode budu kratke i jednostavne, i ako je moguće, da se složene metode razbiju na nekoliko jednostavnijih.

Prema sprovedenoj SHAP analizi oba posmatrana skupa podataka (NASA i GHPR), mogu se izvući nekoliko opštih kombinovanih zaključaka. Glavne softverske metrike koje treba pratiti u pogledu predviđanja defekata, pored očigledne veličine modula (datih u linijama koda) su McCabeova ciklomatska složenost i esencijalna složenost, broj jedinstvenih operanada i ukupnih operanada u izvoru, i maksimalno ugnježdavanje blokova. Kada se posmatraju objektno orijentisani programi, nekoliko dodatnih parametara ukazuje na povećanu složenost modula, kao što su broj zavisnosti posmatrane klase, dubina stabla nasleđivanja i broj statičkih poziva. Kada se posmatra broj metoda posmatrane klase, ako je broj metoda nizak, to obično znači da su velike i vrlo složene, stoga je testiranje mnogo teže i sklonije su defektima. S druge strane, dobro napisan modul obično ima veći broj metoda koje su jednostavne i lako se testiraju i održavaju.

# Poglavlje 7

# Primena u nastavi

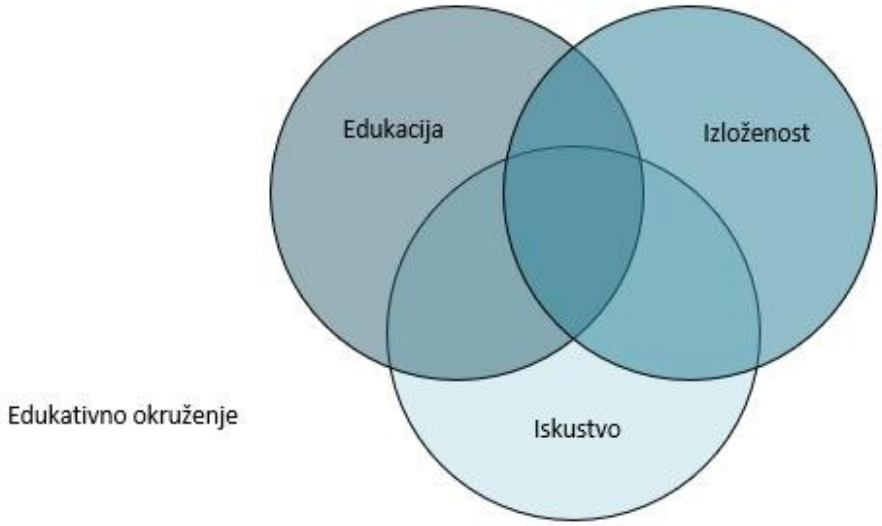
U ovom odeljku opisana je primena različitih simulatora u edukaciji, kao i detaljan pregled edukativnih okruženja koja se koriste u oblasti testiranja softvera. Analizirane su prednosti i mane postojećih rešenja. Na kraju, dat je predlog praktične laboratorijske vežbe u okviru koje studenti ispituju više modela mašinskog učenja na skupu podataka za predikciju defekata.

## 7.1 Primena simulatora u edukaciji

Testiranje softvera je integralni i ključni segment svih projekata razvoja softvera, i često čini razliku između uspeha i neuspeha projekta, ako se pravilno sprovede. Nažalost, zbog dinamične prirode modernih projekata i kratkih rokova, testiranje softvera najviše trpi u pogledu nedostatka vremena i smanjenja radnih sati na projektu. Ova loša praksa će neizbežno dovesti do kasnog otkrivanja defekata, obično od strane krajnjih korisnika kada je sistem već u proizvodnoj fazi. Zlatno pravilo testiranja softvera kaže da ako se greške otkriju kasno, njihovo popravljanje će biti skuplje. Još gore, to bi moglo značiti rizikovanje ljudskih života u nekim kritičnim domenima upotrebe, kao što su medicinski softver, sistemi za kontrolu vazdušnog saobraćaja i softver za avione.

Iako je testiranje softvera očigledno ključni deo razvoja softvera, studenti računarstva često završavaju studije sa vrlo malo znanja i bez praktičnih veština u ovoj oblasti, kako su primetili [107, 108]. U praksi, vešti testeri softvera moraju testirati različite vrste aplikacija i koristiti druge tehnike i pristupe testiranja, i da bi to uradili, oni su dužni da stalno uče [1].

Postoje dve glavne vrste tehnika testiranja softvera, funkcionalno testiranje (poznato i kao testiranje metodom crne kutije, testiranje metodom zatvorene kutije ili testiranje vođeno podacima) i strukturalno testiranje (poznato i kao testiranje metodom staklene kutije, testiranje metodom bele kutije, testiranje metodom prozirne kutije ili testiranje zasnovano na kodu) [109, 110, 111]. Funkcionalno testiranje vidi softver koji se testira kao zatvoreno okruženje, posmatra samo ulaze i izlaze u sistemu, bez razmatranja njegove implementacije. Ova vrsta testiranja se zasniva na zahtevima korisnika koji su obično navedeni u dokumentu sa specifikacijom softvera. Glavna svrha funkcionalnog testiranja je da dokaže da softver radi kako je opisano u dokumentu sa zahtevima. Funkcionalno testiranje se obično koristi u sistemskom testiranju i testiranju od strane krajnjih korisnika, a delimično se može koristiti i u integracionom testiranju, u delovima koji su implementirani. Testiranje metodom bele kutije se bavi kodom i podacima. Njegova glavna svrha je da dokaže da su svi delovi koda testirani i to se meri pokrivenošću koda. Ova tehnika




**Slika 7.1:** Osobine edukacionih okruženja.

se obično koristi u jediničnom i integracionom testiranju. Pravilno testiranje aplikacije bi uvek trebalo da kombinuje ove dve vrste testiranja. Treći tip, tehnika metodom sive kutije, se sve više pominje u modernoj literaturi [112, 113, 114, 115]. U testiranju metodom sive kutije, unutrašnja struktura aplikacije je delimično poznata.

Programeri softvera bi trebalo da budu sposobni da izvrše barem jedinično testiranje koda pre integracije tog koda sa drugim komponentama. Popravljanje grešaka na ovom nivou je najjeftinija opcija, pa je vrlo korisno da programer zna kako to da uradi. Ako se greška ne otkrije na ovom nivou, a i dalje se širi, njeno popravljanje može biti skupo.

Ključno je da akademski kursevi obezbede bolju obuku u oblasti testiranja softvera za testere i programere. Broj univerziteta računarstva koji uključuju kurseve testiranja softvera se povećava, ali studenti retko rešavaju praktične probleme [116, 117, 118, 119]. Izlaganje stvarnim problemima i upoznavanje studenata sa odgovarajućim pristupima testiranja i alatima su ključni (kao što je prikazano na slici 7.1), ali još mnogo je ostalo da se uradi u ovoj oblasti, prema autorima [120]. Više od deset godina kasnije, njihov zaključak je i dalje tačan. Budući da se testiranje softvera bavi apstraktnim konceptima, uloga simulatora na predavanjima je veoma važna - oni su alat koji može pružiti dodatna objašnjenja metoda i tema u predmetima testiranja softvera. Današnje testiranje softvera ima značajnu ulogu u industrijskim projektima, gde se cilja na uvođenje modernih pristupa kao što je razvoj vođen testiranjem (TDD) [121, 122] ili u kombinaciji prirodnih i programskih jezika gde možete pisati na prirodnom jeziku, a generišu se programski kod i test slučajevi [123].

Jedan od nedostataka tradicionalnih okruženja za učenje i simulatora je što često ne uspevaju da uspostave dovoljan nivo angažovanja studenata. Da bi se to rešilo, pristup učenju zasnovan na gejmifikaciji je široko korišćen u mnogim domenima obrazovanja u poslednje vreme. Ova vrsta okruženja se oslanja na igre koje stvaraju angažovanje, što je potrebno za uspešno iskustvo učenja [124, 125, 126].

Agilna metodologija, koja je trenutno najpopularniji model životnog ciklusa razvoja softvera, uspostavlja kolekciju praksi koje znatno utiču na način na koji se testira softver [127, 128]. Stoga bi trebalo da bude adekvatno predstavljena studentima, kako bi ih pripremila za stvarna radna okruženja koja često koriste neki oblik agilnog modela, kao što su ekstremno programiranje [129] ili Scrum [130]. Budući da se moderni projekti razvoja softvera fokusiraju na čestu isporuku softvera, koncepti kontinuirane integracije/kontinuirane isporuke (CI/CD) [131, 132] bi takođe trebalo da budu razmatrani unutar okruženja za učenje. Međutim, u stvarnosti, osnovni program testiranja softvera za studente na dodiplomskim studijama obično ili ne uključuje ove teme ili ih samo spominje bez ulaska u detalje, pošto je fokus na glavnim tehnikama testiranja softvera. Kao rezultat toga, studentima one ostaju nepoznate. Takođe, nastavni plan i program računarstva/softverskog inženjeringa obično organizuje kurseve o upravljanju softverskim projektima, agilnoj metodologiji i procesu razvoja softvera tek nakon osnovnog kursa testiranja softvera.

Ovo poglavlje daje sistematski pregled dostupnih okruženja za učenje koja se koriste na kursevima testiranja softvera na univerzitetima ili koje koriste profesionalci za obuku u radnom okruženju. Istraživanje najnovijih rešenja u ovoj oblasti sprovedeno je sa sledećim istraživačkim pitanjima na umu:

- Koji su nastavni alati korišćeni u domenu testiranja softvera?
- Koja je svrha i priroda svakog od tih pristupa?
- Koji su glavni principi dizajna i koje su najbolje prakse u ovoj oblasti?
- Da li ovi alati pokrivaju najvažnije teme testiranja softvera?

Ova sekcija je organizovana na sledeći način: odeljak 7.2 pruža uvid u opšte zahteve okruženja za učenje testiranja softvera. Pregled okruženja obuhvaćenih ovim istraživanjem dat je u odeljku 7.3. Odeljak 7.4 raspravlja o okruženjima uključenim u ovu komparativnu analizu. Odeljak 7.5 donosi konačni rezime i predlaže smernice za razvoj budućeg okruženja koje će pokušati da prevaziđe nedostatke postojećih analiziranih rešenja.

## 7.2 Postojeći primeri primene

Većina univerziteta prepoznaje testiranje softvera kao važnu temu koja pripada programima računarstva ili informacionih tehnologija, kako se vidi u [107, 108, 120, 133, 134]. Međutim, ovi i drugi istraživački radovi na temu ukazuju da u većini slučajeva predavanja o testiranju softvera ne pokrivaju materijale na odgovarajući način, s obzirom da se u praksi videlo da studenti nemaju adekvatne praktične veštine nakon diplomiranja. Jedan mogući koren ovog problema je to što je na nekim univerzitetima testiranje softvera izborni kurs. Rezultat toga je obično da mnogi softverski inženjeri i programeri izbegavaju odabir ovog kursa, ulaze na tržište rada sa malo ili nimalo obuke i bez poznavanja čak i najosnovnijih metoda testiranja, kako je navedeno u [135].

Istraživanja objavljena u [120] i [108] raspravljaju o činjenici da većina kurikuluma računarstva na univerzitetima širom sveta uključuje razne teme koje se fokusiraju na razvoj softvera, dok zanemaruju testiranje softvera i QA, koji dobijaju samo marginalnu pažnju. Štaviše, istraživanje objavljeno u [107] navodi da profesorima na univerzitetima nedostaju praktična znanja u domenu QA, i oni stoga nastavljaju da stavljaju fokus na kurseve razvoja softvera, žrtvujući teme o tehnikama testiranja softvera. Ovo je potpuno

suprotno modernom tržištu rada, jer čak i za programere softvera, jedan od zahteva je poznavanje osnovnih principa testiranja softvera, koji im mogu pomoći da poboljšaju svoje programersko stručno znanje i stvore pouzdanije aplikacije.

Computer Science Curricula 2013 [136], dokument sa smernicama objavljen od strane ACM i IEEE, preporučuje teme za kurseve računarskih nauka, a teme su označene kao obavezne ili izborne. Štaviše, obavezne teme su podeljene na nivo 1 (obavezne u potpunosti) i nivo 2 (obavezne teme koje studenti treba da pokriju sa najmanje 80 posto). Izborne teme su teme koje proširuju znanje studenata, ali nisu obavezne. Dokument stavlja kurs o testiranju softvera, Verifikacija i validacija softvera, pod oblast Softversko inženjerstvo kao obavezni kurs nivoa 2 sa preporučenih 4 sata predavanja nedeljno. Opšte teme za kurs Verifikacija i validacija softvera preporučene od strane ACM/IEEE su sledeće:

- principi testiranja softvera,
- modeli razvoja softvera,
- nivoi testiranja softvera i tipovi testiranja softvera,
- statičke tehnike testiranja,
- dizajn testova i upravljanje testovima,
- upravljanje defektima,
- ograničenja testiranja softvera u određenim oblastima.

Očekivani rezultati učenja su sledeći:

- Studenti bi trebalo da razumeju razlike između validacije softvera i verifikacije softvera.
- Studenti bi trebalo da budu upoznati sa ulogom alata za testiranje softvera u validaciji softvera.
- Bitno je da studenti shvate kako alati za testiranje softvera doprinose procesu validacije softvera.
- Studenti bi trebalo da steknu razumevanje ključnih termina u testiranju softvera, naročito kada je reč o nivoima i tipovima testiranja softvera. Takođe bi trebalo da budu u stanju da ih razlikuju.
- Studenti bi trebalo da budu upoznati sa metodama potrebnim za identifikaciju najvažnijih test slučajeva koji se koriste za različite tipove testiranja.
- Studenti bi trebalo da mogu da izaberu odgovarajuće test kandidate za setove regresionih testova i da budu sposobni da ih automatizuju.
- Studenti bi trebalo da budu u stanju da prate defekte koristeći odgovarajuće alate za menadžment defekata.
- Studenti bi trebalo da razumeju ograničenja testiranja u specifičnim domenima.

**Tabela 7.1:** Praktične teme obuhvaćene tipičnim osnovnim kursom za testiranje softvera

| Tema | Opis |
|---|---|
| Uvod | Uvod u testiranje softvera, važnost, osnovni termini, greška, kvar, defekt, test, nivoi |
| Crna kutija | Granične vrednosti, klase ekvivalencije, model stanja, grafikoni uzrok-posledica |
| Bela kutija | Pokrivenost iskaza, odluka, putanja, ciklomatska kompleksnost, testiranje toka poda |
| Mutaciono testiranje | Mutacija iskaza, vrednosti i odluka |
| Jedinično testiranje | Praktična implementacija crne kutije i pristupa bele kutije |
| Integraciono testiranje | Integracija odozgo nadole, integracija odozdo nagore, sendvič integracija |
| Sistemsko testiranje | Funkcionalno testiranje, nefunkcionalno testiranje (performanse, skalabilnost itd.) |
| Statičko testiranje | Formalni i neformalni pregledi, upotreba statičkih alata |
| Testiranje GUI | Validacija funkcija vidljivih krajnjim korisnicima |
| Upravljanje defektima | Izveštavanje, životni ciklus defekta, izveštaji o testiranju |

Da bi se postigli ovi rezultati, u praktičnom smislu, Tabela 7.1 sumira teme i relevantne podteme koje moraju biti pokrivene i objašnjene studentima. Ove teme su obavezne prema međunarodnom programu za sertifikaciju softverskih testera na osnovnom nivou (ISTQB) [137], i pokrivene su svim osnovnim kursevima za testiranje softvera.

Dodavanje dopunskih alata za učenje predavanjima o testiranju softvera, poput okruženja za saradnju ili igara, može pomoći profesorima u prenosu znanja i pružiti studentima neophodni praktični uvid u metode testiranja softvera primenjene na stvarne praktične probleme programiranja. Koncept nije nov, jer je o njemu raspravljano još 2001. godine. Autori [138] su primetili da bi studenti trebalo da budu mnogo više izloženi metodama testiranja softvera i da bi trebalo da ih vide u praksi tokom svojih postdiplomskih studija. Među uspešnim ranim primerima, okruženje koje se istaklo bilo je Bug Hunt, koje je predložio [139]. Bug Hunt se sastojao od četiri uvodne teme o testiranju softvera: osnovni termini, metode crne kutije, metode bele kutije i efikasnost skupa testova. Cilj ovog okruženja bio je da vodi studente kroz lekcije i pomogne im da razumeju osnovne koncepte. Još jedan uspešan rani alat pod nazivom MARMOSET predstavljen je u [140, 141]. MARMOSET je podsticao studente da pregledaju svoj kod pre nego što ga predaju alatu na testiranje. Nakon slanja, kod je testiran protiv javnog skupa testova, a ako je prošao, studenti su mogli da ga testiraju i sa tajnim skupom testova (koji sadrži testove koji studentima nisu unapred otkriveni). Nekoliko novijih projekata bilo je zasnovano na alatima za kolaborativno učenje. Na primer, [142] je predložio alat za kolaboraciju u dopunskoj obuci i praksi u oblasti testiranja softvera. Slično tome, u istraživanju objavljenom od strane [143], autori su predložili okruženje za sajber učenje koje bi moglo da se koristi paralelno sa tradicionalnim predavanjima o testiranju softvera.

Najnovija rešenja počinju da koriste elemente zasnovane na igrama kako bi se tehnike testiranja softvera približile mlađim studentima na mnogo prijatniji način. Osnovna ideja iza gejmifikacije je da materijale koji bi mogli biti dosadni i nezanimljivi učini mnogo zanimljivijim. Veoma uspešan primer je igra Code Defenders, koju su predložili [144, 145], gde su autori identifikovali testiranje softvera kao jednu od najvažnijih veština potrebnih za dobrog programera, i pokušali da povećaju angažovanost studenata tokom procesa učenja. Samu igru su implementirali kao igru za dva igrača, gde studenti mogu da igraju jedan protiv drugog napadajući/braneći program dodavanjem grešaka i test slučajeva koji će otkriti te greške, i uče mutaciono testiranje na zabavan način. Još jedno okruženje zasnovano na igrama predložio je [146], gde studenti mogu da usvoje pristupe koji se koriste u funkcionalnom testiranju (crna kutija), strukturalnom (bela kutija) i mutacionom testiranju, učestvujući u 2D akcionoj igri. Mutaciono testiranje je specifičan podtip strukturalnog

testiranja [147, 148, 149]. Zasniva se na promeni operatora (na jednom ili više mesta) u izvornom kodu kako bi se dobili mutirani programi (mutanti). Mutirani programi generišu greške i obezbeđuju da skup testova može da detektuje promene. Najnoviji pristup zasnovan na gejmifikaciji opisan je u [150], gde su autori sproveli kontrolisani eksperiment da bi uporedili nivo angažovanja i postignuća dve grupe studenata na kursu za testiranje softvera. Zaključili su da su studenti koji su pohađali gejmifikovanu verziju kursa imali bolje rezultate, i primetili su da je ključ uspeha dobro dizajnirano gejmifikovano iskustvo.

U pogledu alata za testiranje softvera i njihove primene u obrazovnim institucijama, opsežan pregled literature predstavljen je u [151]. Rad [151] je raspravljao o trendovima i pružio listu od trideset objavljenih radova iz ove oblasti, dobijenih preko Google Scholar u vremenskom okviru 2013-2017, ali nije dao nikakve detalje o bilo kojoj od navedenih publikacija. Drugo istraživanje objavljeno u [152] bavilo se učenjem zasnovanim na igrama u testiranju softvera, međutim, ovaj rad je samo ukratko pomenuo nekoliko pristupa zasnovanih na igrama, bez pružanja dodatnih detalja. Pregledni rad [153] uporedio je pet alata i okruženja za učenje koji se koriste u procesu nastave testiranja softvera, zaključujući da skorašnji trendovi u ovoj oblasti idu u pravcu gejmifikacije. Prema našim saznanjima, to su bili jedini pregledni radovi objavljeni relativno skoro u ovoj oblasti. Nijedan od njih nije pokušao da pruži opsežan pregled alata sa detaljnim opisima, da ih uporedi i detaljno prodiskutuje.

Sprovedeni iscrpni skorašnji pregled literature na ovu temu ukazuje na to da postoji vrlo ograničen broj alata i okruženja koji bi mogli biti iskorišćeni kao dodatak kursevima za testiranje softvera. Ovo je još očiglednije kada se uporedi sa popularnijim oblastima softverskog inženjeringa. Postoji desetine okruženja [154, 155, 156, 157, 158, 159] koja mogu pomoći u nastavi arhitekture i organizacije računara, programiranja, nauke o podacima, bežičnih senzorskih mreža, ugrađenih sistema, itd. Autor ovog pregleda pokušao je da izvrši iscrpan pregled alata koji su korišćeni kao podrška na predavanjima o testiranju softvera i QA. Alati su pretraživani na Google Scholar, u vremenskom okviru 2003-2023 (poslednjih dvadeset godina), koristeći sledeće ključne reči (u različitim kombinacijama): software testing, computer science, teaching environment, education, learning, simulator, tool, gamification, engagement, survey, trends (testiranje softvera, računarska nauka, okruženje za učenje, obrazovanje, učenje, simulator, alat, gejmifikacija, angažovanje, pregled, trendovi).

Konačno, šesnaest pristupa je odabrano na osnovu sledećeg kriterijuma: alati koji su razmatrani u ovom pregledu su razvijeni, primenjeni i korišćeni u studijama na univerzitetskom nivou najmanje jedan semestar kao dodatni deo predavanja o testiranju softvera. Drugim rečima, nisu uzeti u obzir nedovršeni prototipovi i ideje koje nikada nisu praktično iskorišćene kao podrška predavanjima. Izabrani pristupi su takođe ocenjeni u pogledu relevantnih tema koje su pokrivene, kako je navedeno u Tabeli 7.1. Glavni cilj ovog istraživanja je da pokaže uspešne pristupe i koncepte koji su se pokazali efikasnim u praksi, i koji mogu biti iskorišćeni kao solidna polazna tačka za budući rad. Pouke naučene iz ovih šesnaest pristupa mogu se koristiti kao inspiracija za implementaciju novih okruženja koja bi bila izgrađena na dobrim konceptima i prevazišla primećene nedostatke u bliskoj budućnosti.

## 7.3 Pregled edukativnih okruženja u oblasti testiranja softvera

Tradicionalni pristup u nastavi testiranja softvera zasniva se uglavnom na teorijskim predavanjima koja se održavaju u učionicama, gde profesor ima vodeću ulogu, a studenti pasivno slušaju i zapravo su samo primaoci informacija. Profesori dele znanje, a studenti ga primaju, u procesu koji se može ukratko objasniti na sledeći način: studenti imaju cilj da zapamte pružene materijale za kurs i reprodukuju ih kasnije tokom ispita (usmenog ili pismenog) bez dopunske literature. Očigledan problem sa ovom metodom je taj što studenti mogu lako postati pasivni i nemotivisani, sa jedinim ciljem da zapamte sve informacije potrebne za uspešno polaganje ispita. Na kraju, neće imati dublje znanje ili praksu u vezi određene teme [160].

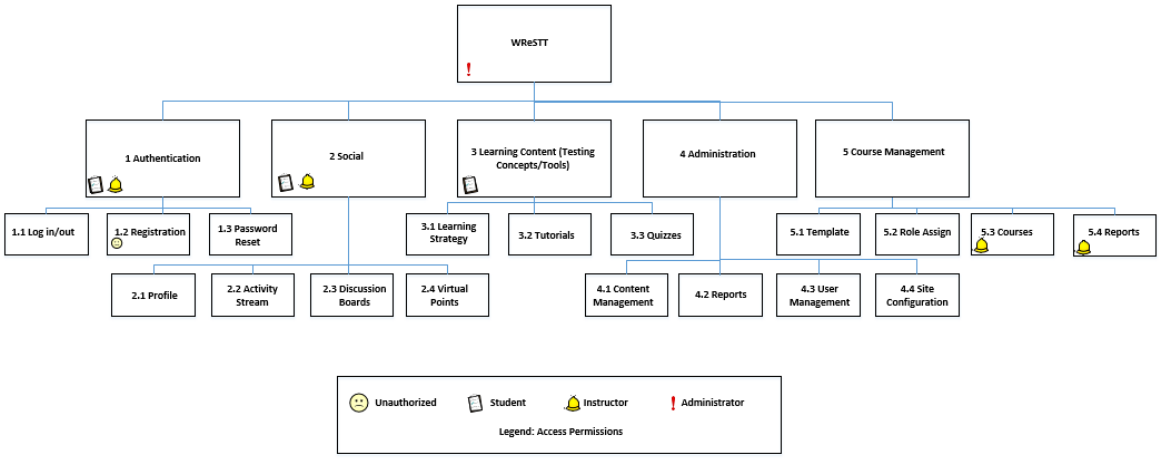
Da bi se poboljšao proces učenja, moderna predavanja treba da uključe inovacije kako bi se studenti podstakli da se više angažuju i razviju svoje umove. Ovaj savremeni pristup ima za cilj da studentima omogući sticanje znanja i praktičnog iskustva koje će im biti korisno i primenljivo nakon završetka studija. U ovom scenariju, profesori i dalje vode nastavni proces. Međutim, studenti sada aktivno učestvuju na predavanjima i uče tako što su izloženi praktičnim problemima. Ovo će više motivisati studente i održati njihovo angažovanje na visokom nivou, što rezultira usvojenim znanjem i dubokim razumevanjem materijala, koji će biti primenljivi kada završe studije i zaposle se [157].

Imajući sve ovo u vidu, ovaj odeljak daje opsežan pregled različitih alata za učenje i kolaborativnih okruženja koja bi mogla biti iskorišćena kao pomoć profesorima u nastavi testiranja softvera i obezbeđenja kvaliteta. Kao što će ovaj pregled pokazati, može se zaključiti da je najbolji pristup za motivisanje studenata i povećanje njihovog angažovanja kombinacija tradicionalnih predavanja u učionici sa alatima zasnovanim na igrama koji će povećati takmičarski duh i nivo zabave među studentima. Ovaj zaključak se zasniva na prethodno objavljenim radovima koji predstavljaju rezultate i dokaze da profesori računarskih nauka treba da razmotre dodavanje dopunskih materijala za učenje zasnovanih na igrama svojim predavanjima, kako je primećeno u [161].

Svih šesnaest posmatranih okruženja opisano je prema principu 7w, koji odgovara na sledeća pitanja - Zašto, Ko, Šta, Kako, od strane Koga, Kada i Gde, i Kako je prošlo (Why, Who, What, How, by Whom, When & Where, and How it Went). Ovaj princip je omogućio korišćenje uniformnog pristupa u daljoj komparativnoj analizi.

### 7.3.1 WReSTT-CyLE/STEM-CyLE

WReSTT (Web-Based Repository of Software Testing Tools) platforma za sajber učenje, koju je predložio [108], na početku je korišćena za hostovanje različitih tutorijala za različite alate za testiranje softvera, sa linkovima ka dopunskim materijalima za čitanje o testiranju softvera. Tokom narednih nekoliko godina, ona je prerasla u veliko okruženje za kolaborativno učenje, koje je ponudilo značajan broj tutorijala, dopunskih materijala, alata za testiranje, itd. Početna verzija okruženja hostovala je jednostavne alate koji su mogli da izračunaju pokrivenost koda za posmatrani program, izvrše statičku analizu koda i izračunaju različite metrike, alate za automatizaciju i izvršavanje testova, i dodatne veb dodatke za omogućavanje emulacije aktivnosti korisnika unutar veb pregledača. Inicijalno, alat je podržavao C++ i Java jezike, testiranje jedinica, i sistemsko testiranje preko pruženog veb GUI-ja. Kasnije, kako je već pomenuto, okruženje se razvilo u složenu arhitekturu na četiri nivoa. Poboljšana verzija WReSTT-a uključivala je autentifikaciju




**Slika 7.2:** WReSTT arhitektura [108].

korisnika, funkcije društvene mreže, upravljanje kursevima (za profesore, za otpremanje predavanja i dodatnih fajlova, zadatke za domaći rad, itd.), a arhitektura sistema je prikazana na slici 7.2.

Funkcionalnost saradnje dodata je u okruženje na osnovu povratnih informacija koje su dali studenti. Takođe, implementirani su i koncepti zasnovani na igrama. Konačna verzija okruženja je implementirala sistem nagrada kroz virtuelne poene koji bi bili dodeljeni korisnicima kada njihov tim završi zadatak. Ovaj koncept je podsticao timski rad, konkurentski duh i generalno drastično povećao nivo angažovanja. Zanimljiva funkcija je uključivanje koncepta društvenih mreža, kroz diskusione forume i forume sa različitim temama, aktivnostima i strimovanjem, podrškom za komentare, itd. Za socijalnu aktivnost dodeljivani su virtuelni novčići.

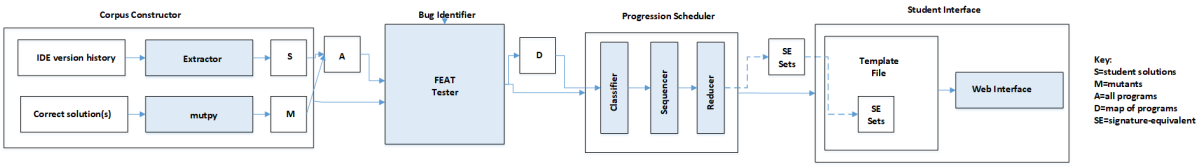
Na kraju, okruženje hostuje razne tutorijale o različitim metodama testiranja i alatima za automatizaciju, i kvizove gde studenti mogu da testiraju svoje razumevanje tema. Završetak i tutorijala i kvizova takođe se nagrađuje virtuelnim novčićima. Rezultat studenata u virtuelnim novčićima učestvuje u ukupnoj oceni kursa. Studenti mogu da pregledaju teme i materijale za kurs, prate vesti za odabrane aktivnosti, komuniciraju sa drugim članovima tima, i tako dalje.

Okruženje WReSTT je razvijeno i praktično upotrebljeno na Florida International University (FIU) 2014. godine. Ovo okruženje je kasnije evoluiralo u STEM-CyLE, i danas se još uvek koristi na najmanje pet drugih institucija.

### 7.3.2 Automatizovani sistem za interaktivno učenje testiranja softvera

Autori u [162] predložili su interaktivno okruženje za učenje testiranja softvera, sa ciljem da podrže i pomognu studentima u učenju kako da kreiraju bolje test slučajeve. Kada studenti koriste ovo okruženje, dobijaju listu programskih zahteva, i od njih se traži da kreiraju testove, sastave skup testova i pošalju ga u okruženje (studentima nije potrebno da implementiraju program, samo su dužni da ga adekvatno testiraju). Nakon što je skup testova poslat, studenti će dobiti povratne informacije od okruženja, i ako je potrebno, mogu poboljšati svoj skup testova iterativno i ponovo ga poslati.

Osnovna hipoteza iza ovog okruženja bila je da studenti ne bi trebalo da testiraju




**Slika 7.3:** Arhitektura automatizovanog sistema za interaktivno učenje [162].

svoje programe, jer ne bi bili objektivni zbog pristrasnosti i nedostajalo bi im motivacije da pronađu defekte, što bi moglo da implicira da nisu u stanju da implementiraju kod koji bi se adekvatno izvršavao. Međutim, ako bi studentima bio dat program koji je napisao neko drugi, mogli bi da održe svoju objektivnost (pošto nisu lično uključeni) i bili bi veoma motivisani da otkriju što više defekata u datom programu. Šire gledano, to može drastično povećati takmičarski duh [162].

Okruženje je koristilo napredne metode testiranja softvera, uključujući mutaciono testiranje i testiranje toka podataka, koje su bile mnogo jače od osnovnih tehnika pokrivenosti koda (pokrivenost izjavama i granama) i mogu lako da otkriju slabe skupove testova. Studenti odmah dobijaju preciznu procenu poslatih skupova testova. Okruženje će takođe predstaviti studentima primere grešaka koje nisu otkrivene njihovim skupom testova. Ovo je ključno za neiskusne studente koji su upisani na uvodni kurs testiranja softvera, jer na početku imaju problema sa identifikacijom svih graničnih vrednosti i graničnih scenarija. Međutim, ako alat pruža povratne informacije koje sadrže jasne primere koji pokazuju nedostatke poslatih skupova testova, studenti mogu da ih analiziraju i otkriju greške koje nisu otkrivene njihovim testovima. Kasnije, mogu da poboljšaju svoje skupove testova, da ih ponovo pošalju i postignu bolje rezultate. Još jedna prednost ovog pristupa je da će studenti naučiti da razumeju i testiraju tuđi kod. Takođe, studenti bi mogli da primene različite strategije za rešavanje istog zadatka i poboljšaju svoje veštine pisanja testova.

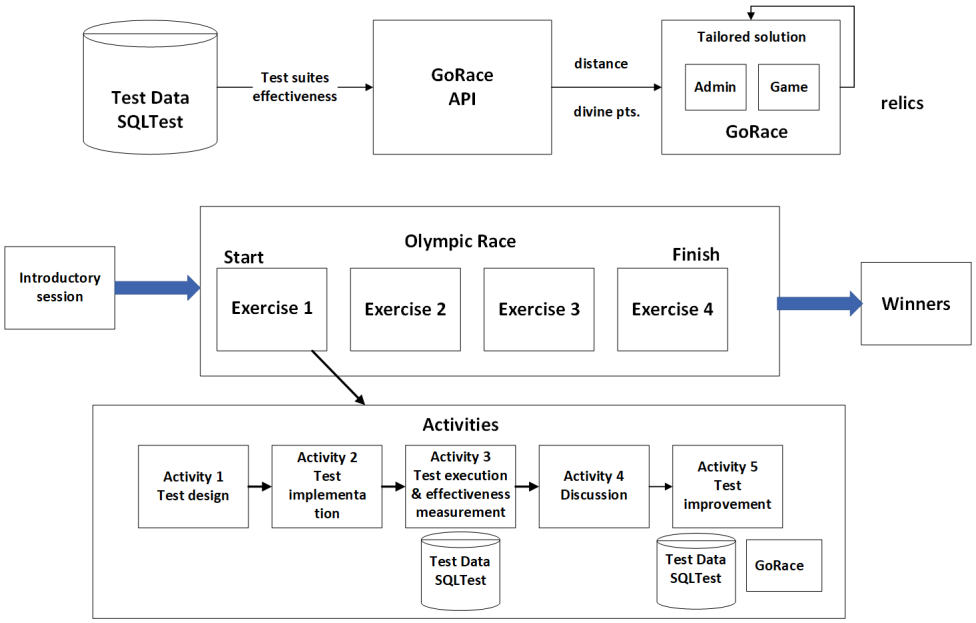
Prvi modul, Corpus Constructor, korišćen je za kreiranje korpusa uzoraka koda. Implementacije koda su izvučene iz onlajn IDE CodeSkulptor [163] (`https://py2.codeskulptor.org/`) koji su koristili drugi studenti dok su implementirali svoje programe za druge kurseve računarskih nauka. Nakon što je kreirana baza uzoraka koda, alat za mutaciono testiranje Mutpy [164] uveo je neke implementacije koje sadrže greške. Drugi modul, nazvan Bug Identifier, koristi se za identifikaciju jedinstvenih test potpisa, i takođe je odgovoran za izbor programa koji bi trebalo da budu vraćeni u povratnim informacijama. Ovaj modul je zasnovan na postojećem alatu FEAT [165] koji je proširen za ovu svrhu. Na kraju, studenti koriste vrlo jednostavan veb interfejs, gde moraju okačiti fajlove za određeni zadatak. Pregled ovog alata je dat na slici 7.3.

Ovaj alat je korišćen na Rice University, Houston, Texas, tokom jesenjeg semestra 2016. godine. Koristilo ga je skoro 150 studenata upisanih na uvodni kurs računarskih nauka koji je koristio Python kao programski jezik. Studija je zaključila da su upisani studenti poboljšali svoje veštine testiranja.

### 7.3.3 SQLTest-GoRace

Najnoviji pristup zasnovan na gejmifikaciji istražili su autori u [150], koji su sproveli eksperiment sa SQLTest [166] i GoRace [167] aplikacijama, koje su integrisane da bi studentima pružile iskustvo zasnovano na igri tokom časova testiranja softvera. Studenti su koristili integrisani alat za izvođenje aktivnosti učenja kroz četiri dizajnirane vežbe.

Prva komponenta sistema, alat SQLTest, razvijen je na Univerzitetu u Oviedu, u




**Slika 7.4:** Arhitektura SQLTest-GoRace sistema [150].

Španiji, i omogućava studentima da izvršavaju svoje setove testova i određuju njihovu efikasnost u odnosu na programe koji su unapred učitani od strane profesora. Kao što ime sugeriše, alat je prvobitno korišćen za izvršavanje SQL skripti, ali je tokom narednih godina proširen da omogući testiranje različitih programa. Alat čuva implementaciju programa i mutante koji zahtevaju testiranje. Student je dužan da dostavi set testova, a alat interno procenjuje testove u odnosu na originalnu implementaciju programa kao i sve mutante. Alat određuje efikasnost testova u smislu procenta otkrivenih defekata u odnosu na ukupan broj defekata koji se mogu otkriti (drugim rečima, broj mutanata). Povratne informacije date studentu takođe uključuju opise defekata koje set testova nije otkrio.

GoRace, koji je druga komponenta predloženog sistema, je alat za igru zasnovanu na višekontekstualnoj naraciji koji je implementirala istraživačka grupa na Univerzitetu u Kadizu, u Španiji. Ovaj alat omogućava korisnicima da automatski generišu veb rešenje za omogućavanje gejmifikacije bilo kog alata treće strane koji se može povezati preko REST API-ja. Celokupno iskustvo GoRace-a postavljeno je kao virtuelna sredina u stilu grčke mitologije, gde učesnici učestvuju u olimpijskoj trci. Dok trče do linije cilja, učesnici sakupljaju virtuelne poene (božanske poene) duž puta, koji im omogućavaju da kupe relikvije koje kasnije mogu koristiti u pogodnom trenutku.

Ova dva gorepomenuta alata su zajedno integrisana, inkorporiranjem strategije zasnovane na igri GoRace, gde se udaljenost koju svaki student pređe u trci i broj božanskih poena izračunava u odnosu na efikasnost njihovih setova testova određenih alatom SQLTest. Dva alata su integrisana kroz GoRace API, koji SQLTest koristi za slanje rezultata koje je svaki student postigao. Nakon toga, GoRace određuje udaljenost i božanske poene koje će dodeliti studentu na osnovu rezultata primljenih od SQLTest-a i ažurira podatke prikazane na grafičkom korisničkom interfejsu. Celokupan radni tok je prikazan na slici 7.4, gde gornji deo prikazuje celokupni arhitektonski koncept, dok donji deo prikazuje koncept implementiranih praktičnih vežbi [150].

Praktične vežbe su se sastojale od četiri aplikacije iz stvarnog života koje su obezbedili nastavnici, gde su studenti bili dužni da primene tehnike naučene na predavanjima. Studentima je dat zadatak da postave i implementiraju test slučajeve, postave okruženje

i izvrše testove. Četiri praktična primera uključivala su problem klasifikacije trougla, onlajn prodavnicu, španski povrat PDV-a i planer rezervacija za laboratoriju. U svaki od posmatranih programa ubačeno je 10-20 defekata. Svaki put kada se set testova izvršavao (studenti su mogli da poboljšaju svoje test slučajeve na osnovu povratnih informacija koje je pružio alat SQLTest kao domaći zadatak), GoRace bi ažurirao poziciju studenta unutar olimpijske trke i izračunao božanske poene. Učesnici bi potom mogli da provere rangiranje u trci, kupe relikvije ili vrše interakciju sa drugim igračima (da ih napadaju ili da se štite).

Alat SQLTest-GoRace je korišćen na Univerzitetu u Oviedu, tokom akademske 2020-2021. godine, sa 135 studenata koji su bili upisani na kurs testiranja softvera. Performanse studenata i nivo angažovanja su upoređeni sa kontrolnom grupom od 100 studenata upisanih na isti kurs godinu dana ranije, u akademskoj 2019-2020. godini, kada pristup zasnovan na igri nije korišćen, a predavanja (zajedno sa istim vežbama) su održavana tradicionalno. Studija [150] je zaključila da je tokom početnih vežbi, nivo angažovanja u eksperimentalnoj grupi bio veći u pogledu stope napuštanja i učešća u aktivnostima. Međutim, primećeno je da je angažovanje pratilo silazni trend tokom kasnijih vežbi. Ipak, performanse studenata su se poboljšale jer je eksperimentalna grupa imala bolje rezultate za svaku vežbu. Studija je takođe zaključila da gejmifikacija može biti vrlo korisna, međutim, stimulativna nagrada mora biti raspoređena tokom celog kursa, kako bi se angažovanje održalo na visokom nivou do samog kraja.

### 7.3.4 Code Defenders

Code Defenders je veb igra za pretraživače, predstavljena od strane [144] 2016. godine, i razvijena da bi učenje mutacionog testiranja bilo zabavnije i prijatnije. Igra je postavljena oko dva igrača (studenta) koji se bore oko jednog programa. Jedan igrač preuzima ulogu napadača, koji stvara mutante, dok je drugi igrač branilac, koji implementira najbolji set testova da bi izložio mutante i branio program koji se testira. Napadač dobija poene za svakog mutanta koji nije otkriven od strane seta testova branitelja, dok branilac dobija poene za test slučajeve koji su ubili mutante koje je dodao napadač. Na kraju, pobednik igre je učesnik sa više poena. Autori u [144] su diskutovali da će oba igrača poboljšati svoje veštine testiranja i da će programer koji je napisao kod koji se koristi u igri imati koristi od dobrih setova testova i lukavih mutanata.

Cilj dizajna ove igre je jasan - pokušati da mutaciono testiranje učini što je moguće zabavnijim, jer se smatra jednom od najdosadnijih i najteže razumljivih tema u domenu testiranja softvera, a studenti obično imaju poteškoće da je usvoje. Igra koristi Java kod uparen sa JUnit testovima. Koncepti igre Code Defenders imaju jasnu ulogu u pomoći studentima da budu motivisaniji i angažovaniji, učeći teške i dosadne teme kroz praktične primere ugrađene u igru, koja će zabaviti studente i podstaći njihov takmičarski duh u procesu. Code Defenders je implementiran kao veb igra za dva igrača na poteze. Budući da je izgrađen oko Java koda i JUnit okvira za testiranje, igra se sastoji od napadača koji cilja da napadne program zasnovan na Javi, a branilac pokušava da ga odbrani stvaranjem JUnit test slučajeva. Implementacije korisničkog interfejsa za obe uloge su različite za napadače i branioca. Napadač prvi poteze, postavljajući mutanta (grešku u kodu koji se testira), koji će pokušati da iskoristi bilo koju slabost i sposobnost otkrivanja defekata seta testova. Nakon što napadač završi svoj potez stvaranjem mutanta, branilac može pokušati da odbrani program dodavanjem jediničnog testa da bi poboljšao postojeći set testova i izložio mutanta.

Igra Code Defenders je prvobitno korišćena na Univerzitetu u Šefildu, od 2016. godine, a kasnije na Univerzitetu u Pasauu, od zimskog semestra 2017-2018. godine na diplomskim i osnovnim studijama, sa studentima koji imaju različite nivoe iskustva u programiranju. Ideja koju je predložila igra CodeDefenders postala je vrlo popularna među studentima, koji su izvestili da su imali povećanu motivaciju za učenje teorije mutacionog testiranja igrajući zanimljivu i prijatnu igru, i preferirali su ovaj pristup u odnosu na tradicionalne metode. Na kraju igre studenti će završiti sa moćnim setovima testova i finim setovima mutanata za programe koji se koriste u igri, poboljšavajući svoje veštine testiranja u procesu. Igra se može igrati onlajn, na `https://code-defenders.org/`, što je omogućilo da se koristi na više drugih univerziteta. Kod igre je otvoren i takođe dostupan i može se naći na `https://github.com/CodeDefenders/CodeDefenders`.

### 7.3.5 Testing game

Još jedan pristup zasnovan na igri predstavili su Valle i saradnici [146] 2017. godine. Ovo istraživanje je predložilo rešenje Testing game, gde su autori pokušali da implementiraju edukativnu igru sa ciljem da je koriste kao dopunu uobičajenim predavanjima iz testiranja softvera. Studentima je dato interaktivno okruženje gde su mogli da nauče najvažnije tehnike testiranja softvera, podeljene na tri različita nivoa igre (metoda crne kutije, metoda bele kutije i mutaciono testiranje).

Testing game igra je razvijena kao arkadna veb igra sa lepom 2D grafikom, sa ciljnom populacijom koju čine studenti osnovnih studija koji su upisani na kurs testiranja softvera. Igra se može koristiti kao dodatni sadržaj standardnim predavanjima i časovima u učionici, omogućavajući studentima koji prate predavanja da poboljšaju svoje veštine i steknu praktična znanja. Igra pokriva glavne metode testiranja koje se obično koriste u praktičnom testiranju softvera. Prvi nivo igre se bavi metodama crne kutije, gde studenti uče kako da analiziraju specifikaciju zahteva za program i primenjuju metode testiranja klasa ekvivalencije i graničnih vrednosti. Drugi nivo pokriva metode bele kutije i počinje sa stvaranjem grafa kontrolnog toka (CFG) i primenom metoda pokrivenosti koda kao što su pokrivenost izjavama i odlukama, zajedno sa pristupom protoku podataka. Konačni, treći nivo igre se bavi metodom testiranja zasnovanom na defektima i daje uvod u mutaciono testiranje.

Igra predstavlja igrača sa avatarom, sa kolekcijom veština i sposobnosti sličnih standardnim 2D igrama. Avatar može koristiti hodanje, trčanje, skakanje, izbegavanje ili borbu protiv neprijatelja. Gorepomenuta tri nivoa igre pokrivaju tri najvažnije teme u testiranju softvera, i svaka tema je označena vratima. Da bi otvorio vrata, igrač mora da završi zadatke i dobije ključeve dostupne u igri. Svaki nivo je podeljen na nekoliko sekcija, a svaka sekcija može da se igra nakon što igrač uspešno završi prethodne sekcije.

Stvarna igra se sastoji od dva nivoa, jedan od njih je nivo testiranja crne kutije (prvi nivo u igri). Ova sekcija zahteva od studenta da analizira specifikaciju zahteva metode koja implementira algoritam sortiranja mehurićima, navodeći da samo nizovi dužine četiri mogu biti obrađeni od strane metode. Student ima cilj da eliminiše neprijatelje i nizove neprihvatljive dužine.

Na drugom nivou, odnosno nivou testiranja bele kutije, igrač dobija pristup kodu i strukturi posmatranog bubble sort algoritma koji se testira. Primarni zadatak za igrača je da locira CFG koji odgovara posmatranom kodu, boreći se protiv neprijatelja i uklanjajući CFG-ove koji nisu validni. Ostale sekcije imaju različite opcije igre gde igrač mora da reši problem testiranja toka podataka, posebno da popuni tabelu Def-Use.

Igra je prvobitno predstavljena na Univerzitetu u Sao Paolu (USP), na Laboratoriji za softversko inženjerstvo Instituta za matematičke i računarske nauke 2016. godine. Nakon implementacije igre, autori Testing game-a su završili studiju izvodljivosti objavljenu u [146] . Na osnovu primljenih povratnih informacija, zaključili su da su studenti uživali u igri i ocenili njen kvalitet kao dobar. Studenti su izjavili da je igra povećala njihov nivo motivacije i konkurentski duh, što je povezano sa zabavnom stranom učenja igrajući igru. Kao nekoliko nedostataka, studenti su istakli suviše jednostavne opcije navigacije i tutorijal koji predstavlja kontrole igre igraču. Takođe, multiplayer mod nije bio implementiran u prvoj verziji igre, a studenti su smatrali da bi moglo biti mnogo zanimljivije da mogu da igraju jedni protiv drugih. Prema dostupnim informacijama, igra se i dalje koristi na USP-u.

### 7.3.6 Eksperiment sa kartama u nastavi testiranja softvera

Metoda zasnovana na saradnji je korišćena za kreiranje igre sa kartama, predložene u [168]. Ovaj pristup je pokazao da nije neophodno implementirati elektronsku video igru kako bi se motivisali studenti, povećala njihova angažovanost i podstakla saradnja među njima. Tokom igranja, studenti takođe dele svoje znanje među sobom. Glavna motivacija za stvaranje ovog modela bila je integracija igre u tradicionalna predavanja, sa ciljem da se poboljša nastavni proces podsticanjem studenata da učestvuju i dele svoja iskustva i znanje, a da pri tome razvijaju takmičarski duh. Iskustvo slično igri bi ih motivisalo da nastave da igraju više puta kako bi usvojili nove veštine i dalje unapredili svoju stručnost. Predavač bi mogao da postavi težinu igre tako da prati trenutne nivoe znanja studenata, sa opcijom kasnije modifikacije u toku kursa, a na osnovu stvarnog napretka studenata.

Igra se zasniva na teorijskoj literaturi koju je odredio Međunarodni odbor za kvalifikacije u testiranju softvera (ISTQB). ISTQB predstavlja svetsku mrežu sertifikata specijalizovanu za testere softvera i inženjere za kontrolu kvaliteta, podeljenih u različite domene testiranja (tester mobilnih aplikacija, agilni tester, itd.) i nivoe stručnosti (osnovni nivo, napredni nivo, specijalista, itd.). Početni sertifikat, koji je često preduslov za sve napredne nivoe testiranja, naziva se tester softvera – osnovni nivo [137], često je zahtev za poslove testiranja softvera, i očekuje se da bi svi ozbiljni testeri softvera trebali da ga polože. Ovaj sertifikat osnovnog nivoa je inspirisao svaku aktivnost implementiranu u igri. Program ovog sertifikata na početnom nivou može da se primeni na studente upisane na osnovne kurseve testiranja softvera. Prateći program studija za ovaj sertifikat, sadržaj igre uključuje sledeće teme: osnove testiranja softvera, životni ciklus testiranja softvera, dizajn i implementaciju testa, statičko testiranje, upravljanje testovima i primenu alata za testiranje.

Ciljevi učenja igre su takođe definisani prema ciljevima u programu ISTQB-a. Ciljevi su podeljeni u četiri grupe:

- Prva grupa: očekuje se da studenti zapamte i prepoznaju osnovne termine kasnije.
- Druga grupa: zahteva razumevanje, sposobnost objašnjavanja i vršenja poređenja, davanje primera, klasifikaciju i kategorizaciju.
- Treća grupa: razmatra primenu usvojenih tehnika i sposobnost njihove upotrebe u praktičnom radu.
- Četvrta grupa: zahteva od studenata da budu u stanju da vrše inspekciju koda.

Većina karata u igri je inspirisana ovim četiri grupe.

Pošto mnogi studenti vole da igraju popularne igre sa kartama u slobodno vreme, dizajn karata je bio osmišljen tako da bude po njihovom ukusu (da podseća na igru Magic: The Gathering). Svaka pojedinačna karta se sastoji od nekoliko elemenata. Na primer, labela 2 sadrži podatak o tome da li je karta za jednog ili više igrača. Labela 3 daje dodatne informacije potrebne za rešavanje zadatka. Labela 5 pokazuje očekivani nivo znanja u ISTQB sistemu, dok labela 7 prikazuje problem koji treba rešiti. Labela 8 označava poene koje će igrač dobiti ako se zadatak završi. Rešenja za probleme navedene na kartama nalaze se na poleđini samih karata. Da bi se sprečilo varanje (u smislu da igrač može pročitati rešenje pre davanja odgovora), rešenje je šifrovano i vidljivo je samo ako se posmatra kroz poseban crveni film.

Igra sa kartama se igra na sledeći način. Moguće je igrati režimu sa jednim ili više igrača, gde je maksimalni broj igrača ograničen na 4. Pre početka igre, špil karata se promeša i svaki igrač uzima četiri karte. Pri svakom potezu, igrač izvlači kartu sa vrha špila. Nakon toga, igrač mora da izabere jednu od karata u ruci i da odigra sa njom tako što je stavlja na sto i čita zadatak naglas. Igrač zatim ima mogućnost da sam kaže rešenje ili da izabere saveznika među ostalim učesnicima u igri. Nakon što je rešenje pruženo, cela grupa odlučuje da li je prihvatljivo (igrač će dobiti poene navedene na karti) ili ne (ne dodeljuju se poeni). Pobednik igre je prvi igrač koji dostigne 30 poena.

Igra je uključena u nastavu softverskog inženjeringa koja se održava u četvrtom semestru studija za bachelor diplomu iz oblasti Mehatroničkog inženjeringa na Ostbayerische Technische Hochschule (OTH Regensburg), 2015. godine.

Iskustva sa korišćenjem ove igre u kombinaciji sa klasičnim predavanjima ukazuju da ova metoda grupnog učenja može biti korisna na nekoliko načina. Najvažnije je to što su studenti u svojim povratnim informacijama naveli da su uživali u takmičenju, osećali se visoko motivisano i voleli su što su deo grupe.

### 7.3.7 Light views

Internet okruženje za učenje principa testiranja objektno orijentisanog softvera, nazvano Light views, predloženo je od strane [169]. To je bilo jedno od prvih okruženja korišćenih za edukaciju o testiranju softvera i jedno od prvih koje je prepoznalo važnost onlajn alata za e-učenje. Autori su takođe prepoznali da su u tradicionalnim osnovnim studijama računarstva i softverskog inženjeringa, studenti učili tehnike za dizajniranje i razvijanje relativno malih aplikacija, sa gotovo nikakvim akcentom na testiranje softvera. Predloženi sistem sadržao je animacije, slike, UML dijagrame i interaktivne lekcije kako bi pomogao studentima da nauče koncepte testiranja objektno orijentisanog softvera. Tehnologije korišćene za razvoj ove veb aplikacije uključivale su Java aplete, JavaScript i Perl.

Testiranje objektno orijentisanog softvera smatra se složenijim od testiranja tradicionalnog proceduralnog softvera, zbog paradigme objektno orijentisanog softvera koja uključuje nasleđivanje, polimorfizam i apstrakciju. Predloženi sistem je imao za cilj da pomogne studentima da nauče ove koncepte vizualizacijom procesa testiranja objektno orijentisanog softvera kroz interaktivne lekcije. Očekivalo se da će studenti prihvatiti izazov kroz vizualizaciju putanja izvršavanja softvera i tranzicija stanja i stvoriti bolje test slučajeve koji bi pokrili sva moguća stanja komponenti uz pomoć simulacija i animacija.

Light views platforma je ovaj interaktivni kurs izložila kroz veb interfejs, kako bi omogućila studentima da aktivno uče, kroz saradnju, u svom ritmu kod kuće, preko inter-

neta. Alat je korišćen početkom 2000-ih na Školi za računarstvo i softversko inženjerstvo, Monash University, Caulfield, Australija. Koncept učenja uključivao je tekstualne opise izazova i pasivne materijale uključujući UML modele problema. Veb interfejs je takođe uključivao linkove ka teoriji objektno orijentisanog programiranja i teoriji testiranja softvera. Studenti su evaluirani kroz upitnike i kvizove za svaku studiju slučaja uključenu u kurs. Sistem je uključivao upitnike kako bi primio povratne informacije od studenata i dalje unapredio sistem. Kvizovi su imali za cilj da testiraju znanje studenata kroz širok spektar pitanja, od jednostavnih da/ne i pitanja sa više izbora do složenijih interaktivnih pitanja sa dijagramima stanja i tranzicijama. Obuhvaćene teme uključivale su testiranje metodama crne i bele kutije, testiranje na osnovu događaja i testiranje distribuiranih komponenti.

### 7.3.8 Sistem za učenje lake strukture na bazi Androida

Sistem za učenje lake strukture na bazi Android platforme kao podrška za kurseve testiranja softvera predložen je od strane [170]. Autori su odlučili da iskoriste popularnost i rasprostranjenost mobilnih telefona među studentima i izabrali Android kao ciljanu platformu za aplikaciju, budući da je njegov tržišni udeo mnogo veći od iOS-a. Cilj ovog sistema bio je da obezbedi interaktivnu aplikaciju koja je sposobna da prezentuje resurse za učenje kako bi poboljšala učenje na kampusu.

Principi na kojima je sistem izgrađen bili su masovni otvoreni onlajn kursevi (MOOC) i mali privatni onlajn kursevi (SPOC) [171]. U predstavljenom sistemu, nastavnici su pažljivo dizajnirali i uređivali mikro-video snimke, a od studenata se tražilo da samostalno gledaju video snimke izvan učionice (unutar kampusa), koristeći svoje pametne telefone. Sistem je takođe pružao podršku za onlajn testiranje i interakciju sa ostalim studentima i nastavnicima. Klijentska strana predloženog sistema implementirana je na Android platformi, dok je backend strana izgrađena kao SSH (Struts, Spring i Hibernate) za upravljanje različitim materijalima za kurseve. Kako je većina autonomnih platformi namenjena za onlajn učenje, predloženi sistem uključivao je dve glavne komponente: osnovnu funkcionalnost i pomoćnu funkcionalnost. Osnovna komponenta se bavila osnovnim informacijama o kursu, upravljanjem resursima i vežbama, dok je pomoćna komponenta pružala podršku za onlajn ispite, diskusije i evaluacije. Kompletna funkcionalnost je bila izložena kroz Android frontend.

Predloženi sistem je korišćen 2018. godine na Univerzitetu Anhui SanLian u Kini, i korišćen je za starije studente koji su se opredelili za računarstvo i softversko inženjerstvo, koji su uzeli predmet testiranja softvera kao izborni kurs, izvan redovnih časova. Nakon prijave na sistem, studenti mogu pregledati materijale za kurs, preuzeti potrebne resurse i prisustvovati forumskoj diskusiji o predmetu učenja koji ih zanima, zajedno sa ostalim studentima i nastavnicima. Nastavnici, s druge strane, mogu objaviti različite materijale za učenje, video snimke i vežbe, kao i odgovoriti na pitanja studenata pristupanjem forumu. Sistem je testiran na kampusu gde je tri stotine studenata završilo onlajn kurs koristeći svoje Android telefone.

### 7.3.9 Bug Hunt

Bug Hunt, veb-bazirani tutorijal koji pokriva uvodne teme o testiranju softvera, predložen je od strane [139]. Razvijen je sa ciljem da poveća angažovanje studenata i smanji radni teret nastavnika pružajući automatske procene znanja. Svaka lekcija sadrži niz iza-

zova koji daju trenutne povratne informacije kako bi se povećalo angažovanje studenata. Studenti mogu da prate lekcije u svom ritmu i da vežbaju osnovne pristupe u testiranju softvera.

Aplikacija je izgrađena sa tradicionalnim Java 2 Enterprise Edition tehnologijama, kao standardna MVC arhitektura. Aplikacija je dizajnirana da postepeno gradi znanje studenata o tehnikama testiranja softvera. Početna lekcija daje uvod u osnovne termine i koncepte, dok sledeće lekcije obrađuju metode crne kutije i metode bele kutije. Na kraju, poslednja lekcija spaja sve koncepte i uvodi automatizaciju testiranja. Svaka lekcija ima niz uputstava koja objašnjavaju metodu testiranja i daju izazov koji treba rešiti. Pored toga, svaka lekcija sadrži kolekciju artefakata od interesa za datu temu (dokument specifikacije zahteva, izvorni kod, itd.). Cilj studenata je da razviju test slučajeve (u obliku kolekcije ulaza i očekivanih izlaza za program) za svaku lekciju i predaju ih sistemu. Nakon podnošenja test slučajeva, Bug Hunt će ih izvršiti i pružiti trenutne povratne informacije studentima.

Aplikacija Bug Hunt je implementirana tokom dva semestra na Univerzitetu Nebraska-Lincoln i pet drugih institucija (jesenji semestar 2004. i prolećni semestar 2005.), sa obećavajućim rezultatima. Oko dvesta studenata koristilo je sistem i generalno su bili vrlo zadovoljni njime, prema rezultatima objavljenim od strane [139]. Prednost za nastavnike koji koriste Bug Hunt bila je automatizovana procena studenata, kroz automatsko generisanje izveštaja studenata. Aplikacija beleži sve test slučajeve, ulazne vrednosti, očekivane izlaze, stvarne izlaze i rezultate izvršenja testa. Kako bi svim studentima u grupi mogao biti dodeljen isti identifikacioni broj klase, bilo je moguće meriti individualne performanse studenata u odnosu na ukupne performanse grupe.

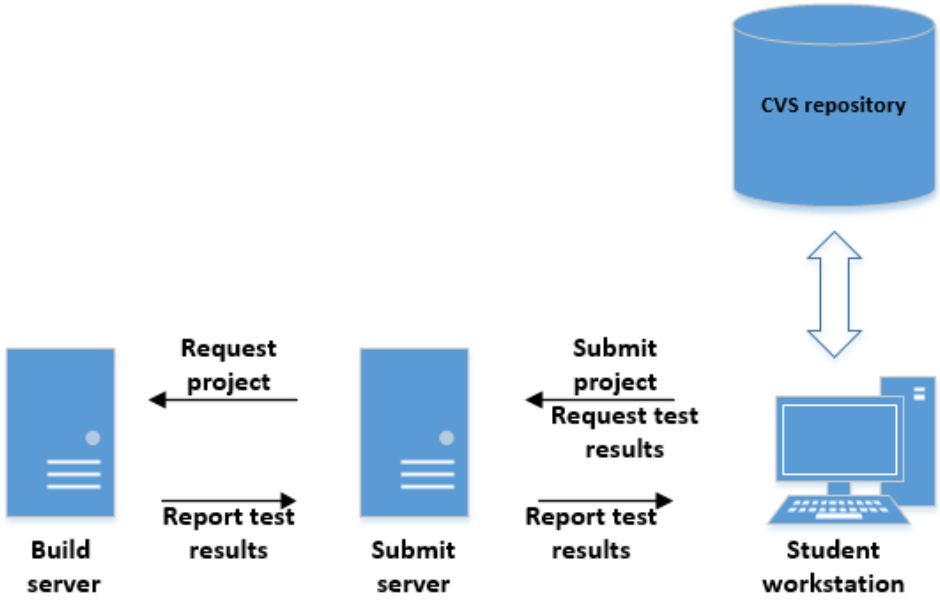
### 7.3.10 MARMOSET

MARMOSET alat je bio jedan od prvih projekata koji su pružali podršku za kurseve softverskog testiranja, i predložen je od strane [140, 141]. Početni cilj ovog projekta bio je da pruži podršku za testiranje uglavnom za uvodne kurseve programiranja, kroz predaju projekata i testno okruženje koje je podsticalo studente da rano počnu da rade na svojim projektima i kritički razmišljaju o njima. MARMOSET je razvijen kao tradicionalni J2EE veb server, koji se oslanja na servlets, jsp, GWT i JavaScript.

Proces u MARMOSET sistemu bio je sledeći: studenti bi predali svoje projekte kroz Eclipse okruženje (koristeći opciju koju pruža plugin) ili koristeći alate komandne linije. Server za predaju bio je korišćen za prijem predatih projekata. Odvojeni build server korišćen je za izgradnju projekata i testiranje prema postavci testa, koja je sadržala sve neophodne biblioteke i fajlove za izvršenje testa. Postojala su četiri glavna tipa testova koje je MARMOSET sistem bio u stanju da izvršava:

- Studentske testove (razvijene od strane samih studenata);
- Javne testove (testove dostupni studentima);
- Finalne testove za release (poverljivi test set nastavnika i rezultati su objavljivani studentima pre roka);
- Tajne testove (poverljivi test set nastavnika, sa rezultatima koji su objavljivani studentima tek nakon roka za predaju projekta).

Nakon predaje projekta sistemu, projekat bi bio kompajliran i testiran od strane alata, i alat bi pružio trenutni izveštaj sa rezultatima test seta studenata i javnih test setova.




**Slika 7.5:** Marmoset dijagram toka [140].

Rezultati finalnog test seta za release bi bili otkriveni blizu roka za predaju projekta (da bi se omogućile neke finalne izmene projekata), dok bi rezultati tajnog test seta bili otkriveni nakon roka. Rezultati tajnog test seta su korišćeni za ocenjivanje projekata studenata. Ceo proces je prikazan na slici 7.5 [140].

Test set za release imao je neke specifične karakteristike. Nakon prolaska javnog test seta, studenti su imali opciju da sprovedu testiranje release-a svojih projekata. Nakon zahteva za testiranje release verzije projekta, sistem bi otkrio samo broj neuspešnih testova studentu, sa imenima samo prva dva testa koja su pala (nisu svi neuspešni testovi bili otkriveni studentima). Da bi podstakao studente da kritički analiziraju svoj projekat sami pre ponovne predaje projekta, sistem je imao ograničen broj release testova koje studenti mogu izvršiti za svoj projekat, gde bi svako puštanje release testova potrošilo po jedan token (koji bi se regenerisao nakon nekog predefinisanog vremena). Standardna postavka sistema dozvoljavala je studentima tri tokena, i jedan dan vremena za regeneraciju za svaki projekat. Drugim rečima, studenti su mogli da izvedu release testiranja na svojim projektima tri puta dnevno.

Nastavnici, s druge strane, mogli su da prate napredak studenata, gledajući detaljne izveštaje o predatim projektima, izvedenim testovima (sa statusom prošao ili pao) za svakog studenta, itd. MARMOSET je bio u stanju da obradi Java projekte i predaje zasnovane na make fajlu (napisane na bilo kom programskom jeziku koji se može kompajlirati i testirati sa make alatom, kao što su C i Ruby, na primer). Java projekti su se oslanjali na JUnit biblioteku za izvršenje testa. Alat je inicijalno bio veoma dobro prihvaćen od strane studenata i nastavnika i korišćen je više od pet godina u praksi, od 2006. do 2011. godine, na Univerzitetu u Merilendu, SAD. MARMOSET alat sa demo serverom i izvornim kodom je dostupan na `https://marmoset.cs.umd.edu/`.

### 7.3.11 Bug Hide-and-Seek

Bug Hide-and-Seek predstavlja pristup zasnovan na igri koji se koristi za istraživanje tačnosti verifikacije testova, a predložen je od strane [172]. Autori ovog istraživanja su imali za cilj da razviju pedagoški pristup za uvođenje jediničnog testiranja u kurseve razvoja softvera. Došli su do igre Bug Hide-and-Seek, koja zahteva od studenata da implementiraju i ispravne i defektne programe (sa namernim greškama) i da razviju adekvatne testove koji će pomoći u otkrivanju grešaka i identifikovanju ispravnih implementacija. Pristup je uključivao najpopularniji alat za jedinično testiranje za specificiranje individualnih testova, uključujući GoogleTest (C++), JUnit (Java) i unittest(Python). Igra je implementirana na dva različita načina.

Prva implementacija igre pretpostavljala je da su studentima date različite instrukcije. Pre početka igre, studentima su nasumično dodeljene jedna od dve moguće uloge - sakrivač ili tragač. Tragači su imali cilj da implementiraju robustne i stabilne programe, koji će proći kroz njihove sopstvene testove, kao i kroz testove razvijene od strane svih ostalih studenata na času. Sakrivači, s druge strane, dobili su uputstva da namerno dodaju greške u svoje programe, sa ciljem da sakriju greške na način da se njihovi programi ponašaju ispravno u nekim slučajevima upotrebe, ali da ne uspeju u drugim slučajevima. Svaki student u grupi je takođe bio obavezan da preda test set zajedno sa implementacijom. Glavni cilj igre bio je da se implementiraju testovi koji će biti sposobni da razlikuju ispravna i neispravna rešenja. Sekundarni cilj je bio različit na osnovu uloge studenta: tragači su pokušavali da implementiraju program dovoljno robustan da bi prošao što više testnih paketa celog razreda, a sakrivači su pokušavali da sakriju greške u programu da bi prevarili što više testnih paketa da ga prođu (ne otkrivajući grešku).

Druga varijanta igre nije razdvajala studente u dve uloge. Umesto toga, svakom studentu je naloženo da sakrije grešku u određenoj metodi, dok su sve ostale metode u projektu ispravno implementirane. Glavni cilj igre je ostao isti - studenti su pokušavali da implementiraju skrivene greške koje su teško otkriti i da kreiraju test set koji bi bio sposoban da ispravno identifikuje prihvatljive i neispravne implementacije. Nasumična dodela metoda gde su studenti bili obavezni da implementiraju skrivene greške osigurala je da su greške distribuirane preko različitih metoda kada su svi predati projekti uzeti u obzir.

U obe varijante igre, programski zadatak bio je razvijanje klase koja predstavlja model podataka za igru Tic-Tac-Toe sa skupom metoda (koje su sprovedene implementiranjem obezbeđenog interfejsa), zajedno sa odgovarajućim testovima. Igra je evaluirana tokom dva semestra na Državnom univerzitetu u Čiku, SAD, od strane studenata koji su imali prethodno iskustvo sa programskim jezicima, ali nisu imali kontakt sa testiranjem softvera, sa obećavajućim rezultatima.

### 7.3.12 iLearnTest

iLearnTest je framework za edukativne igre koji je predložen od strane [173]. iLearnTest, zapravo, omogućava lako kreiranje edukativnih igara, kako bi podržao različite oblasti obrazovanja iz softverskog inženjeringa i pomogao studentima povećavajući njihovo interesovanje i angažovanje na temu kursa. Framework pruža nekoliko funkcija koje ga čine zanimljivim i za instruktore i za studente:

- Nudi kolekciju šablona igara koji olakšavaju kreiranje igara.
- Odvaja sadržaj igre od njene implementacije.

- Daje mogućnost studentima da uče svojim tempom;
- Pruža povratne informacije o rezultatu igre, tačnim odgovorima i greškama studentima nakon svake igre, tako da mogu da pokušaju da se poboljšaju i da ostvare bolje rezultate sledeći put kada igraju igru.

Framework pruža kolekciju šablona igara, koji mogu lako da se prilagode za bilo koju temu softverskog inženjeringa i da se ponovo koriste za različite kurseve. Framework je izgrađen na osnovi Construct2 endžina koji obezbeđuje osnovni set funkcija. Sadržaj igre je specificiran unutar XML fajla, koji se čita kroz AJAX i koristi se za automatsko generisanje igre. Igra se generiše u HTML5 omogućavajući simultanu igru više studenata kroz veb interfejs. Kako je iLearnTest onlajn veb platforma, takođe je prilagođena za učenje testiranja softvera [174].

Sadržaj igre trebalo je da bude prilagođen da prati Foundation Level silabus šeme sertifikacije ISTQB-a. Prva verzija igre uključivala je materijale iz dve sekcije Foundation Level silabusa - statičko testiranje i tehnike koncipiranja testa. Materijali iz drugih poglavlja ovog silabusa, uključujući osnove testiranja, upravljanje testovima i test alate, biće dodati u kasnijim verzijama igre.

Za svako poglavlje dostupne su različite opcije igranja, uključujući:

- pogodi koncept,
- opcija prevuci i otpusti,
- pronađi put,
- kviz pitanja.

Ove opcije mogu da se modifikuju u zavisnosti od tipa pitanja. Igra je evaluirana na Fakultetu inženjeringa Univerziteta u Portu. Alatu se može pristupiti onlajn na `https://web.fe.up.pt/~apaiva/iLearnTest/`, međutim, autori moraju biti kontaktirani za kreiranje važećeg naloga.

### 7.3.13 TestEG

Igra TestEG (Test educational game) je predložena od strane [175, 176], sa ciljem da poboljša kako teorijske tako i praktične aspekte procesa učenja softverskog testiranja. Igra je razvijena u Unity3D okruženju. Ova edukativna igra slična kvizu simulira kancelarijsko okruženje, sa virtuelnim novcem koji se može potrošiti na zapošljavanje maksimalno tri zaposlena. Ovi radnici se kasnije mogu obučavati sa ciljem da pomognu igraču u softverskom testiranju.

Radnici takođe mogu pomoći sa čitanjem informacija o različitim temama softverskog testiranja. Kako igrač zapošljava radnike i obučava ih za softversko testiranje, igra zapravo daje igraču ulogu menadžera tima za osiguranje kvaliteta, koji vodi tim softverskih testera koji će mu pomoći tokom procesa testiranja. Ovo je jedinstven pristup, koji takođe pokriva deo upravljanja timom projekata softverskog testiranja, što se ne vidi u drugim pristupima zasnovanim na igrama pokrivenim u ovom preglednom radu.

Koncept igre je jednostavan - igraču i njegovom timu biće dati različiti problemi sa kodom, a igrač ima cilj da ih reši u ograničenom vremenskom okviru. Osnovna postavka igre zahteva od igrača da odgovori na deset pitanja u roku od deset minuta. Nakon

što igrač završi dati kviz, igra daje povratne informacije o tačnim i pogrešnim odgovorima. Nastavnici takođe mogu kontrolisati okruženje i pratiti napredak studenata i njihove odgovarajuće rezultate.

Pristup dat u TestEG je evaluiran korišćenjem upitnika kako bi se dobile povratne informacije od studenata. Kako je fokus TestEG-a podrška procesu učenja i nastave povezanom sa softverskim testiranjem, studenti Federalnog univerziteta u Lavrasu koji su pohađali relevantne kurseve koji se bave softverskim testiranjem i profesionalci koji rade u toj oblasti su pozvani putem e-maila da budu evaluatori. Pored toga, poslata je i e-mail pozivnica SBC-u (Sociedade Brasileira de Computação). Upitnik je uključivao pitanja o igri, angažovanju i nivou uživanja kako bi se proverila prihvaćenost predložene igre među studentima. Dobijeni rezultati jasno su pokazali da TestEG ispunjava osnovne zahteve edukativnog softvera i da se može koristiti kao efikasan alat koji bi mogao da podrži kurs za učenje softverskog testiranja.

### 7.3.14 JoVeTest

Igra zasnovana na Java programskom jeziku, JoVeTest, je predložena od strane [177]. Cilj igre bio je da zadovolji potrebe vezane za obuku novih profesionalaca i predstavljanje sadržaja na efikasniji i zabavniji način. JoVeTest je digitalna igra zasnovana na poznatoj igri Tic Tac Toe koja podržava učenje i proučavanje koncepta softverskog testiranja. JoVeTest prati istu dinamiku kao popularna igra Tic Tac Toe. Postoje dva igrača, svaki ima simbol (X ili O), i svaki igrač igra naizmenično. Cilj igre je da se simbol spoji u red, kolonu ili dijagonalu. Ko god prvi dostigne cilj pobeđuje. Ono što karakteriše JoVeTest kao nastavni alat je to što igrač, da bi imao pravo da obeleži svoj simbol, mora tačno da odgovori na pitanje sa više ponuđenih odgovora vezano za softversko testiranje.

JoVeTest je razvijen u programskom jeziku Java, kao samostalna desktop aplikacija. Veoma je jednostavan, jer ne koristi bazu podataka, a pitanja dolaze u tekstualnom fajlu koji je kompresovan unutar JAR fajla. Danas JoVeTest sadrži 144 pitanja preuzeta iz ISTQB Certified Tester Foundation level silabusa. Nova pitanja iz bilo kog drugog izvora mogu se dodati u aplikaciju preko txt fajla, omogućavajući lako povećanje početne kolekcije pitanja, i samim tim prilagođavanje svakom sadržaju kursa koji se pokriva na predavanjima. Kompletan projekat je otvorenog koda i dostupan je na `http://experts.icomp.ufam.edu.br/jovetest/`.

Kada igrač odabere poziciju gde želi da postavi svoj simbol, pitanje se izvlači iz teme odabrane u početnim podešavanjima sistema i koje još nije izvučeno u tekućoj igri. Zatim se pitanje prikazuje na tabli. Igrač mora tačno da odgovori na pitanje da bi mogao da obeleži svoj simbol na izabranom mestu. Ako igrač izabere pogrešan odgovor, prikazuje se poruka o statusu odgovora, zajedno sa tačnim odgovorom koji je istaknut zelenom bojom i nema obeležavanja simbola igrača na izabranom mestu. Ako igrač izabere tačan odgovor, prikazuje se obaveštenje da je odgovor tačan, obeležena opcija je istaknuta zelenom bojom, a simbol igrača je obeležen na izabranom mestu.

Aplikacija je implementirana na Univerzitetu u Amazonasu, Brazil, i testirana od strane studenata na kursu verifikacije i validacije 2016. godine, sa dobrim povratnim informacijama. Kasnije je aplikacija uglavnom korišćena kao alat za pripremu za ISTQB Foundation Level ispita.

### 7.3.15 GreaTest

Još jedan pristup zasnovan na kartama pod nazivom GreaTest predložen je u [178]. GreaTest igra sa kartama je igra za više igrača kreirana za podučavanje softverskog testiranja na zabavniji način. Kreacija ove igre bila je zasnovana na drugim igrama poput Munchkin [179], u kojoj igrači moraju da reše izazove sa kartama koje drže u rukama da bi osvojili nagrade nakon svakog uspešnog izazova. U predloženoj igri, izazovi su scenariji upotrebe aplikacije i igrači moraju da pogode vrstu testa koji kada se pokrene može otkriti situaciju opisanu u izazovu.

Cilj ove igre sa kartama je da podstakne učenje primene različitih tehnika testiranja u situacijama koje se mogu dogoditi prilikom korišćenja aplikacije i na igraču je da odredi koja vrsta testa je potrebna. Vrste testova uključene u igru su:

- Test prihvatljivosti: proverava da li se sistem ponaša prema željenoj funkcionalnosti navedenoj u specifikaciji zahteva.
- Test performansi: proverava da li sistem zadovoljava zahteve za performanse, kao što su vreme odziva i stopa prenosa.
- Stres test: proverava kako sistem reaguje pod izuzetno velikim opterećenjima, obično simuliranjem velikog broja konkurentnih korisnika.
- Funkcionalni test: proverava da li je potrebna funkcionalnost ispravno implementirana u sistemu.
- Test bezbednosti: proverava da li je sistem dovoljno robustan da izdrži situacije kada je izložen sigurnosnim napadima.
- Test upotrebljivosti: proverava izgled i iskustvo sa sistemom, drugim rečima, da li korisnici mogu lako da interaguju sa sistemom.

Igra se sastoji od tri špila karata. Dva špila su inspirisana Munchkin igrom, jedan sa zadacima koji treba da se reše, i drugi sa bonus kartama za uspešna rešenja zadataka. Maksimalan broj igrača je postavljen na šest, a igra podstiče interakciju između igrača. Špilovi karata su definisani na sledeći način:

- Špil sa izazovima sadrži 38 situacija korišćenja aplikacije, koje mogu predstavljati ispravnu upotrebu ili problem. Aplikacije uključuju bankarski sistem, aplikaciju za video pozive i sistem za registraciju.
- Igrački špil sadrži 48 karata sa vrstama testova, 8 sa testerima, i 16 dodatnih, vezanih za primenu alata ili sastanak, ukupno 72 karte.
- Bonus špil sadrži 36 karata sa nagradama za tačne odgovore.

Igra je inspirisana Munchkin igrom. Pre početka, svaki igrač dobija pet karata iz špila, a svaki igrač, tokom svog poteza, izvlači dve karte iz istog špila. Prvi igrač može da odigra svoju kartu, spuštajući kartu testera na sto, ako je ima u ruci. Ako igrač nema ovu kartu, preskače svoj potez i daje ga igraču sa svoje leve strane. Ako izvučena karta bude tester, on bira do dve situacije među onima koje su na stolu za rešavanje, uzimajući u obzir karte testova koje može imati u ruci. Treba napomenuti da na stolu igrača može biti samo jedan tester, osim ako nema "Kooperativni rad", koji mu omogućava da ima dve karte

tipa tester. Da bi se rešila situacija iz špila izazova, potrebno je povezati kartu testa sa izabranim izazovom. Zatim igrač okreće kartu izazova da vidi koji su testovi primenljivi u kontekstu opisane situacije i ako je igrač odigrao odgovarajući test, proverava koje brojeve treba da izvuče na kockicama. Sve vrste testova su navedene na poleđini karte, zajedno sa vrednostima koje su im dodeljene. Ako igrač ukloni brojeve koje su naznačili bacanjem kockica, dobija poen i može izvući kartu iz bonus špila. U situaciji kada se ne koriste bonus karte može zaraditi do dva poena po rundi. Igrač koji sakupi ukupno sedam poena pobeđuje u igri.

Predložena igra sa kartama GreaTest planirana je da deluje kao podrška nastavi softverskog testiranja kroz interakciju grupa studenata. Ograničenje predloženog pristupa je da je igra trenutno podržana samo na portugalskom jeziku. Sadržaj igre može se dobiti na linku `http://pesquisa.great.ufc.br/en.html` besplatno, kontaktiranjem autora. Igra se koristi na Federalnom univerzitetu u Ceará, Brazil, od 2018. godine.

### 7.3.16 Test trainer

Alat Test trainer predložen je u [180]. Okruženje sadrži veb interfejs sa materijalima za obuku i alatom za obuku u oblasti testiranja softvera koji studenti mogu koristiti za vežbanje svojih veština i dobijanje trenutnih povratnih informacija koje mogu pomoći u procesu učenja. Tutorijali izloženi preko veb interfejsa pružaju detaljne vodiče za različite tehnike testiranja, sa grafičkim primerima koji pomažu studentima da vizualizuju proces. Tutorijali sadrže lekcije o osnovnim principima testiranja, metodama crne i bele kutije, uključujući analizu graničnih vrednosti, podelu na klase ekvivalencije, i uzročno-posledični graf.

Nakon završetka tutorijala, studenti su u mogućnosti da vežbaju tehnike alatom Test trainer, koji je implementiran kao standardna Java aplikacija. Studenti mogu da preuzmu specifikaciju programa koji treba da se testira, da razviju testove (navođenjem ulaznih vrednosti i očekivanih izlaznih vrednosti) i i da ih pošalju preko alata. Kada se proces testiranja završi, studenti mogu da posmatraju rezultate i da vide individualne test statuse (da li su testovi prošli ili pali). Na kraju, student prima povratne informacije sa statistikama - broj otkrivenih defekata, broj nedetektovanih defekata i pokrivenost testa. Moguće je videti koje su zahteve pokrili testovi, a koje su propustili, omogućavajući studentima da razumeju gde su pogrešili. Alat je takođe sposoban da prikaže redundantne testove.

S druge strane, nastavnici mogu koristiti alat Trainer kako bi obezbedili program koji treba da se testira, zajedno sa zahtevima (kreiranim korišćenjem šablona za strukturiranje zahteva na pravilan način i olakšavanje procesa parsiranja). Nastavnici takođe mogu obezbediti provere za identifikaciju defekata i zahteve koji bi trebalo da budu testirani.

Arhitektura predloženog sistema sadrži četiri glavna modula:

- GUI - korisnički interfejs izgrađen sa Java Swing, korišćenjem pristupa programiranja vođenog događajima.
- parser - koristi se za parsiranje ulaznog teksta (zahtevi, ulazne i izlazne promenljive, itd.) i java datoteka.
- executor - koristi se za stvaranje okruženja u realnom vremenu, omogućavajući manipulaciju i izvršavanje tvrdnje na osnovu programa u odnosu na ulaz testa.
- metrics - koristi se za generisanje metrika o pokrivenosti postignutoj testovima koje je obezbedio student.

Alat je testiran na grupi studenata programiranja na početnom nivou na Državnom univerzitetu Arizona, SAD, u 2005. godini. Objavljeni rezultati ukazuju da je pokrivenost testova koju su postigli studenti drastično povećana nakon korišćenja tutorijala i alata Trainer.

## 7.4 Prednosti i mane upotrebe simulatora u edukaciji

Tradicionalni način predavanja teme testiranja softvera nije prilagođen zahtevima savremenih studenata. Pre svega, materijali za kurs mogu biti dosadni i suvoparni ako nastavnik koristi tradicionalni pristup sa kredom i tablom, što može da demotiviše i frustrira studente koji očekuju da vide praktične primere metoda testiranja softvera. Drugo, tehnologije koje se mogu koristiti kao podrška nastavnom procesu napreduju i poboljšavaju se svakodnevno. Veb sajtovi, mobilne aplikacije, igre, pa čak i virtuelna stvarnost, sve to može da se koristi za unapređenje nastavnog procesa i pruža beskrajne mogućnosti.

Dve glavne metodologije mogu da se koriste za poboljšanje nastavne prakse uopšte: učenje zasnovano na istraživanju i učenje zasnovano na rešavanju problema. Prva metodologija je usredsređena na podsticanje studenata da sami otkrivaju stvari, izvode eksperimente sa svojim istraživačkim projektima i sarađuju sa drugim studentima kako bi međusobno podelili znanje. Ova metodologija cilja na nivo angažovanja studenata, radoznalost, pedantnost, kritičko razmišljanje i sposobnost da primene naučene tehnike u praksi, a sve to se smatra visoko poželjnim sposobnostima koje dobar tester softvera treba da ima.

Druga metodologija se odnosi na pristup gde nastavnici koriste rešavanje problema kako bi sistematski upoznali studente sa konceptima koji će se koristiti za pronalaženje rešenja za različite zadatke. Ova metoda se oslanja na nastavnika, koji daje zadatak studentima, pružajući im vođstvo i podršku dok pokušavaju sami da analiziraju i reše zadatak, primenjujući koncepte naučene kroz predavanja. Ovaj pristup se često koristi na kursevima računarstva i softverskog inženjeringa, i prirodno je dobar izbor za kurseve testiranja softvera [181].

Obe opisane metodologije se lako mogu integrisati i implementirati u savremena okruženja za učenje i saradnju. Veb platforme WReSTT-CyLE (Odeljak 7.3.1), Automatizovani sistem za interaktivno učenje testiranja softvera (Odeljak 7.3.2) i Light views (Odeljak 7.3.7), su primeri ovakvih alata. Ove platforme za saradnju imaju mogućnost da povećaju posvećenost, angažovanje i motivaciju studenata, i podstaknu saradnju i deljenje znanja među studentima. Pored toga, ove platforme se mogu koristiti i na mobilnim uređajima, koncept koji je prikazan u Odeljku 7.3.8 na Android platformi. Potencijalni problem koji može nastati je trajna integracija ovog tipa alata u akademske institucije, jer većina tih institucija ima tendenciju da se drži tradicionalnijih pristupa nastavnom procesu, i često su otporne na prihvatanje i uključivanje savremenih tehnologija. Drugi problem je trošak potreban za implementaciju i održavanje takvog sistema, koji se u većini slučajeva ne može ignorisati. S druge strane, prednosti korišćenja obrazovnih alata kao korisnog dodatka tradicionalnim predavanjima su značajne:

1. Studenti su motivisaniji i njihovo angažovanje je značajno povećano. Oni su aktivni učesnici u procesu prenosa znanja, učestvujući u diskusijama i istraživanjima kroz individualne i grupne projekte. Veštine koje mogu steći su praktičnije i primenljivije kada završe studije i zaposle se.

2. Nivo kreativnosti je drastično povećan. Studenti su u mogućnosti da izvode svoja istraživanja, izvršavaju simulacije, učestvuju u projektima, i dobijaju povratne informacije o svom radu, što ih može podstaći da rade još više.

3. Tehnološka infrastruktura koja je potrebna za ovakvo okruženje postaje dostupna i pristupačna sve većem broju akademskih institucija svakog dana.

4. Studenti sa posebnim potrebama se lako mogu uključiti u virtuelno okruženje za učenje, i mogu učestvovati u većem broju aktivnosti u poređenju sa tradicionalnom metodom nastave.

5. Dodatni zadaci mogu biti nagrađeni dodatnim bodovima za aktivnost, koji se mogu smatrati dodatkom na ukupnu ocenu predmeta.

Među nedostacima korišćenja alata za učenje u nastavnom procesu, treba napomenuti mogući problem preopterećenja studenata sa previše materijala za učenje. Ako sadržaj nije pravilno izbalansiran, to može dovesti do gubitka fokusa i smanjenja nivoa angažovanja. Druga potencijalna opasnost leži u sistemu nagrađivanja - ako platforma dodeljuje dodatne bodove za dodatne zadatke, koji se računaju u ukupnu ocenu koju studenti zarade, oni mogu biti u iskušenju da na kraju pokušaju da pronađu ranjivost ili grešku u dizajnu sistema, sigurnosti i iskoriste alat. Poslednji izazov leži u činjenici da nastavnici ponekad nisu upoznati sa najnovijim tehnologijama, dok su nove generacije studenata, rođene 2000-ih godina, bile u bliskom kontaktu sa tim tehnologijama od samog početka. Ovaj tehnološki jaz između nastavnika i studenata može predstavljati potencijalni problem koji može usporiti integraciju savremenih tehnologija u nastavni proces, kako je primetio [182].

Ovo opsežno istraživanje je takođe pokazalo da se pristup zasnovan na igrama često koristi za povećanje interesa i angažovanja studenata. To ima savršen smisao, jer sirovi materijali mogu brzo postati dosadni i suvoparni, a gejmifikacija bi mogla biti savršeno rešenje. Pristupi zasnovani na igrama, kao što su SQLTest-GoRace (Odeljak 7.3.3), Code Defenders (Odeljak 7.3.4), Testing game (Odeljak 7.3.5), Bug hunt (Odeljak 7.3.9), Bug Hide-and-Seek (Odeljak 7.3.11), TestEG (Odeljak 7.3.13) i JoVeTest (Odeljak 7.3.14) su razvijeni kao kompjuterske igre, a povratne informacije studenata su bile odlične. Takođe je korišćen i pristup zasnovan na kartama, kao što je prikazano u primerima Card game (Odeljak 7.3.6) i GreaTest (Odeljak 7.3.15). Ove igre sa kartama su inspirisane uspešnim i široko popularnim igrama sa kartama, kao što su Munchkin i Magic: The Gathering, igre sa kojima su studenti upoznati. I kompjuterske igre i igre sa kartama drastično povećavaju nivo angažovanja studenata, jer se osećaju motivisano da aktivno učestvuju u okruženjima nalik igrama, iz očiglednog razloga - podseća ih na prave video igre. Uspeh pristupa gejmifikaciji se oslanja na još jedan važan faktor - takmičarski duh, jer studenti uživaju u takmičenju jedni protiv drugih, drastično povećavajući angažovanje. Ovo se može koristiti za poboljšanje njihovog razumevanja nastavnih materijala.

Glavne prednosti pristupa zasnovanog na igrama kao dodatka procesu učenja testiranja softvera su sledeće:

1. Nivo motivacije, konkurentnosti i angažovanja studenata se drastično poboljšava. Stavljanje u poznata okruženja nalik igrama može im pomoći da se osećaju udobnije, otvorenije i raspoloženije za inače monotone i potencijalno dosadne materijale.

2. Gejmifikacija predstavlja mnogo prijatniji način učenja, posebno kada se uporedi sa tradicionalnim metodama nastave.

3. Gejmifikacija može povećati entuzijazam i samopouzdanje.

4. Korišćenje multiplayer moda može poboljšati timski rad i saradnju.

5. Prenos znanja i diskusije postaju prirodni studentima koji učestvuju u igri zajedno, što im može pomoći da razumeju neke složenije teme.

6. Teške teme se mogu predstaviti na prijatniji i zanimljiviji način, što može pomoći studentima da ih shvate kroz praktične primere ugrađene u igru.

7. Gejmifikacija može pomoći studentima koji imaju problema sa socijalnim veštinama i interakcijom sa drugim ljudima, pružajući im snažnu motivaciju da učestvuju i budu član tima.

Kao nedostatak gejmifikacije, može se videti da brojni kritičari tvrde da ona ne dodaje ništa korisno učenju, čak da studenti samo gube vreme i neće usvojiti ništa važno. S druge strane, profesori često nisu voljni da koriste pristupe zasnovane na igrama u učionici na nivou postdiplomskih studija, jer to možda neće izgledati prikladno tradicionalnijim nastavnicima i kolegama. Čak i ako je pristup zasnovan na igrama usvojen za kurs, ponekad profesori ne mogu da odrede da li studenti stvarno uče, ili se samo igraju i zabavljaju se. Pored toga, kako je primećeno u [150], dizajn igre, korisničko iskustvo, a naročito sistem nagrada su ključni za angažovanje studenata, jer čak i ako su privučeni igri na početku, nivo angažovanja će opasti kako kurs napreduje. Motivacija će pratiti silazni trend kada studenti shvate da neće dobiti više nagrada za svoj rad. Moguće rešenje za ovaj izazov bi bilo da se motivacioni stimulus rasporedi tokom celog kursa, kako bi se studenti zadržali zainteresovanim i angažovanim.

Komparativna analiza šesnaest alata obuhvaćenih ovim pregledom je data u Tabeli 7.2. Očigledno je da dominiraju metode zasnovane na igrama, jer se više od polovine obuhvaćenih alata odnosi na gejmifikaciju. To je zapravo bilo očekivano, jer je pristup zasnovan na igrama najperspektivniji za povećanje interesa i angažovanja studenata, pošto sirovi materijali mogu biti dosadni. Takođe se može istaći da samo jedan od šesnaest razmatranih alata (WReSTT) zapravo implementira i nudi virtuelne poene koji se mogu sakupljati kroz dodatne aktivnosti. Svi razmatrani okviri ciljaju na motivaciju i angažovanje studenata podsticanjem njihovog takmičarskog duha. Međutim, neka vrsta virtuelnih poena ili koncepta nagrade se može preporučiti za još veću motivaciju studenata, jer se ti virtuelni poeni mogu koristiti za dodavanje ocenama studenata dobijenim tradicionalnim metodama ispitivanja. Tabela 7.3 pruža dodatne detalje o tehnologijama korišćenim za razvoj svakog od obuhvaćenih alata, kao i veb adresu gde se alatu može pristupiti (ako je informacija dostupna), i podršku za jezik. Tabela 7.4 prikazuje detalje o korišćenju okruženja uključenih u pregled (institucija gde su razvijeni, godina početnog uvođenja, institucije koje ih koriste, i period korišćenja).

**Tabela 7.2:** Poređenje okruženja uključenih u istraživanje

| Okruženje | Tip | Teme | Sistem nagrada | glavne karakteristike |
|---|---|---|---|---|
| WReSTT-CyLE/STEM-CYLE | Veb aplikacija | Tutorijali i kvizovi | Da | Socijalna platforma i saradnja |
| Automated system | Veb aplikacija | Mutaciono testiranje i tok podataka | Ne | Evaluacija skupova testova |
| SQLTest-GoRace | Igra | Crna i bela kutija, mutaciono testiranje | Da | Dobra grafika, takmičarsko iskustvo |
| Code Defenders | Igra | Mutaciono testiranje | Ne | Bazirano na Java i JUnit |
| Testing game | Igra | Crna i bela kutija, mutaciono testiranje | Ne | Dobra grafika, nekoliko nivoa |
| Card game | Igra | ISTQB Foundation | Ne | Priprema za ISTQB |
| Light views | Veb aplikacija | Crna i bela kutija, testiranje bazirano na dogadajima | Ne | Testiranje objektno-orijentisanog softvera |
| Android learning system | Mobilna aplikacija | Opšte testiranje softvera | Ne | Mikro-video, onlajn testiranje |
| Bug Hunt | Veb aplikacija | Crna i bela kutija, automatizacija | Ne | Uvodni tutorijali za testiranje, automatizacija |
| MARMOSET | Veb aplikacija | - | Ne | Izvršavanje nekoliko vrsta testova nad studentskim radovima |
| Bug Hide-and-Seek | Igra | Jedinično testiranje | Ne | Različite uloge za studente - skrivači i tražioci |
| iLearnTest | Igra | ISTQB Foundation | Ne | Priprema za ISTQB |
| TestEG | Igra | Uvod u testiranje softvera | Ne | Igra pruža uvid u test menadžment |
| JoVeTest | Igra | ISTQB Foundation | Ne | Priprema za ISTQB |
| GreaTest | Igra | Sistemsko funkcionalno i nefunkcionalno testiranje | Ne | Izazovi uključuju različite vrste grešaka, igrač mora da primeni odgovarajuće rešenje |
| Test trainer | Veb aplikacija | Crna i bela kutija | Ne | Razvijanje testova na osnovu graničnih vrednosti, klasa ekvivalencije i uzročno-posledičnih grafova |

**Tabela 7.3:** Dodatne informacije o okruženjima uključenim u istraživanje

| Okruženje | Tehnologija/programski jezik | url | Podrška za jezike | Publikacija |
|---|---|---|---|---|
| WReSTT-CyLE/STEM-CYLE | Konfigurabilni sistem za upravljanje učenjem | https://stem-cyle.cis.fiu.edu/ | engleski | Clarke et al. [108] |
| Automated system | CodeSkulptor, FEAT, nepoznata veb tehnologija | N/A | engleski | Smith et al. [162] |
| SQLTest-GoRace | GoRace veb aplikacija, SQLTest REST API | https://gorace.uca.es/en, https://in2test.lsi.uniovi.es/sqlrules/ | engleski | Blanco et al. [150] |
| Code Defenders | Java, JUnit | https://code-defenders.org/ | engleski | Rojas et al. [141] |
| Testing game | Nepoznata veb tehnologija | N/A | engleski | Valle et al. [146] |
| Card game | Fizička kartaška igra | N/A | engleski | Soska et al. [168] |
| Light views | Java applets, JavaScript, Perl | N/A | engleski | Ramakrishnan [169] |
| Android learning system | Android | N/A | engleski | Yu [170] |
| Bug Hunt | Java | N/A | engleski | Elbaum et al. [139] |
| MARMOSET | Java, servleti, jsp, GWT, JavaScript | https://marmoset.cs.umd.edu/ | engleski | Spacco et al. [140, 141] |
| Bug Hide-and-Seek | GoogleTest, JUnit, unittest | N/A | engleski | Buffardi et al. [172] |
| iLearnTest | Construct2, AJAX, HTML5 | https://web.fe.up.pt/~apaiva/iLearnTest/ | engleski | Paiva et al. [173] |
| TestEG | Unity3D | N/A | portugalski | Oliveira et al. [175, 176] |
| JoVeTest | Java | http://experts.icomp.ufam.edu.br/jovetest/ | portugalski | Barbosa et al. [177] |
| GreaTest | Fizička kartaška igra | http://pesquisa.great.ufc.br/en.html | portugalski | Beppe et al. [178] |
| Test trainer | Java | N/A | engleski | Collofello et al. [180] |

**Tabela 7.4:** Detalji o upotrebi okruženja uključenih u istraživanje

| **Okruženje** | **Razvijen od** | **Godina** | **Institucije gde se koristi** | **Period upotrebe** |
|---|---|---|---|---|
| WReSTT-CyLE/STEM-CYLE | Florida International University | 2014 | Florida International University | 2014-sadašnjost |
| | | | Alabama A&M University | |
| | | | Florida A&M University | |
| | | | Florida Gulf Coast University | |
| | | | Georgia Southern University | |
| | | | Miami University | |
| Automated system | Rice University, Houston | 2016 | Rice University, Houston | 2016-2017 |
| SQLTest-GoRace | University of Oviedo | 2020 | University of Oviedo | 2020-2021 |
| Code Defenders | University of Sheffield | 2016 | University of Sheffield | 2017-sadašnjost |
| | | | University of Passau | |
| Testing game | University of Sao Paulo | 2016 | University of Sao Paulo | 2016-sadašnjost |
| Card game | OTH Regensburg | 2015 | OTH Regensburg | 2015-sadašnjost |
| Light views | Monash University, Caulfield, Australia | 2000 | Monash University, Caulfield, Australia | rane 2000-te |
| Android learning system | Anhui SanLian University, China | 2018 | Anhui SanLian University, China | 2018-2019 |
| Bug Hunt | University of Nebraska-Lincoln | 2004 | University of Nebraska-Lincoln and five other institutions | 2004-2005 |
| MARMOSET | University of Maryland | 2006 | University of Maryland | 2006-2011 |
| Bug Hide-and-Seek | California State University, Chico | 2017 | California State University, Chico | 2017-2018 |
| iLearnTest | University of Porto, Portugal | 2016 | University of Porto, Portugal | 2016-sadašnjost |
| TestEG | Federal University of Lavras, Brazil | 2015 | Federal University of Lavras, Brazil | N/A |
| JoVeTest | Universidade do Amazonas, Brazil | 2016 | Universidade do Amazonas, Brazil | 2016-sadašnjost |
| GreaTest | Universidade Federal do Ceara, Brazil | 2018 | Universidade Federal do Ceara, Brazil | 2018-sadašnjost |
| Test trainer | Arizona State University | 2005 | Arizona State University | 2005-2006 |

Na kraju, Tabela 7.5 pruža pregled tema koje pokriva svako okruženje o kojem se raspravlja, na osnovu identifikovanih tema predstavljenih u Tabeli 7.1. Treba napomenuti da su informacije o pokrivenim temama izvedene iz odgovarajućih publikacija koje su predstavile i opisale svaki alat, kao i sa zvaničnih veb stranica (ako postoje). Ovaj pristup evaluaciji je uzet jer su opisana okruženja konceptualizovana na različite načine, a ovo je određeno kao najmanji zajednički delilac. Moguće je zaključiti da nijedan od alata ne pokriva sve teme. Alati koji su postigli najveći procenat pokrivenih tema su obično alati zasnovani na programu ISTQB Foundation level ili alati koji pružaju dodatne dokumente za učenje (sa niskom interaktivnošću). Alati sa visokom interaktivnošću obično se fokusiraju na najvažnije praktične teme, koje uključuju crnu kutiju, belu kutiju i mutaciono testiranje. Takođe je moguće zaključiti da većina alata pokriva statičko testiranje i izveštavanje u nekoj meri, jer se od studenata traži da razumeju specifikaciju zahteva, analiziraju pokrivenost koda i osnovno izveštavanje o greškama. Dodatni zaključak koji se može izvući je da većina alata ne pokriva teme testiranja GUI-a, iako je to važan deo procesa testiranja sa specifičnim izazovima. Integracija i sistemsko testiranje su takođe podržani u ograničenoj meri, uglavnom pružanjem tutorijala o tome kako bi trebalo da se izvode, bez praktičnih zadataka koji bi mogli da uključuju testiranje performansi. Poslednje, ali ne manje važno, agilna metodologija se jedva spominje, i praktično samo od strane okruženja strukturiranih oko ISTQB Foundation level programa (kao što su dve opisane kartaške igre).

**Tabela 7.5:** Glavne teme koje pokrivaju posmatrana okruženja i njihov nivo interaktivnosti

| Okruženje/Teme | Crna kutija | Bela kutija | Mutaciono | Integraciono | Sistemsko | GUI testiranje | Statičko | Izveštavanje | % pokrivenosti | Interaktivnost |
|---|---|---|---|---|---|---|---|---|---|---|
| WReSTT-CyLE/STEM-CYLE | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | 87.5% | srednja |
| Automated system | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | 75% | srednja |
| SQLTest-GoRace | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | 75% | visoka |
| Code Defenders | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | 50% | visoka |
| Testing game | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | 50% | visoka |
| Card game | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | 87.5% | srednja |
| Light views | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | 50% | niska |
| Android learning system | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | 75% | niska |
| Bug Hunt | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | 50% | srednja |
| MARMOSET | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | 50% | srednja |
| Bug Hide-and-Seek | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | 37.5% | srednja |
| iLearnTest | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | 50% | visoka |
| TestEG | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | 75% | srednja |
| JoVeTest | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | 75% | srednja |
| GreaTest | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | 62.5% | srednja |
| Test trainer | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | 50% | niska |

## 7.5 Zaključna razmatranja postojećih rešenja

Svet se neprestano menja zbog napretka tehnologije, a obrazovanje mora da ga prati. Inovacije i njihovo dodavanje u proces učenja su neophodni za nove mlade generacije studenata, koje su oblikovane tehnologijom od rođenja. Obrazovanje se mora prilagoditi modernom svetu i prihvatiti nove tehnologije, ili bi moglo ostati daleko iza. Ova činjenica posebno važi za predmete iz oblasti računarstva i softverskog inženjeringa, koji moraju uvek ići u korak sa najnovijim trendovima, jer nema smisla učiti studente o starim i zastarelim tehnologijama. Isto važi i za testiranje softvera, koje je integralni deo polja softverskog inženjeringa, povezan sa svim ostalim predmetima programiranja.

Istraživanje predstavljeno u ovom članku ima za cilj da pruži pregled različitih okruženja, alata i igara koje bi se mogle koristiti kao dopunska sredstva tradicionalnoj nastavi testiranja softvera. Glavna motivacija za uključivanje dopunskih alata u časove testiranja softvera je ta što bi sirova predavanja mogla biti teška za razumevanje, monotona i dosadna. Alati imaju za cilj da povećaju angažovanje studenata, motivišu ih i daju im više praktičnog znanja koje mogu koristiti nakon što diplomiraju. Ovo istraživanje je pokušalo da pokrije sva okruženja koja su na jedan ili drugi način korišćena kao dopuna kursevima testiranja softvera u poslednjih dvadeset godina.

Ovo istraživanje je pokazalo da se većina alata koji se koriste kao dopuna kursevima testiranja softvera implementiraju kao alati za kolaborativno učenje ili igre. Međutim, mnogi nastavnici nisu baš voljni da dodaju bilo kakva dopunska obrazovna sredstva svojim časovima, uglavnom iz dva razloga. Prvo, ne veruju da bi ti alati doneli nešto korisno i poboljšali razumevanje studenata o temi. Drugo, mnogi nastavnici nemaju dovoljno tehničkog iskustva da koriste moderne tehnologije. S druge strane, mnogo veći broj nastavnika smatra ove alate korisnim i ukazuje na to da njihova upotreba može drastično povećati motivaciju u učionici ako se adekvatno kombinuju sa tradicionalnim predavanjima. Evaluacija alata koja je sprovedena u odnosu na broj pokrivenih tema testiranja softvera takođe pokazuje da nijedan alat ne pokriva kompletni set. Konačno, neke važne teme kao što su testiranje GUI-a i agilna metodologija razvoja softvera se jedva pominju i to samo u nekoliko alata.

Budući rad u ovoj oblasti treba da se fokusira na dalje učenje iz iskustava dobijenih iz praktične primene alata predstavljenih u ovom istraživanju. Svako okruženje koje je uključeno u istraživanje ispunjava svoju svrhu i uspešno se koristi kao dodatak nastavnim aktivnostima. Međutim, nijedan od njih ne pokriva potpuno sve teme navedene u smernicama ACM/IEEE. Krajnji cilj bi bio da se na tom znanju i naučenim lekcijama izgradi i implementira novo okruženje za učenje testiranja softvera koje bi moglo da se nosi sa nedostacima pronađenim u trenutno dostupnim okruženjima.

## 7.6 Primer laboratorijske vežbe

U sklopu ove doktorske disertacije, razvijen je i predlog laboratorijske vežbe u okviru koje studenti ispituju više različitih modela mašinskog učenja na JM1 NASA skupu podataka [29]. Na početku, od studenata je zahtevano da urade eksploratornu analizu podataka (engl. Exploratory Data Analysis, skr. EDA), kako bi razumeli podatke sa kojima rade. Od studenata je zatim konkretno zahtevano da ispitaju mogućnosti AdaBoost modela [183], koji nije ranije ispitan na ovom skupu podataka, i da uporede sa rezultatima drugih tradicionalnih modela mašinskog učenja. Nakon toga, potrebno je da studenti ispitaju više metaheurističkih algoritama kako bi optimizovali parametre AdaBoost modela

za konkretan skup podataka i dobili što je moguće bolje rezultate na posmatranom skupu podataka. Na kraju, urađena je evaluacija uticaja upotrebe ovakve laboratorijske vežbe na proces predavanja iz oblasti veštačke inteligencije i testiranja softvera, tako što je studentima koji su učestvovali u testiranju upotrebe modela mašinskog učenja dat anonimni upitnik u kome su ocenjivali različite aspekte ovakvog praktičnog primera u nastavi.

### 7.6.1 Eksploratorna analiza podataka

Skup podataka za predviđanje defekata sadrži brojne važne karakteristike i metrike o povezanom softveru, koje bi mogle biti iskorišćene za uspešno predviđanje da li određeni softverski modul ima defekte. Ovaj set karakteristika je uspostavljen još sedamdesetih godina sa ciljem da objektivno karakteriše svojstva koda relevantna za kvalitet softvera, i danas se još uvek koristi.

Karakteristike uključuju nekoliko metrika koje je predstavio Thomas McCabe [21], kao što su broj linija koda (LOC), ciklomatska složenost (tj. CC ili v(g)), esencijalna složenost (ev(g)), i dizajnerska složenost (iv(g)). Pored toga, posmatrane su i nekoliko metrika koje je predložio Maurice Halstead [22, 23], uključujući četiri osnovne i osam izvedenih metrika, kao što su ukupni operatori + operandi, dužina programa, procena vremena, broj linija, broj linija koje sadrže komentare, i prazne linije.

Konačno, skup podataka takođe razmatra jedinstvene operatore, jedinstvene operande, ukupne operatore, ukupne operande, i broj grana (neophodno za testiranje pokrivenosti grana/odluke, izvedeno iz kontrolnog toka grafa). Ciljna promenljiva je defekt, sa mogućim vrednostima tačno ili netačno, što ukazuje na to da li posmatrani modul ima ili nema jedan ili više prijavljenih defekata. Ovaj problem pripada klasičnim izazovima binarne klasifikacije. Toplotna mapa koja pokazuje korelaciju između karakteristika je prikazana na slici 7.6, dok su neki od najrelevantnijih indikatora prikazani na slici 7.7. Konačno, slika 7.8 prikazuje dijagram koji prikazuje distribuciju klasa posmatranog skupa podataka.

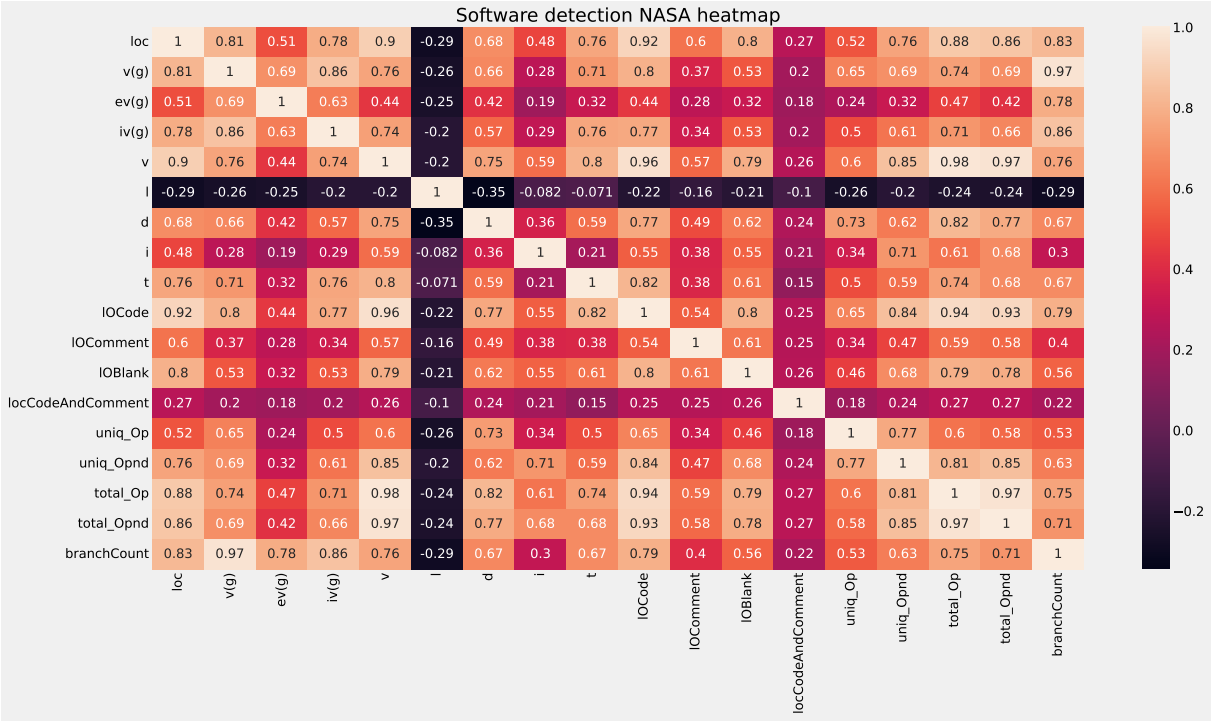
Takođe se može primetiti da je ovaj konkretni skup podataka nebalansiran, jer 77,5% skupa podataka predstavlja unose bez defekata, a preostalih 22,5% označava instance sa defektima.

### 7.6.2 Metrike koje se koriste u vežbi

U vežbi se koriste tradicionalne metrike mašinskog učenja bazirane na tačno pozitivnim (TP), tačno negativnim (TN), lažno pozitivnim (FP), i lažno negativnim (FN) predikcijama. Ovaj set metrika omogućava izračunavanje važnih indikatora mašinskog učenja kao što su tačnost, preciznost, osetljivost i F-skor. Za NASA skup podataka, koji je nebalansiran, optimizuje se Koenov kapa koeficijent.

### 7.6.3 Evaluacija tradicionalnih modela mašinskog učenja na NASA skupu

U ovom delu vežbe, od studenata se traži da testiraju nekoliko modela mašinskog učenja na NASA skupu podataka. Među posmatranim modelima je i AdaBoost, koji nije ranije ispitivan u ovom konkretnom scenariju. Ostvareni rezultati izvršavanja AdaBoost modela, sa podrazumevanim podešavanjima, prikazani su u tabeli 7.6. Može se uočiti da AdaBoost postiže vrlo osrednje rezulate sa podrazumevanim podešavanjima, čime se




**Slika 7.6:** Toplotna karta atributa PROMISE NASA skupa podataka

dodatno naglašava neophodnost podešavanja hiperparametara svakog modela mašinskog učenja za svaki konkretan problem.

**Tabela 7.6:** Poređenja različitih modela mašinskog učenja na NASA skupu podataka

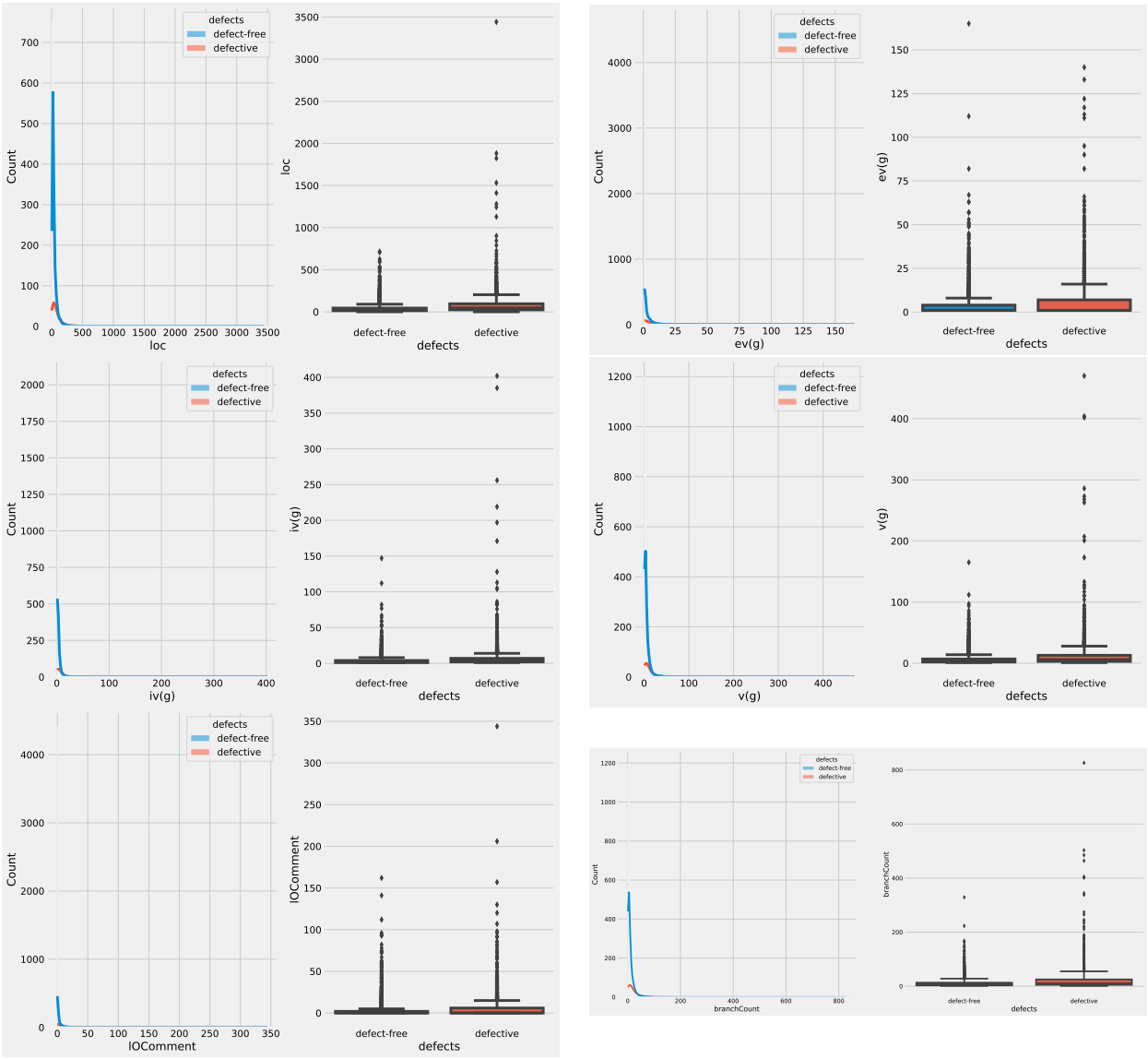
| **Methods** | accuracy | error rate | Cohen's kappa |
|---|---|---|---|
| AdaBoost | 68.58% | 0.314217 | 0.124913 |
| XGBoost | 74.34% | 0.256610 | 0.144685 |
| RF | 78.11% | 0.218856 | 0.089953 |
| SVM | 77.52% | 0.224841 | 0.012657 |
| DNN 1 hidden layer | 75.77% | 0.242277 | 0.0872 |
| DNN 2 hidden layers | 79.68% | 0.20324 | 0.09175 |

### 7.6.4 Simulacije sa optimizacijom AdaBoost modela

Studentima je dat zadatak da uz pomoć metaheurističkih algorithama optimizuju AdaBoost model za dati skup podataka o defektima softvera. Kolekcija hiperparametara modela AdaBoost koji se optimizuju, zajedno sa njihovim odgovarajućim ograničenjima prostora pretrage i tipovima promenljivih su navedeni u Tabeli 7.7.

**Tabela 7.7:** AdaBoost hiperparametri optimizovani u ovoj vežbi

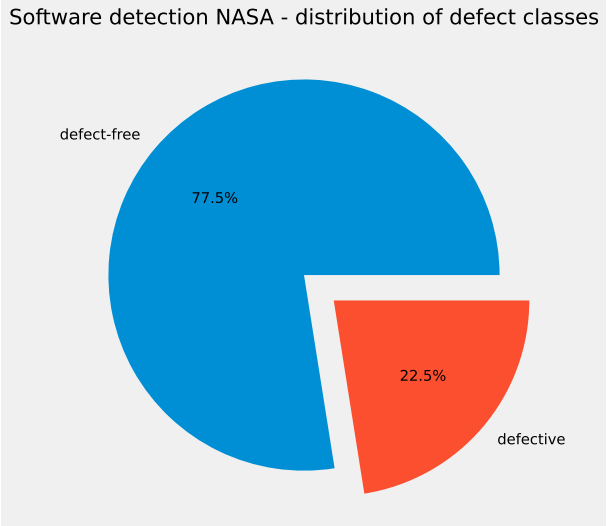
| AdaBoost hiperparametar | granice pretrage | tip promenljive |
|---|---|---|
| broj estimatora | $[10, 30]$ | celobrojna vrednost |
| dubina | $[1, 5]$ | celobrojna vrednost |
| stopa učenja | $[0.01, 2]$ | realna vrednost |




**Slika 7.7:** Vizuelizacija relevantnih atributa

Ostali brojni hiperparametri modela AdaBoost su fiksirani na podrazumevane vrednosti tokom eksperimenata. Opisani okvir je kreiran koristeći Python programski jezik, i oslanja se na često korišćene Python biblioteke za mašinsko učenje, kao što su numpy, scipy, i pandas. XGBoost model je implementiran koristeći Python biblioteku scikit-learn. Šema kodiranja koju koristi ovaj okvir je da se svaka jednika posmatra kao vektor dužine $l$, gde $l$ prikazuje broj optimizovanih hiperparametara. Kao što je ranije rečeno, tri AdaBoost hiperparametara je prošlo kroz proces optimizacije, stoga je u ovoj vežbi, $l$ bio fiksiran na 3.

Algoritmi odabrani za optimizaciju AdaBoost modela su: metod krda losova (engl. elk herding optimizer, skr. EHO)[184], GA [50], PSO [56], algoritam kojota (engl. coyote optimization algorithm, skr. COA) [185], SCA [62] i WOA [100]. Svaki od algoritama se izvršava sa 8 jedinki u populaciji, 10 iteracija po izvršavanju, i 20 nezavisnih izvršavanja (pošto su eksperimenti zahtevni). Koenov kapa koeficijent se koristi kao funkcija cilja u eksperimentima.




**Slika 7.8:** Distribucija klasa

### 7.6.5 Rezultati simulacija

Rezultati dobijeni nakon izvršavanja navedenih algoritama su dati u nastavku. Tabela 7.8 prikazuje rezultate funkcije cilja (Koenov kapa koeficijent), dok su rezultati indikatorske funkcije (stopa greške u klasifikaciji) dati u Tabeli 7.9.

**Tabela 7.8:** Rezultati funkcije cilja za sve posmatrane algoritme.

| Metoda | Best | Worst | Mean | Median | Std | Var |
|---|---|---|---|---|---|---|
| AB-EHO | 0.195173 | 0.163105 | 0.181935 | 0.185901 | 0.012611 | 1.56E-04 |
| AB-GA | 0.196269 | 0.181283 | 0.188697 | 0.188885 | 0.005392 | 2.91E-05 |
| AB-PSO | 0.181253 | 0.173224 | 0.177188 | 0.177096 | 0.002693 | 7.41E-06 |
| AB-COA | 0.261202 | 0.165103 | 0.194524 | 0.187268 | 0.031657 | 1.02E-03 |
| AB-SCA | 0.188107 | 0.167954 | 0.178706 | 0.180724 | 0.006498 | 4.26E-05 |
| AB-WOA | 0.194548 | 0.162723 | 0.180287 | 0.179703 | 0.010844 | 1.14E-04 |

**Tabela 7.9:** Rezultati indikator funkcije - stopa greške u klasifikaciji.

| Metoda | Best | Worst | Mean | Median | Std | Var |
|---|---|---|---|---|---|---|
| AB-EHO | 0.245424 | 0.229652 | 0.245205 | 0.247094 | 0.014826 | 2.19E-04 |
| AB-GA | 0.245424 | 0.243887 | 0.235703 | 0.234203 | 0.007305 | 5.28E-05 |
| AB-PSO | 0.237196 | 0.225156 | 0.230794 | 0.231027 | 0.007991 | 6.39E-05 |
| AB-COA | 0.234533 | 0.254802 | 0.234767 | 0.232708 | 0.009914 | 9.79E-05 |
| AB-SCA | 0.231195 | 0.234201 | 0.226607 | 0.225816 | 0.007357 | 5.29E-05 |
| AB-WOA | 0.252905 | 0.233103 | 0.235495 | 0.230667 | 0.013092 | 1.69E-04 |

Zatim, potrebno je prikazati detaljne metrike za najbolje izvršavanje svakog od algoritama, kao što je prikazano u Tabeli 7.10. Može se uočiti da je accuracy značajno poboljšan, na primer, iznosi oko 75.4% u slučaju EHO i GA algoritama, što je značajno poboljšanje rezultata osnovnog neoptimizovanog AdaBoost modela.

Na kraju, od studenata se traži da prikažu vrednosti hiperparametara AdaBoost modela za svaki od korišćenih algoritama, kao što je prikazano u Tabeli 7.11.

Od studenata je na kraju zahtevano i da vizuelizuju rezultate eksperimenata, prikazom odgovarajućih box plot dijagrama za sve posmatrane metaheurističke algoritme, kao što je prikazano na Slici 7.9.

**Tabela 7.10:** Detaljne metrike najboljih izvršavanja algoritama.

| Metoda | Metrika | Bez defekata | Sa defektima | Accuracy | Macro avg | Weighted avg |
|---|---|---|---|---|---|---|
| AB-EHO | precision | 0.809982 | 0.429306 | 0.754583 | 0.619644 | 0.724391 |
| | recall | 0.892857 | 0.277870 | 0.754583 | 0.585364 | 0.754583 |
| | f1-score | 0.849403 | 0.337374 | 0.754583 | 0.593388 | 0.734278 |
| AB-GA | precision | 0.810254 | 0.429668 | 0.754583 | 0.619961 | 0.724683 |
| | recall | 0.892375 | 0.279534 | 0.754583 | 0.585954 | 0.754583 |
| | f1-score | 0.849334 | 0.338710 | 0.754583 | 0.594022 | 0.734525 |
| AB-PSO | precision | 0.773789 | 0.216066 | 0.698466 | 0.494928 | 0.648390 |
| | recall | 0.863417 | 0.129784 | 0.698466 | 0.496600 | 0.698466 |
| | f1-score | 0.816150 | 0.162162 | 0.698466 | 0.489156 | 0.669106 |
| AB-COA | precision | 0.773789 | 0.216066 | 0.698466 | 0.494928 | 0.648390 |
| | recall | 0.863417 | 0.129784 | 0.698466 | 0.496600 | 0.698466 |
| | f1-score | 0.816150 | 0.162162 | 0.698466 | 0.489156 | 0.669106 |
| AB-SCA | precision | 0.773794 | 0.217391 | 0.687617 | 0.495593 | 0.648692 |
| | recall | 0.843629 | 0.149750 | 0.687617 | 0.496690 | 0.687617 |
| | f1-score | 0.807204 | 0.177340 | 0.687617 | 0.492272 | 0.665585 |
| AB-WOA | precision | 0.811607 | 0.413395 | 0.747101 | 0.612501 | 0.722073 |
| | recall | 0.877413 | 0.297837 | 0.747101 | 0.587625 | 0.747101 |
| | f1-score | 0.843228 | 0.346228 | 0.747101 | 0.594728 | 0.731482 |

**Tabela 7.11:** Parametri najboljeg AdaModela dobijeni sa svakim od posmatranih algoritama.

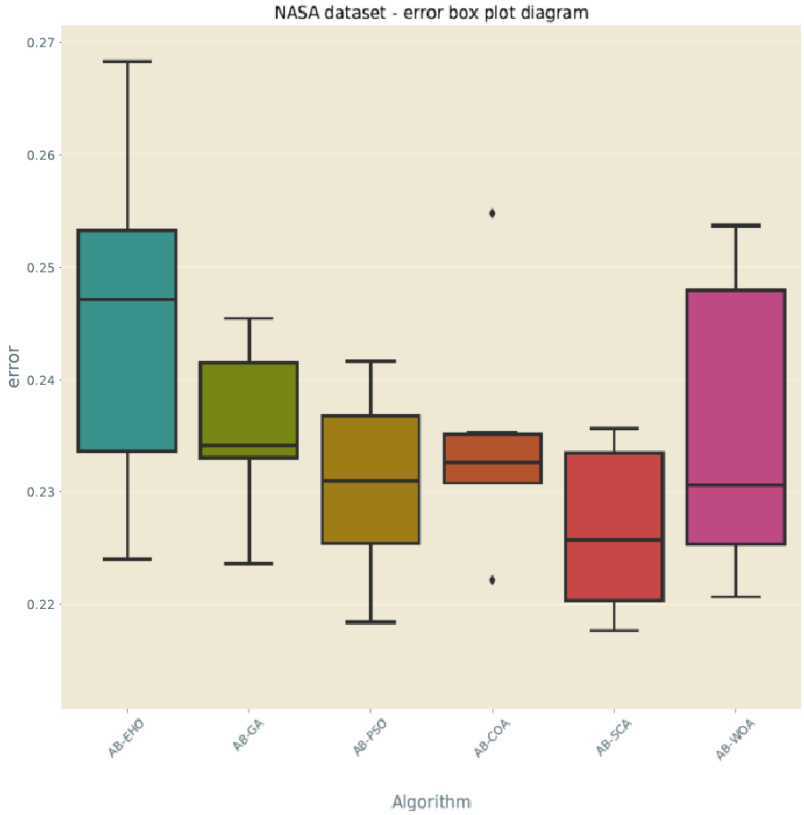
| Metoda | Broj estimatora | dubina | Stopa učenja |
|---|---|---|---|
| AB-EHO | 30 | 4 | 1.401188 |
| AB-GA | 30 | 5 | 1.026934 |
| AB-PSO | 22 | 4 | 1.144578 |
| AB-COA | 24 | 2 | 1.997424 |
| AB-SCA | 12 | 4 | 1.741467 |
| AB-WOA | 30 | 5 | 1.328102 |

### 7.6.6 Studentska evaluacija predloženog rešenja

U cilju evaluacije predložene metodologije u procesu predavanja iz predmeta veštačke inteligencije i testiranja softvera, studenti su testirali predloženo rešenje. Nakon rada sa okruženjem, studenti su dobili anonimni upitnik, u kome su davali ocene (u rasponu 1-5) za različite aspekte predloženog rešenja. Ukupni utisci ispitanika, sa prosečnim vrednostima iz svih poslatih anketa, dati su u Tabeli 7.12.

**Tabela 7.12:** Lakoća korišćenja i prednosti upotrebe predloženog rešenja.

| | Prosečna ocena |
|---|---|
| Lakoća korišćenja (1 - teško, 5 - lako) | 3.9 |
| Razumevanje tema u odnosu na materijale koji nisu uključeni u vežbu (1 - mnogo teže, 5 - mnogo lakše) | 4.4 |
| Težina materijala (1 - lako, 5 - teško) | 4.1 |
| Efekat upotrebe predloženog rešenja na učenje (1 - nema uticaj, 5 - značajan uticaj) | 4.5 |
| Uticaj predloženog rešenja na zainteresovanost studenata za temu (1 - nema, 5 - značajan) | 4.2 |




**Slika 7.9:** Vizuelizacija box plot dijagrama za sve posmatrane algoritme.

Iako su materijali upotrebljeni u predloženoj vežbi mnogo složeniji i teži od drugih dostupnih materijala koji se koriste na konvencionalnim kursevima, studenti su uglavnom procenili razumevanje materijala znatno superiornije nego za druge teme iz ovih oblasti, koje nisu pokrivene ovom vežbom. Ovakvo rešenje je značajan alat koji može pomoći naročito u slučaju težih tema, gde su prednosti mogućnosti vizuelizacije i praktičnih primera na konkretnim skupovima podataka velike.

Uopšteniji odgovori o uticaju koji predloženo rešenje ima na tradicionalni proces učenja su priloženi u Tabeli 7.13. Dobijeni podaci indikuju značaj upotrebljavanja odgovarajućeg alata u paraleli sa tradicionalnim materijalima za kurs. Sveukupno gledano, nivo razumevanja studenata, kao i njihova zainteresovanost za temu su bili značajno veći nego bez upotrebe predloženog rešenja. Studenti su imali opciju da dodaju i komentare u obliku slobodnog teksta, gde su naročito podvukli da je ovakav pristup dobar način da se studentima pomogne u razumevanju materijala, jer na ovakav način mogu da dobiju praktično znanje i veliki broj odgovora na pitanja u vezi tradicionalnih teorijskih materijala.

**Tabela 7.13:** Značaj upotrebe praktičnih vežbi u okviru kurseva veštačke inteligencije i testiranja softvera.

| | Prosečna ocena |
|---|---|
| Značaj upotrebe alata koji predstavlja podršku za kurs (1 - mali, 5 - veliki) | 4,7 |
| Značaj mogućnosti vizuelizacije (1 - nema, 5 - veoma važno) | 4.8 |
| Značaj lake pripreme i realizacije primera za vežbu (1 - nema, 5, veoma veliki) | 4.2 |
| Značaj mogućnosti samostalnog podešavanja modela ML (1 - nema uticaj, 5 - značajan uticaj) | 3.9 |
| Poverenje u materijale za kurs (1 - nema, 5 - značajan) | 3.7 |
| Relevantnost predloženog rešenja (1 - nema, 5 - visoko) | 4.1 |

# Poglavlje 8

# Zaključak

Zadatak predviđanja defekata je od vitalnog značaja u razvoju softvera, jer može pomoći u fokusiranijem testiranju modula sklonih greškama, a to može rezultirati efikasnim procesom testiranja i ranim otkrivanjem defekata. Osim toga, to može uštedeti vreme, novac i sprečiti brojne probleme nakon implementacije softvera. Pristup koji je izabran u ovoj tezi koristi hibridnu strukturu predviđanja defekata softvera koja bi mogla biti primenjena za ovaj važan izazov. Uvedeni model se oslanja na novi HARSA metaheuristički algoritam, razvijen dodavanjem FA postupka pretrage osnovnom RSA algoritmu. Ovaj hibridni algoritam niskog nivoa koristi prednosti snažne eksploracije RSA i snažne eksploatacije FA, gde jake tačke svakog algoritma prevazilaze njihove odgovarajuće nedostatke. HARSA metod počinje izvršavanje koristeći RSA postupak pretrage, ali nakon određenog broja iteracija, počinje da rotira RSA i FA postupke pretrage kako bi postigao snažniju eksploataciju.

Ovaj novi HARSA algoritam je implementiran kao komponenta sistema mašinskog učenja, i upotrebljen je da optimizuje hiperparametre XGBoost strukture. Ovaj pristup je nazvan XG-HARSA, i evaluiran je na dva značajna skupa podataka za klasifikaciju defekata softvera (NASA i GHPR). Važna razlika između ova dva skupa podataka je da je NASA nebalansiran, dok je GHPR savršeno balansiran skup podataka. Dobijeni rezultati su upoređeni sa rezultatima dobijenim od osam konkurentnih modernih snažnih optimizacionih algoritama, korišćenih u identičnoj konfiguraciji sa identičnim ciljem da se fino podese hiperparametri XGBoost-a. U slučaju NASA skupa podataka, uvedeni XG-HARSA model postigao je vrhunsku tačnost klasifikacije od 79,39% (stopa greške kao funkcija cilja) i nešto nižu tačnost od 78,26% i najbolji i značajno poboljšani F1 rezultat kada je Koenova kapa korišćena kao funkcija cilja. Na GHPR skupu podataka, koji je balansiran, samo je stopa greške korišćena kao funkcija cilja, gde je predloženi XG-HARSA postigao najbolju tačnost klasifikacije od 81,33%.

Na kraju, sprovedena je SHAP analiza uz interpretaciju rezultata najboljeg modela (XG-HARSA) na oba skupa podataka, sa ciljem razumevanja eksperimentalnih rezultata i određivanja najznačajnijih karakteristika. Ova analiza potvrdila je posmatranja iz prakse, gde su najvažnije karakteristike svojstva koda koji ukazuju na složenost, poput broja linija koda, ukupnog broja operanada, nivoa ugnježavanja, dubine stabla nasleđivanja, broja zavisnosti koje posmatrana klasa poseduje i broja metoda u klasi. Ako je kod izuzetno složen, vrlo je teško pravilno ga testirati i pokriti sve scenarije, što dovodi do povećane šanse da će neki defekti ostati nepokriveni, a moguće je da će se preneti u fazu produkcije gde bi ih neizbežno primetili krajnji korisnici. Jasna preporuka ovde bi bila da se prate ove ključne metrike, i ako se složenost povećava tokom razvoja, preporučuje se refaktorisanje

koda.

Osnovni cilj ove teze bila je realizacija sistema za predikciju defekata u softverskim modulima, primenom modela mašinskog učenja čiji su hiperparametri optimizovani pomoću metaheurističkih algoritama. Realizovani sistem je sposoban da na osnovu skupa softverskih metrika efikasno i precizno predvidi da li je posmatrani modul podložan greškama ili ne.

Za određivanje odgovarajućih vrednosti hiperparametara modela za konkretan problem klasifikacije upotrebljeni su metaheuristički algoritmi, zato što su sposobni da efikasno pretraže prostor mogućih vrednosti hiperparametara i pronađu one vrednosti koje daju najbolje performanse modela. Optimizovani sistem radi binarnu klasifikaciju na osnovu skupa softverskih metrika za posmatrani modul.

Rezultati različitih metaheurističkih algoritama za ovaj konkretan problem su analizirani. Data je i interpretacija najboljih modela primenom SHAP metode, koja omogućava dublje razumevanje kako posmatrani model donosi odluke, i što može biti korisno za buduća dalja unapređenja modela. Obavljena je i detaljna statistička analiza primenom statističkih testova koji su pokazali koji modeli su najbolji za ovaj konkretan problem klasifikacije.

Realizovano rešenje se može koristiti u planiranju procesa testiranja, pošto omogućava identifikovanje modula podložnih greškama na osnovu specifičnih metrika, nakon čega se testiranje može fokusirati na njih. Dodatno, implementirana je i laboratorijska vežba koja ilustruje ceo proces optimizacije hiperparametara modela mašinskog učenja na konkretnom primeru skupa podataka za identifikaciju defekata u modulima.

Ostvareni nivo performansi modela mašinskog učenja na problemu klasifikacije softverskih modula podložnim greškama je vrlo obećavajući, a budući eksperimenti bi trebali biti usmereni na uspostavljanje još većeg nivoa poverenja u predloženi XG-HARSA model. Ovaj zadatak će definitivno uključivati testiranje na dodatnim skupovima podataka i metrikama iz živih projekata, kao i uzimanje u obzir dodatnih podataka i metrika koje su dostupne na otvorenim Git repozitorijumima.

Predloženi algoritam se može testirati i na drugim problemima klasifikacije, pošto je neophodno optimizovati hiperparametre modela na svakom pojedinačnom problemu zasebno. Dodatno, moguće su i dalje modifikacije samog algoritma RSA, kako bi se ostvarile još bolje performanse.

# Prilozi

# Prilog A

# Primeri koda

## A.1 Implementacija osnovnog RSA algoritma u Python programskom jeziku

**Listing A.1:** Osnovni RSA algoritam

```
1
2  from utilities.solutionFSHT import Solution
3  from utilities.fs_utils import transfer_function
4  import copy
5  import numpy as np
6  import sys
7
8  #faFSHT - firefly feature selection hyper-parameters
       tuning
9  #implementacija algoritma koja je prilagodjena i za fs i
       za hipertuning
10 class RSA:
11     def __init__(self, n, function, tf, nfeatures):
12
13         self.N = n
14         self.function = function
15         self.population = []
16         self.best_solution = [None] * self.function.D
17         self.gamma = 1          # fixed light absorption
              coefficient
18         self.beta_zero = 1          # attractiveness at
              distance zero
19         self.beta = 1
20         self.alpha = 0.5          # randomization
              parameter
21         self.FFE = self.N;  #parametar za fitness
              function evaluations, omdah se racuna za fazu
              inicijalizacije i jednak je broju resenja
22         self.tf = tf; # ovo je transfer function za
              transformaciju u binary space
```

```
23          self.nfeatures = nfeatures; #ovo je max broj
               features u datasetu, prvih nfeatures
               parametara su za features, ostali su za hiper-
               param optimiation
24
25
26      def initial_population(self):
27          for i in range(0, self.N):
28              local_solution = Solution(self.function,self
                   .tf,self.nfeatures)
29              self.population.append(local_solution)
30          self.population.sort(key=lambda x: x.
               objective_function)
31          self.best_solution = copy.deepcopy(self.
               population[0].x)
32
33      def update_position(self, t, max_iter):
34
35          delta = 1 - (10**(-4)/0.9)**(1/max_iter)
36          #self.alpha = ( 1- delta) * self.alpha
37
38          lb = self.function.lb
39          ub = self.function.ub
40
41          scale = []
42          for i in range(self.function.D):
43              scale.append(np.abs(ub[i] - lb[i]))
44
45          #temp_population = copy.deepcopy(self.population
               )
46          temp_population = self.population.copy()
47
48          for i in range(self.N):
49
50              for j in range(self.N):
51
52                  if self.population[i].objective_function
                       > temp_population[j].
                      objective_function:
53                      #print(self.nfeatures)
54                      r = np.sqrt(np.sum((np.array(self.
                           population[i].x) - (np.array(
                           temp_population[j].x)))**2))
55                      beta = (self.beta - self.beta_zero)
                           * np.exp(-self.gamma * r ** 2) +
                           self.beta_zero
56                      temp = self.alpha * (np.random.rand(
                           self.function.D) - 0.5) * scale
```

```
57                      sol = np.array(self.population[i].x)
                            * (1 - beta) + np.array(
                           temp_population[j].x) * beta +
                           temp
58
59                      sol[0:self.nfeatures] =
                           transfer_function(sol[0:self.
                           nfeatures], self.tf, self.
                           nfeatures)  #ovde koristimo
                           transfer funckiju za prvih
                           nfeatures parametara
60                      sol = self.checkBoundaries(sol)
61
62                      solution = Solution(self.function,
                           self.tf,self.nfeatures,sol.tolist
                           ())
63                      self.FFE = self.FFE + 1
64
65                      if solution.objective_function <
                           self.population[i].
                           objective_function:
66                          self.population[i] = solution
67          #self.qr()
68      def sort_population(self):
69
70          self.population.sort(key=lambda x: x.
               objective_function)
71          self.best_solution = self.population[0].x
72
73      def get_global_best(self):
74          return (self.population[0].objective_function,
               self.population[0].error,self.population[0].
               y_proba, self.population[0].y,self.population
               [0].feature_size,
75                    self.population[0].model)
76
77
78          #return self.population[0].objective_function
79
80      def get_global_worst(self):
81          return self.population[-1].objective_function
82
83      def optimum(self):
84          print('f(x*) = ', self.function.minimum, 'at x* =
               ', self.function.solution)
85
86      def algorithm(self):
87          return 'FA'
```

```
88
89      def objective(self):
90
91          result = []
92
93          for i in range(self.N):
94              result.append(self.population[i].
                    objective_function)
95
96          return result
97
98      def average_result(self):
99          return np.mean(np.array(self.objective()))
100
101     def std_result(self):
102         return np.std(np.array(self.objective()))
103
104     def median_result(self):
105         return np.median(np.array(self.objective()))
106
107
108     def print_global_parameters(self):
109             for i in range(0, len(self.best_solution)):
110                 print('X:␣{}'.format(self.best_solution
                        [i]))
111
112     def get_best_solutions(self):
113         return np.array(self.best_solution)
114
115     def get_solutions(self):
116
117         sol = np.zeros((self.N, self.function.D))
118         for i in range(len(self.population)):
119             sol[i] = np.array(self.population[i].x)
120         return sol
121
122
123     def print_all_solutions(self):
124         print("******all␣solutions␣objectives**********"
                )
125         for i in range(0,len(self.population)):
126               print('solution␣{}'.format(i))
127               print('objective:{}'.format(self.
                      population[i].objective_function))
128               print('solution␣{}:␣'.format(self.
                      population[i].x))
129               print('
                      ------------------------------------')
```

```
130
131     def get_global_best_params(self):
132         return self.population[0].x
133
134     #proverava boundaries za hiper-parametre
135     def checkBoundaries(self,Xnew):
136         for j in range(self.nfeatures,self.function.D):
137             if Xnew[j] < self.function.lb[j]:
138                 Xnew[j] = self.function.lb[j]
139
140             elif Xnew[j] > self.function.ub[j]:
141                 Xnew[j] = self.function.ub[j]
142         return Xnew
143
144
145
146     def getFFE(self):
147         return self.FFE
148
149     # funkcija koja vraca najbolji global_best_solution
150     def get_global_best_solution(self):
151         # ovde pravimo liste sa objective i indicator za
                celu populaciji
152
153         indicator_list = []  # ovo je indikator, sta god
                da je u pitanju
154         objective_list = []  # ovo je objective, sta god
                da je u pitanju
155         objective_indicator_list = []
156         for i in range(len(self.population)):
157             indicator_list.append(
158                 self.population[i].error)  # ovo je za
                    error, mada je to bilo koji drugi
                    indikator, samo se tako zoeve
159             objective_list.append(self.population[i].
                   objective_function)  # ovo je objective
160         objective_indicator_list.append(objective_list)
161         objective_indicator_list.append(indicator_list)
162         self.population[0].diversity =
               objective_indicator_list
163
164         return self.population[0]
165
166
167
168
169     def qr(self):
170     #metoda za quasi-reflextive learning
```

```
171
172         lb = self.function.lb
173         ub = self.function.ub
174         qr_solution = [None] * self.function.D
175         for i in range(self.N):
176             for j in range(self.function.D):
177                 if self.population[i].x[j] < (ub[j] + lb
                        [j]) / 2:
178                     qr_solution[j] = self.population[i].
                            x[j] + (
179                             (ub[j] + lb[j]) / 2 - self.
                                population[i].x[j]) * np.
                                random.uniform()
180                 else:
181                     qr_solution[j] = (ub[j] + lb[j]) / 2
                            + (
182                             self.population[i].x[j] - (
                                ub[j] + lb[j]) / 2) * np.
                                random.uniform()
183             qr_solution_add = Solution(self.function,
                    qr_solution)
184             self.population.append(qr_solution_add)
185         self.population.sort(key=lambda x: x.
                objective_function)
186         # delete elements from population that are not
                needed
187         del self.population[(self.N):len(self.population
                )]
188         # self.FFE = self.FFE + self.N  # dodajemo FFE
                za QRBL mechanism
```

## A.2 Implementacija predloženog HARSA algoritma u Python programskom jeziku

**Listing A.2:** Predloženi HARSA algoritam

```
1
2  from utilities.solutionFSHT import Solution
3  from utilities.fs_utils import transfer_function
4  import copy
5  import numpy as np
6  import sys
7
8  #implementacija algoritma koja je prilagodjena i za fs i
       za hipertuning
9  class HARSA:
10     def __init__(self, n, function, tf, nfeatures):
```

```
11
12          self.N = n
13          self.function = function
14          self.population = []
15          self.best_solution = [None] * self.function.D
16          self.gamma = 1          # fixed light absorption
               coefficient
17          self.beta_zero = 1          # attractiveness at
               distance zero
18          self.beta = 1
19          self.alpha = 0.5          # randomization
               parameter
20          self.FFE = self.N;  #parametar za fitness
               function evaluations, omdah se racuna za fazu
               inicijalizacije i jednak je broju resenja
21          self.tf = tf; # ovo je transfer function za
               transformaciju u binary space
22          self.nfeatures = nfeatures; #ovo je max broj
               features u datasetu, prvih nfeatures
               parametara su za features, ostali su za hiper-
               param optimiation
23
24
25      def initial_population(self):
26          for i in range(0, self.N):
27              local_solution = Solution(self.function,self
                   .tf,self.nfeatures)
28              self.population.append(local_solution)
29          self.population.sort(key=lambda x: x.
               objective_function)
30          self.best_solution = copy.deepcopy(self.
               population[0].x)
31
32      def update_position(self, t, max_iter):
33
34          delta = 1 - (10**(-4)/0.9)**(1/max_iter)
35          #self.alpha = ( 1- delta) * self.alpha
36
37          lb = self.function.lb
38          ub = self.function.ub
39
40          scale = []
41          for i in range(self.function.D):
42              scale.append(np.abs(ub[i] - lb[i]))
43
44          #temp_population = copy.deepcopy(self.population
               )
45          temp_population = self.population.copy()
```

```
46
47          #ovo je tweak radimo duplo u poslednjim
               iteracijama
48          if t>int(max_iter/2):
49              k = 2
50          else:
51              k =1
52
53          for p in range(k):
54
55              for i in range(self.N):
56
57                  for j in range(self.N):
58
59                      if self.population[i].
                           objective_function >
                           temp_population[j].
                           objective_function:
60                          #print(self.nfeatures)
61                          r = np.sqrt(np.sum((np.array(
                               self.population[i].x) - (np.
                               array(temp_population[j].x)))
                               **2))
62                          beta = (self.beta - self.
                               beta_zero) * np.exp(-self.
                               gamma * r ** 2) + self.
                               beta_zero
63                          temp = self.alpha * (np.random.
                               rand(self.function.D) - 0.5) *
                                scale
64                          sol = np.array(self.population[i
                               ].x) * (1 - beta) + np.array(
                               temp_population[j].x) * beta +
                                temp
65
66                          sol[0:self.nfeatures] =
                               transfer_function(sol[0:self.
                               nfeatures], self.tf, self.
                               nfeatures)  #ovde koristimo
                               transfer funckiju za prvih
                               nfeatures parametara
67                          sol = self.checkBoundaries(sol)
68
69                          solution = Solution(self.
                               function, self.tf,self.
                               nfeatures,sol.tolist())
70                          self.FFE = self.FFE + 1
71
```

```
72                          if solution.objective_function <
                                self.population[i].
                               objective_function:
73                              self.population[i] =
                                   solution
74
75          self.qr()
76          #self.qr()
77      def sort_population(self):
78
79          self.population.sort(key=lambda x: x.
               objective_function)
80          self.best_solution = self.population[0].x
81
82      def get_global_best(self):
83          return (self.population[0].objective_function,
               self.population[0].error,self.population[0].
               y_proba, self.population[0].y,self.population
               [0].feature_size,
84                   self.population[0].model)
85
86
87          #return self.population[0].objective_function
88
89      def get_global_worst(self):
90          return self.population[-1].objective_function
91
92      def optimum(self):
93          print('f(x*) = ', self.function.minimum, 'at x* =
               ', self.function.solution)
94
95      def algorithm(self):
96          return 'FA'
97
98      def objective(self):
99
100         result = []
101
102         for i in range(self.N):
103             result.append(self.population[i].
                   objective_function)
104
105         return result
106
107     def average_result(self):
108         return np.mean(np.array(self.objective()))
109
110     def std_result(self):
```

```
111         return np.std(np.array(self.objective()))
112
113     def median_result(self):
114         return np.median(np.array(self.objective()))
115
116
117     def print_global_parameters(self):
118             for i in range(0, len(self.best_solution)):
119                 print('X: {}'.format(self.best_solution
                        [i]))
120
121     def get_best_solutions(self):
122         return np.array(self.best_solution)
123
124     def get_solutions(self):
125
126         sol = np.zeros((self.N, self.function.D))
127         for i in range(len(self.population)):
128             sol[i] = np.array(self.population[i].x)
129         return sol
130
131
132     def print_all_solutions(self):
133         print("******all solutions objectives**********"
                )
134         for i in range(0,len(self.population)):
135               print('solution {}'.format(i))
136               print('objective:{}'.format(self.
                      population[i].objective_function))
137               print('solution {}: '.format(self.
                      population[i].x))
138               print('
                      ---------------------------------------')
139
140     def get_global_best_params(self):
141         return self.population[0].x
142
143     #proverava boundaries za hiper-parametre
144     def checkBoundaries(self,Xnew):
145         for j in range(self.nfeatures,self.function.D):
146             if Xnew[j] < self.function.lb[j]:
147                 Xnew[j] = self.function.lb[j]
148
149             elif Xnew[j] > self.function.ub[j]:
150                 Xnew[j] = self.function.ub[j]
151         return Xnew
152
153
```

```
154
155     def getFFE(self):
156         return self.FFE
157
158     # funkcija koja vraca najbolji global_best_solution
159     def get_global_best_solution(self):
160         # ovde pravimo liste sa objective i indicator za
                celu populaciji
161
162         indicator_list = []  # ovo je indikator, sta god
                da je u pitanju
163         objective_list = []  # ovo je objective, sta god
                da je u pitanju
164         objective_indicator_list = []
165         for i in range(len(self.population)):
166             indicator_list.append(
167                 self.population[i].error)  # ovo je za
                    error, mada je to bilo koji drugi
                    indikator, samo se tako zoeve
168             objective_list.append(self.population[i].
                   objective_function)  # ovo je objective
169         objective_indicator_list.append(objective_list)
170         objective_indicator_list.append(indicator_list)
171         self.population[0].diversity =
               objective_indicator_list
172
173         return self.population[0]
174
175
176
177
178     def qr(self):
179     #metoda za quasi-reflextive learning
180
181         lb = self.function.lb
182         ub = self.function.ub
183         qr_solution = [None] * self.function.D
184         for i in range(self.N):
185             for j in range(self.function.D):
186                 if self.population[i].x[j] < (ub[j] + lb
                       [j]) / 2:
187                     qr_solution[j] = self.population[i].
                           x[j] + (
188                             (ub[j] + lb[j]) / 2 - self.
                                   population[i].x[j]) * np.
                                   random.uniform()
189                 else:
```

```
190                     qr_solution[j] = (ub[j] + lb[j]) / 2
                            + (
191                             self.population[i].x[j] - (
                                   ub[j] + lb[j]) / 2) * np.
                                   random.uniform()
192             qr_solution_add = Solution(self.function,
                   self.tf,self.nfeatures,qr_solution)
193             self.population.append(qr_solution_add)
194         self.population.sort(key=lambda x: x.
               objective_function)
195         # delete elements from population that are not
               needed
196         del self.population[(self.N):len(self.population
               )]
197         # self.FFE = self.FFE + self.N  # dodajemo FFE
               za QRBL mechanism
```

## A.3 Optimizacija XGBoost modela (stopa greške) u Python programskom jeziku

**Listing A.3:** XGBoost (Error min)

```
1
2
3   #XGBOOST funkcija za feature selection i tuning
4
5
6   import math
7   import numpy as np
8   from utilities.load_dataset import load_dataset
9   from ml_models.ELM import  ELM
10  import xgboost as xgb
11  import matplotlib.pyplot as plt
12  from sklearn.preprocessing import LabelEncoder
13  from sklearn.metrics import accuracy_score
14  from sklearn.model_selection import train_test_split
15  import matplotlib.pyplot as plt
16  from sklearn import metrics
17  from sklearn.metrics import roc_auc_score
18  from xgboost import XGBClassifier
19  #ovo je za cohen kappa score
20  from sklearn.metrics import cohen_kappa_score
21
22  class XGBoostFunction:
23      def __init__(self, D, no_classes,nfeatures,
           fitness_type,alpha,x_train,x_test,y_train,y_test,
           intParams,bounds,features_list=None):
```

```
24          #D je broj parametara resenja (broj
               hiperparametara koji se optimizuju)
25          #bounds je dictionary sa lower and upper bounds,
                ovaj dictionary se ubacuje iz spoljnog koda
26          #parametri za tuning
27          # x[0] - learning rate (lr), eta parametar float
28          # x[1] - min_child_weight, float
29          #x[2] - subsample, float
30          #x[3] - collsample_bytree
31          #x[4] - max_depth
32          #x[5] - gamma
33
34          #postavljamo datasetove
35          self.x_train = x_train
36          self.x_test = x_test
37          self.y_train = y_train
38          self.y_test = y_test
39
40          self.y_train1  = np.zeros(shape=len(y_train))
41          self.y_test1  = np.zeros(shape=len(y_test))
42
43          self.nfeatures = nfeatures
44
45          self.intParams = intParams
46
47          # koji objective koristimo, ovo mora biti prvi
               parametar koji se vraca
48          # moze kombinacija broja features i error rate-a
               , posto se radi o minimizaciji
49          # fitness type - 0 je samo error, a 1 je
               composite error i number of features
50          self.fitness_type = fitness_type
51          # ovo su weighted coefficient za error i broj
               selektovanih features
52          self.alpha = alpha
53          self.beta = 1 - alpha  # alpha je obicnao za
               error, a beta je obicno za broj feature-a
54
55          # ako se radi bez feature selection, onda se
               prosledjuje features list
56          # dakle, ako je features_list=None (po default),
                onda se radi fs i optimizacija, a ako ima
               necega, onda se radi samo optimizacija
57          # za svaki slucaj konvertujemo sve elemente u
               int
58          self.features_list = [int(x) for x in
               features_list]
59          print(self.features_list)
```

```
60
61
62          #converting y_test and y_train to single label
               classification
63
64          #fajl za load datasetova kodira target varijablu
               , pa pravi kolone u zavisnosti broja klasa,
               npr. 0 0 1, 0 1 0, itd.
65          #medjutim, XGBoost kao i DMatrix ne koriste one
               hot encoding, vec se koriste u target koloni
               klase, 0,1,2,3, itd.
66          #zbog toga pravimo pomocne setove y_test1 i
               y_train1 gde se koriste 0,1,2,3,4 za target,
               umesto one hot encoding
67
68          for i in range(len(self.y_test)):
69              self.y_test1[i] = np.argmax(self.y_test[i])
70
71
72          for i in range(len(self.y_train1)):
73              self.y_train1[i] = np.argmax(self.y_train[i
                   ])
74
75          #print(f'y_test shape: {self.y_test1.shape}' )
76          #print(f'y_train shape: {self.y_train1.shape}')
77
78          #ovo je za DMatrix
79
80          #koristimo bez encodinga za target
81          #pravimo Dmatrix jer XGBoost radi bolje
82          #self.d_train = xgb.DMatrix(self.x_train, self.
               y_train1)
83          #self.d_test = xgb.DMatrix(self.x_test, self.
               y_test1)
84
85
86          self.no_classes = no_classes
87          self.D = D
88
89          self.ub = [None] * self.D
90          self.lb = [None] * self.D
91          self.minimum = 0
92          self.solution = '(0,...,0)'
93          self.name = "XGBoost␣Function"
94
95          # pomocna promenljiva koja pokazuje da li radimo
               feature selection
```

```
96          # postavljamo ovu promenjivu samo na osnovu
               neatures
97          if self.nfeatures > 0:
98              self.fs = True
99          else:
100             self.fs = False
101
102
103         if self.fs:
104             for i in range(self.nfeatures):
105                 self.lb[i] = 0
106                 self.ub[i] = 1
107
108             self.lb[self.nfeatures] = bounds['lb_lr']  #
                    lower bound of learning rate
109             self.ub[self.nfeatures] = bounds['ub_lr']  #
                    upper bound of learning rate
110             self.lb[self.nfeatures + 1] = bounds['lb_mcw
                   ']  # lower bound of min_child_weight
111             self.ub[self.nfeatures + 1] = bounds['ub_mcw
                   ']  # upper bound of min_child_weight
112             self.lb[self.nfeatures + 2] = bounds['lb_ss'
                   ]  # lower bound of sub-sample
113             self.ub[self.nfeatures + 2] = bounds['ub_ss'
                   ]  # upper bound of sub-sample
114             self.lb[self.nfeatures + 3] = bounds['lb_cst
                   ']  # lower bound of coll sample by tree
115             self.ub[self.nfeatures + 3] = bounds['ub_cst
                   ']  # upper bound of coll sample by tree
116             self.lb[self.nfeatures + 4] = bounds['lb_md'
                   ]  # lower bound of maximum depth of tree,
                    ovo je int
117             self.ub[self.nfeatures + 4] = bounds['ub_md'
                   ]  # upper bound of maximum depth of tree,
                    ovo je int
118             self.lb[self.nfeatures + 5] = bounds['lb_g']
                     # lower bound of maximum depth of tree
119             self.ub[self.nfeatures + 5] = bounds['ub_g']
                     # upper bound of maximum depth of tree
120
121             # SADA DEFINISEMO PARAMETRE AKO SE NE RADI
                   FEATURE SELECTION
122         else:
123
124             self.lb[self.nfeatures] = bounds['lb_lr']  #
                    lower bound of learning rate
125             self.ub[self.nfeatures] = bounds['ub_lr']  #
                    upper bound of learning rate
```

```
126             self.lb[self.nfeatures + 1] = bounds['lb_mcw
                    '] # lower bound of min_child_weight
127             self.ub[self.nfeatures + 1] = bounds['ub_mcw
                    '] # upper bound of min_child_weight
128             self.lb[self.nfeatures + 2] = bounds['lb_ss'
                    ] # lower bound of sub-sample
129             self.ub[self.nfeatures + 2] = bounds['ub_ss'
                    ] # upper bound of sub-sample
130             self.lb[self.nfeatures + 3] = bounds['lb_cst
                    '] # lower bound of coll sample by tree
131             self.ub[self.nfeatures + 3] = bounds['ub_cst
                    '] # upper bound of coll sample by tree
132             self.lb[self.nfeatures + 4] = bounds['lb_md'
                    ] # lower bound of maximum depth of tree,
                     ovo je int
133             self.ub[self.nfeatures + 4] = bounds['ub_md'
                    ] # upper bound of maximum depth of tree,
                     ovo je int
134             self.lb[self.nfeatures + 5] = bounds['lb_g']
                      # lower bound of maximum depth of tree
135             self.ub[self.nfeatures + 5] = bounds['ub_g']
                      # upper bound of maximum depth of tree
136
137         #podesavanje granica parametara
138         #typical values: https://www.analyticsvidhya.com
                /blog/2016/03/complete-guide-parameter-tuning-
                xgboost-with-codes-python/
139         #za razliku od implementacije bez feature
                selection, ovde se sve postavlja u odnosu na
                broj features,
140         #npr. learning rate je index nfeatures, pa
                sledeci je nfeatures+1, itd.
141
142
143
144
145
146         #self.x_train, self.y_train, self.x_test, self.
                y_test = load_dataset(path_train, path_test,
                self.no_classes, normalize, test_size)
147         '''
148         self.x_train=x_train
149         self.y_train=y_train
150         self.x_test=x_test
151         self.y_test = y_test
152         '''
153         # coverting train and test datasets to DMatrix
                da bi radilo brze
```

```
154
155         #self.d_train = xgb.DMatrix(self.x_train, self.
               y_train)
156         #self.d_test = xgb.DMatrix(self.x_test, self.
               y_test)
157
158
159         #ovo nam koristi za solution
160         self.y_test_length = len(y_test)
161
162         '''
163         self.D = D
164         #D is weighs*input features length + 1
165         #in second experiment we use NN as well as the
               first argument of solution
166         self.lb_w = lb_w
167         self.ub_w = ub_w
168         self.lb_nn= lb_nn
169         self.ub_nn = ub_nn
170          #ub[0],lb[0] is for hidden neurons size
171
172         self.lb[0] = lb_nn
173         self.ub[0] = ub_nn
174
175
176         for i in range(1, self.D):
177             self.lb[i] = self.lb_w
178             self.ub[i] = self.ub_w
179         #self.lb[0] = self.lb_nn
180         #self.ub[0] = self.ub_nn
181         '''
182
183     def function(self, x):
184         x = np.array(x)
185
186         # parametri za XGBoost
187         params = {
188             'booster': 'gbtree',
189             'max_depth': int(round(x[self.nfeatures
                   +4],0)),
190             'learning_rate': x[self.nfeatures+0],
191             'sample_type': 'uniform',
192             'normalize_type': 'tree',
193             #'objective': 'binary:hinge',
194             #'objective': 'multi:softprob', #logistic
                   function da bismo dobili y_proba
195             #'objective':'binary:logistic',
196             'objective':'multi:softprob',
```

```
197             'rate_drop': 0.1,
198             'n_estimators': 100,
199             'min_child_weight': x[self.nfeatures+1],
200             'subsample': x[self.nfeatures+2],
201             'colsample_bytree': x[self.nfeatures+3],
202             'random_state': 23,
203             'seed': 23,
204             'silent': 1, #logging mode za silent
205             'num_class':self.no_classes,
206             'gamma':x[self.nfeatures+5],
207             'verbosity':0
208             # 'num_boost_round':10
209         }
210         #params = {'max_depth':10,'num_class':self.
               no_classes, 'rate_drop': 0.1,
211             #'#n_estimators':100}
212
213         #sada se prvo kao u fazi inicijalizacije igramo
               sa selektovanim features i dmatrix
214         #sada gledamo prvo da li koristimo feature
               selection
215         if self.fs:
216
217             x_train_fs = self.x_train[:, x[0:self.
                   nfeatures] == 1]
218             x_test_fs = self.x_test[:, x[0:self.
                   nfeatures] == 1]
219
220             # broj odabranih featrues
221             feature_size = np.sum(x[0:self.nfeatures])
222
223             if(feature_size==0):
224                 return 1,1,0,0,0
225
226             #sada pravimo novi dmatrix na osnovu
                   odabranih features
227
228             self.d_train = xgb.DMatrix(x_train_fs, self.
                   y_train1)
229             self.d_test = xgb.DMatrix(x_test_fs, self.
                   y_test1)
230         # sada gledamo ako ne radimo feature selection i
                onda gledamo unapred prosledjenu listu
               features, koja predstavlja koji se features
               uzimaju u obzir
231         else:
232
233             # print("OVO RADI")
```

```
234             x_train_fs = self.x_train[:, np.array(self.
                    features_list) == 1]
235             x_test_fs = self.x_test[:, np.array(self.
                    features_list) == 1]
236             # broj odabranih featrues
237             feature_size = np.sum(self.features_list)
238             # print(x_train_fs.shape[1])
239             # print(x_train_fs)
240             # ovo za svaki slucaj ako neko prosledi sve
                    nule
241             if feature_size == 0:
242                 return 1, 1, 0, 0, 0
243             self.d_train = xgb.DMatrix(x_train_fs, self.
                    y_train1)
244             self.d_test = xgb.DMatrix(x_test_fs, self.
                    y_test1)
245
246
247         '''
248         xgb_clf = xgb.train(params, self.d_train)
249
250         y_proba = xgb_clf.predict(self.d_test)
251
252         y = np.zeros((len(y_proba), y_proba.shape[1]))
253         for i in range(len(y_proba)):
254             y[i][np.argmax(y_proba[i])] = 1
255
256         acc = np.round(accuracy_score(self.y_test,
                y_proba) * 100, 2)
257
258         error = 1-acc
259
260         return (error,y_proba,y)
261         '''
262         correct = 0
263         xgb_clf = xgb.train(params, self.d_train)
264         y_proba = xgb_clf.predict(self.d_test)
265         total = y_proba.shape[0]
266         #print(y_proba)
267         #print(y_proba.shape)
268         for i in range(total):
269             predicted = np.argmax(y_proba[i])
270             # test = np.argmax(self.y_test[i])
271             test = self.y_test1[i]
272             correct = correct + (1 if predicted == test
                    else 0)
273         #sada pravimo i predkcije po klasama za kohen
                kappa
```

```
274         y_cappa = np.zeros(len(y_proba))
275         for i in range(total):
276             y_cappa[i] = np.argmax(y_proba[i])
277 
278 
279         y = np.zeros((len(y_proba), y_proba.shape[1]))
280         for i in range(len(y_proba)):
281             y[i][np.argmax(y_proba[i])] = 1
282         # classification error
283         error = round(1 - (correct / total),30)
284         #print(f'error: {error}')
285 
286         #print("OVO JE Y1: ", self.y_test1)
287         #print("OVO JE Y: ",y)
288 
289         #racunamo cohen kappa za statistiku, ovo je
              indikator
290         cohen_kappa = cohen_kappa_score(y_cappa,self.
              y_test1)
291 
292 
293 
294         if self.fitness_type == 0:
295             return (error,cohen_kappa,y_proba,y,
                      feature_size,xgb_clf)
296         else:
297             objective = self.alpha*error + self.beta*(
                      feature_size/self.nfeatures) #ovo je za
                      minimization
298             return (objective,error,y_proba,y,
                      feature_size,xgb_clf)
```

## A.4 Optimizacija XGBoost modela (Kohen kapa koeficijent) u Python programskom jeziku

**Listing A.4:** XGBoost (Cohen kappa)

```
1 
2 
3 #XGBOOST funkcija za feature selection i tuning
4 #metrikom cohen cappa, radi se minimization (1-cohen
      kappa), sto je veci cohen kappa, to je bolje
5 
6 import math
7 import numpy as np
8 from utilities.load_dataset import load_dataset
9 from ml_models.ELM import  ELM
```

```
10 import xgboost as xgb
11 import matplotlib.pyplot as plt
12 from sklearn.preprocessing import LabelEncoder
13 from sklearn.metrics import accuracy_score
14 from sklearn.model_selection import train_test_split
15 import matplotlib.pyplot as plt
16 from sklearn import metrics
17 from sklearn.metrics import roc_auc_score
18 from xgboost import XGBClassifier
19 #ovo je za cohen kappa score
20 from sklearn.metrics import cohen_kappa_score
21
22 class XGBoostFunction:
23     def __init__(self, D, no_classes,nfeatures,
       fitness_type,alpha,x_train,x_test,y_train,y_test,
       intParams,bounds,features_list=None):
24         #D je broj parametara resenja (broj
             hiperparametara koji se optimizuju)
25         #bounds je dictionary sa lower and upper bounds,
               ovaj dictionary se ubacuje iz spoljnog koda
26         #parametri za tuning
27         # x[0] - learning rate (lr), eta parametar float
28         # x[1] - min_child_weight, float
29         #x[2] - subsample, float
30         #x[3] - collsample_bytree
31         #x[4] - max_depth
32         #x[5] - gamma
33
34         #postavljamo datasetove
35         self.x_train = x_train
36         self.x_test = x_test
37         self.y_train = y_train
38         self.y_test = y_test
39
40         self.y_train1  = np.zeros(shape=len(y_train))
41         self.y_test1  = np.zeros(shape=len(y_test))
42
43         self.nfeatures = nfeatures
44
45         self.intParams = intParams
46
47         # koji objective koristimo, ovo mora biti prvi
             parametar koji se vraca
48         # moze kombinacija broja features i error rate-a
             , posto se radi o minimizaciji
49         # fitness type - 0 je samo error, a 1 je
             composite error i number of features
50         self.fitness_type = fitness_type
```

```
51          # ovo su weighted coefficient za error i broj
               selektovanih features
52          self.alpha = alpha
53          self.beta = 1 - alpha  # alpha je obicnao za
               error, a beta je obicno za broj feature-a
54
55          # ako se radi bez feature selection, onda se
               prosledjuje features list
56          # dakle, ako je features_list=None (po default),
                onda se radi fs i optimizacija, a ako ima
               necega, onda se radi samo optimizacija
57          # za svaki slucaj konvertujemo sve elemente u
               int
58          self.features_list = [int(x) for x in
               features_list]
59          print(self.features_list)
60
61          #converting y_test and y_train to single label
               classification
62
63          #fajl za load datasetova kodira target varijablu
               , pa pravi kolone u zavisnosti broja klasa,
               npr. 0 0 1, 0 1 0, itd.
64          #medjutim, XGBoost kao i DMatrix ne koriste one
               hot encoding, vec se koriste u target koloni
               klase, 0,1,2,3, itd.
65          #zbog toga pravimo pomocne setove y_test1 i
               y_train1 gde se koriste 0,1,2,3,4 za target,
               umesto one hot encoding
66
67          for i in range(len(self.y_test)):
68              self.y_test1[i] = np.argmax(self.y_test[i])
69
70          for i in range(len(self.y_train1)):
71              self.y_train1[i] = np.argmax(self.y_train[i
                   ])
72
73          #print(f'y_test shape: {self.y_test1.shape}' )
74          #print(f'y_train shape: {self.y_train1.shape}')
75
76          #ovo je za DMatrix
77
78          #koristimo bez encodinga za target
79          #pravimo Dmatrix jer XGBoost radi bolje
80          #self.d_train = xgb.DMatrix(self.x_train, self.
               y_train1)
81          #self.d_test = xgb.DMatrix(self.x_test, self.
               y_test1)
```

```
82
83
84          self.no_classes = no_classes
85          self.D = D
86
87          self.ub = [None] * self.D
88          self.lb = [None] * self.D
89          self.minimum = 0
90          self.solution = '(0,...,0)'
91          self.name = "XGBoost␣Function"
92
93          # pomocna promenljiva koja pokazuje da li radimo
                feature selection
94          # postavljamo ovu promenjivu samo na osnovu
               neatures
95          if self.nfeatures > 0:
96              self.fs = True
97          else:
98              self.fs = False
99
100
101         if self.fs:
102             for i in range(self.nfeatures):
103                 self.lb[i] = 0
104                 self.ub[i] = 1
105
106             self.lb[self.nfeatures] = bounds['lb_lr']  #
                    lower bound of learning rate
107             self.ub[self.nfeatures] = bounds['ub_lr']  #
                    upper bound of learning rate
108             self.lb[self.nfeatures + 1] = bounds['lb_mcw
                   ']  # lower bound of min_child_weight
109             self.ub[self.nfeatures + 1] = bounds['ub_mcw
                   ']  # upper bound of min_child_weight
110             self.lb[self.nfeatures + 2] = bounds['lb_ss'
                   ]  # lower bound of sub-sample
111             self.ub[self.nfeatures + 2] = bounds['ub_ss'
                   ]  # upper bound of sub-sample
112             self.lb[self.nfeatures + 3] = bounds['lb_cst
                   ']  # lower bound of coll sample by tree
113             self.ub[self.nfeatures + 3] = bounds['ub_cst
                   ']  # upper bound of coll sample by tree
114             self.lb[self.nfeatures + 4] = bounds['lb_md'
                   ]  # lower bound of maximum depth of tree,
                    ovo je int
115             self.ub[self.nfeatures + 4] = bounds['ub_md'
                   ]  # upper bound of maximum depth of tree,
                    ovo je int
```

```
116             self.lb[self.nfeatures + 5] = bounds['lb_g']
                    # lower bound of maximum depth of tree
117             self.ub[self.nfeatures + 5] = bounds['ub_g']
                    # upper bound of maximum depth of tree
118
119             # SADA DEFINISEMO PARAMETRE AKO SE NE RADI
                  FEATURE SELECTION
120         else:
121
122             self.lb[self.nfeatures] = bounds['lb_lr']  #
                    lower bound of learning rate
123             self.ub[self.nfeatures] = bounds['ub_lr']  #
                    upper bound of learning rate
124             self.lb[self.nfeatures + 1] = bounds['lb_mcw
                  ']  # lower bound of min_child_weight
125             self.ub[self.nfeatures + 1] = bounds['ub_mcw
                  ']  # upper bound of min_child_weight
126             self.lb[self.nfeatures + 2] = bounds['lb_ss'
                  ]  # lower bound of sub-sample
127             self.ub[self.nfeatures + 2] = bounds['ub_ss'
                  ]  # upper bound of sub-sample
128             self.lb[self.nfeatures + 3] = bounds['lb_cst
                  ']  # lower bound of coll sample by tree
129             self.ub[self.nfeatures + 3] = bounds['ub_cst
                  ']  # upper bound of coll sample by tree
130             self.lb[self.nfeatures + 4] = bounds['lb_md'
                  ]  # lower bound of maximum depth of tree,
                    ovo je int
131             self.ub[self.nfeatures + 4] = bounds['ub_md'
                  ]  # upper bound of maximum depth of tree,
                    ovo je int
132             self.lb[self.nfeatures + 5] = bounds['lb_g']
                    # lower bound of maximum depth of tree
133             self.ub[self.nfeatures + 5] = bounds['ub_g']
                    # upper bound of maximum depth of tree
134
135         #podesavanje granica parametara
136         #typical values: https://www.analyticsvidhya.com
              /blog/2016/03/complete-guide-parameter-tuning-
              xgboost-with-codes-python/
137         #za razliku od implementacije bez feature
              selection, ovde se sve postavlja u odnosu na
              broj features,
138         #npr. learning rate je index nfeatures, pa
              sledeci je nfeatures+1, itd.
139
140
141
```

```
142
143
144         #self.x_train, self.y_train, self.x_test, self.
               y_test = load_dataset(path_train, path_test,
               self.no_classes, normalize, test_size)
145         '''
146         self.x_train=x_train
147         self.y_train=y_train
148         self.x_test=x_test
149         self.y_test = y_test
150         '''
151         # coverting train and test datasets to DMatrix
               da bi radilo brze
152
153         #self.d_train = xgb.DMatrix(self.x_train, self.
               y_train)
154         #self.d_test = xgb.DMatrix(self.x_test, self.
               y_test)
155
156
157         #ovo nam koristi za solution
158         self.y_test_length = len(y_test)
159
160         '''
161         self.D = D
162         #D is weighs*input features length + 1
163         #in second experiment we use NN as well as the
               first argument of solution
164         self.lb_w = lb_w
165         self.ub_w = ub_w
166         self.lb_nn= lb_nn
167         self.ub_nn = ub_nn
168          #ub[0],lb[0] is for hidden neurons size
169
170         self.lb[0] = lb_nn
171         self.ub[0] = ub_nn
172
173
174         for i in range(1, self.D):
175             self.lb[i] = self.lb_w
176             self.ub[i] = self.ub_w
177         #self.lb[0] = self.lb_nn
178         #self.ub[0] = self.ub_nn
179         '''
180
181     def function(self, x):
182         x = np.array(x)
183
```

```
184         # parametri za XGBoost
185         params = {
186             'booster': 'gbtree',
187             'max_depth': int(round(x[self.nfeatures
                +4],0)),
188             'learning_rate': x[self.nfeatures+0],
189             'sample_type': 'uniform',
190             'normalize_type': 'tree',
191             #'objective': 'binary:hinge',
192             #'objective': 'multi:softprob', #logistic
                function da bismo dobili y_proba
193             #'objective':'binary:logistic',
194             'objective':'multi:softprob',
195             'rate_drop': 0.1,
196             'n_estimators': 100,
197             'min_child_weight': x[self.nfeatures+1],
198             'subsample': x[self.nfeatures+2],
199             'colsample_bytree': x[self.nfeatures+3],
200             'random_state': 23,
201             'seed': 23,
202             'silent': 1, #logging mode za silent
203             'num_class':self.no_classes,
204             'gamma':x[self.nfeatures+5],
205             'verbosity':0
206             # 'num_boost_round':10
207         }
208         #params = {'max_depth':10,'num_class':self.
              no_classes, 'rate_drop': 0.1,
209             #'#n_estimators':100}
210
211         #sada se prvo kao u fazi inicijalizacije igramo
              sa selektovanim features i dmatrix
212         #sada gledamo prvo da li koristimo feature
              selection
213         if self.fs:
214
215             x_train_fs = self.x_train[:, x[0:self.
                nfeatures] == 1]
216             x_test_fs = self.x_test[:, x[0:self.
                nfeatures] == 1]
217
218             # broj odabranih featrues
219             feature_size = np.sum(x[0:self.nfeatures])
220
221             if(feature_size==0):
222                 return 1,1,0,0,0
223
```

```
224             #sada pravimo novi dmatrix na osnovu
                   odabranih features
225
226             self.d_train = xgb.DMatrix(x_train_fs, self.
                   y_train1)
227             self.d_test = xgb.DMatrix(x_test_fs, self.
                   y_test1)
228         # sada gledamo ako ne radimo feature selection i
                onda gledamo unapred prosledjenu listu
               features, koja predstavlja koji se features
               uzimaju u obzir
229         else:
230
231             # print("OVO RADI")
232             x_train_fs = self.x_train[:, np.array(self.
                   features_list) == 1]
233             x_test_fs = self.x_test[:, np.array(self.
                   features_list) == 1]
234             # broj odabranih featrues
235             feature_size = np.sum(self.features_list)
236             # print(x_train_fs.shape[1])
237             # print(x_train_fs)
238             # ovo za svaki slucaj ako neko prosledi sve
                   nule
239             if feature_size == 0:
240                 return 1, 1, 0, 0, 0
241             self.d_train = xgb.DMatrix(x_train_fs, self.
                   y_train1)
242             self.d_test = xgb.DMatrix(x_test_fs, self.
                   y_test1)
243
244
245         '''
246         xgb_clf = xgb.train(params, self.d_train)
247
248         y_proba = xgb_clf.predict(self.d_test)
249
250         y = np.zeros((len(y_proba), y_proba.shape[1]))
251         for i in range(len(y_proba)):
252             y[i][np.argmax(y_proba[i])] = 1
253
254         acc = np.round(accuracy_score(self.y_test,
                y_proba) * 100, 2)
255
256         error = 1-acc
257
258         return (error,y_proba,y)
259         '''
```

```
260         correct = 0
261         xgb_clf = xgb.train(params, self.d_train)
262         y_proba = xgb_clf.predict(self.d_test)
263         total = y_proba.shape[0]
264         #print(y_proba)
265         #print(y_proba.shape)
266         for i in range(total):
267             predicted = np.argmax(y_proba[i])
268             # test = np.argmax(self.y_test[i])
269             test = self.y_test1[i]
270             correct = correct + (1 if predicted == test
                else 0)
271         #sada pravimo i predkcije po klasama za kohen
               kappa
272         y_cappa = np.zeros(len(y_proba))
273         for i in range(total):
274             y_cappa[i] = np.argmax(y_proba[i])
275
276
277         y = np.zeros((len(y_proba), y_proba.shape[1]))
278         for i in range(len(y_proba)):
279             y[i][np.argmax(y_proba[i])] = 1
280         # classification error
281         error = round(1 - (correct / total),30)
282         #print(f'error: {error}')
283
284         #print("OVO JE Y1: ", self.y_test1)
285         #print("OVO JE Y: ",y)
286
287         #racunamo cohen kappa za statistiku, ovo je
               indikator
288         cohen_kappa = cohen_kappa_score(y_cappa,self.
               y_test1)
289
290
291         #posto idemo na minimizaciju prvo je 1-
               cohen_kappa
292         if self.fitness_type == 0:
293             return ((1-cohen_kappa),error,y_proba,y,
                    feature_size,xgb_clf)
294         else:
295             objective = self.alpha*error + self.beta*(
                    feature_size/self.nfeatures) #ovo je za
                    minimization
296             return (objective,error,y_proba,y,
                    feature_size,xgb_clf)
```

# Literatura


[1] Živković, T., Živković, M., "Comparative analysis of techniques for testing object oriented programs", in 2020 Zooming Innovation in Consumer Technologies Conference (ZINC). IEEE, 2020, str. 270–275.

[2] McDonald, M., Musson, R., Smith, R., The practical guide to defect prevention. Microsoft Press, 2007.

[3] Committee, I. C. S. S. E. T., IEEE Standard Glossary of Software Engineering Terminology: An American National Standard. IEEE, 1990.

[4] Jones, C., Bonsignour, O., The economics of software quality. Addison-Wesley Professional, 2011.

[5] Boehm, B., Basili, V. R., "Top 10 list [software development]", Computer, Vol. 34, No. 1, 2001, str. 135–137.

[6] Khoshgoftaar, T. M., Gao, K., Seliya, N., "Attribute selection and imbalanced data: Problems in software defect prediction", in 2010 22nd IEEE International conference on tools with artificial intelligence, Vol. 1. IEEE, 2010, str. 137–144.

[7] Xu, Z., Khoshgoftaar, T. M., Allen, E. B., "Prediction of software faults using fuzzy nonlinear regression modeling", in Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000). IEEE, 2000, str. 281–290.

[8] Li, Z., Reformat, M., "A practical method for the software fault-prediction", in 2007 IEEE international conference on information reuse and integration. IEEE, 2007, str. 659–666.

[9] Menzies, T., Greenwald, J., Frank, A., "Data mining static code attributes to learn defect predictors", IEEE transactions on software engineering, Vol. 33, No. 1, 2006, str. 2–13.

[10] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A., "Defect prediction from static code features: current results, limitations, new approaches", Automated Software Engineering, Vol. 17, No. 4, 2010, str. 375–407.

[11] Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J., "A general software defect-proneness prediction framework", IEEE transactions on software engineering, Vol. 37, No. 3, 2010, str. 356–370.

[12] Lessmann, S., Baesens, B., Mues, C., Pietsch, S., "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", IEEE Transactions on Software Engineering, Vol. 34, No. 4, 2008, str. 485–496.

[13] Gayatri, N., Nickolas, S., Reddy, A., Reddy, S., Nickolas, A., "Feature selection using decision tree induction in class level metrics dataset for software defect predictions", in Proceedings of the world congress on engineering and computer science, Vol. 1, 2010, str. 124–129.

[14] Catal, C., Sevim, U., Diri, B., "Practical development of an eclipse-based software fault prediction tool using naive bayes algorithm", Expert Systems with Applications, Vol. 38, No. 3, 2011, str. 2347–2353.

[15] Denaro, G., "Estimating software fault-proneness for tuning testing activities", in Proceedings of the 22nd international conference on Software engineering, 2000, str. 704–706.

[16] Khoshgoftaar, T. M., Seliya, N., "Tree-based software quality estimation models for fault prediction", in Proceedings Eighth IEEE Symposium on Software Metrics. IEEE, 2002, str. 203–214.

[17] Khoshgoftaar, T. M., Seliya, N., Gao, K., "Assessment of a new three-group software quality classification technique: An empirical case study", Empirical Software Engineering, Vol. 10, No. 2, 2005, str. 183–218.

[18] Park, B.-J., Oh, S.-K., Pedrycz, W., "The design of polynomial function-based neural network predictors for detection of software defects", Information Sciences, Vol. 229, 2013, str. 40–57.

[19] Wang, Q., Yu, B., Zhu, J., "Extract rules from software quality prediction model based on neural network", in 16th IEEE international conference on tools with artificial intelligence. IEEE, 2004, str. 191–195.

[20] Zheng, J., "Cost-sensitive boosting neural networks for software defect prediction", Expert Systems with Applications, Vol. 37, No. 6, 2010, str. 4537–4543.

[21] McCabe, T. J., "A complexity measure", IEEE Transactions on software Engineering, No. 4, 1976, str. 308–320.

[22] Halstead, M. H., Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., 1977.

[23] Halstead, M. H., "Advances in software science", in Advances in Computers. Elsevier, 1979, Vol. 18, str. 119–172.

[24] Abaei, G., Selamat, A., "A survey on software fault detection based on different prediction approaches", Vietnam Journal of Computer Science, Vol. 1, 2014, str. 79–95.

[25] Catal, C., "A comparison of semi-supervised classification approaches for software defect prediction", Journal of Intelligent Systems, Vol. 23, No. 1, 2014, str. 75–82.

[26] Chen, T., Guestrin, C., "Xgboost: A scalable tree boosting system", in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, str. 785–794.

[27] Abualigah, L., Abd Elaziz, M., Sumari, P., Geem, Z. W., Gandomi, A. H., "Reptile search algorithm (rsa): A nature-inspired meta-heuristic optimizer", Expert Systems with Applications, Vol. 191, 2022, str. 116158.

[28] Yang, X.-S., "Firefly algorithms for multimodal optimization", in International symposium on stochastic algorithms. Springer, 2009, str. 169–178.

[29] Sayyad Shirabad, J., Menzies, T., "The PROMISE Repository of Software Engineering Databases.", School of Information Technology and Engineering, University of Ottawa, Canada, dostupno na: http://promise.site.uottawa.ca/SERepository 2005.

[30] Xu, J., Wang, F., Ai, J., "Defect prediction with semantics and context features of codes based on graph representation learning", IEEE Transactions on Reliability, 2021.

[31] Menzies, T., DiStefano, J., Orrego, A., Chapman, R., "Assessing predictors of software defects", in Proc. Workshop Predictive Software Models, 2004.

[32] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., "A systematic literature review on fault prediction performance in software engineering", IEEE Transactions on Software Engineering, Vol. 38, No. 6, 2011, str. 1276–1304.

[33] Wahono, R. S., "A systematic literature review of software defect prediction", Journal of Software Engineering, Vol. 1, No. 1, 2015, str. 1–16.

[34] Thota, M. K., Shajin, F. H., Rajesh, P. *et al.*, "Survey on software defect prediction techniques", International Journal of Applied Science and Engineering, Vol. 17, No. 4, 2020, str. 331–344.

[35] Roy, P., Mahapatra, G., Rani, P., Pandey, S., Dey, K. N., "Robust feedforward and recurrent neural network based dynamic weighted combination models for software reliability prediction", Applied Soft Computing, Vol. 22, 2014, str. 629–637.

[36] Ghaffarian, S. M., Shahriari, H. R., "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey", ACM Computing Surveys (CSUR), Vol. 50, No. 4, 2017, str. 1–36.

[37] Wang, S., Yao, X., "Using class imbalance learning for software defect prediction", IEEE Transactions on Reliability, Vol. 62, No. 2, 2013, str. 434–443.

[38] Elish, K. O., Elish, M. O., "Predicting defect-prone software modules using support vector machines", Journal of Systems and Software, Vol. 81, No. 5, 2008, str. 649–660.

[39] Bai, X., Zhou, H., Yang, H., "An hvsm-based gru approach to predict cross-version software defects.", International Journal of Performability Engineering, Vol. 16, No. 6, 2020.

[40] Tong, H., Liu, B., Wang, S., "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning", Information and Software Technology, Vol. 96, 2018, str. 94–111.

[41] Stegherr, H., Heider, M., Hähner, J., "Classifying metaheuristics: Towards a unified multi-level classification system", Natural Computing, 2020, str. 1–17.

[42] Emmerich, M., Shir, O. M., Wang, H., "Evolution strategies", in Handbook of Heuristics. Springer, 2018, str. 89–119.

[43] Fausto, F., Reyna-Orta, A., Cuevas, E., Andrade, Á. G., Perez-Cisneros, M., "From ants to whales: metaheuristics for all tastes", Artificial Intelligence Review, Vol. 53, No. 1, 2020, str. 753–810.

[44] Goldberg, D. E., Richardson, J. *et al.*, "Genetic algorithms with sharing for multimodal function optimization", in Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms, Vol. 4149. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[45] Booker, L. B., Goldberg, D. E., Holland, J. H., "Classifier systems and genetic algorithms", Artificial intelligence, Vol. 40, No. 1-3, 1989, str. 235–282.

[46] Bartz-Beielstein, T., Branke, J., Mehnen, J., Mersmann, O., "Evolutionary algorithms", Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Vol. 4, No. 3, 2014, str. 178–195.

[47] Karaboğa, D., Ökdem, S., "A simple and global optimization algorithm for engineering problems: differential evolution algorithm", Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 12, No. 1, 2004, str. 53–60.

[48] Ahmad, M. F., Isa, N. A. M., Lim, W. H., Ang, K. M., "Differential evolution: A recent review based on state-of-the-art works", Alexandria Engineering Journal, Vol. 61, No. 5, 2022, str. 3831–3872.

[49] Qin, A. K., Suganthan, P. N., "Self-adaptive differential evolution algorithm for numerical optimization", in 2005 IEEE congress on evolutionary computation, Vol. 2. IEEE, 2005, str. 1785–1791.

[50] Mirjalili, S., Mirjalili, S., "Genetic algorithm", Evolutionary Algorithms and Neural Networks: Theory and Applications, 2019, str. 43–55.

[51] Zahir, A. M., Alhady, S. S. N., Wahab, A., Ahmad, M., "Objective functions modification of ga optimized pid controller for brushed dc motor", International Journal of Electrical and Computer Engineering, Vol. 10, No. 3, 2020, str. 2426.

[52] Cuk, A., Bezdan, T., Bacanin, N., Zivkovic, M., Venkatachalam, K., Rashid, T. A., Devi, V. K., "Feedforward multi-layer perceptron training by hybridized method between genetic algorithm and artificial bee colony", Data science and data analytics: opportunities and challenges, 2021, str. 279.

[53] Rahman, I., Mohamad-Saleh, J., "Hybrid bio-inspired computational intelligence techniques for solving power system optimization problems: A comprehensive survey", Applied Soft Computing, Vol. 69, 2018, str. 72–130.

[54] Beni, G., "Swarm intelligence", Complex Social and Behavioral Systems: Game Theory and Agent-Based Models, 2020, str. 791–818.

[55] Abraham, A., Guo, H., Liu, H., "Swarm intelligence: foundations, perspectives and applications", in Swarm intelligent systems. Springer, 2006, str. 3–25.

[56] Kennedy, J., Eberhart, R., "Particle swarm optimization", in Proceedings of ICNN'95-international conference on neural networks, Vol. 4. IEEE, 1995, str. 1942–1948.

[57] Karaboga, D., Basturk, B., "On the performance of artificial bee colony (ABC) algorithm", Applied soft computing, Vol. 8, No. 1, 2008, str. 687–697.

[58] Dorigo, M., Birattari, M., Stutzle, T., "Ant colony optimization", IEEE computational intelligence magazine, Vol. 1, No. 4, 2006, str. 28–39.

[59] Yang, X.-S., "A new metaheuristic bat-inspired algorithm", in Nature inspired cooperative strategies for optimization (NICSO 2010). Springer, 2010, str. 65–74.

[60] Yang, X.-S., Gandomi, A. H., "Bat algorithm: a novel approach for global engineering optimization", Engineering computations, 2012.

[61] Yang, X.-S., Slowik, A., "Firefly algorithm", in Swarm Intelligence Algorithms. CRC Press, 2020, str. 163–174.

[62] Mirjalili, S., "Sca: a sine cosine algorithm for solving optimization problems", Knowledge-based systems, Vol. 96, 2016, str. 120–133.

[63] Abualigah, L., Diabat, A., Mirjalili, S., Abd Elaziz, M., Gandomi, A. H., "The arithmetic optimization algorithm", Computer methods in applied mechanics and engineering, Vol. 376, 2021, str. 113609.

[64] Wolpert, D. H., Macready, W. G., "No free lunch theorems for optimization", IEEE transactions on evolutionary computation, Vol. 1, No. 1, 1997, str. 67–82.

[65] Zivkovic, M., Bacanin, N., Venkatachalam, K., Nayyar, A., Djordjevic, A., Strumberger, I., Al-Turjman, F., "Covid-19 cases prediction by using hybrid machine learning and beetle antennae search approach", Sustainable Cities and Society, Vol. 66, 2021, str. 102669.

[66] Zivkovic, M., Venkatachalam, K., Bacanin, N., Djordjevic, A., Antonijevic, M., Strumberger, I., Rashid, T. A., "Hybrid genetic algorithm and machine learning method for covid-19 cases prediction", in Proceedings of International Conference on Sustainable Expert Systems: ICSES 2020, Vol. 176. Springer Nature, 2021, str. 169.

[67] Bacanin, N., Bezdan, T., Tuba, E., Strumberger, I., Tuba, M., Zivkovic, M., "Task scheduling in cloud computing environment by grey wolf optimizer", in 2019 27th Telecommunications Forum (TELFOR). IEEE, 2019, str. 1–4.

[68] Bezdan, T., Zivkovic, M., Antonijevic, M., Zivkovic, T., Bacanin, N., "Enhanced flower pollination algorithm for task scheduling in cloud computing environment", in Machine Learning for Predictive Analysis. Springer, 2020, str. 163–171.

[69] Zivkovic, M., Bezdan, T., Strumberger, I., Bacanin, N., Venkatachalam, K., "Improved harris hawks optimization algorithm for workflow scheduling challenge in cloud–edge environment", in Computer Networks, Big Data and IoT. Springer, 2021, str. 87–102.

[70] Zivkovic, M., Bacanin, N., Tuba, E., Strumberger, I., Bezdan, T., Tuba, M., "Wireless sensor networks life time optimization based on the improved firefly algorithm", in 2020 International Wireless Communications and Mobile Computing (IWCMC). IEEE, 2020, str. 1176–1181.

[71] Zivkovic, M., Bacanin, N., Zivkovic, T., Strumberger, I., Tuba, E., Tuba, M., "Enhanced grey wolf algorithm for energy efficient wireless sensor networks", in 2020 Zooming Innovation in Consumer Technologies Conference (ZINC). IEEE, 2020, str. 87–92.

[72] Bacanin, N., Tuba, E., Zivkovic, M., Strumberger, I., Tuba, M., "Whale optimization algorithm with exploratory move for wireless sensor networks localization", in International Conference on Hybrid Intelligent Systems. Springer, 2019, str. 328–338.

[73] Zivkovic, M., Zivkovic, T., Venkatachalam, K., Bacanin, N., "Enhanced dragonfly algorithm adapted for wireless sensor network lifetime optimization", in Data Intelligence and Cognitive Informatics. Springer, 2021, str. 803–817.

[74] Bezdan, T., Cvetnic, D., Gajic, L., Zivkovic, M., Strumberger, I., Bacanin, N., "Feature selection by firefly algorithm with improved initialization strategy", in 7th Conference on the Engineering of Computer Based Systems, 2021, str. 1–8.

[75] Zivkovic, M., Bacanin, N., Antonijevic, M., Nikolic, B., Kvascev, G., Marjanovic, M., Savanovic, N., "Hybrid cnn and xgboost model tuned by modified arithmetic optimization algorithm for covid-19 early diagnostics from x-ray images", Electronics, Vol. 11, No. 22, 2022, str. 3798.

[76] Zivkovic, M., Bacanin, N., Rakic, A., Arandjelovic, J., Stanojlovic, S., Venkatachalam, K., "Chaotic binary ant lion optimizer approach for feature selection on medical datasets with covid-19 case study", in 2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS). IEEE, 2022, str. 581–588.

[77] Sarac, M., Mravik, M., Jovanovic, D., Strumberger, I., Zivkovic, M., Bacanin, N., "Intelligent diagnosis of coronavirus with computed tomography images using a deep learning model", Journal of Electronic Imaging, Vol. 32, No. 2, 2022, str. 021406.

[78] Strumberger, I., Tuba, E., Zivkovic, M., Bacanin, N., Beko, M., Tuba, M., "Dynamic search tree growth algorithm for global optimization", in Doctoral Conference on Computing, Electrical and Industrial Systems. Springer, 2019, str. 143–153.

[79] Jovanovic, D., Antonijevic, M., Stankovic, M., Zivkovic, M., Tanaskovic, M., Bacanin, N., "Tuning machine learning models using a group search firefly algorithm for credit card fraud detection", Mathematics, Vol. 10, No. 13, 2022, str. 2272.

[80] Petrovic, A., Bacanin, N., Zivkovic, M., Marjanovic, M., Antonijevic, M., Strumberger, I., "The adaboost approach tuned by firefly metaheuristics for fraud detection", in 2022 IEEE World Conference on Applied Intelligence and Computing (AIC). IEEE, 2022, str. 834–839.

[81] Bacanin, N., Jovanovic, L., Zivkovic, M., Kandasamy, V., Antonijevic, M., Deveci, M., Strumberger, I., "Multivariate energy forecasting via metaheuristic tuned long-short term memory and gated recurrent unit neural networks", Information Sciences, Vol. 642, 2023, str. 119122.

[82] Stoean, C., Zivkovic, M., Bozovic, A., Bacanin, N., Strulak-Wójcikiewicz, R., Antonijevic, M., Stoean, R., "Metaheuristic-based hyperparameter tuning for recurrent deep learning: Application to the prediction of solar energy generation", Axioms, Vol. 12, No. 3, 2023, str. 266.

[83] Bacanin, N., Stoean, C., Zivkovic, M., Rakic, M., Strulak-Wójcikiewicz, R., Stoean, R., "On the benefits of using metaheuristics in the hyperparameter tuning of deep learning models for energy load forecasting", Energies, Vol. 16, No. 3, 2023, str. 1434.

[84] Bacanin, N., Sarac, M., Budimirovic, N., Zivkovic, M., AlZubi, A. A., Bashir, A. K., "Smart wireless health care system using graph lstm pollution prediction and dragonfly node localization", Sustainable Computing: Informatics and Systems, Vol. 35, 2022, str. 100711.

[85] Jovanovic, L., Jovanovic, G., Perisic, M., Alimpic, F., Stanisic, S., Bacanin, N., Zivkovic, M., Stojic, A., "The explainable potential of coupling metaheuristics-optimized-xgboost and shap in revealing vocs' environmental fate", Atmosphere, Vol. 14, No. 1, 2023, str. 109.

[86] Bacanin, N., Zivkovic, M., Stoean, C., Antonijevic, M., Janicijevic, S., Sarac, M., Strumberger, I., "Application of natural language processing and machine learning boosted with swarm intelligence for spam email filtering", Mathematics, Vol. 10, No. 22, 2022, str. 4173.

[87] Stankovic, M., Antonijevic, M., Bacanin, N., Zivkovic, M., Tanaskovic, M., Jovanovic, D., "Feature selection by hybrid artificial bee colony algorithm for intrusion detection", in 2022 International Conference on Edge Computing and Applications (ICECAA). IEEE, 2022, str. 500–505.

[88] Dobrojevic, M., Zivkovic, M., Chhabra, A., Sani, N. S., Bacanin, N., Amin, M. M., "Addressing internet of things security by enhanced sine cosine metaheuristics tuned hybrid machine learning model and results interpretation based on shap approach", PeerJ Computer Science, Vol. 9, 2023, str. e1405.

[89] Milosevic, S., Bezdan, T., Zivkovic, M., Bacanin, N., Strumberger, I., Tuba, M., "Feed-forward neural network training by hybrid bat algorithm", in Modelling and Development of Intelligent Systems: 7th International Conference, MDIS 2020, Sibiu, Romania, October 22–24, 2020, Revised Selected Papers 7. Springer International Publishing, 2021, str. 52–66.

[90] Bacanin, N., Zivkovic, M., Al-Turjman, F., Venkatachalam, K., Trojovskỳ, P., Strumberger, I., Bezdan, T., "Hybridized sine cosine algorithm with convolutional neural networks dropout regularization application", Scientific Reports, Vol. 12, No. 1, 2022, str. 1–20.

[91] Bacanin, N., Stoean, C., Zivkovic, M., Jovanovic, D., Antonijevic, M., Mladenovic, D., "Multi-swarm algorithm for extreme learning machine optimization", Sensors, Vol. 22, No. 11, 2022, str. 4204.

[92] Jovanovic, L., Jovanovic, D., Bacanin, N., Jovancai Stakic, A., Antonijevic, M., Magd, H., Thirumalaisamy, R., Zivkovic, M., "Multi-step crude oil price prediction based on lstm approach tuned by salp swarm algorithm with disputation operator", Sustainability, Vol. 14, No. 21, 2022, str. 14616.

[93] Bukumira, M., Antonijevic, M., Jovanovic, D., Zivkovic, M., Mladenovic, D., Kunjadic, G., "Carrot grading system using computer vision feature parameters and a cascaded graph convolutional neural network", Journal of Electronic Imaging, Vol. 31, No. 6, 2022, str. 061815.

[94] Khishe, M., Mosavi, M. R., "Chimp optimization algorithm", Expert systems with applications, Vol. 149, 2020, str. 113338.

[95] Gurrola-Ramos, J., Hernàndez-Aguirre, A., Dalmau-Cedeño, O., "Colshade for real-world single-objective constrained optimization problems", in 2020 IEEE congress on evolutionary computation (CEC). IEEE, 2020, str. 1–8.

[96] Kumar, A., Das, S., Zelinka, I., "A self-adaptive spherical search algorithm for real-world constrained optimization problems", in Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 2020, str. 13–14.

[97] Shapiro, S. S., Francia, R., "An approximate analysis of variance test for normality", Journal of the American statistical Association, Vol. 67, No. 337, 1972, str. 215–216.

[98] Cohen, J., "A coefficient of agreement for nominal scales", Educational and psychological measurement, Vol. 20, No. 1, 1960, str. 37–46.

[99] McHugh, M. L., "Interrater reliability: the kappa statistic", Biochemia medica, Vol. 22, No. 3, 2012, str. 276–282.

[100] Mirjalili, S., Lewis, A., "The whale optimization algorithm", Advances in engineering software, Vol. 95, 2016, str. 51–67.

[101] Abdullah, J. M., Ahmed, T., "Fitness dependent optimizer: inspired by the bee swarming reproductive process", IEEE Access, Vol. 7, 2019, str. 43 473–43 486.

[102] Breiman, L., "Random forests", Machine learning, Vol. 45, No. 1, 2001, str. 5–32.

[103] Vapnik, V., Estimation of dependences based on empirical data. Springer Science & Business Media, 2006.

[104] LaTorre, A., Molina, D., Osaba, E., Poyatos, J., Del Ser, J., Herrera, F., "A prescription of methodological guidelines for comparing bio-inspired optimization algorithms", Swarm and Evolutionary Computation, Vol. 67, 2021, str. 100973.

[105] Wilcoxon, F., "Individual comparisons by ranking methods", in Breakthroughs in statistics. Springer, 1992, str. 196–202.

[106] Lundberg, S. M., Lee, S.-I., "A unified approach to interpreting model predictions", Advances in neural information processing systems, Vol. 30, 2017.

[107] Lemos, O. A. L., Silveira, F. F., Ferrari, F. C., Garcia, A., "The impact of software testing education on code reliability: An empirical assessment", Journal of Systems and Software, Vol. 137, 2018, str. 497–511.

[108] Clarke, P. J., Davis, D., King, T. M., Pava, J., Jones, E. L., "Integrating testing into software engineering courses supported by a collaborative learning environment", ACM Transactions on Computing Education (TOCE), Vol. 14, No. 3, 2014, str. 1–33.

[109] Nidhra, S., Dondeti, J., "Black box and white box testing techniques-a literature review", International Journal of Embedded Systems and Applications (IJESA), Vol. 2, No. 2, 2012, str. 29–50.

[110] Kumar, M., Singh, S. K., Dwivedi, R., "A comparative study of black box testing and white box testing techniques", International Journal of Advance Research in Computer Science and Management Studies, Vol. 3, No. 10, 2015.

[111] Verma, A., Khatana, A., Chaudhary, S., "A comparative study of black box testing and white box testing", Int. J. Comput. Sci. Eng, Vol. 5, No. 12, 2017, str. 301–304.

[112] Jehan, S., Pill, I., Wotawa, F., "Soa grey box testing–a constraint-based approach", in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops. IEEE, 2013, str. 232–237.

[113] Khan, M. E., Khan, F., "A comparative study of white box, black box and grey box testing techniques", International Journal of Advanced Computer Science and Applications, Vol. 3, No. 6, 2012.

[114] Cai, Y., Karakostas, G., Wassyng, A., "Secure and trusted partial grey-box verification", International Journal of Information Security, Vol. 18, No. 6, 2019, str. 677–700.

[115] Acharya, S., Pandya, V., "Bridge between black box and white box–gray box testing technique", International Journal of Electronics and Computer Science Engineering, Vol. 2, No. 1, 2012, str. 175–185.

[116] Lethbridge, T. C., Diaz-Herrera, J., Richard Jr, J., Thompson, J. B. *et al.*, "Improving software practice through education: Challenges and future trends", in Future of Software Engineering (FOSE'07). IEEE, 2007, str. 12–28.

[117] Scatalon, L. P., Barbosa, E. F., Garcia, R. E., "Challenges to integrate software testing into introductory programming courses", in 2017 IEEE Frontiers in Education Conference (FIE). Ieee, 2017, str. 1–9.

[118] de Andrade, S. A., de Oliveira Neves, V., Delamaro, M. E., "Software testing education: dreams and challenges when bringing academia and industry closer together", in Proceedings of the XXXIII Brazilian Symposium on Software Engineering, 2019, str. 47–56.

[119] Zheng, W., Bai, Y., Che, H., "A computer-assisted instructional method based on machine learning in software testing class", Computer Applications in Engineering Education, Vol. 26, No. 5, 2018, str. 1150–1158.

[120] Astigarraga, T., Dow, E. M., Lara, C., Prewitt, R., Ward, M. R., "The emerging role of software testing in curricula", in 2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments. IEEE, 2010, str. 1–26.

[121] Beck, K., Test-driven development: by example. Addison-Wesley Professional, 2003.

[122] Janzen, D., Saiedian, H., "Test-driven development concepts, taxonomy, and future direction", Computer, Vol. 38, No. 9, 2005, str. 43–50.

[123] "Avantes - advancing novel textual similarity-based solutions in software development", dostupno na: https://avantes.etf.bg.ac.rs/ Accessed 05-Jul-2022.

[124] Arnold, B. J., "Gamification in education", Proceedings of the American Society of Business and Behavioral Sciences, Vol. 21, No. 1, 2014, str. 32–39.

[125] Manzano-León, A., Camacho-Lazarraga, P., Guerrero, M. A., Guerrero-Puerta, L., Aguilar-Parra, J. M., Trigueros, R., Alias, A., "Between level up and game over: A systematic literature review of gamification in education", Sustainability, Vol. 13, No. 4, 2021, str. 2247.

[126] Swacha, J., "State of research on gamification in education: A bibliometric survey", Education Sciences, Vol. 11, No. 2, 2021, str. 69.

[127] Black, R., Coleman, G., Walsh, M., Cornanguer, B., Forgacs, I., Kakkonen, K., Sabak, J., Black, R., "Agile testing foundations: An istqb foundation level agile tester guide". BCS, 2017.

[128] Barraood, S. O., Mohd, H., Baharom, F., "A comparison study of software testing activities in agile methods", in Knowledge Management International Conference (KMICe), Vol. 2021, 2021.

[129] Beck, K., "Embracing change with extreme programming", Computer, Vol. 32, No. 10, 1999, str. 70–77.

[130] Schwaber, K., "Scrum development process", in Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas. Springer, 1997, str. 117–134.

[131] Virmani, M., "Understanding devops & bridging the gap from continuous integration to continuous delivery", in Fifth international conference on the innovative computing technology (intech 2015). IEEE, 2015, str. 78–82.

[132] Bobrovskis, S., Jurenoks, A., "A survey of continuous integration, continuous delivery and continuos deployment.", in BIR workshops, 2018, str. 314–322.

[133] Aniche, M., Hermans, F., Van Deursen, A., "Pragmatic software testing education", in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 2019, str. 414–420.

[134] Garousi, V., Rainer, A., Lauvås Jr, P., Arcuri, A., "Software-testing education: A systematic literature mapping", Journal of Systems and Software, Vol. 165, 2020, str. 110570.

[135] Wong, E., "Improving the state of undergraduate software testing education", in American Society for Engineering Education. American Society for Engineering Education, 2012.

[136] Draft, S., "Computer science curricula 2013", ACM and IEEE Computer Society, Incorporated: New York, NY, USA, 2013.

[137] Van Veenendaal, E., Graham, D., Black, R., Foundations of software testing: ISTQB certification. Delmar Learning, 2012.

[138] Jones, E. L., "An experiential approach to incorporating software testing into the computer science curriculum", in 31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No. 01CH37193), Vol. 2. IEEE, 2001, str. F3D–7.

[139] Elbaum, S., Person, S., Dokulil, J., Jorde, M., "Bug hunt: Making early software testing lessons engaging and affordable", in 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007, str. 688–697.

[140] Spacco, J., Strecker, J., Hovemeyer, D., Pugh, W., "Software repository mining with marmoset: An automated programming project snapshot and testing system", in Proceedings of the 2005 international workshop on Mining software repositories, 2005, str. 1–5.

[141] Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., Padua-Perez, N., "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses", ACM Sigcse Bulletin, Vol. 38, No. 3, 2006, str. 13–17.

[142] Pech, J. P. U., Vera, R. A. A., Gómez, O. S., "Software testing education through a collaborative virtual approach", in International Conference on Software Process Improvement. Springer, 2017, str. 231–240.

[143] Yujian Fu, P., Clarke, P. J., "Integrating software testing to cs curriculum using wrestt-cyle", 2015.

[144] Rojas, J. M., Fraser, G., "Code defenders: a mutation testing game", in 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2016, str. 162–167.

[145] Fraser, G., Gambi, A., Kreis, M., Rojas, J. M., "Gamifying a software testing course with code defenders", in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 2019, str. 571–577.

[146] Valle, P. H. D., Toda, A. M., Barbosa, E. F., Maldonado, J. C., "Educational games: A contribution to software testing education", in 2017 IEEE Frontiers in education Conference (FIE). IEEE, 2017, str. 1–8.

[147] Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., Harman, M., "Mutation testing advances: an analysis and survey", in Advances in Computers. Elsevier, 2019, Vol. 112, str. 275–378.

[148] Coles, H., Laurent, T., Henard, C., Papadakis, M., Ventresque, A., "Pit: a practical mutation testing tool for java", in Proceedings of the 25th international symposium on software testing and analysis, 2016, str. 449–452.

[149] Pizzoleto, A. V., Ferrari, F. C., Offutt, J., Fernandes, L., Ribeiro, M., "A systematic literature review of techniques and metrics to reduce the cost of mutation testing", Journal of Systems and Software, Vol. 157, 2019, str. 110388.

[150] Blanco, R., Trinidad, M., Suárez-Cabal, M. J., Calderón, A., Ruiz, M., Tuya, J., "Can gamification help in software testing education? findings from an empirical study", Journal of Systems and Software, Vol. 200, 2023, str. 111647.

[151] Lauvås Jr, P., Arcuri, A., "Recent trends in software testing education: A systematic literature review", in Norsk IKT-konferanse for forskning og utdanning, 2018.

[152] Šošić, S., Ristić, O., Milošević, M., "Game-based learning of software testing", in 2020 8th International Scientific Conference Technics and Informatics in Education, 2020.

[153] Zivkovic, T., Zivkovic, M., "Survey of learning environments for software testing education", in 7th Conference on the Engineering of Computer Based Systems, 2021, str. 1–9.

[154] Živković, M., Nikolić, B., Protić, J., Popović, R., "A survey and classification of wireless sensor networks simulators based on the domain of use", Adhoc and Sensor Wireless Networks, Vol. 20, 2014.

[155] Nikolic, B., Radivojevic, Z., Djordjevic, J., Milutinovic, V., "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization", IEEE Transactions on Education, Vol. 52, No. 4, 2009, str. 449–458.

[156] Prasad, P., Alsadoon, A., Beg, A., Chan, A., "Using simulators for teaching computer organization and architecture", Computer Applications in Engineering Education, Vol. 24, No. 2, 2016, str. 215–224.

[157] Živković, M. *et al.*, "Pregled primena virtuelnih okruženja u obrazovanju", in Sinteza 2019-International Scientific Conference on Information Technology and Data Related Research. Singidunum University, 2019, str. 99–106.

[158] Branovic, I., Popovic, R., Jovanovic, N., Giorgi, R., Nikolic, B., Zivkovic, M., "Integration of simulators in virtual 3d computer science classroom", in 2014 IEEE Global Engineering Education Conference (EDUCON). IEEE, 2014, str. 1–4.

[159] Balamuralithara, B., Woods, P. C., "Virtual laboratories in engineering education: The simulation lab and remote lab", Computer Applications in Engineering Education, Vol. 17, No. 1, 2009, str. 108–118.

[160] Dolgopolovas, V., Dagienė, V., "Computational thinking: Enhancing steam and engineering education, from theory to practice", Computer Applications in Engineering Education, Vol. 29, No. 1, 2021, str. 5–11.

[161] Caulfield, C., Xia, J. C., Veal, D., Maj, S., "A systematic survey of games used for software engineering education", Modern Applied Science, Vol. 5, No. 6, 2011, str. 28–43.

[162] Smith, R., Tang, T., Warren, J., Rixner, S., "An automated system for interactively learning software testing", in Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, 2017, str. 98–103.

[163] Tang, T., Rixner, S., Warren, J., "An environment for learning interactive programming", in Proceedings of the 45th ACM technical symposium on Computer science education, 2014, str. 671–676.

[164] Derezinska, A., Halas, K., "Experimental evaluation of mutation testing approaches to python programs", in 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops. IEEE, 2014, str. 156–164.

[165] Tang, T., Smith, R., Rixner, S., Warren, J., "Data-driven test case generation for automated programming assessment", in Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, 2016, str. 260–265.

[166] Tuya, J., Suarez-Cabal, M. J., De La Riva, C., "Sqlmutation: A tool to generate mutants of sql database queries", in Second Workshop on Mutation Analysis (Mutation 2006-ISSRE Workshops 2006). IEEE, 2006, str. 1–1.

[167] Trinidad, M., Calderón, A., Ruiz, M., "Gorace: a multi-context and narrative-based gamification suite to overcome gamification technological challenges", IEEE Access, Vol. 9, 2021, str. 65 882–65 905.

[168] Soska, A., Mottok, J., Wolff, C., "An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education", in 2016 IEEE Global Engineering Education Conference (EDUCON). IEEE, 2016, str. 576–584.

[169] Ramakrishnan, S., "Lightviews-visual interactive internet environment for learning oo software testing", in Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium. IEEE, 2000, str. 692–695.

[170] Yu, J., "Design of a lightweight autonomous learning system for the course of software testing based on android", in Journal of Physics: Conference Series, Vol. 1288, No. 1. IOP Publishing, 2019, str. 012051.

[171] Baggaley, J., "Mooc rampant", Distance education, Vol. 34, No. 3, 2013, str. 368–378.

[172] Buffardi, K., Valdivia, P., "Bug hide-and-seek: An educational game for investigating verification accuracy in software tests", in 2018 IEEE Frontiers in Education Conference (FIE). IEEE, 2018, str. 1–8.

[173] Paiva, A. C., Flores, N. H., Barbosa, A. G., Ribeiro, T. P., "ilearntest–framework for educational games", Procedia-Social and Behavioral Sciences, Vol. 228, 2016, str. 443–448.

[174] Gomes, J., Salgado, M. T., "Serious game to learn software testing", 2017.

[175] de Oliveira, B. C., "Testeg-um software educacional para o ensino de teste de software", Universidade Federal de Lavras, 2013.

[176] Oliveira, B., Afonso, P., Costa, H., "Testeg—a computational game for teaching of software testing", in 2016 35th International Conference of the Chilean Computer Science Society (SCCC). IEEE, 2016, str. 1–10.

[177] Barbosa, A., Neves, L., Neto, A., "Jovetest-jogo da velha para auxiliar no ensino e estudo de teste de software", IX Fórum de Educação em Engenharia de Software, 2016.

[178] Beppe, T. A., de Araújo, Í. L., Aragão, B. S., de Sousa Santos, I., Ximenes, D., Andrade, R. M. C., "Greatest: A card game to motivate the software testing learning", in Proceedings of the XXXII Brazilian Symposium on Software Engineering, 2018, str. 298–307.

[179] Jackson, S., "Munchkin", https://munchkin.game/, (Accessed on 11/12/2021). 2021.

[180] Collofello, J., Vehathiri, K., "An environment for training computer science students on software testing", in Proceedings Frontiers in Education 35th Annual Conference. IEEE, 2005, str. T3E–6.

[181] européenne. Commission européenne, U., européenne. Direction générale de la recherche, U., Science education now: A renewed pedagogy for the future of Europe. Office for Official Publications of the European Communities, 2007.

[182] Velev, D., Zlateva, P., "Virtual reality challenges in education and training", International Journal of Learning and Teaching, Vol. 3, No. 1, 2017, str. 33–37.

[183] Hastie, T., Rosset, S., Zhu, J., Zou, H., "Multi-class adaboost", Statistics and its Interface, Vol. 2, No. 3, 2009, str. 349–360.

[184] Al-Betar, M. A., Awadallah, M. A., Braik, M. S., Makhadmeh, S., Doush, I. A., "Elk herd optimizer: a novel nature-inspired metaheuristic algorithm", Artificial Intelligence Review, Vol. 57, No. 3, 2024, str. 48.

[185] Pierezan, J., Coelho, L. D. S., "Coyote optimization algorithm: a new metaheuristic for global optimization problems", in 2018 IEEE congress on evolutionary computation (CEC). IEEE, 2018, str. 1–8.

# Biografija autora

Tamara Živković, master inženjer elektrotehnike i računarstva, rođena je 28.11.1982. u Valjevu, Republika Srbija. Osnovnu školu „Žikica Jovanović Španac“ završila je u Valjevu kao nosilac Vukove diplome. Gimnaziju prirodno-matematičkog smera završila je u Valjevu sa odličnim uspehom. Od rane mladosti iskazivala je veliko interesovanje za prirodne nauke. Pored interesovanja za prirodne nauke pokazala je interesovanje i za debatu, književnost i strane jezike. Poseduje dve diplome Kembridž univerziteta za engleski jezik (nivoi B2 i C1).

Elektrotehnički fakultet u Beogradu, smer telekomunikacije i informacione tehnologije, upisala je 2001. godine. U decembru 2007. godine diplomirala je sa prosečnom ocenom 7.71. Diplomski rad „Asimetrični kriptografski algoritam eliptična kriva“ pod mentorstvom prof. dr Aleksandre Smiljanić odbranila je u decembru 2007. godine sa ocenom 10.

Master akademske studije na Elektrotehničkom fakultetu u Beogradu na studijskom programu Diplomske akademske studije Elektrotehnika i računarstvo – modul Audio i video tehnologije upisala je 2008. godine, a diplomirala u oktobru 2010. godine sa prosečnom ocenom 9.14. Master rad „Verifikacija jedne metode proračuna zvučne izolacije“ pod mentorstvom prof. dr Dragane Šumarac Pavlović odbranila je u oktobru 2010. godine sa ocenom 10.

Doktorske akademske studije na Elektrotehničkom fakultetu Univerziteta u Beogradu, na modulu Softversko inženjerstvo upisala je u oktobru 2012. godine. Položila je sve ispite sa ocenom 10 i ostvarila 120 ESPB.

Od aprila 2008. godine bila je zaposlena u beogradskom odseku kompanije P3 communications, čije je sedište u Ahenu, Nemačka. Radila je na projektima testiranja softvera mobilnih telefona u državama Evrope, Azije i Afrike, na poziciji test inženjera. Od januara 2012. godine do danas zaposlena je u američkoj kompaniji Merit Solutions. Radi na poziciji tim lidera i menadžera kvaliteta softvera.

Nosilac je međunarodno priznatih sertifikata iz oblasti testiranja softvera: ISTQB Certified Tester Foundation Level, ISTQB Certified Tester Advanced Level Test Manager i ISTQB Certified Tester Foundation Level Agile Tester. Fokus njenog istraživačkog rada su testiranje softvera i veštačka inteligencija.

# ИЗЈАВА О АУТОРСТВУ

Име и презиме аутора: **Тамара Живковић**
Број индекса: **2012/5040**

**Изјављујем**

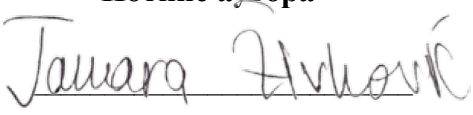
да је докторска дисертација под насловом

**Предвиђање дефеката у софтверу применом модела машинског учења оптимизованих метахеуристикама**

- резултат сопственог истраживачког рада;
- да дисертација у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио/ла интелектуалну својину других лица.

**Потпис аутора**

У Београду, 28.05.2024. год.



# ИЗЈАВА О ИСТОВЕТНОСТИ ШТАМПАНЕ И ЕЛЕКТРОНСКЕ ВЕРЗИЈЕ ДОКТОРСКОГ РАДА

Име и презиме аутора: **Тамара Живковић**

Број индекса: **2012/5040**

Студијски програм: **Електротехника и рачунарство, модул Софтверско инжењерство**

Наслов рада: **Предвиђање дефеката у софтверу применом модела машинског учења оптимизованих метахеуристикама**

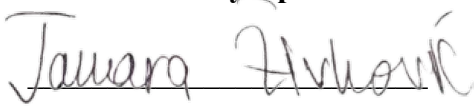
Ментор: **др Дражен Драшковић, ванредни проф.**

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла ради похрањена у **Дигиталном репозиторијуму Универзитета у Београду.** Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

**Потпис аутора**

У Београду, 28.05.2024. год.



# ИЗЈАВА О КОРИШЋЕЊУ

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

**Предвиђање дефеката у софтверу применом модела машинског учења оптимизованих метахеуристикама**

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство (CC BY)
2. Ауторство – некомерцијално (CC BY-NC)
3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)
4. **Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)**
5. Ауторство – без прерада (CC BY-ND)
6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.

Кратак опис лиценци је саставни део ове изјаве).

**Потпис аутора**

У Београду, 28.05.2024. год.

Тамара Лукић

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.