Predviđanje pogrešaka izvornoga kôda zasnovano na otkrivanju anomalija korištenjem semantičkih značajki izlučenih primjenom autoenkodera na leksičke reprezentacije izvornoga kôda

Afrić, Petar

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Petar Afrić

PREDVIĐANJE POGREŠAKA IZVORNOGA KÔDA ZASNOVANO NA OTKRIVANJU ANOMALIJA KORIŠTENJEM SEMANTIČKIH ZNAČAJKI IZLUČENIH PRIMJENOM AUTOENKODERA NA LEKSIČKE REPREZENTACIJE IZVORNOGA KÔDA

DOKTORSKI RAD

Zagreb, 2023.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Petar Afrić

PREDVIĐANJE POGREŠAKA IZVORNOGA KÔDA ZASNOVANO NA OTKRIVANJU ANOMALIJA KORIŠTENJEM SEMANTIČKIH ZNAČAJKI IZLUČENIH PRIMJENOM AUTOENKODERA NA LEKSIČKE REPREZENTACIJE IZVORNOGA KÔDA

DOKTORSKI RAD

Mentor: Izv. prof. dr. sc. Marin Šilić

Zagreb, 2023.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Petar Afrić

SOURCE CODE DEFECT PREDICTION BASED ON ANOMALY DETECTION USING SEMANTIC FEATURES EXTRACTED BY APPLYING AUTOENCODERS TO LEXICAL SOURCE CODE REPRESENTATIONS

DOCTORAL THESIS

Supervisor: Associate professor Marin Šilić, PhD

Zagreb, 2023

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Mentor: izv. prof. dr. sc. Marin Šilić

Doktorski rad ima: 134 stranica

Doktorski rad br.: _____

O mentoru

Marin Šilić je rođen 1983. godine u Sarajevu u Bosni i Hercegovini. Osnovno obrazovanje pohađao je u Makarskoj u Hrvatskoj. Diplomirao je 2007. godine na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva. Kao student bio je primatelj stipendije Ministarstva Znanosti Republike Hrvatske i dio naprednog programa završetka diplomskog studija s posebnim fokusom na istraživački rad. Nakon diplome od strane fakulteta dodijeljena mu je nagrada "Josip Lončar" kao jednom on najboljih studenata generacije u području računarstva. Od 2007. godine zaposlen je kao doktorand na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva. Tijekom doktorskog studija boravio je 6 mjeseci na znanstvenom usavršavanju u New Yorku u kompaniji Google kao član *Google Docs* tima radeći na razvoju sustava *Google Spreadsheets*. Doktorirao je na Sveučilištu u Zagrebu, Fakultetu Elektrotehnike i Računarstva 2013. godine gdje je trenutno zaposlen kao izvanredni profesor. Rezultate svojih istraživanja opisao je u radovima koji su objavljeni u uglednim strukovnim časopisima kao što su *IEEE Transactions on Services Computing*, *IEEE Transactions od Dependable and Secure Computing*, *Journal of Systems and Software*, *Knowledge Based Systems* i *IEEE Access*. Nadalje, svoja istraživanja objavio je i na prestižnim skupovima istraživača područja programskog inženjerstva *ACM SIGSOFT Symposium on the Foundations of Software Engineering* i *IEEE International Conference on Software Quality, Reliability and Security*. Sudjelovao je kao voditelj i suradnik na više stručnih i znanstvenih projekata na FER-u gdje je do sada uspješno mentorirao dvije doktorske disertacije. Fokus njegovog istraživanja trenutno su računarstvo zasnovano na uslugama, sustavi za preporučivanje, analiza velikih skupova podataka, umjetna inteligenciju i strojno učenje te previđanje pogrešaka programskog koda. Član je strukovne udruge IEEE.

About the Supervisor

Marin Šilić was born in 1983. in Sarajevo in Bosnia and Herzegovina. He attended in basic education in Makarska in Croatia. He graduated in 2007. from the Faculty of Electrical Engineering and Computing, University of Zagreb. As a student he received a scholarship from the Croatian Ministry of Science and was part of an advanced program with special emphasis on the research work. Upon graduation he was awarded the "Josip Lončar" as one of the best graduating students in computing of his generation. He has worked at the Faculty of Electrical Engineering and Computing, University of Zagreb since 2007. During his Ph.D. he spent 6 months in New York, US at Google working as part of the *Google Docs* team focusing on the development of the *Google Spreadsheets List View System*. He got his Ph.D. from the Faculty of Electrical Engineering and Computing, University of Zagreb in 2013. and is currently employed there as an Associate Professor. He has published several papers in *IEEE Transactions*

on Services Computing, *IEEE Transactions on Dependable and Secure Computing*, *Journal of Systems and Software*, *Knowledge-Based Systems* and *IEEE Access*. Also, he has published his research results at the *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* and at the *IEEE International Conference on Software Quality, Reliability and Security*. He has been a lead and an associate on multiple practical and scientific projects at the FER where he has also successfully mentored two Ph.D. candidates. His research interests include service-oriented computing, recommender systems, big data analysis, artificial intelligence, machine learning, and software defect prediction. He is a member of IEEE.

Ovaj doktorski rad želim posvetiti svojoj supruzi Pauli i našoj djeci.

Zahvale

Za početak želim se zahvaliti svom mentoru izv. prof. dr. sc. Marinu Šiliću koji mi je svojim znanjem i iskustvom bio glavna podrška i nit vodilja kroz doktorski studij.

Želim se zahvaliti svojoj obitelji za svu podršku koju su mi pružili za vrijeme doktorskog studija.

Posebno se želim se zahvaliti i svim djelatnicima i suradnicima Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu koji su na razne načine doprinijeli mom iskustvu pohađanja doktorata.

Sažetak

Cilj predviđanja pogrešaka u izvornom programskom kôdu je otkrivanje neispravnih programskih modula kako bi se što bolje alocirali ograničeni resursi za osiguravanje kvalitete programskog kôda. Prilikom izgradnje podatkovnih skupova, za razvoj modela za predviđanje pogrešaka u izvornom programskom kôdu, podatci se uzimaju iz sustava za praćenje zadataka i sustava za kontrolu verzija izvornog programskog kôda. Kvaliteta podataka u ovim izvorima utječe na rad i evaluaciju razvijenih modela. Klasifikacija zadataka je bitna jer se kasnije koristi za određivanje ispravnosti programskih modula koji će biti korišteni za treniranje modela. Dodatno, podatkovni skupovi korišteni za predviđanje pogrešaka programske potpore su često neuravnoteženi, sadržavajući veći broj primjera programskih modula ne-sklonih pogreškama nego onih sklonih pogreškama. U sklopu ove disertacije istražen je utjecaj klasifikacije zadataka na kvalitetu podatkovnih skupova za predviđanje pogrešaka programske potpore i rad modela treniranih na tim skupovima. Dodatno, predložen je model koji problem tretira kao problem detekcije anomalije čime zaobilazi problem neuravnoteženosti razreda.

Ključne riječi: predviđanje pogrešaka programske potpore, kvaliteta programske potpore, otkrivanje anomalija, klasifikacija zadataka, neuravnoteženost razreda

Extended Abstract

Source code defect prediction based on anomaly detection using semantic features extracted by applying autoencoders to lexical source code representations

Today's importance of software is irrefutable and continues to grow exponentially, primarily due to the inherent reality that software has seamlessly woven itself into the very fabric of an ever-expanding array of products across industries. Indeed, software has transcended its conventional boundaries to proliferate into an extensive array of specialized variants, each tailored to cater to distinct functionalities within the technological landscape. These variants encompass firmware, operating systems, device drivers, compilers, user applications, and a myriad of other specialized categories. Moreover, user applications, in themselves, exhibit a noteworthy diversification, encompassing desktop applications, web-based applications, mobile applications, and a multitude of other specialized applications serving unique purposes.

This intricate web of software manifestations not only underscores the omnipresence of software in the contemporary world but also highlights its remarkable complexity. In essence, the role of software in modern society is not only pivotal but also multi-dimensional, continually evolving to meet the ever-growing demands of a technologically-driven global landscape.

Software defects can have significant ramifications, extending beyond mere financial losses to encompass critical safety concerns that imperil both individuals and organizations. A software defect, in essence, refers to any instance where a software product deviates from its intended or expected behavior, whether through the manifestation of erroneous outcomes, unexpected actions, or unanticipated responses. Software defects are usually a result of a lack of software development methodology or a lack of software specification. In order to reduce the impact of software defects, quality assurance has become an integral part of software development. Quality assurance is a term referring to all activities which aim to improve the quality and reliability of software products. These activities can include pair programming, code review, software testing, software defect prediction and many others.

The realm of quality assurance is an endeavor that demands a substantial investment of both time and resources. Within the broader context of software development, it is not uncommon to observe that a significant proportion of the allocated resources is dedicated to the intricate process of software testing. However, software testing is not always possible due to a lack of time and resources. Software defect prediction aims to reduce the needed time and resources by flagging high risk software modules, thus focusing testing efforts. This is achieved by identifying and singling out software modules that pose a heightened risk of harboring defects. By pinpointing these high-risk components, software defect prediction effectively directs and concentrates testing efforts in a strategic and efficient manner.

At its core, the concept of software defect prediction hinges upon the premise that sof-

tware projects sharing similarities, whether in terms of their development context or underlying characteristics, are likely to exhibit analogous behaviors, including the manifestation of similar software defects. Researchers mine version control systems and issue tracking systems to construct software defect prediction datasets and develop software defect prediction models.

Researchers rely on commit to issue links and issue classification when constructing software defect prediction datasets. Code adjusted by commits linking to bug related issues is considered as defect prone and is the main interest of software defect prediction. However, this process is prone to noise as modules where a defect is present, but not reported will be considered non defective. Similarly, if links between commits and issues can not be established an uncertainty about their defectiveness will be unavoidable. If we presume all such modules are defective then some non defective modules will be mislabeled. On the other had, if we presume all are non-defective than some defective modules will be mislabeled. Finally, even when links between issues and commits are established, researchers often do not question the quality of issue classification inside issue tracking systems. On top of the noise related problems, software defect prediction datasets often suffer from class imbalance problems, having far more non-defect-prone examples than defect-prone ones. This is a consequence of how these datasets are created, but also of software development as such. Software development inherently strives to avoid software defects, both during development and with additional quality assurance methods. Additionally, defective software which is never reported as such further increases the imbalance problem.

In this dissertation the author investigates the impact of issue classification on the quality of software defect prediction datasets and resulting models. Furthermore, the dissertation introduces a novel software defect prediction approach grounded in the principles of anomaly detection. This innovative methodology reimagines the identification of defect-prone software modules by considering them as anomalies within the broader software ecosystem. This unique perspective implicitly addresses the class imbalance problem that often plagues traditional defect prediction models. By reframing the problem in this manner, the proposed approach offers a fresh perspective on defect prediction, one that sidesteps the conventional pitfalls associated with skewed data distributions.

In sum, this dissertation represents a contribution to the field of software defect prediction. It underscores the importance of issue classification, shedding light on its far-reaching implications for the quality of datasets and predictive models. Simultaneously, the introduction of an anomaly-based approach adds a novel dimension to the ongoing discourse in defect prediction, promising to advance the state of the art by mitigating class imbalance issues and fostering more robust predictive capabilities in software quality assurance.

The dissertation is organized as follows.

Chapter 1 (*Uvod*) serves as a foundational introduction to the field of software defect pre-

diction, it shortly presents and contextualizes the author's contributions to the field, and lays out the rest of the dissertation charting the road-map for the remainder of this comprehensive dissertation.

Chapter 2 (*Predviđanje pogrešaka programske potpore*) describes software defect prediction in more detail. It provides details about the overall process of software defect prediction, SDP dataset development, the common problems of SDP datasets, publicly available SDP datasets, features used for SDP, field divisions based on the type of prediction and field division based on the models used for software defect prediction. Finally, it presents the metric used for model evaluation, and statistical methods used for result validation. Model development is a stochastic process, meaning that if the same model is trained on the same dataset, it will not always result in the same final model. This can be a consequence of random model initialization or data shuffling after every epoch of model training. Given this stochastic nature of developed models it is not enough to compare single instances of different models. Instead, evaluation results of repeated model training need to be compared between different models using statistical methods.

Chapter 3 (*Određivanje klase zadatka dodijeljenog razvijatelju programskog kôda*) describes the field of issue classification. Research in this field deals with issue classification without considering software defect prediction. As part of this dissertation the author investigated the impact of issue classification on the quality of software defect prediction datasets and resulting models.

Chapters 4 and 5 stand as the cornerstone of the author's scientific contributions as part of this dissertation. Chapter 4 presents the analysis of how issue classification influences the amount of noise in software defect prediction datasets and how that influences the performance of resulting software defect prediction models. Chapter 5 presents a novel approach to software defect prediction based on the idea of anomaly detection. Such an approach inherently sidesteps the problem of class imbalance models often have to deal with.

Chapter 4 (*Utjecaj klasifikacije zadataka na podatkovne skupove i rad modela predviđanja pogrešaka programske potpore*) presents the analysis of how issue classification influences the quality of software defect prediction datasets and resulting models. For the purpose of this analysis, 7 open-source repositories were mined for data. More specifically, the version control system and issue tracking system of each repository was mined, all of them coming from the popular open-source platform GitHub. The collected data was used to construct datasets for issue classification and, further on, datasets for software defect prediction. In order to create issue classification datasets a labeling application was developed, while software defect prediction datasets were constructed based on the labeled issues and features constructed using both the source code and the process of its development. Thus, the resulting software defect prediction datasets consist of both classic and semantic code complexity features and simple code development process features. Four methods for issue classification were investigated: *a*

keyword heuristic, *an advanced keyword heuristic*, *FastText model* and *RoBERTa model*. The obtained results show that using RoBERTa for issue classification results in the best datasets and models for software defect prediction. On average, datasets constructed using the RoBERTa model contain 14.3641 % of mislabeled instances. In 65 out of 84 cases, models trained on these datasets achieve superior performance when compared to models trained on datasets constructed using other issue classification methods. In 55 out of those 65 cases the results are shown to be statistically relevant using a normality test and depending on the results either a student t-test or a Mann-Whitney U test. Furthermore, in 17 out of 28 cases there is no statistical difference between software defect prediction models trained on datasets created using the RoBERTa model for issue classification, and models trained on datasets derived from manual issue classification. The created datasets and obtained results represent scientific contributions of the author of this dissertation.

Chapter 5 (*Predviđanje pogrešaka programske potpore zasnovano na otkrivanju anomalija*) presents a novel approach to software defect prediction based on the idea of anomaly detection. Software defect prediction datasets often suffer from class imbalance, having far more examples of non-defect-prone code than of defect-prone code. This class imbalance problem is a result of the fact that developers try to minimize the occurrence of software defects, but also the fact that a lot of effort is put into software quality assurance and the fact that there is a bias towards false negatives as defective software which was never marked as such will be considered as non-defective. As a way to get around the class imbalance problem the proposal is to treat the problem as an anomaly detection problem. All code is considered as non-defective, aka. healthy code, while defect-prone code represents an anomaly. The proposed model is called REPD which stands for Reconstruction Error Probability Distribution. It is based on the idea that an autoencoder which has learned to reconstruct non-defective example will not be good at reconstructing defective examples as they are anomalies. The proposed model is compared to five other models: Logistic Regression, Decision Tree, Naive Bayes, K-Nearest Neighbors and Hybrid SMOTE-Enseble. The first four are models often used in the field of source code defect prediction, while the last model Hybrid SMOTE-Enseble represents a State-of-the-Art model. Comparison of the proposed model and the alternative models is done on 5 datasets based on classic code complexity software defect prediction features, and, an addition, 24 datasets based on semantic features which are constructed by applying autoencoders and deep belief networks to the lexical representations of the source code. The obtained results show the superior performance of the proposed model with it improving the obtained F1 score, in some cases, up to 7.12 %. Finally, the robustness of the proposed model to the class imbalance problem is investigated. This is done by over-sampling and under-sampling several datasets, changing the share of defective modules from 20% to 5% with a 1% step and tracking the influence this has on the average F1 score of models used for comparison. The method for constructing semantic

features using an autoencoder model on the lexical representations of the source code and the proposed REPD model are scientific contributions of the author of this dissertation.

Finally, chapter 6 (*Zaključak*) draws this dissertation to a close, it reaffirms and underscores the scientific contributions of this dissertation and takes the opportunity to shed light on their broader implications and importance within the broader academic and practical context.

Key words: software defect prediction, software quality, anomaly detection, issue classification, class imbalance

Sadržaj

Poglavlje 1

Uvod

Godine 1945. razvijen je ENIAC (engl. Electronic Numerical Integrator and Computer, skr. ENIAC) [1] koji je bio prvo programabilno računalo opće primjene. Imao je 27 tona i prvenstveno ga je koristila američka vojska za artiljerijske izračune. Od tada, do pisanja ovog rada prošlo je skoro 80 godina, a računala su od masivnih strojeva visoko specijalizirane namjene postala svakodnevna pojava.

Prva računala zahtijevala su razvoj programa (engl. Computer Program) u jezicima koji su bili specifični za računalo na kojem su se programi izvršavali. Radilo se o ozbiljnom ograničenju jer su napisani programi bili vezani za računalo za koje su razvijeni. Godine 1956. Chomsky [2] je opisao hijerarhiju jezika. Radilo se o važnim teorijskim temeljima za razvoj područja prevođenja programskih jezika. Razvojem područja nastali su viši programski jezici i programi za njihovo prevođenje u stroju razumljive instrukcije. Ovo je omogućilo inženjerima razvoj programske potpore (engl. Software) koja nije bila usko vezana uz fizički stroj na kojem su se programi izvršavali. Umjesto toga, razvijeni su programi koji bi programsku potporu prvo preveli u stroju razumljiv jezik, a potom bi bilo moguće njeno izvršavanje. Ovo je omogućilo značajno brži razvoj programske potpore jer je jednom napisane programe bilo moguće puno lakše ponovno primjenjivati.

Razvoj programskih jezika i programske potpore pratio je i rapidni razvoj računalnog hardvera. Postavljen je *Moore-ov zakon* [3] koji kaže da se broj tranzistora na čipu udvostruči svakih 24 mjeseca. Radi se o empirijskoj opservaciji o brzini razvoja računalnog hardvera. Slično Moore-ovom zakonu postavljen je i *Kryder-ov zakon* [4] koji kaže da se količina podataka na tvrdom disku udnostruči svakih 13 mjeseci.

Današnji značaj programske potpore je neosporan i svakim danom sve veći s obzirom na to da programska potpora postaje sastavni dio sve većeg broja proizvoda [5]. Procjenjuje se da je 2019. godine u svijetu postojalo 2 milijarde osobnih računala, 5 milijardi pametnih telefona, 8 milijuna podatkovnih centara i 20 milijardi IoT uređaja (engl. Internet of Things, skr. IoT) [6]. Svi navedeni uređaji zahtijevaju programsku potporu, a do danas njihov broj se sigurno

povećao.

Uz velik broj uređaja koji zahtijevaju programsku potporu, sama potpora postala je izuzetno raznolika. Razvijeni su *programi trajno upisani u memoriju* (engl. Firmware) za kontrolu fizičkih uređaja, *operacijski sustavi* (engl. Operating System, skr. OS) koji olakšavaju korištenje računala pružajući grafičko sučelje za njegovo korištenje, standardizirani način instalacije aplikacija i mnoge druge funkcionalnosti, *pokretači uređaja* (engl. Device Driver) koji omogućuju komunikaciju između operacijskih sustava i fizičkih uređaja, *prevoditelji programskih jezika* koji omogućuju izvođenje programske potpore na različitim računalnim arhitekturama i operacijskim sustavima te mnogi *korisnički programi* (engl. Applications). Ponovno, raznolikost korisničkih aplikacija je izuzetno velika, pa tako postoje *mrežne aplikacije* (engl. Web Application), *aplikacije za osobna računala* (engl. Desktop Application), *mobilne aplikacije* (engl. Mobile Applications) koje se i same granaju u velik broj podtipova.

Uz svoju raznolikost, moderna programska potpora izuzetno je velika i često zahtjeva detaljne specifikacije te velike timove inženjera za njen razvoj i održavanje. Primjera radi, operacijski sustav Windows ima više od 70 milijuna linija kôda kada se uključi sva programska potpora potrebna za njegov rad [7].

Specifikacija programske potpore (engl. Software Specification) je dokument koji navodi zahtjeve koje programska potpora mora ispunjavati. Uključuje *funkcionalne* (engl. Functional), *ne-funkcionalne* (engl. Non-Functional) i *domenske* (engl. Domain) zahtjeve. *Funkcionalni zahtjevi* opisuju funkcionalnost programske potpore. *Ne-funkcionalni zahtjevi* opisuju parametre rada programske potpore. *Domenski zahtjevi* su specifični za domenu za koju se razvija programska potpora.

Pogreška programske potpore (engl. Software Bug) je defektno ponašanje programske potpore uzrokovano kršenjem pravila rada računalnog programa. Pogreške programske potpore najčešće nastaju kao posljedica neprikladnog razvoj programske potpore i pogrešaka u specifikaciji programske potpore [8]. Pogreške programske potpore uzrokuju neočekivano ili netočno ponašanje programske potpore [9]. Ovi problemi mogu biti uzrokovani sintaksnim pogreškama u izvornom programskom kôdu, logičkim pogreškama, problemima s memorijom, mrežnim problemima i mnogim drugim uzrocima.

Američki institut za standarde i tehnologiju definirao je radni okvir za pogreške programske potpore [*]. Radi se o strukturiranoj i potpunoj klasifikaciji pogrešaka programske potpore, koja nije ovisna o specifičnom programskom jeziku.

Pogreške u programskoj potpori mogu dovesti do velikih financijskih gubitaka pa čak i narušavanja kritične sigurnosne infrastrukture. Istraživači procjenjuju da su pogreške programske potpore i loše osiguranje kvalitete 2020 godine ekonomiju Sjedinjenih Američkih Država koštali $2.08 bilijuna dolara [10]. Dodaju li se ovom procjene da će do 2030 godine u svijetu

[*] `https://www.nist.gov/itl/ssd/software-quality-group/samate/bugs-framework`

nedostajati 85.2 milijuna programskih inženjera, što će imati negativan ekonomski učinak od 8.452 bilijuna dolara, [11] jasna postaje važnost osiguranja kvalitete programske potpore.

Osiguravanje kvalitete programske potpore (engl. Software Quality Assurance) opisuje niz aktivnosti koje imaju zajednički cilj poboljšanja kvalitete i pouzdanosti programske potpore te osiguranja da ona radi u skladu sa svojom specifikacijom i zadovoljava potrebe korisnika. Pronalaženje i uklanjanje pogrešaka programske potpore izuzetno je važno za osiguravanje kvalitete programske potpore.

Aktivnosti osiguranja kvalitete programske potpore uključuju aktivnosti planiranja, kontrole i nadzora procesa razvoja programske potpore, kao i praćenje i izvješćivanje o rezultatima testiranja i evaluacije kvalitete. Primjeri aktivnosti osiguranja kvalitete programske potpore su *razvoj programske potpore u paru* (engl. Pair Programming) [12], *pregledavanje izvornog programskog kôda* (engl. Code Inspections) [13], *prezentacija izvornog programskog kôda* (engl. Code Walkthroughs) [14], *testiranje programske potpore* (engl. Software Testing) [15, 16] te *predviđanje pogrešaka programske potpore* (engl. Software Defect Prediction skr. SDP) [5, 17].

Osiguranje kvalitete programske potpore zahtijeva značajne količine vremena i resursa. Testiranje programske potpore često koristi velik udio resursa dodijeljenih za njen razvoj [9, 18, 19]. Postoje različite vrste testiranja programske potpore. *Testiranje jedinica* (engl. Unit Testing) je testiranje pojedinačnih komponenti izvornog programskog kôda kako bi se provjerila njihova funkcionalnost i ispravnost. *Integracijsko testiranje* (engl. Integration Testing) testira kako pojedinačne komponente izvornog programskog kôda rade zajedno i kako se ponašaju u međusobnoj interakciji. *Sistemsko testiranje* (engl. System Testing) testira cjelokupnu programsku potporu kako bi se provjerila funkcionalnost i performanse na različitim razinama. *Prihvatljivost programske potpore korisniku* (engl. User Acceptance testing) je vrsta testiranja koje se provodi nad krajnjim korisnicima kako bi se provjerilo zadovoljava li programska potpora njihove potrebe i zahtjeve. *Sigurnosno testiranje* (engl. Security Testing) je vrsta testiranja koje provjerava sigurnost programske potpore i nastoji otkriti potencijalne sigurnosne propuste. *Testiranje kvalitete izvođenja*(engl. Performance Testing) provjerava brzinu, skalabilnost i stabilnost programske potpore. Svaka vrsta testiranja ima svoju važnost i doprinosi poboljšanju kvalitete programske potpore. Izbor vrste testiranja ovisi o zahtjevima i specifikacijama programske potpore.

Detaljno testiranje programske potpore nije uvijek moguće jer bi zahtijevalo previše vremena i resursa. Predviđanje pogrešaka programske potpore nastoji skratiti vrijeme i bolje rasporediti resurse za osiguranje kvalitete programske potpore predviđanjem modula programske potpore koji sadrže pogreške ili su im skloni [9]. Modul programske potpore predstavlja jedinicu izvornog programskog kôda za koju se vrši predviđanje pogrešaka. Predviđanje pogrešaka programske potpore zasniva se na ideji da će slična programska potpora ili programska potpora razvijena pod sličnim uvjetima biti sklona pogreškama ako je i izvorna programska potpora sa-

državala pogreške [20]. SDP omogućuje fokusiranje resursa za osiguranje kvalitete programske potpore na njene visoko rizične dijelove [21]. Abaei i Selamat navode kako SDP zadovoljava tri važna zahtjeva. Prvo, pomaže u procjeni napretka programske potpore i planiranju procesa testiranja. Drugo, pomaže u procjeni kvalitete programske potpore. Konačno, doprinosi pouzdanosti programske potpore [22]. Podatci potrebni za izgradnju modela za predviđanje pogrešaka programske potpore sakupljaju se iz sustava za upravljanje verzijama kôda (engl. Version Control System, skr. VCS) i sustava za praćenje zadataka (engl. Issue Tracking System, skr. ITS).

Sustav upravljanje verzijama izvornog programskog kôda je programska potpora koja se koristi za pohranu izvornog programskog kôda i praćenje njegovih promjena te pohranu i praćenje promjena u dodatnim resursima potrebnim za rad programske potpore. Sustav omogućuje inženjerima suradnju na projektima, pregled promjena kroz vrijeme te povratak na prijašnje verzije izvornog programskog kôda. Dodatno, moguća je provjera autora i točno vrijeme promjena izvornog programskog kôda. Primjeri VCS sustava su Git, SVN i Mercurial, a postoje i drugi.

Sustav za praćenje zadataka je programska potpora koja se koristi za praćenje zadataka u projektima koji razvijaju i održavaju programsku potporu. Zadatke je moguće dijeliti ovisno o vrsti zahtjeva na koji se odnose, pa mogu biti zahtjev za popravkom pogreške, zahtjev za novom funkcionalnošću, zahtjev za ažuriranjem rada programske potpore ili neka druga vrsta zahtjeva. Sustav za praćenje zadataka omogućuje inženjerima dodjelu zadatka odgovarajućem inženjeru ili grupi inženjera. Također, omogućuje praćenje napretka u rješavanju zadatka. Primjer sustava za praćenje zadataka je Jira, a postoje i drugi.

Temeljem izvornog programskog kôda sakupljenog iz sustava za kontrolu verzija kôda grade se značajke kojima se opisuju programski moduli. Temeljem informacija o zadacima prikupljenih iz sustava za praćenje zadataka i promjenama koje su bile potrebne u izvornom programskom kôdu prikupljenih iz sustava za kontrolu verzija, određuje se sklonost programskih modula pogreškama. Značajke koje opisuju programske module i informacije o njihovoj sklonosti pogreškama čine podatkovni skup za predviđanje pogrešaka programske potpore. Koristeći navedene podatkovne skupove razvijaju se modeli za predviđanje pogrešaka programske potpore. Razvijeni modeli se vrednuju na dijelu skupa koji nije bio korišten za vrijeme njihova razvoja. Ako se pokažu zadovoljavajućim, ugrađuju se u ciklus razvoja programske potpore kako bi doprinijeli osiguranju njene kvalitete. Kako veličina programske potpore raste, predviđanje njenih pogrešaka imat će sve važniju ulogu u pouzdanom plasiranju programske potpore na tržište [9]. Catal i Diri navode kako je predviđanje pogrešaka programske potpore postalo jedno od najvažnijih područja istraživanja u razvoju programske potpore[23].

Područje predviđanja programske potpore izuzetno je aktivno i puno izazova. Li et al. navode kako se broj radova objavljenih u ovom području konstantno povećava [21], ali isto tako Thota et al. navode kako velik broj predloženih pristupa daje loše rezultate kada se testira na

većem broju podatkovnih skupova [9].

Potencijalni razlog za nepouzdan rad velikog broja modela za predviđanje pogrešaka programske potpore mogla bi biti kvaliteta podatkovnih skupova koji se koriste za njihovu izgradnju. Naime, istraživači se oslanjaju na veze između *zadataka* (engl. Issue) i *predaja promjene programskog kôda* (engl. Commit) pri izgradnji podatkovnih skupova za predviđanje pogrešaka programske potpore. Kôd ažuriran *predajom promjene* povezanom sa zadatkom koji je označen kao *zahtjev za popravkom pogreške* smatra se *sklon pogreškama* (engl. Defect Prone). Problem je što istraživači često slijepo vjeruju *oznakama* (engl. Label) zadataka u sustavima za praćenje zadataka, iako istraživanja pokazuju da su oznake često pogrešno pridijeljene zadatcima [24, 25, 26, 27]. Dodatno, podatkovni skupovi za predviđanje pogrešaka programske potpore često pate od problema neuravnoteženosti razreda (engl. Class Imbalance) [28, 29, 30, 31, 32, 33, 34, 35]. Naime, primjera kôda koji nije sklon pogreškama ima mnogo više nego primjera kôda koji je sklon pogreškama. Ovo predstavlja ozbiljan izazov mnogim modelima za predviđanje pogrešaka programske potpore i jedan je od razloga zašto se za njihovo vrednovanje ne koristi mjera točnosti (engl. Accuracy). Točnost (eng. Accuracy) je mjera kvalitete rada modela koja opisuje koliko često model daje točne odgovore u odnosu na ukupan broj predviđanja. U situacijama neuravnoteženosti razreda model može postići visoku točnost predviđanja većine primjera, ali će biti loš u predviđanju rijetkih primjera. U takvim situacijama, točnost ne daje potpunu sliku o kvaliteti rada modela. Dodatno, točnost ne uzima u obzir posljedice pogrešnih predviđanja. Na primjer, programski modul označen kao nesklon pogreškama svojim pogrešnim radom može izazvati veliku financijsku štetu, pa je bolje da model programski modul nesklon pogreškama proglasi sklonim pogreškama nego obrnuto.

U ovoj disertaciji analiziran je utjecaj krivo označenih zadataka, u sustavima na praćenje zadataka, na kvalitetu podatkovnih skupova i modela za predviđanje pogrešaka programske potpore [36] te model za predviđanje pogrešaka programske potpore koji se zasniva na ideji otkrivanja anomalija [37] tretirajući pojavu izvornog programskog kôda koji sadrži pogreške kao anomaliju, a ispravni kôd kao normalne primjere kôda.

U nastavku disertacije detaljno je opisana analiza utjecaja krivo označenih zadataka te razvijeni model zasnovan na ideji otkrivanja anomalija i rezultati njegova vrednovanja na nekoliko različitih skupova za predviđanje pogrešaka programske potpore.

U poglavlju 2 detaljno je opisano područje predviđanja pogrešaka programske potpore. Opisan je općeniti proces predviđanja pogrešaka programske potpore, proces izgradnje skupova podataka za razvoj modela, česti nedostatci tih podatkovnih skupova, javno dostupni skupovi podataka, značajke koje se koriste za opis programskih modula, podjela područja ovisno o tipu predviđanja i ovisno o modelima koji se koriste za predviđanje, mjere za vrednovanje modela i statističke metode za validaciju rezultata vrednovanja.

U poglavlju 3 dan je kratak pregled područja klasifikacije zadataka tj. određivanja klase

zadatka dodijeljenog razvijatelju programskog kôda. Istraživanja u ovom području obično se bave klasifikacijom zadataka ne razmatrajući njen utjecaj na predviđanje programske potpore. U sklopu ove disertacije napravljena je analiza utjecaja klasifikacije zadataka na kvalitetu podatkovnih skupova i modela za predviđanje pogrešaka programske potpore.

Poglavlja 4 i 5 predstavljaju znanstvene doprinose autora ove disertacije.

U poglavlju 4 prezentirana je analiza utjecaja klasifikacije zadataka na kvalitetu podatkovnih skupova i rad modela predviđanja pogrešaka programske potpore. Za potrebe analize prikupljeni su podatci 7 javno dostupnih repozitorija otvorenog programskog kôda. Koristeći prikupljene podatke izgrađeni su podatkovni skupovi za klasifikaciju zadataka i podatkovni skupovi za predviđanje pogrešaka programske potpore. Razmotrene su različite metode klasifikacije zadataka: *heuristika zasnovana na ključnim riječima*, *unaprijeđena heuristika zasnovana na ključnim riječima*, *FastText model* i *RoBERTa model*. Prikupljeni rezultati pokazuju da korištenje RoBERTa modela rezultira najboljim podatkovnim skupovima za predviđanje pogrešaka programske potpore. Podatkovni skupovi izgrađeni primjenom RoBERTa modela u prosjeku sadrže 14.3641 % krivo označenih primjera. U 65 od 84 slučaja, modeli trenirani na podatkovnim skupovima dobivenim korištenjem RoBERTa modela ostvaruju bolje rezultate vrednovanja nego isti modeli trenirani na podatkovnim skupovima dobivenim korištenjem drugih pristupa klasifikacije zadataka. Od 65 slučajeva 55 pokazuje statistički značajnu razliku. Nadalje, u 17 od 28 slučajeva ne postoji statistički značajna razlika u rezultatima vrednovanja modela treniranih na podatkovnim skupovima dobivenim korištenjem RoBERTa modela i podatkovnim skupovima dobivenim ručnim označavanjem zadataka. Napravljeni podatkovni skupovi su javno objavljeni i uz rezultate analize predstavljaju znanstvene doprinose autora. Cijelo istraživanje je objavljeno u obliku znanstvenog članka [36], a programski kôd napravljen je javno dostupnim [38].

U poglavlju 5 prezentiran je novi pristup predviđanju pogrešaka programske potpore. Podatkovni skupovi za razvoj modela za predviđanje pogrešaka programske potpore često pate od problema neuravnoteženosti razreda, zbog toga predloženi pristup tretira SDP kao problem otkrivanja anomalija. Anomalije u ovom slučaju predstavljaju primjere programskih modula koji sadrže pogreške. Predstavljeni pristup naziva se REPD (engl. Reconstruction Error Probability Distribution, skr. REPD) što je skraćenica za vjerojatnosnu distribuciju grešaka rekonstrukcije. Rad modela uspoređen je s radom pet alternativnih modela na 5 podatkovnih skupova koji koriste klasične značajke složenosti programskog kôda. Dodatno, uspoređen je i na 24 podatkovna skupa koji koriste semantičke značajke izgrađene primjenom autoenkoder modela na leksičke jedinke izvornog programskog kôda. Rezultati vrednovanja REPD modela pokazuju superiornost njegova rada povećavajući iznos ostvarene F1 mjere za 7.12 %. Za kraj analiziran je utjecaj neuravnoteženosti razreda na rad REPD modela. Metoda izgradnje semantičkih značajki primjenom autoenkoder modela na leksičke jedinke izvornog programskog kôda i REPD model za

predviđanje pogrešaka programske potpore zasnovan na ideji otkrivanja anomalija predstavljaju znanstvene doprinose autora i objavljeni su u obliku znanstvenog članka [37], a programski kôd napravljen je javno dostupnim [39].

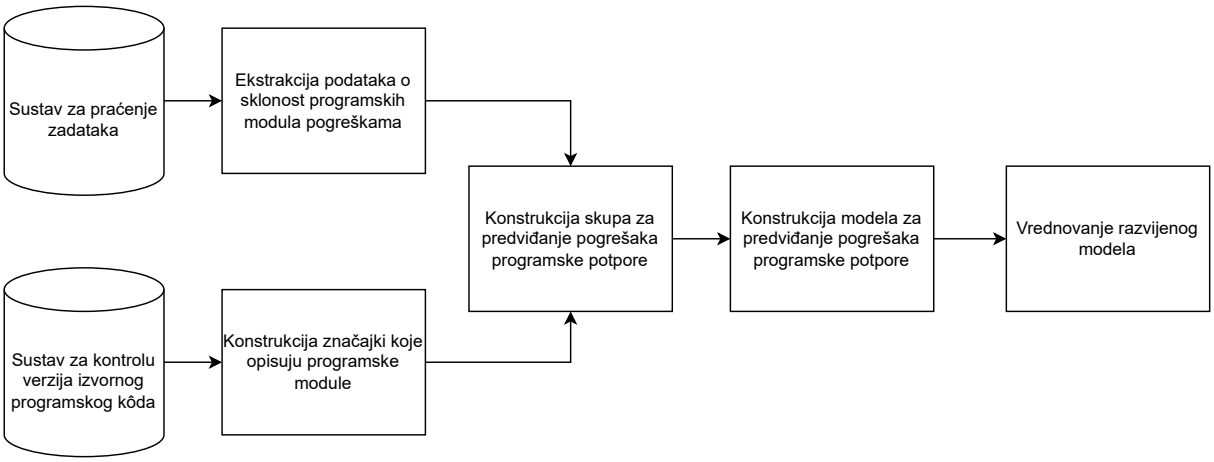
Konačno, u poglavlju 6 zaključuje se ova doktorska disertacija, navode njeni znanstveni doprinosi i njihova važnost.

Poglavlje 2

Predviđanje pogrešaka programske potpore

Znanstveno područje predviđanja pogrešaka programske potpore (engl. Software Defect Prediction skr. SDP) započeo je Akiyama 1971 godine objavom članka [40] u kojem je proučavao vezu između broja linija kôda (engl. Lines of Code, skr. LOC), kao mjere složenosti programskog kôda i sklonosti ne-ispravnog rada programske potpore. Pokazao je kako postoji pozitivna korelacija između broja linija kôda i broja programskih pogrešaka.

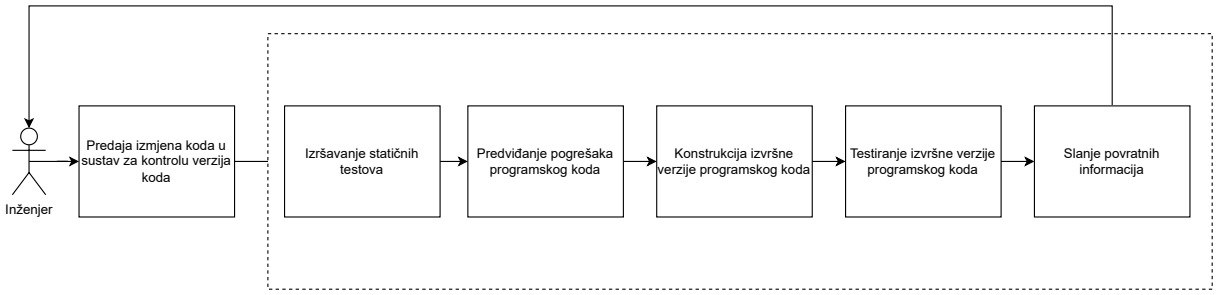
Predviđanje pogrešaka programske potpore tipično se sastoji od konstrukcije značajki koje opisuju modul programske potpore, prikupljanje povijesnih podataka o sklonosti pogreškama navedenih modula, obrade prikupljenih podataka, konstrukcije podatkovnog skupa za SDP, razvoja modela za SDP i njegovog vrednovanja. Slika 2.1 prikazuje općeniti pregled previđanja pogrešaka programske potpore.



Slika 2.1: Pregled predviđanja pogrešaka programske potpore

Razvijeni SDP model moguće je ugraditi u sustav *kontinuirane integracije* (engl. Continuous Integration) kako bi označio potencijalno pogrešne programske module čim se pogreška unese u njih. Alternativno, model se može izvršavati neovisno, kada inženjeri procjene da je

potrebno. Kod integracije u sustav kontinuirane integracije model se tipično ugrađuje nakon izvršavanja statičkih testova i prije konstrukcije izvršne verzije programskog kôda. Na ovaj način, osigurava se rano dobivanje povratne informacije od modela. Neovisno izvršavanje se pokreće prije alokacije resursa za osiguravanje kvalitete programske potpore. Slika 2.2 prikazuje mjesto ugradnje razvijenog modela u sustav kontinuirane integracije.



Slika 2.2: Ugradnja predviđanja pogrešaka programske potpore u sustav kontinuirane integracije

Od Akiyamova članka [40] područje je izuzetno aktivno i mnogi članci su objavljeni razmatrajući sve dijelove predviđanja pogrešaka programske potpore od prikupljanja podataka, konstrukcije značajki, odabira značajki, obrade značajki, izgradnje modela i konačno vrednovanja modela za predviđanje pogrešaka programske potpore.

U nastavku ovog poglavlja detaljno su opisani različiti aspekti područja predviđanja pogrešaka programske potpore. U poglavlju 2.1 opisani su podatkovni skupovi za SDP, tj. opisan je proces njihove izgradnje, opisani su njihovi česti nedostatci i navedeni su često korišteni, javno dostupni, podatkovni skupovi. U poglavlju 2.2 opisane su različite vrste značajki korištene za opis programskih modula nad kojima se vrši predviđanje pogrešaka programske potpore. U poglavlju 2.3 opisane su različite veličine programskih modula nad kojima je moguće vršiti predviđanje pogrešaka programske potpore. U poglavlju 2.4 opisana je podjela područja s obzirom na tip predviđanja. U poglavlju 2.5 opisana je podjela područja s obzirom na vrstu modela koji se koriste za predviđanje pogrešaka programske potpore. U poglavlju 2.6 prezentirane su mjere za vrednovanje modela korištene u provedenom istraživanju i, općenito, standardno korištene u ovom području. Konačno, u poglavlju 2.7 prezentirane su statističke metode korištene za statističku potvrdu rezultata korištene u provedenom istraživanju i, općenito, standardno korištene u ovom području.

2.1 Podatkovni skupovi

Podatkovni skupovi za razvoj modela za predviđanje programskih pogrešaka izgrađuju se sakupljanjem podataka iz sustava za praćenje zadataka (engl. Issue Tracking System, skr. ITS) i sustav za kontrolu verzija izvornog programskog kôda (engl. Version Control System, skr.

VCS). Svaka predaja promjene kôda (engl. Commit) u VCS sustavu se analizira kako bi se povezala sa zadatkom u ITS sustavu. Ako se pronađe veza sa zadatakom označenim kao *zahtjev za popravak pogreške* (engl. Bug Fix Request) tada se prethodno stanje kôda smatra pogrešnim, a stanje nakon predaje u sustav se smatra ispravnim. Alternativno, sva stanja ovog kôda se smatraju sklonim programskim pogreškama. Dodatno, može se tražiti predaja promjene kôda koja je unijela programsku pogrešku, tada se ta predaja promjena smatra onom koja unosi pogrešku u programsku potporu (engl. Bug Inducing Commit). Ovaj pristup koristi se pri izgradnji podatkovnih sustava za izučavanje predaja promjena koje uzrokuju pogreške u programskom kôdu. Veza između predaje u sustav za kontrolu verzija izvornog programskog kôda i zadatka se smatra pronađena ako se u poruci promjene pronađe broj koji odgovara broju zadatka i ključne riječi poput *bug* i/ili *fix* [41, 42, 43, 44].

U poglavlju 2.1.1 navedeni su nedostatci podatkovnih skupova za predviđanje pogrešaka programske potpore, a u poglavlju 2.1.2 navedeni su podatkovni skupovi često korišteni u području predviđanja pogrešaka programske potpore.

2.1.1 Nedostatci

Neki istraživači ukazali su na potencijalne probleme s podatkovnim skupovima za predviđanje pogrešaka programske potpore [45, 46, 47, 48].

Pri izgradnji podatkovnih skupova za predviđanje pogrešaka programske potpore podatci se prikupljaju iz više izvora. Prikupljaju se podatci za opis programske potpore tj. njenih modula iz sustava za upravljanje verzijama izvornog programskog kôd. Iz sustava za praćenje zadataka prikupljaju se podatci o obavljenim zadatcima koje je bilo potrebno izvršiti u sklopu razvoja i održavanja programske potpore. Temeljem prikupljenih podatka izgrađuje se podatkovni skup za predviđanje pogrešaka programske potpore. Pri izgradnji podatkovnog skupa potrebno je uklanjanje nepotrebnih atributa, popunjavanje praznih vrijednosti i po potrebi druge obrade podataka.

Proces izgradnje podatkovnih skupova za predviđanje pogrešaka programske potpore ima dvije kritične točke u kojima šum može biti unesen u izgrađeni podatkovni skup. Prvo, šum može biti unesen ako se ne može identificirati veza između predaje promjene kôda u sustavu za kontrolu verzija izvornog programskog kôda i zadatka u sustavu za praćenje zadataka. U ovom slučaju, programski kôd može biti tretiran kao ispravan, a zapravo je postojao zadatak koji je *zahtjev za popravkom pogreške* u navedenom programskom kôdu tj. navedeni programski kôd je neispravan (engl. Defective) ili sklon pogreškama (engl. Defect-Prone). Drugi slučaj unosa šuma u podatkovni skup za predviđanje pogrešaka programske potpore događa se kada je zadatak u sustavu za praćenje zadataka krivo označen [49].

Uz dvije navedene kritične točke šum u podatkovnom skupu za predviđanje pogrešaka programske potpore može nastati i kada prije izgradnje podatkovnog skupa pogreška u programskoj

potpori ne bude zabilježena. Tada se programski modul koji sadrži pogrešku smatra ispravnim, iako je neispravan, ali nije zabilježena manifestacija pogreške.

Zbog nedostataka u skupovima za predviđanje pogrešaka potrebni su postupci odabira značajki [50, 51, 52, 53, 54, 55], normalizacije [56, 57] i nošenja sa šumom u podacima [58, 59].

Istraživači su se posvetili problemu šuma u podatkovnim skupovima za predviđanje pogrešaka programske potpore. Na primjer, Bird et al. [60] su u svom istraživanju ustanovili da broj popravljenih pogrešaka u izvornom programskom kôdu ne odgovara broju zadataka označenih kao *zahtjevi za popravak pogreške*. Rezultat su dijelovi izvornog programskog kôda koji se smatraju ispravni ili neskloni pogreškama, iako to zapravo nisu. Kako navode Zhiqiang et al. [21] ovo je posljedica toga što se sakupljanje podataka za razvoj modela za predviđanje pogrešaka programske potpore zasniva na traženju ključnih riječi koje inženjeri često ne pišu u poruke predaje promjene programskog kôda u sustav za kontrolu verzije kôda [41, 42, 43, 44, 61, 62, 63].

Wu et al. [59] su predložili pristup kojeg su nazvali *ReLink* kako bi adresirali problem nedostajućih veza između zadataka i predaja promjene kôda u sustav kontrole verzije kôda. Ručno su analizirali poruke predaja promjena kôda u VCS i otkrili da veze imaju učestale pravilnosti koje su potom iskoristili za izgradnju algoritma za otkrivanje nedostajućih veza. Motivirani radom Wu et al. [59], Nguyen et al. [64] su razvili *MLink*. Radi se o višeslojnom pristupu koji u obzir uzima i karakteristike u porukama predaja promjene kôda i karakteristike izvornog programskog kôda te je u stanju pomoću tih karakteristika identificirati nedostajuće veze između predaja promjena kôda u sustavu za praćenje verzija kôda i zadataka u sustavu za praćenje zadataka.

Herzig et al. [25] su ručno analizirali više od 7000 zadataka prikupljenih iz sustava za praćenje zadataka od 5 različitih projekata slobodno dostupnog izvornog programskog kôda. Ustanovili su da je 33.8% svih zadataka označenih kao *zahtjev za popravak pogreške* krivo označeno i da je 39% programskih modula označenih kao skloni pogreškama ispravno i nikad nisu sadržavali pogrešku. Na pogrešno označavanje zadataka upozorili su i Antoniol et al. [24] koji su analizirali zadatke označene kao *zahtjev za popravak programske pogreške*. Našli su da ovi zadatci mogu zapravo biti *zahtjevi za održavanjem kôda*, *zahtjevi za ažuriranjem kôda*, pozivi na raspravu ili jednostavno *zahtjevi za pomoć pri korištenju programskog sučelja*. Svoje istraživanje proveli su na 3 repozitorija *Mozilla*, *Eclipse* i *JBoss* i pokazali da modeli poput stabla odluke, naivan Bayes i logistička regresija mogu predvidjeti oznake zadataka temeljem njihova opisa i naslova postižući F1 rezultat od 0.70. Utjecaj krivo označenih zadataka na rad modela za predviđanje pogrešaka programske potpore istraživali su Tantithamthavorn et al. [49] te su u svom istraživanju koristili 3931 ručno označenih zadataka iz *Apache Jackrabbit* i *Lucene* sustava. Njihovi rezultati pokazuju da krivo označeni zadatci nisu potpuno nasumični te da ne-nasumični šum ne utječe na preciznost modela, ali može smanjiti odziv modela za 32% - 44%.

Istraživači Kim et al. [58] su proučavali utjecaj šuma u podatkovnim skupovima za predviđanje pogrešaka programske potpore na rad modela za predviđanje pogrešaka programske potpore. Ustanovili su da šum u podatkovnim skupovima za razvoj modela za predviđanje pogrešaka programske potpore značajno utječe na njihov rad ako podatkovni skup sadrži 20% - 35% *lažnih pozitiva* (engl. False Positive, skr. FP) i *lažnih negativa* (engl. False Negative, skr. FN). Predložili su algoritam za pronalaženje i uklanjanje krivo označenih primjera u podatkovnim skupovima. Uklanjanjem šuma iz podatkovnih skupova za predviđanje pogrešaka programske potpore bavili su se i Khan et al. [65] istražujući utjecaj 9 različitih načina za uklanjanje šuma iz podatkovnih skupova za razvoj modela za predviđanje pogrešaka programske potpore. Umjesto nasumično generiranog šuma koristili su podatkovne skupove bez šuma i unosili šum koristeći heurističke pristupe. Njihovi rezultati pokazuju da metode za uklanjanje šuma nisu uspješne i teško unaprjeđuju rad modela.

Uz šum u podatcima, čest problem je neuravnoteženosti razreda (engl. Class Imbalance) [28, 29, 30, 31, 32, 33, 34, 35]. Naime, podatkovni skupovi za predviđanje pogrešaka programske potpore često sadrže daleko manje primjera koji su označeni kao skloni pogreškama nego primjera koji su označeni kao neskloni pogreškama [21]. Postoji nekoliko razloga zašto može doći do ove pojave: rijetkost pogrešaka programske potpore, posljedice pogrešaka programske potpore i težina prikupljanja podataka o pogreškama programske potpore. Pogreške programske potpore su relativno rijetke pojave s obzirom na ukupan broj programskih modula što može otežati prikupljanje adekvatnog broja primjera programskih modula sklonih pogreškama. Posljedice pogrešaka programske potpore mogu biti katastrofalne pa organizacije često ulažu velike napore u njihovu prevenciju. S obzirom na to da se nastanak pogrešaka programske potpore aktivno nastoji spriječiti ovo umanjuje broj primjera programskih modula sklonih pogreškama. Konačno, postupak prikupljanja podataka za predviđanje pogrešaka programske potpore je zahtjevan i oslanja se na njihovu sustavnu evidenciju u sustavima za praćenje zadataka. Ako se sustavna evidencija pogrešaka programske potpore ne provodi izgrađeni podatkovni skupovi sadržavat će umjetno nizak broj primjera programskih modula sklonih pogreškama.

Istraživači su mnogo pažnje posvetili problemu neuravnoteženosti razreda pa tako Thota et al. [9] navode kako je ozbiljan izazov u području predviđanja pogrešaka programske potpore činjenica da je primjera programskih pogrešaka značajno manje nego primjera bez programskih pogrešaka.

Kako bi smanjili utjecaj neuravnoteženosti razreda Jing et al. [66] su predložili *unificirani radni okvir* (engl. Unified Framework) za predviđanje pogrešaka programske potpore. Drugi istraživači su razmatrali metode pod-uzorkovanja i ponovnog uzorkovanja kako bi se nosili s problemom neuravnoteženosti razreda. Tako su Tan et al. [30] predložili korištenje ponovnog uzorkovanja i klasifikacijskih metoda koje omogućuju *kontinuirano učenje* (engl. Online Learning), a Goyal [35] je predložio novu metodu pod-uzorkovanja podataka nazvanu *pod-*

uzorkovanje zasnovano na susjedstvu (engl. Neighbourhood based Under-Sampling). Malhotra et al. [67] i Bennin et al. [68] su istraživali različite metode uzorkovanja s ciljem uravnoteženja podatkovnog skupa za predviđanje pogrešaka programske potpore. Obje skupine autora došle su do zaključka da je ponovno uzorkovanje bolje od drugih metoda, a pogotovo bolje od metode pod-uzorkovanja.

C. Seiffert et al. [69] su analizirali problem neuravnoteženosti razreda u podatkovnim skupovima i utjecaj šuma na rad razvijenih modela. Njihovi rezultati potvrđuju rezultate od Kim et al.[58]. Problemima šuma i neuravnoteženosti razreda bavili su se i Pandey et al. [70] koji su proveli istraživanje metoda za nošenje sa šumom i istraživanje problema neuravnoteženosti razreda u podatkovnim skupovima za razvoj modela za predviđanje pogrešaka programske potpore. Njihovi rezultati ukazuju da je dovoljno da podatkovni skup sadrži 10% krivo označenih podataka da bi *stopa dobro označenih pozitivnih primjera* (engl. True Positive Rate, skr. TPR) i *stopa dobro označenih negativnih primjera* (engl. True Negative Rate, skr. TNR) bile smanjene za 20% - 30%, a vrijednost *krivulje rada prijemnika* (engl. Receiver Operating Curve, skr. ROC) smanjena za 40% - 50%.

2.1.2 Javno dostupni podatkovni skupovi za predviđanje pogrešaka programske potpore

Kako bi se rezultati različitih istraživanja mogli lakše uspoređivati istraživači su podatkovne skupove za predviđanje pogrešaka programske potpore napravili javno dostupnim.

Korištenje javno dostupnih i standardiziranih podatkovnih skupova ima niz prednosti pri razvoju modela za predviđanje pogrešaka programske potpore. Prije svega, omogućuje istraživačima pristup većim i raznolikim podatkovnim skupovima nego što bi im inače bili dostupni u internim sustavima. To povećava šanse za pronalaženje uzoraka i trendova u podacima, što dovodi do boljih modela za predviđanja pogrešaka programske potpore. Nadalje, istraživači ne moraju trošiti vrijeme na označavanje podataka, već se mogu usredotočiti na razvoj modela. Također, korištenje javnih skupova podataka omogućuje usporedbu različitih modela i tehnika predviđanja između različitih istraživača. To dovodi do bržeg napretka u području predviđanja pogrešaka programske potpore i može pomoći u identificiranju najboljih praksi. Konačno, korištenje javnih skupova podataka poboljšava transparentnost i pouzdanost modela, jer drugi istraživači mogu pregledati i ponovno analizirati rezultate.

Ukratko, korištenje javnih standardiziranih skupova podataka donosi brojne koristi u razvoju modela za predviđanje pogrešaka programske potpore, uključujući veću raznolikost podataka, transparentnost i usporedivost modela.

Najpoznatiji skupovi podataka za predviđanje pogrešaka programske potpore su *NASA* [45, 56, 71], *PROMISE* [72], *Bug Prediction Dataset*, *Eclipse Bug Data Repository* [41], *ReLink*

[59], Eclipse JDT/PDE [73] i *A Unified Bug Dataset for Java* [74, 75]. Ferenc et al. [75] unificirali su mnoge često korištene skupove podataka.

2.2 Oblikovanje i vrste značajki

Rana znanstvena istraživanja u području predviđanja pogrešaka programske potpore fokusirala su se na definiranje značajki koje se mogu koristiti za predviđanje pogrešaka programske potpore. Značajke su karakteristike koje se koriste za opisivanje podatkovnih modula za koje se vrši predviđanje pogrešaka. Značajke su obično numeričke, ali mogu biti i kategoričke vrijednosti. S vremenom su definirane mnoge različite značajke. Značajke je općenito moguće podijeliti na *značajke složenosti programskog kôda* (engl. Code Complexity Features) i *procesne značajke* (engl. Process Features). Značajke složenosti programskog kôda nastoje opisati složenost izvornog programskog kôda, a procesne značajke nastoje opisati proces njegova razvoja.

U poglavlju 2.2.1 navedene su značajke složenosti programskog kôda koje se koriste za predviđanje pogrešaka programske potpore, a u poglavlju 2.2.2 navedene su procesne značajke koje se koriste za predviđanje pogrešaka programske potpore.

2.2.1 Značajke složenosti kôda

Značajke složenosti izvornog programskog kôda (engl. Code Complexity Features) nastoje sažeti informacije o njegovoj složenosti i sadržaju kako bi se mogle iskoristiti za predviđanje pogrešaka programske potpore.

McCabe [76] je predložio značajke zasnovane na ciklometrijskoj i strukturalnoj složenosti izvornog programskog kôda. Ciklometrijska složenost kvantificira broj linearno neovisnih izvršavanja izvornog programskog kôda. Radi se o složenosti kontrolne strukture. Općenito, složenost kontrolne strukture mjeri koliko je složena kontrola toka u programu, uključujući petlje, grananja i rekurzivne funkcije. Složenije kontrolne strukture mogu biti teže razumjeti i podložnije pogreškama.

Halstead [77] je predložio značajke za opis složenosti kôda poput *broj jedinstvenih operatora*, *broj jedinstvenih operanada*, *ukupan broj operatora* i *ukupan broj operanada*. Izvorni programski kôd s većim brojem operacija složeniji je što smanjuje njegovu čitljivost i čini ga podložnim pogreškama

Pojavom objektno-orijentiranih programskih jezika pojavila se potreba za značajkama koje opisuju programske module tj. programsku potporu napisanu u tim jezicima. *Objektno-orijentirani programski jezici* (engl. Object-oriented Programming Languages) temelje se na objektno-orijentiranoj paradigmi koja se temelji na objektima, koji su instance različitih razreda, čime nastoji olakšati organizaciju i upravljanje kompleksnim programskim sustavima. U objektno-

orijentiranom programiranju programska potpora se sastoji od objekata koji komuniciraju jedni s drugima putem metoda, poruka i svojstava. Svaki objekt se sastoji od svojstva koja opisuju njegovo stanje i metode koje modeliraju njegovo ponašanje. Razredi opisuju zajedničke značajke i ponašanja objekata iste vrste. Chidamber i Kemerer [78] su predložili inicijalne objektno orijentirane značajke nazvane *CK značajke*. Primjeri ovih značajki su broj razreda s kojima je objekt povezan, udio metoda koje mijenjaju svojstva objekta i broj direktnih pod-razreda pojedinog razreda. Uz CK značajke, Harrison et al. [79] predložili su *MOOD značajke* zasnovane na polimorfnim svojstvima objektno orijentiranog kôda. Mnogi drugi istraživači predložili su dodatne značajke za opis objektno-orijentiranog kôda [80, 81, 82, 83, 84].

McCabe značajke, Halstead značajke, CK značajke, MOOD značajke i njima slične značajke učestalo se nazivaju *klasične značajke* ili *tradicionalne značajke*.

Novija istraživanja definirala su dinamičke značajke koje nastoje opisati povezivanje i složenost programskih objekata za vrijeme izvršavanja izvornog programskog kôda [85, 86, 87, 88], a zadnja istraživanja značajki za predviđanje pogrešaka programske potpore fokusirana su na razvoj semantičkih značajki primjenom modela dubokog učenja na izvorni programski kôd [89, 90, 91, 92]. Semantičke značajke za predviđanje pogrešaka programske potpore temelje se na semantici, tj. značenju kôda, i njegovoj strukturi, a ne samo na njegovoj sintaksi. Ovim značajkama nastoji se opisati stil pisanja kôda, strukturu kôda, način njegova korištenja, kontekst u kojem se nalazi i cilj njegovog izvršavanja.

Za ovu disertaciju bitan je GraphCodeBERT model [92] koji je zasnovan na BERT arhitekturi i može se iskoristiti za generiranje semantičkih reprezentacija izvornog programskog kôda generiranih temeljem strukture kôda (leksičkih tokena i sintaksnog stabla) te njegovih podatkovnih struktura. Radi se o višejezičnom modelu s potporom za iduće programske jezike: Python, Java, JavaScript, PHP, Ruby i Go.

2.2.2 Procesne značajke

Uz značajke složenosti izvornog programskog kôda istraživači su definirali i procesne značajke (engl. Process Features) koje opisuju proces razvoja programske potpore, a ne sam izvorni programski kôd. Ove značajke temelje se na ideji da proces razvoja programske potpore utječe na pojavu pogrešaka u programskoj potpori. Na primjer, razmatranjem veličine tima koji radi na razvoju programske potpore moglo bi se doći do zaključka da veći timovi mogu imati više problema s komunikacijom i koordinacijom, pa razvijena programska potpora posljedično sadrži više pogrešaka. Moglo bi se razmatrati i iskustvo inženjera koji razvija programsku potporu što bi moglo rezultirati zaključkom da iskusni inženjeri rade manje pogrešaka u programskoj potpori.

Zbog razlika između različitih inženjera kao što su stil programiranja, učestalost predaja promjena izvornog programskog kôda i razina iskustva, Jiang et al. [93] su predložili razvoj

personaliziranih modela za predviđanje pogrešaka programske potpore tvrdeći da kombiniranje pogrešaka različitih inženjera dovodi do narušavanja rada modela za predviđanje pogrešaka programske potpore. Ovim istraživanjem pokazali su da nije bitan sam kôd već i način njegova nastanka.

Pinzger et al. [94] su pokazali da je broj inženjera koji su radili na programskom kôdu dobra značajka za predviđanje pogrešaka programske potpore i da su centralni moduli programskog kôda skloniji pogreškama od perifernih programskih modula. Slično, Bird et al. [95] su analizirali značajke koje opisuju vlasništvo izvornog programskog kôda stavljajući naglasak na inženjera koji je pisao kôd umjesto samog programskog kôda. Pokazali su da programski moduli na kojima je velik broj inženjera radio male izmjene češće sadrže pogreške. Nadalje, Matsumoto et al. [96] su predložili značajke zasnovane na inženjerima koji razvijaju programsku potporu. Predložene značajke uključuju broj predaja promjena izvornog programskog kôda svakog od inženjera i broj različitih inženjera koji su radili na programskom modulu. Isto tako su pokazali su da što više inženjera radi na programskom modulu to je on skloniji pogreškama. Konačno, Madeyski et al. [97] su razmatrali koje procesne značajke najviše doprinose predviđanju pogrešaka programske potpore i ustanovili da je broj inženjera jedna od najefektnijih značajki.

Dinamika i sadržaj promjena kôda isto se smatraju procesnim značajkama. Istraživači Nachiappan et al. [98] su predložili značajke za opis promjene kôda poput broja dodanih linija kôda, broja uklonjenih linija kôda i broja promijenjenih linija kôda, a Nagappan i Ball [99] su predložili značajke zasnovane na učestalosti promjene programskog kôda. Nadalje, Šikić et al. [100] su predložili agregirane značajke promjene izvornog programskog kôda. Agregirane značajke ne opisuju samo jednu promjenu već ukupnu promjenu programskog kôda za vrijeme njegova dosadašnjeg razvoja. Korištenje značajki koje opisuju promjene programskog kôda istražili su Moser et al. [101] i pokazali da takove značajke mogu biti efikasnije u predviđanju pogrešaka programske potpore nego značajke koje opisuju njegovu složenost. Drugi istraživači su dodatno proširili skup značajki za opis promjene kôda [102, 103].

Uz značajke koje opisuju inženjere koji rade na razvoju programske potpore i značajke koje opisuju promjene izvornog programskog kôda istraživači su nastojali razviti i značajke koje opisuju timove i interakcije timova koji razvijaju programsku potporu. Tako su Meneely et al. [104] predložili značajke koje opisuju društvene mreže inženjera koji rade na razvoju programske potpore, a Bacchelli et al. [105] značajke koje opisuju popularnost različitih dijelova izvornog programskog kôda analizirajući e-mail arhive i mjereći o kojim komponentama se najviše razgovaralo. Pokazali su kako inženjeri učestalo razgovaraju o kôdu sklonom pogreškama.

Istraživači su predložili još mnoge procesne značajke [106, 107, 108, 109, 110].

2.3 Veličina programskog modula nad kojim se vrši predviđanje

Predviđanje pogrešaka programske potpore moguće je raditi za različite veličine programskih modula. Modul programske potpore predstavlja jedinicu izvornog programskog kôda. Predviđanje na razini manjih modula olakšava inženjerima postupak pronalaženja programskih pogrešaka. Predviđanje na većim razinama je bolje prilagođeno za dodjelu resursa za osiguravanje kvalitete programske potpore [9]. Dodatno, predviđanje na manjim razinama zahtjeva više vremena jer je ista programska potpora opisana s većim brojem primjera.

Predviđanje je moguće provoditi na razini *komponente* [111, 112, 113, 114, 115], *datoteke* [101, 104, 116, 117, 118, 119, 120, 121], *programskog razreda* [122, 123], *programske metode* [116, 124, 125, 126] ili na *razini promjene* [58, 127, 128, 129, 130, 131]. Izbor granularnosti ovisi o potrebama projekta i dostupnosti podataka.

2.4 Podjela područja ovisno o tipu predviđanja

Područje predviđanja pogrešaka programske potpore moguće je podijeliti u dva pod-područja ovisno o podacima koji se koriste za razvoj i vrednovanje modela. U prvim istraživanjima istraživači bi prikupili podatke o projektu, potom bi istrenirali model za predviđanje pogrešaka programske potpore i taj model primijenili na projekt, čiji su podatci prikupljeni, kako bi predvidjeli pogreške programske potpore u njemu. Kasnija istraživanja fokusirala su se na mogućnost treniranja modela na podatcima jednog projekta i njegovu primjenu na drugom projektu. Takav pristup omogućio bi predviđanje pogrešaka programske potpore kod projekata bez povijesnih podataka. U slučaju primjene modela na projekt čiji su podatci korišteni za razvoj modela radi se o pristupu predviđanja pogrešaka programske potpore unutar projekta, a u slučaju primjene modela na projekt čiji podatci nisu korišteni za njegov razvoj radi se o predviđanju pogrešaka programske potpore između projekata.

U poglavlju 2.4.1 detaljnije je opisano predviđanje pogrešaka programske potpore unutar istog projekta, a u poglavlju 2.4.2 detaljnije je opisano predviđanje pogrešaka programske potpore između projekata.

2.4.1 Predviđanje pogrešaka unutar istog projekta

Prvo pod-područje je *predviđanje pogrešaka programske potpore unutar istog projekta* (engl. Within-Project Software Defect Prediction, skr. WP-SDP) [132]. U sklopu ovog područja model za predviđanje pogrešaka trenira se nad podacima prikupljenim od projekta i potom se dobiveni model koristi za predviđanje pogrešaka unutar istog projekta, ali na prethodno ne

viđenim dijelovima projekta [111, 119, 133, 134, 135, 136, 137, 138, 139, 140]. Ovakav pristup se najčešće koristi u kontekstu velikih projekata, gdje postoji dovoljno podataka za treniranje modela. Prednost ovog pristupa leži u činjenicu što se mogu uzeti u obzir specifičnosti projekta, što može dovesti do boljeg rada modela za predviđanje pogrešaka.

WP-SDP moguće je dodatno podijeliti na predviđanje pogrešaka programske potpore unutar iste verzije projekta (engl. Inner-Version Within-Project Software Defect Prediction, skr. IV-WP-SDP) i između različitih verzija istog projekta (engl. Cross-Version Within-Project Software Defect Prediction, skr. CV-WP-SDP) [132].

WP-SDP ima problem hladnog početka (engl. Cold Start) što znači da je modele nemoguće razviti za potpuno nove projekte bez povijesnih podataka ili male projekte bez dovoljne količine podataka. Ipak, istraživači i dalje rade na razvoju modela za WP-SDP. Neka od posljednjih istraživanja u ovom području navedena su u nastavku.

Bhutamapuram et al. [136] su predložili učenje raznolikih ansambl modela tehnikom bootstrap agregacije. Malohtra et al. [137] su predložili korištenje duboke konvolucijske mreže za predviđanje pogrešaka programske potpore. Palma et al. [138] su predložili WP-SDP za predviđanje pogrešaka programske potpore koja opisuje računalnu infrastrukturu. Liang et al. [139] su predložili WP-SDP metodu aktivnog učenja za predviđanje pogrešaka programske potpore uzrokovanih starenjem programske potpore, a Šikić et al. [140] su predložili WP-SDP metodu zasnovanu na neuronskim mrežama za grafove.

2.4.2 Predviđanje pogrešaka između projekata

Motivirani problemom hladnog početka istraživači su se okrenuli izučavanju mogućnosti razvoja modela za predviđanje pogrešaka programske potpore između različitih projekata. Ovi pristupi čine drugo pod-područje u predviđanju pogrešaka programske potpore i naziva se *predviđanje pogrešaka programske potpore između projekata* (engl. Cross-Project Software Defect Prediction, skr. CP-SDP) [141]. CP-SDP modeli za predviđanje pogrešaka razvijaju se nad podacima prikupljenim na jednom projektu i potom primjenjuju na druge projekte [57, 141, 142, 143, 144, 145, 146, 147, 148]. Projekt čiji podatci se koriste za razvoj modela zove se *izvorni projekt* (engl. Source Project), a projekt na koji se model primjenjuje naziva se *ciljni projekt* (engl. Target Project).

Zimmermann et al. [141] su pokazali da je CP-SDP izazovan zadatak. Točnije, pokazali su da nije lako pronaći na kojem projektu bi model trebao biti treniran kako bi dobro predviđao pogreške na drugom projektu. U njihovom istraživanju od 622 predviđanja između projekata samo 3.4 % postiglo je obećavajuće rezultate.

Osnovna pretpostavka strojnog učenja je da su podatci za treniranje i podatci za testiranje uzorkovani iz iste podatkovne distribucije. Činjenica da modeli trenirani na jednom projektu često ne rade dobro na drugim projektima ukazuje da je ova pretpostavka narušena tj. da se po-

datkovne distribucije projekata razlikuju do te mjere da razlika onemogućuju razvoj i primjenu modela. Iz navedenog razlog, modeli za predviđanje između projekata imaju značajno lošije rezultate nego modeli za predviđanje unutar istog projekta [142, 146].

Istraživači su probali koristiti stablo odluke kako bi predvidjeli mogućnost predviđanja između projekata [141, 142]. Ovakvi pristupi mogu se nazvati meta učenjem jer se razvija model čiji je cilj usmjeriti učenje drugog modela koji će biti specijaliziran za rješavanje problema od interesa.

Istraživači su nastojali adresirati problem mogućnosti predviđanja između projekata kroz nekoliko metoda: *ažuriranjem značajki* (engl. Metric Compensation) [149, 150], *identificiranjem najbližih susjednih projekata* (engl. Nearest Neighbor Filtering) [151], *meta učenjem* (engl. Meta Learning) [152], *prijenosom naivnog Bayesa* (engl. Transfer Naive Bayes), *korištenjem ansambla modela* (engl. Ensemble Model) i *analizom prijenosnih komponenti* (engl. Transfer Component Analysis) [57, 145, 153, 154].

Ažuriranje značajki nastoji normalizirati značajke *ciljanoga projekta* koristeći značajke *izvornog projekta*. Cilj normalizacije je smanjenje razlika u distribucijama izvornog i ciljnog projekta [149].

Identificiranje najbližih susjednih projekata nastoji pronaći izvorne projekte čija je podatkovna distribucija najsličnija podatkovnoj distribuciji ciljanog projekta. Tada se za razvoj modela koristi samo *N* najbližih izvornih projekata [151].

Meta učenje nastoji predvidjeti koji metodu koristiti kako bi rezultati predviđanja između projekata bili što bolji [152]. Općenitije govoreći, meta učenjem se mogu smatrati svi pristupi koji razvijaju modele čija je funkcija usmjeriti učenje drugih modela.

Prijenos naivnog Bayesa preračunava vjerojatnosti modela naivan Bayes temeljem sličnosti izvornog i ciljnog projekta. Navedena metoda ima inherentno ograničenje da je primjenjiva samo pri korištenju tog model [143].

Korištenje ansambla modela trenira više modela za predviđanje između projekata i potom ih povezuje u jedan konačni model. Na primjer, Xia et al. [155] su predložili model kojeg su nazvali HYDRA. Model kombinira genetski algoritam i učenje u ansamblu minimizirajući pogrešku na malom označenom skupu ciljnog projekta pri konstrukciji modela čija je kompozicija. Još jedan primjer korištenja ansambla modela je istraživanje Panichella et al. [148] koji su predložili CODEP model koji korištenjem klasifikacijskog modela logističke regresije ili naivnog Bayesa kombinira 6 pod-modela pri predviđanju pogrešaka između projekata.

Analiza prijenosnih komponenti nastoji transformirati značajke izvornog i ciljnog projekta u latentni prostor značajki na takav način da njihove podatkovne distribucije postanu što sličnije [145]. Ova metoda daje dobre rezultate ali je osjetljiva na metodu normalizacije značajki koja se koristi kao dio njihove transformacije [57]. Zato su Nam et al. [57] predložili algoritam TCA+ koji je napredna verzija analize prijenosnih komponenti. TCA+ ugrađuje pravila za donošenje

odluke o metodi transformacije značajki. TCA+ se smatra jednim od trenutačno najboljih pristupa za predviđanje između projekata. Alternativno, Jin et al. [156] su predložili metodu koja isto tako ažurira značajke izvornog i ciljnog projekta kako bi se povećala mogućnost uspješnog predviđanja pogrešaka između projekta. Metoda koristi KTSVMs (engl. Kernel Twin Support Vector Machines) [157] optimiran korištenjem roja kvantnih čestica (engl. Quantum Particle Swarm Optimization) [158] za predviđanje pogrešaka između projekata.

Posljednja istraživanja nastoje omogućiti korištenje projekata opisanih različitim značajkama kako bi se ostvarilo učenje između projekata. Ovo bi značajno povećalo količinu podataka na raspolaganju pri izgradnji modela. Ovi pristupi nazivaju se *heterogenim predviđanjem pogrešaka programske potpore* [159, 160].

S obzirom na to da je i kod *predviđanja unutar istog projekta* i kod *predviđanja između projekata* potrebno dosta vremena dok inženjeri dobiju povratnu informaciju o kvaliteti izvornog programskog kôda istraživači su se posvetili izučavanju *promjena koje uzrokuju pogreške u programskom kôdu* (engl. Just-In-Time Software Defect Prediction, skr. JIT-SDP) [161, 162, 163, 164, 165, 166]. Kod klasičnih WP-SDP i CP-SDP pristupa modeli za predviđanje pogrešaka izvršavaju se prije dodjele resursa za osiguravanje kvalitete programske potpore. U tom trenutku inženjerima može biti problem prisjetiti se promjena koje su unijeli u sustav. Zbog toga JIT-SDP modeli izvršavaju se čim inženjeri unose promjene u sustav i pružaju im neposredne povratne informacije. Ovakav pristup potiče osiguravanje kvalitete programske potpore odmah u vrijeme razvoja.

2.5 Podjela područja ovisno o modelima korištenim za predviđanje

Područje predviđanja pogrešaka programske potpore moguće je podijeliti ovisno o tipu modela koji se koristi za predviđanje. Većina modela zasnovani su na tehnikama strojnog učenja [21].

Modeli mogu biti zasnovani na binarnoj klasifikaciji, tada model klasificira sadrži li programski modul pogrešku ili ne, odnosno klasificira dali je programski modul sklon pogreškama ili nije. Postoji mnogo modela koji se mogu koristiti za binarne klasifikaciju poput logističke regresije, stablo odluke, naivan Bayes, k najbližih susjeda i mnogi drugi. Odabir odgovarajućeg modela ovisi o dostupnom podatkovnom skupu. Modeli mogu biti zasnovani i na regresiji, u tom slučaju programski modul predviđa broj pogrešaka unutar programske potpore [167, 168]. I modeli klasifikacije i modeli regresije spadaju u nadzirane modele. Uz nadzirane modele postoje i polu-nadzirani i ne-nadzirani modeli.

U sklopu ove disertacije razvijen je model koji predviđanje pogrešaka programske potpore tretira kao problem otkrivanja anomalija. Pri radu na disertaciji nađen je samo jedan rad koji je ovom problemu pristupio na ovakav način. Kamrun et al. [169] prvo provode odabir značajki u

skladu sa [56], a potom oblikuju ispravne primjere programskih modula kao univarijantne [170] ili multivarijantni [171] Gaussove distribucije. Primjeri iz skupa za testiranje se potom klasificiraju kao skloni pogreškama ako su predaleko od određene distribucije. Članak [37] objavljen u sklopu ove disertacije inspirirao je druge istraživače [172, 173, 174, 175]. Tako su Zhang et al. [172] iskoristili BiGAN (engl. Bidirectional Generative Adversarial Network) za otkrivanje pogrešaka programske potpore tretirajući ih kao anomalije. Moussa et al. [173] su iskoristili *jedno razredni potporni stroj* (engl. One-Class Support Vector Machine, skr. One-Class SVM) za predviđanje pogrešaka programske potpore zasnovano na ideji otkrivanja anomalija. Lomio et al. [174] su primijenili pristup tretiranja problema otkrivanja pogrešaka programske potpore kao problema otkrivanja anomalija na JIT predviđanje pogrešaka programske potpore, a Akimova et al. [175] su konstruirali semantički zasnovanu metriku anomaličnosti izvornog programskog kôda.

Ovisno o dostupnosti i korištenju ciljnog razreda prilikom razvoja modela, modele je općenito moguće podijeliti na *nadzirane* (engl. Supervised) [93, 113, 129, 140, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188], *polu-nadzirane* (engl. Semi-Supervised) [189, 190], *ne-nadzirane* (engl. Unsupervised) [163, 191, 192, 193, 194] i *specijalizirane* [119].

Specijalizirani modeli specifični su za područje predviđanja grešaka programske potpore. Primjer takvog modela je BugCache. Kim et al. [119] su predložili BugCache algoritam. Algoritam održava liste informacija o lokacijama gdje su prethodno zabilježene pogreške programske potpore te pomoću njih određuje programske module koji su skloniji pogreškama.

U poglavlju 2.5.1 izloženo je više informacija o nadziranim modelima za predviđanje pogrešaka programske potpore. U poglavlju 2.5.2 izloženo je više informacija o polu-nadziranim modelima za predviđanje pogrešaka programske potpore. Konačno, u poglavlju 2.5.3 izloženo je više informacija o ne-nadziranim modelima za predviđanje pogrešaka programske potpore.

2.5.1 Nadzirani modeli

Nadzirano stojno učenje, tj. nadzirani modeli koriste označene podatke kako bi predvidjeli oznake do sada neviđenih primjera. Može se smatrati da nadzirani modeli modeliraju funkciju koja ulazne podatke mapira u oznake primjera. Velika većina istraživanja u području predviđanja pogrešaka programske potpore koristi *nadzirane* modele [9, 93, 113, 129, 140, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 195].

S obzirom na velik broj takvih istraživanja van je dosega ove disertacije da navodi sva istraživanja koja koriste nadzirane modele. Umjesto toga čitatelji se upućuju na pregledne članke [8, 9, 22, 134, 145, 176, 178, 179, 196, 197] te je navedeno nekoliko najvažnijih (sortirano po broju citata na sustavu Google Scholar) nedavnih istraživanja. Pa tako, Alsawalqah et al. [195] su predložili metodu koja trenira više ansambl modela, odabire model s najboljim rezultatima vrednovanja te ga potom ponovno trenira na podatkovnom skupu koji je balansiran koristeći

SMOTE (engl. Synthetic Minority Over-sampling Technique, skr. SMOTE) [198] metodu. Potom, Wang et al. [184] su predložili GH-LSTM (engl. Gated Hierarchical Long Short-Term Memory, skr. GH-LSTM) model koji je u stanju iz apstraktnoga sintaksnog stabla izvornog programskog kôda konstruirati semantičke i klasične značajke te ih primijeniti pri klasifikaciji programskih modula. Potom, Wang et al. [185] su predložili LASSO–SVM koji kombinira odabir i kompresiju minimalnih apsolutnih vrijednosti značajki te stroj potpornih vektora. Potom, Goyal [186] je predložio novu metodu za filtriranje nazvanu FILTER koja je prilagođena za kasnije efikasno korištenje stroja potpornih vektora. Potom, Pan et al. [187] su predložili primjernu CodeBERT modela [199] za predviđanje pogrešaka programske potpore. Konačno, Khurma et al. [188] su predložili korištenje otočne binarne optimizacije plamenih moljaca (engl. Island Binary Moth Flame Optimization) popraćene strojem potpornih vektora. Uz navedene, predloženo je još mnogo nadziranih modela za predviđanje pogrešaka programske potpore.

2.5.2 Polu-nadzirani modeli

Polu-nadzirani modeli su modeli koji pri svom razvoju, za razliku od nadziranih modela koji koriste isključivo označene primjere, koriste malu količinu označenih primjera i veliku količinu ne označenih primjera. U polu-nadzirani modeli nastoje iskoristiti informacije iz neoznačenih primjera kako bi se poboljšali svoj rad. Idealni su za korištenje u situacijama kada je teško dobiti veliku količinu označenih primjera, ali je dostupan veliki broj neoznačenih podataka. Predviđanje pogrešaka programske potpore upravo je takvo, jer je lako izgraditi značajke za velike količine izvornog programskog kôda, ali nije lako izgraditi potpune podatkovne skupove. Stoga, istraživači razmatraju mogućnost njihova korištenja [189, 190, 200, 201, 202] u predviđanju pogrešaka programske potpore s obzirom na to da bi omogućili korištenje informacija o programskim modulima za koje nije određena sklonost programskih pogreškama.

Primjeri istraživanja koja su koristila polu-nadzirane modele navedeni su u nastavku. Lu et al. [189] su koristili redukciju dimenzionalnosti značajki kojima su opisani programski moduli i polu-nadzirano učenje te dobili rezultate sumjerljive s onima nadziranih modela. Wu et al. [190] su predložili korištenje *učenja zasnovanog na mapama* (engl. Dictionary Learning) pri provođenju predviđanja unutar i između projekata. Wang et al. [200] su predstavili polu-nadzirani ansambl model kojeg nazivaju NSSB (engl. Non-negative Sparse-Based SemiBoost). Koristeći izgradnju *ne negativne matrice sličnosti* programskih modula omogućuju njihovo grupiranje i time korištenje pri učenju ansambl algoritma. Zhang et al. [201] su predstavili NSGLP (engl. Non-negative Sparse Graph-based Label Propagation). Koristeći strategiju uzorkovanja zasnovanu na Laplace metrici uzorkuju primjere programskih modula koji nisu skloni pogreškama i konstruiraju uravnotežen podatkovni skup. Zatim izračunaju ne negativne težine veza između programskih modula, koje koriste kao indikatore za njihovo grupiranje. Konačno, koriste propagacijski algoritam kako bi propagirali oznake programskih modula kroz graf dobiven njihovim

grupiranjem. Konačno, Meng et al. [202] su predstavili polu-nadzirani model koji kombinira normalizaciju značajki, ponovno uzorkovanje primjera i Tri algoritam (engl. Tri Algorithm) te ga vrednovali na NASA podatkovnom skupu.

2.5.3 Ne-nadzirani modeli

Ne-nadzirani modeli koriste isključivo neoznačene primjere pri svojoj izgradnji. Ovi modeli nastoje otkriti uzorke i strukturu u podatcima. Ne-nadzirani modeli idealni su za korištenje u situacijama kada je teško dobiti označene primjere, ali je dostupan veliki broj neoznačenih podataka i postoji osnovana pretpostavka da su podatci strukturirani. Istraživači su ispitali korištenje *ne-nadziranih* modela za potrebe predviđanja pogrešaka programske potpore [163, 192, 193, 194]. Prednost ne-nadziranih modela leži u činjenici da nije potrebno prikupiti podatke o tome sadrže li programski moduli pogreške ili ne, što značajno pojednostavljuje proces razvoja modela za otkrivanje pogrešaka programske potpore.

Primjeri istraživanja koja su koristila ne-nadzirane modele navedeni su u nastavku. Yang et al. [163] su pokazali kako jednostavan nenadziran model u velikom broju slučajeva radi bolje od mnogih predloženih nadziranih modela. Ipak, važno je napomenuti su Fu et al. [192] neuspješno probali rekreirati njihove rezultate. Nam et al. [193] su predstavili nove pristupe nazvane CLA i CLAMI koji automatizirano, bez ljudske intervencije, grupiraju module programske potpore ta temeljem iznosa njihovih značajki određuju sklonost sadržavanju programske potpore. Konačno, Zhang et al. [194] su koristili *spektralno grupiranje* (engl. Spectral Clustering, skr. SC) kao način nošenja s razlikama između izvornih i ciljnih projekata kod previđanja između projekta.

2.6 Mjere za vrednovanje modela

Modeli predviđanja pogrešaka programske potpore za programski modul određuju oznaku *sklon pogrešci/sadrži pogrešku* (engl. Defective) ili *nesklon pogrešci/ne sadrži pogrešku* (engl. Non-Defective). Moduli koji sadrže pogrešku smatraju se pozitivnim primjerima (engl. Positive), dok moduli koji ne sadrže poruku dobivaju oznaku negativnih primjera (engl. Negative). S obzirom na to da se problem prirodno predstavlja kao klasifikacijski problem, koriste se standardne metrike za vrednovanje klasifikacijskih modela.

U poglavlju 2.6.1 opisana je matrica zabune. U poglavlju 2.6.2 opisana je metrika preciznosti. U poglavlju 2.6.3 opisana je metrika odziva. U poglavlju 2.6.4 opisana je metrika F-mjere. Konačno, u poglavlju 2.6.5 opisana je metrika Matthewsov koeficijent korelacije.

2.6.1 Matrica zabune

Matrica zabune (engl. Confusion Matrix) definira se temeljem stvarne oznake primjera i oznake koju model dodijeli primjeru. Definicija matrice prikazane je u Tablici 2.1.

Tablica 2.1: Matrica zabune

		Predviđeno (*engl. Predicited*)	
		Negativno	Pozitivno
Stvarno (*engl. Actual*)	Negativno	TN	FP
	Pozitivno	FN	TP

Matrica se sastoji od četiri vrijednosti *stvaran pozitiv* (engl. True Positive, skr. TP), *stvaran negativ* (engl. True Negative, skr. TN), *lažan pozitiv* (engl. False Positive, skr. FP) i *lažan negativ* (engl. False Negative, skr. TN). Vrijednost *stvaran pozitiv* označava broj primjera koji su skloni pogreškama i model im je pridijelio oznaku da su skloni pogreškama. Vrijednost *stvaran negativ* označava broj primjera koji nisu skloni pogreškama i model im je pridijelio oznaku da nisu skloni pogreškama. Vrijednost *lažan negative* označava broj primjera koji su skloni pogreškama, ali im je model pridijelio oznaku da nisu skloni pogreškama. Konačno, vrijednost *lažan pozitiv* označava broj primjera koji nisu skloni pogreškama, ali im je model pridijelio oznaku da su skloni pogreškama.

2.6.2 Preciznost

Preciznost (engl. Precision, skr. P) je mjera za vrednovanje modela koja govori koliki udio primjera koji su označeni kao pozitivni uistinu jesu pozitivni, tj. od primjera koje model označi, koliki udio je uistinu značajan. Raspon vrijednosti preciznosti je između 0 i 1, gdje 0 označava model kod kojeg ni 1 označeni primjer zapravo nije značajan, dok 1 označava model kod kojeg su svi označeni primjeri značajni. Dakle, veća vrijednost preciznosti indikacije je boljeg modela. Matematička definicija preciznosti prikazana je u jednadžbi 2.2.

$$P = \frac{TP}{TP + FP} \tag{2.1}$$

2.6.3 Odziv

Odziv (engl. Recall, skr. R) je mjera za vrednovanje modela koja govori koliki udio primjera je označen od ukupnog broja primjera koji su uistinu trebali biti označeni, tj. koliko primjera je označeno od svih značajnih primjera. Raspon vrijednosti odziva je između 0 i 1, gdje 0 označava model kod kojeg ni 1 od primjera koji su trebali biti označeni nije označen, dok 1

označava model kod kojeg su svi primjeri koji su trebali biti označeni uistinu označeni. Dakle, veća vrijednost odziva indikacija je boljeg modela. Matematička definicija odziva prikazana je u jednadžbi 2.2.

$$R = \frac{TP}{TP+FN} \tag{2.2}$$

2.6.4 F-mjera

F-mjera (engl. F-score) ili kako se još naziva *F1-mjera* (engl. F1-score, skr. F1) je mjera za vrednovanje modela koja predstavlja harmonijsku sredinu preciznosti i odziva. Na ovaj način dvije vrijednosti su predstavljene jednim brojem što olakšava usporedbu različitih modela. Vrijednost F1 je između 0 i 1, a veći broj označava bolji model. Matematička definicija F1 prikazana je u jednadžbi 2.3. Iako se koristi u većini istraživanja, F1 je osjetljiva na neuravnoteženost podataka [32].

$$F1 = 2 * \frac{P*R}{P+R} \tag{2.3}$$

2.6.5 Matthewsov koeficijent korelacije

Matthewsov koeficijent korelacije (engl. Matthews Correlation Coefficient, skr. MCC) [203] statistička je mjera za vrednovanje modela zasnovana na stvarnim i predviđenim oznakama primjera. Mjera je ekvivalentna *Hi-kvadrat* (engl. Chi-square) statistici za tablicu kontingencije dimenzija $2*2$. Za razliku od F1, MCC nije osjetljiva na neuravnoteženost razreda u podacima te se stoga učestalo koristi pri vrednovanju modela za predviđanje pogrešaka programske potpore. Vrijednost MCC je između -1 i 1, gdje bi vrijednost -1 označavala model koji savršeno, ali obrnuto, pridjeluje oznake primjerima, vrijednost 0 bi označavala model koji nasumično pridjeljuje oznake primjerima, a vrijednost 1 bi označavao model koji savršeno dodjeljuje oznake primjerima. Dakle, veća vrijednost MCC indikacija je boljeg model. Ipak, model čija bi MCC vrijednost bila negativna može se iskoristi tako da se obrnu oznake dobivene od modela. Matematička definicija MCC prikazana je u jednadžbi 2.4.

$$MCC = \frac{TN*TP - FN*FP}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{2.4}$$

2.7 Statističke metode za validaciju modela

Modeli predviđanja pogrešaka programske potpore često su stohastičke naravi. To znači da ako se isti model trenira više puta nad istim podacima, neće uvijek na kraju treniranja rezultirati

jednako dobrim modelom. Stohastička narav modela najčešće proizlazi iz nasumične inicijalizacije njegovog početnog stanja, ali može biti i posljedica stohastičkih elemenata u proceduri za njegovo učenje, na primjer, ako se koristi *optimizacija u malim grupama* (engl. Mini-Batch Optimization) moguće je, i često korisno, nakon svake epohe optimizacijskog procesa nasumično promiješati primjere po grupama. Zbog stohastičke prirode modela, kako bi ih ispravno usporedili, nije dovoljno svaki model istrenirati jednom i potom usporediti rezultate vrednovanja različitih modela. Umjesto toga, potrebno je svaki model istrenirati više puta (empirijski se pokazalo da je 30 dovoljno za dobru usporedbu) te usporediti distribucije rezultata vrednovanja različitih modela. U ovu svrhu postoji nekoliko statističkih metoda koje se učestalo koriste za usporedbu modela.

U poglavlju 2.7.1 detaljnije je opisan Kolmogorov-Smirnov Test. U poglavlju 2.7.2 detaljnije je opisan test normalne distribucije. U poglavlju 2.7.3 detaljnije je opisan Student t-test. U poglavlju 2.7.4 detaljnije je opisan Mann-Whitney U test. Konačno u poglavlju 2.7.5 opisana je Cohenova d vrijednost.

2.7.1 Kolmogorov-Smirnov Test

Kolmogorov–Smirnov test (engl. Kolmogorov–Smirnov test, skr. KS Test) [204] je ne-parametarski test za usporedbu vjerojatnosnih distribucija. Ne-parametarski testovi ne pretpostavljaju vrste distribucija koje uspoređuju. U svojoj biti on odgovara na pitanje: kolika je vjerojatnost dobiti dva uzorkovana skupa primjera, ako su izvučeni iz iste vjerojatnosne distribucije. Nulta hipoteza testa je da su primjeri uzorkovani iz dvije iste distribucije. Ako se nulta hipoteza odbaci, tada je sigurno da distribucije nisu iste. Ako se nulta hipoteza ne uspije odbaciti, tada nije moguće sa sigurnosti utvrditi jednakost distribucija.

2.7.2 Test normalne distribucije

Test normalne distribucije (engl. Normality Test) [205] statistički je test koji daje odgovor na pitanje kolika je vjerojatnost da uzorkovani skup podataka dolazi iz distribucije koja se ponaša poput normalne distribucije. Nulta hipoteza testa je da primjeri dolaze iz normalne distribucije. Ako se testom odbaci nultu hipotezu, tada je sigurno da primjeri ne dolaze iz normalne distribucije. Ako se nulta hipoteza ne uspije odbaciti, tada nije moguće sa sigurnosti utvrditi da primjeri dolaze iz normalne distribucije. Ipak, istraživači često u slučaju kada nulta hipoteza nije odbačena pretpostavljaju da primjeri dolaze iz normalne distribucije.

2.7.3 Student t-test

Student t-test (engl. Student t-test) [206] statistički je test koji uspoređuje srednje vrijednosti dvaju uzorkovanih skupova. Nulta hipoteza testa je da su srednje vrijednosti dvaju uzorkova-

nih skupova jednake. Ako se testom odbaci nultu hipotezu, tada je sigurno da skupovi nemaju isti srednju vrijednost. Ako se testom ne odbaci nultu hipotezu, tada nije moguće sa sigurnosti utvrditi da skupovi imaju istu srednju vrijednost. S obzirom na to da se Student t-testom uspoređuje srednje vrijednosti distribucija, one su reprezentativne karakteristike tih distribucija. Kako bi takova usporedba bila smislena, prije Student t-testa potrebno je provesti test normalne distribucije za distribucije koje se uspoređuju jer je srednja vrijednost smislena reprezentativna karakteristika za takav tip distribucije.

2.7.4 Mann-Whitney U test

Mann-Witney U test [207] je ne-parametarski statistički test čija je nulta hipoteza da su primjeri jednog skupa i primjeri drugog skupa uzorkovani iz iste distribucije. Ako se odbaci nultu hipotezu, tada je sigurno da dva skupa vrijednosti nisu uzorkovana iz iste distribucije. Ako se testom ne uspije odbaciti nultu hipotezu, tada nije moguće sa sigurnosti utvrditi da primjeri ne dolaze iz iste distribucije. Ne-parametarski testovi ne pretpostavljaju vrste distribucija koje uspoređuju.

2.7.5 Choenov d

Choenov d (engl. Cohen's d Test) [208] mjera je *veličine učinka* (engl. Effect Size) razlike između dvije srednje vrijednosti. Dakle, daje odgovor na pitanje koliko je razlika između srednjih vrijednosti dviju distribucija značajna. Granice se mogu proizvoljno postaviti ovisno o području u kojem se vrši mjerenje, ali empirijski su se ustalile iduće granice. Razlika je *trivijalna* ako je *d vrijednost* manja od 0.2. Razlika je *mala* ako je *d vrijednost* između 0.2 i 0.5. Razlika je *značajna* ako je *d vrijednost* između 0.5 i 0.8. Konačno, razlika je *velika* ako je *d vrijednost* veća od 0.8.

Poglavlje 3

Određivanje klase zadatka dodijeljenog razvijatelju programskog kôda

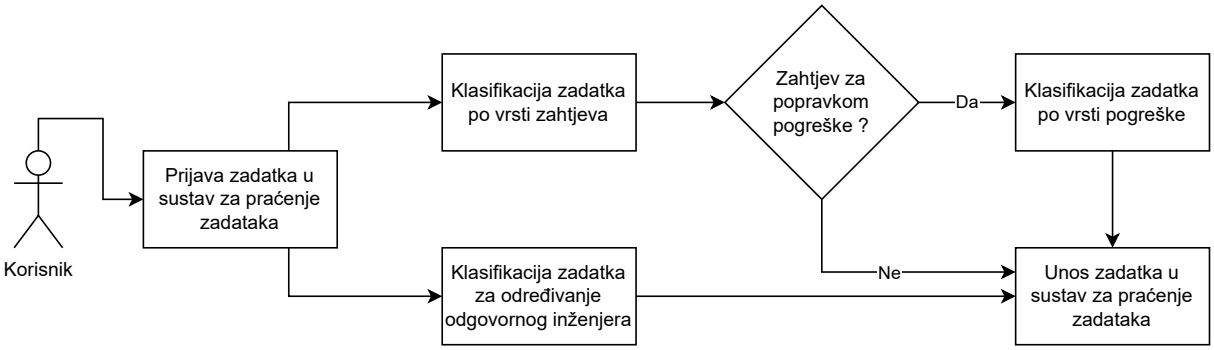
Sustavi za praćenje zadataka (engl. Issue Tracking System, skr. ITS) su programska potpora koja se koristi za praćenje i upravljanje zadacima u organizacijama koje vrše razvoj programske potpore. Korištenje ovih sustava omogućuje suradnju više inženjera te im omogućuje učinkovito i sustavno vođenje *lista zadataka* (engl. Backlog) koje je potrebno adresirati prilikom razvoja i održavanja programske potpore. Dodatno, omogućuje im upravljanje životnim ciklusima zadataka, od prijave do rješenja zadatka. Za svaki zadatak, sustavi za praćenje zadataka tipično pružaju mogućnost *dodavanja oznaka* (engl. Label Assignment), *dodjeljivanja zadatka* (engl. Task Assignment) članu tima, *definiranje vremenskog roka* (engl. Time Estimate) za obavljanje zadatka i mnoge druge mogućnosti. Sustavi za praćenje zadataka također omogućuju generiranje izvještaja o aktivnostima, izvještaja o efikasnosti rješavanja zadataka i izvještaja o drugim relevantnim statistikama, što može pomoći u poboljšanju procesa upravljanja zadacima u organizaciji. Uz navedeno, sustavi za praćenje zadataka omogućuju i dodatnu potporu za upravljanje projektom kao što je *potpora za upravljanje resursima*, *upravljanje vremenom*, *određivanje prioriteta*, itd.

Svaki *zadatak* (engl. Issue) u sustavu za praćenje zadataka sastoji se od naslova i opisa. Svaki zadatak ima svoj jedinstveni identifikator i za svaki zadatak moguće je praćenje povijesti aktivnosti vezanih uz taj zadatak. Ovo olakšava praćenje napretka na zadatku i poboljšava komunikacija između inženjera. Svakom zadatku moguće je dodijeliti oznake čime se pružaju dodatne informacije o zadatku i olakšava snalaženje u sustavu za praćenje zadataka. *Oznake* (engl. Label) koje se dodjeljuju zadatku obično je moguće proizvoljno definirati i koristiti, te nisu obavezne. Ipak, većina inženjera koristi ih za indikaciju tipa zadatka tj. radi li se o *zahtjevu za novom funkcionalnošću* (engl. Feature request), *zahtjevu za popravak pogreške* (engl. Bug request), *zahtjevu za ažuriranjem postojeće funkcionalnosti* (engl. Modification Request) ili nečem drugom. Pridjeljivanje oznaka zadacima obično se provodi ručno, od strane inženjera

koji rade na projektu. Oznake im omogućuju lakše snalaženje u listi zadataka koje je potrebno obaviti.

Klasifikacija zadataka (eng. Issue Classification) je proces klasificiranja ili razvrstavanja zadataka u odgovarajuće razrede kako bi se olakšalo upravljanje zadacima i njihovo rješavanje. Klasifikacija zadataka koristi se u različitim kontekstima, poput upravljanja korisničkom podrškom, razvoju programske potpore ili analizi podataka. Cilj klasifikacije zadataka je poboljšati učinkovitost i produktivnost rada, smanjiti vrijeme potrebno za rješavanje zadataka i poboljšati zadovoljstvo korisnika. Pri provođenju klasifikacije zadataka moguće je zadatke klasificirati po nekoliko različitih kriterija. Klasifikacijom zadataka može se određivati oznaka tipa zadatka, tj. radi li se o *zahtjevu za novom funkcionalnošću*, *zahtjevu za popravak pogreške*, *zahtjevu za ažuriranjem postojeće funkcionalnosti* ili nečem drugom [209, 210, 211, 212, 213, 214, 215]. Klasifikacija zadataka može se koristiti i za određivanje podtipa zadatka. Na primjer, ako je zadatak označen kao *zahtjev za popravak pogreške* moguće je provoditi detaljniju klasifikaciju tipa pogreške [216]. Tada se radi klasifikacija pogrešaka u određene skupine prema njihovom uzroku ili svojstvima, što omogućuje razumijevanje učestalosti, distribucije i težine problema. Klasifikacija pogrešaka programske potpore prema vrsti i svojstvima korisna je za identificiranje glavnih pogrešaka programske potpore i utvrđivanje uzroka njihove pojave. Jedna moguća klasifikacija pogrešaka programske potpore po vrsti bila bi podjela na sintaksne pogreške, logičke pogreške i pogreške okoline. *Sintaksna pogreška* nastaje pri nepravilnom korištenju programskog jezika. *Logička pogreška* nastaje kada programski kôd ne radi u skladu s očekivanjima jer postoji greška u njegovoj logici, a *pogreška okoline* kada programski kôd ne radi zbog loše interakcije s okolinom u kojoj se izvršava. Primjer loše interakcije s okolinom bio bi pokušaj spajanja na bazu podataka koja je u tom trenu nedostupna. Alternativno, zahtjevi za popravkom pogreške mogu se klasificirati po svom prioritetu [217]. Klasifikacija pogrešaka programske potpore omogućuje inženjerima da usmjeravaju svoje napore i resurse na prioritetne popravke koji su ključni za kvalitetu programske potpore i zadovoljstvo korisnika. Dakle, klasifikacija zadataka važan je korak u procesu upravljanja pogreškama programske potpore. Klasifikacija zadataka može određivati i podtip ovisno o dijelu sustava u kojem je potrebno intervenirati. Alternativno, klasifikacija zadataka može određivati kojem inženjeru ili skupini inženjera je potrebno dodijeliti zadatak. Određivanje inženjera kojem je najbolje dodijeliti zadataka ako se radi o zahtjevu za popravkom pogreške naziva se trijaža pogreške programske potpore (engl. Bug Triage) [218, 219]. Ovisno o vrsti klasifikacije zadataka, klasifikacijom se određuju oznake koje je potrebno dodijeliti zadatku ili drugi atributi vezani uz zadatak. Klasifikacija zadataka po vrsti zahtjeva od primarnog je interesa za ovu disertaciju. Slika 3.1 prikazuje moguće klasifikacije zadatka prilikom predaje u sustav za praćenje zadataka.

Automatska klasifikacija zadataka u sustavima za praćenje zadataka ima nekoliko prednosti i koristi za organizaciju, uključujući uštedu vremena, poboljšanje točnosti, jednolikost i dos-



Slika 3.1: Klasifikacija zadatka pri predaji u sustav za praćenje zadataka

ljednost u označavanju i skalabilnost označavanja.

Ušteda vremena postiže se jer automatska klasifikacija zadataka može značajno smanjiti vrijeme koje je potrebno za ručno kategoriziranje zadataka. Algoritmi klasifikacije mogu brzo i precizno klasificirati zadatke, što omogućuje inženjerima da se usredotoče na njihovo rješavanje umjesto na ručno praćenje i klasificiranje.

Poboljšanje efikasnosti rezultat je činjenice da automatska klasifikacija zadataka može pomoći inženjerima da brže i učinkovitije rješavaju zadatke. Kada se zadatci automatski klasificiraju, timovi mogu brzo identificirati probleme koji zahtijevaju hitnu pažnju, prioritet ili specifične vještine.

Poboljšanje točnosti može se postići primjenom automatske klasifikacija zadataka. Algoritmi klasifikacije mogu učiti na temelju povijesti prethodno klasificiranih zadataka i poboljšati svoju sposobnost prepoznavanja sličnih zadataka u budućnosti.

Jednolikost i dosljednost može se postići primjenom automatske klasifikacija zadataka jer će model za klasifikaciju zadatku uvijek pridijeliti isti oznaku. To može poboljšati komunikaciju između inženjera i smanjiti mogućnost pogreške.

Skalabilnost se postiže jer automatska klasifikacija zadataka može olakšati rukovanje velikim brojem zadataka. Algoritmi klasifikacije mogu brzo i precizno klasificirati veliki broj zadataka, što može olakšati upravljanje i održavanje ITS sustava u organizaciji.

U poglavlju 3.1 opisani su podatkovni skupovi za automatsku klasifikaciju zadataka i njihovi nedostatci, a u poglavlju 3.2 opisano je nekoliko modela za automatsku klasifikaciju zadataka.

3.1 Podatkovni skupovi i njihovi nedostatci

Podatkovni skupovi za automatsku klasifikaciju zadataka izgrađuju se prikupljanjem podataka iz sustava za praćenje zadataka. Inženjeri zadatcima pridjeljuju oznake i zadatke dodjeljuju inženjerima zaduženim za njihovo rješavanje čime implicitno grade podatkovni skup za klasifikaciju zadataka. Za izgradnju podatkovnog skupa potrebno je samo preuzeti navedene podatke iz sustava za praćenje zadataka. Međutim, istraživači su ukazali na probleme s kvalitetom oz-

naka zadataka u sustavima za praćenje zadataka. Tako su Herbold et al. [210] istaknuli da klasifikacija zadatka u sustavima za njihovo praćenje često ne odgovara njihovom opisu te su nastojali poboljšati automatsku klasifikaciju zadataka uključujući prethodno ručno uneseno znanje o zadacima. Nadalje, Pandey. et al. [26] su ustanovili da su zadatci često krivo označeni što razvojnim inženjerima otežava posao i oduzima vrijedno vrijeme prilikom njihovog adresiranja. Motivirani time, testirali su nekoliko modela strojnog učenja kako bi analizirali njihovu uspješnost u automatskoj klasifikaciji zadataka. Na tragu kvalitete označavanja zadataka, Ohira et al. [27] su primijetili da korisnici nekad zadatke označe kao *zahtjev za popravak pogreške* čak i kad oni to nisu, a isto tako, nekad *zahtjeve za popravkom pogreške* označe kao *zahtjev za novom funkcionalnošću*, iako se radi o pogrešci u programskoj potpori. Ručno su analizirali 1000 *zahtjeva za ispravkom pogreške* i 1000 *zahtjeva za novom funkcionalnošću* kako bi ustanovili efekt krivog označavanja i zašto do njega dolazi. Od 1000 *zahtjeva za popravkom pogreške* ustanovili su 61 zahtjev koji je ukazivao na probleme s radom centralnog dijela sustava, a od 1000 *zahtjeva za novom funkcionalnošću* 59 ih se odnosilo na centrali dio sustava. Od 61 *zahtjeva za popravkom pogreške* za točno označene zadatke bio potreban 1 dan da se dodijele inženjeru i adresiraju dok je za krivo označene zadatke bilo potrebno 7 dana da se dodijele inženjeru i još 6 dana da se adresiraju. Od 59 *zahtjeva za novom funkcionalnošću* točno označenima je bilo potrebno 3.6 dana da se dodijele inženjeru i 1 dan da se adresiraju, dok je krivo označenim zahtjevima bilo potrebno 8 dana da se dodijele inženjeru i preko 25 dana da se adresiraju. Ustanovili su da do pogrešne klasifikacije obično dolazi kada su u pitanju zadatci koji zahtijevaju veće promjene u sustavu pa se njihovo adresiranje dugo odgađa.

Bettenburg et al. [220] su ustanovili da se zadatci prijavljeni u sustavu za praćenje zadataka značajno razlikuju po svojoj kvaliteti. Dok neki korisnici pruže mnogo informacija i sudjeluju u rješavanju zadatka, drugi pružaju jako malo informacija ili čak pogrešne informacije, što može dovesti do pogrešnog adresiranja zadataka. Motivirani razlikom u kvaliteti prijavljenih zadataka Fang et al. [221] napravili su model za klasifikaciju zadataka označenih kao zahtjev za popravkom pogreške u razrede informativnih i ne informativnih zahtjeva.

Zhu et al. [222] su pokazali kako *zahtjevi za popravkom pogreške* obično imaju veći prioritet od ostalih zahtjeva. Sustavi za automatsku klasifikaciju zadataka često provode detaljniju klasifikaciju tj. ne klasificiraju samo dali je zadatak *zahtjev za popravkom pogreške* ili ne. Zbog prioriteta *zahtjeva za ispravkom pogreške* autori su predložili model koji označava samo radi li se o takvom zahtjevu ili ne.

Klasifikacija zadataka važna je za predviđanje budućih pogrešaka programske potpore. Oznake zadataka su bitne pri određivanju sadrži li neki kôd ili neka njegova verzija pogrešku ili ne. Analiza prethodnih pogrešaka pomaže u izgradnji modela za predviđanje pogrešaka programske potpore koji će identificirati vjerojatne uzroke budućih grešaka i predvidjeti njihovu vjerojatnost. To omogućuje proaktivno upravljanje pogreškama prije nego što se one pojave,

što može značajno smanjiti vrijeme i troškove potrebne za njihovo rješavanje.

Istraživači u području predviđanja pogrešaka programske potpore oslanjaju se na oznake zadataka, ne vodeći računa o njihovoj kvaliteti, iako postoje istraživanja koja kvalitetu oznaka zadataka dovode u pitanje. Tako, Antoniol et al. [24], Herzig et al. [25], Pandey. et al. [26] i Ohira et al. [27] su svi u svojim istraživanjima, ručnom analizom zadataka, pokazali kako je velik broj zadataka krivo označen.

Postoji nekoliko čimbenika koji mogu dovesti do pogrešnog označavanja zadataka.

Nedostatak jasno definiranih i preciznih kriterija za klasifikaciju zadataka je jedan od razloga. Ako nema jasnih smjernica ili ako su kriteriji previše općeniti, korisnici sustava mogu pogrešno zadacima dodijeliti pogrešne oznake, što može dovesti do pogrešne analize i upravljanja pogreškama programske potpore.

Povezan razlog koji može dovesti do pogrešnog označavanja zadataka je *nepravilna obuka ili nedovoljno znanje korisnika* ITS sustava o klasifikaciji zadataka. Ako korisnici nisu upoznati s kriterijima za klasifikaciju zadataka ili nisu educirani o najboljim praksama, mogu napraviti pogreške prilikom označavanja zadataka.

Još jedan razlog za pogrešno označavanje zadataka je *nedostatak jasno definiranih procesa praćenja i nadzora nad klasifikacijom zadataka*. Ako nema procesa za provjeru točnosti i dosljednost klasifikacije zadataka, može doći do nepravilnosti u označavanju i upravljanju zadatcima.

Konačno, zadatci u ITS sustavima mogu biti pogrešno označeni zbog *nedostatka kvalitete podataka ili nepotpunih informacija o zadatcima*. Ako se ne prikupljaju sve relevantne informacije o zadatku, ili ako su informacije netočne ili nepotpune, korisnici ITS sustava mogu napraviti pogrešnu procjenu o vrsti zadatka.

Dakle, nejasni kriteriji, nedovoljno obučeni korisnici, nedostatak procesa nadzora i nepotpuni podaci mogu uzrokovati pogrešno označavanje zadataka u sustavima za praćenje zadataka.

3.2 Modeli za automatsku klasifikaciju zadataka

Automatsko određivanje oznaka zadataka od interesa je istraživačima. Cilj je smanjiti vrijeme koje inženjeri provode ručno određujući oznake zadataka [209, 210, 211, 212, 213, 214, 215]. Istraživanja u ovom području, u pravilu, ne razmatraju utjecaj klasifikacije na predviđanje pogrešaka programske potpore. Modeli koji se koriste za klasifikaciju zadataka koriste naslov i opis zadatka kako bi izvršili klasifikaciju. Dakle, radi se o modelima zasnovanim na obradi prirodnog jezika. Alternativno, ako se u opis zadatka priloži ispis programskog stoga može se provoditi klasifikacija zadatka korištenjem informacija iz programskog stoga. Korištenje informacija programskog stoga koristi se najčešće pri klasifikaciji vrste pogreške na koju se odnosi zadatak. Nekoliko posljednjih istraživanja u ovom području navedeno je u nastavku.

Motivirani ciljem smanjenja vremena koje inženjeri provode ručno označavajući zadatke Wang et al. [209] su koristili BERT kako bi preporučili GitHub oznake zadataka temeljem njihova opisa. Naime, oni koji prijave zadatak u sustav često ne dodijele oznaku, pa ovaj posao pada na inženjere zadužene za održavanje programskog repozitorija. Ovaj posao može postati vremenski vrlo zahtjevan pa je automatsko označavanje poželjno kako bi oslobodilo vrijeme inženjera. Slično, Siddiq et al. [212] us ukazali na teškoću ručnog označavanja zadataka unutar sustava za praćenje zadataka te su primijenili BERT na zadatke unutar sustava GitHub i usporedili njegov rad s radom FastText modela. Nadalje, Bharadwaj et al. [214] razmotrili su različite verzije BERT modela za klasifikaciju zadataka u sustavu GitHub te dali preporuke kako koristiti BERT model za klasifikaciju zadataka. Colavito et al. [215] iskoristili su RoBERTa model za klasifikaciju GitHub zadataka. Alternativni pristup predložili su Kallis et al. [211]. Oni su predložili TICKET TAGGER koji se može koristiti za automatsku dodjelu oznaka zadacima unutar GitHub sustava. I u njihovom slučaju motivacija je isto bila smanjiti vrijeme koje inženjeri provode ručno označavajući zadatke. Još jedan alternativni pristup predložili su Li et al. [213]. Oni su predložili DeepLabel model za automatsko označavanje zadataka u sustavima za praćenje zadataka. Radi se a ansambl modelu koji sadrži više pod-modela specifičnih za različita polja (naslov i opis) zadatka. Koristi kombinaciju Word2Vec i dvosmjernog LSTM modela zasnovanog na pažnji (engl. Attention-based Bi-directional LSTM, skr. ABLSTM).

Iz izloženih posljednjih istraživanja u području klasifikacije zadataka vidljivo je učestalo korištenje BERT modela za klasifikaciju zadataka. Ovo ne čudi s obzirom na činjenicu da se zadatak svodi na klasifikaciju zasnovanu na obradi prirodnog jezika, a BERT modeli su jedni od trenutno najboljih (engl. State-of-the-Art) modela u tom području.

Poglavlje 4

Utjecaj klasifikacije zadataka na podatkovne skupove i rad modela predviđanja pogrešaka programske potpore

Razvoj modela za predviđanje pogrešaka programske potpore (engl. Software Defect Prediction skr. SDP) oslanja se na specijalizirane podatkovne skupove dobivene prikupljanjem informacija iz sustava za praćenje verzija kôda (engl. Version Control System, skr. VCS) i sustava za praćenje zadataka (engl. Issue Tracking System, skr. ITS).

Predaje promjene kôda (engl. Commit) u sustav za praćenje verzija kôda uparuju se sa zadacima u sustavima za praćenje zadataka. Temeljem izvornog programskog kôda prikupljenog iz sustava za praćenje verzija grade se značajke za opis programskih modula koje se koriste za predviđanje pogrešaka programske potpore. Temeljem zadataka prikupljenih iz sustava za praćenje zadataka, točnije njihovih oznaka (engl. Label), te uparenih *predaja promjena kôda* određuje se *sklonost pogreškama* programskih modula. Značajke koje opisuju programske module i njihova *sklonost pogreškama* čine skup za razvoj modela za predviđanje pogrešaka programske potpore.

U procesu izgradnje podatkovnog skupa za razvoj modela za predviđanje pogrešaka programske potpore postoje dvije situacije koje unose šum u podatkovni skup, narušavajući njegovu kvalitetu i posljedično kvalitetu razvijenih modela. Prva situacija nastaje kada se *predaje promjene kôda* ne mogu povezati sa zadacima. Ako se kôd nad kojim su vršene promjene tretira kao nesklon pogreškama, iako nisu uspješno određeni zadatci koji su bili razlog tih promjena, tada je moguće da će se kôd koji je sklon pogreškama tretirati kao nesklon pogreškama. Ako bi se takav kôd uvijek tretirao kao sklon pogreškama, tada bi problem bio upravo suprotan. Bilo bi moguće da se kôd koji nije sklon pogreškama proglasi sklonim pogreškama. Druga situacija kada se šum može unijeti u izgrađeni skup za razvoj modela za predviđanje pogrešaka

programske potpore je kada su zadatci upareni s *predajama promjene kôda* krivo označeni.

Iako su istraživači ukazali na krivo označene zadatke u sustavima za praćenje zadataka [24, 25], na činjenicu da šum unesen u podatkovne skupove za predviđanje pogrešaka programske potpore ima negativan utjecaj na modele za predviđanje pogrešaka programske potpore [49, 58, 69, 70] i na činjenicu da je jednom unesen šum teško ukloniti iz podatkovnog skupa [65], istraživači se i dalje često slijepo oslanjaju na ispravnost oznaka zadataka. Istraživači često traže referencu na zadatak u porukama *predaje promjene kôda* i ključne riječi poput *fix* i *bug* te u slučaju njihova pronalaska asocirani kôd proglašavaju sklonim pogreškama [41, 42, 43, 44, 61].

Antoniol et al. [24] su pronašli da je velik broj zadataka označenih kao *zahtjev za popravkom pogreške* to zapravo nisu. Razmatrali su mogućnost automatskog označavanja zadataka koristeći modele stablo odluke (engl. Decision Trees), naivan Bayes (engl. Naive Bayes) i logističku regresiju (engl. Logistic Regression). Zadatke i izvorni programski kôd prikupili su od tri projekta (*Mozilla*, *Eclipse* i *JBoss*) zasnovana na programskom jeziku Java. Pokazali su da navedeni modeli mogu uspješno označavati zadatke postižući F1 rezultat od 0.70, ali nisu direktno vrednovali utjecaj na rad modela za predviđanje pogrešaka programske potpore.

Od objave njihova istraživačkog rada, razvijeni su napredni modeli za obradu prirodnog teksta poput BERT [223] modela. Istraživači su ih primijenili za označavanje zadataka [209, 210, 211, 212] uz obećavajuće rezultate. Ipak, ni jedno od ovih istraživanja nije klasifikaciju zadataka razmatralo u kontekstu predviđanja pogrešaka programske potpore, tj. ni jedno od ovih istraživanja nije razmatralo utjecaj kvalitete klasifikacije zadataka na rad modela za predviđanje pogrešaka programske potpore.

U sklopu ove disertacije razmatran je utjecaj kvalitete automatskog označavanja zadataka na kvalitetu skupova i rad modela za predviđanje pogrešaka programske potpore.

Detaljnije govoreći, prikupljeni su podatci 7 programskih repozitorija slobodno dostupnog kôda tj. podatci iz njihovih sustava za kontrolu verzija i sustava za praćenje zadataka. Temeljem prikupljenih podataka, izgrađeni su skupovi za predviđanje pogrešaka programske potpore. Prikupljeni su podatci o svim *predajama promjene kôda* i svim zadacima. Za svaki od repozitorija i svaku *predanu promjenu kôda* tražene su veze s prikupljenim zadatcima. Oni zadatci koji su povezani s barem jednom *predajom promjene kôda* smatraju se *zadacima od interesa* (engl. Issues of Interest, skr. IOI). *Predaje promjene kôda* koje sadrže vezu na barem jedan zadatak smatraju se *predajama od interesa* (engl. Commits of Interest, skr. COI). Za svaku datoteku izvornog programskog kôda, određene su sve *predaje promjena kôda* koje ju ažuriraju. Ukoliko su sve takve *predaje promjena kôda* određene kao COI tada se datoteka smatra *datotekom od interesa* (engl. File of Interest, skr. FOI). Svaka verzija datoteke izvornog programskog kôda, dohvaćena je iz programskog repozitorija. Potom je dekodirana, uklonjeni su svi komentari i izgrađene su semantičke značajke koristeći GraphCodeBERT [92]. Uz semantičke značajke, konstruirane su klasične značajke složenosti kôda i procesne značajke. Sve navedene značajke zajedno opisuju

programski modul (u ovom slučaju datoteku izvornog programskog kôda) nad kojim se vrši predviđanje pogrešaka programske potpore. Modul se smatra sklonim pogrešci ako je ikad u svojoj povijesti sadržavao programsku pogrešku, tj. ako je i jedna *predaja promjene kôda* nad datotekom tog modula imala cilj adresiranja zadatka označenog kao *zahtjev za popravak pogreške*. Kako bi se konstruirao podatkovni skup za predviđanje pogrešaka programske potpore bez krivo označenih zadataka, za svaki repozitorij ručno je analizirano i označeno barem 1000 IOI. Nakon izgradnje podatkovnog skupa, analizirana je količina šuma unesena u podatkovni skup ako se ne koriste ručno određene oznake zadataka, nego se one automatski određuju koristeći nekoliko različitih pristupa. Ti pristupi su *jednostavna heuristika zasnovana na ključnim riječima* (engl. Simple Keyword Heuristic, skr. KWM), *unaprijeđena heuristika zasnovana na ključnim riječima* (engl. Improved Keyword Heuristic, skr. IKWM), *FastText* model i *RoBERTa* model. Za svaki od podatkovnih skupova trenirani su i vrednovani idući modeli za predviđanje pogrešaka programske potpore: *logistička regresija* (engl. Logistic Regression, skr. LR), *stablo odluke* (engl. Decision Tree, skr. DTC), *naivan Bayes* (engl. Naive Bayes, skr. NB) i *k najbližih susjeda* (engl. K-Nearest Neighbors, skr. KNN). Razmatran je utjecaj šuma na njihov rad.

U nastavku ovog poglavlja dan je kratak pregled područja obrade prirodnog jezika, detaljno je opisan proces prikupljanja podataka, izgradnja skupa i razvoj modela za klasifikaciju zadataka te postupak predviđanja pogrešaka i analiza utjecaja šuma na rezultate modela za predviđanje pogrešaka programske potpore.

U poglavlju 4.1 dan je kratak opis obrade prirodnog jezika. Područje obrade prirodnog jezika je veliko i detaljan opis područja van je opsega ove disertacije, umjesto toga dan je kratak pregled područja potreban za razumijevanje modela i tehnika korištenih pri provođenju ovog istraživanja. U poglavlju 4.2 opisan je postupak prikupljanja podataka. U poglavlju 4.3 opisana je izgradnja podatkovnog skupa za klasifikaciju zadataka i korišteni modeli. U poglavlju 4.4 opisan je postupak izgradnje podatkovnog skupa za predviđanje pogrešaka programske potpore i korišteni modeli. U poglavlju 4.5 prezentirani su i analizirani dobiveni rezultati. Konačno, u poglavlju 4.6 predstavljeni su rizici valjanosti provedenog istraživanja i opisan je način smanjenja istih.

4.1 Obrada prirodnog jezika

Obrada prirodnog jezika (engl. Natuaral Language Processing, skr. NLP) područje je *umjetne inteligencije* (engl. Artificial Intelligence) koje se bavi razvojem modela koji računalima omogućuju razumijevanje ljudskog jezika [224].

NLP uključuje mnoge zadatke kao što je *prepoznavanje govora* (engl. Speech Recognition), *automatsko sažimanje teksta* (engl. Automatic Text Summarization), *ispravljanje gramatičkih pogrešaka* (engl. Grammatical Error Correction), *strojno prevođenje* (engl. Machine Transla-

tion) i još mnoge druge zadatke.

Detaljna analiza NLP-a van je okvira ove disertacije. Ipak, dodan je kratak opis pojmova i modela zbog lakšeg razumijevanja provedenog istraživanja.

Morfološka analiza (engl. Morphological analysis) pod-područje je NLP-a i lingvistike koje izučava strukture riječi i kako se riječi formiraju interakcijom različitih morfema. *Morfem* (engl. Morpheme) je najmanja jedinica jezika koja sadrži neko značenje. *Stem* (engl. Stem) i *afiks* (engl. Affix) primjeri su morfema. *Stem* je glavni dio riječi koji joj daje njeno primarno značenje. *Afiks* riječi daju dodatno značenje. Primjeri afiksa su *sufiks* (engl. Suffix) i *prefiks* (engl. Prefix). *Prefiks* je skup slova koji se dodaje na početak riječi i time doprinosi njenom značenju. *Sufix* je skup slova koji se dodaje na kraj riječi i time doprinosi njenom značenju. *Steming* (engl. Stemming) je postupak svođenja riječi na njen *stem*.

Semantička analiza (engl. Semantic Analysis) pod-područje je NLP-a koje se bavi razumijevanjem sadržaja teksta. Osnovni problem NLP-a je dvosmislenost prirodnog jezika. Dvosmislenost može biti na razini riječi, dijelova rečenica pa čak i cijelih rečenica. Za razrješavanje dvosmislenosti potrebno je uzeti širi kontekst teksta koji se razmatra [225]. Razvijeni su mnogi modeli za razumijevanje sadržaja teksta, a mogu se podijeliti na simboličke, statističke i konektivističke tj. neuronske [226].

Konektivistička istraživanja u području obrade prirodnog jezika fokusirala su se na *Word2Vec* modele [227], *konvolucijske* modele [228, 229, 230], *ponavljajuće* modele [231, 232] i *modele zasnovane na pozornosti* [233, 234].

Istraživači su pokazali prednosti pred-treniranja modela na velikim količinama teksta za potrebe korištenja tih modela za učenje klasifikacije teksta i ostale zadatke obrade prirodnog jezika [235]. Korištenjem pred-treniranih modela moguće je izbjeći treniranje modela od samog početka, čime se značajno skraćuje vrijeme za njihovo treniranje i izgrađuju modeli koji rade bolje nego da su trenirani samo za specifičan zadatak.

Korištenje velikih pred-treniranih modela i njihovo kasnije usavršavanje na specifičnim zadacima ostvarilo je značajna postignuća u nekoliko područja obrade prirodnog jezika [223, 236].

FastText [237] je metoda za ugrađivanje (engl. Embedding) riječi koja je proširenje *Word2Vec* modela. Smatra se modelom *vreće riječi* (engl. Bag of Words) koji svaku riječ predstavlja kao skup *n-grama*. U izvornom članku je pokazan kao puno brži model od dubokih modela s komparativnim rezultatima.

Modeli *BERT* [223] i *RoBERTa* [236] su danas u širokoj primjeni. Ime RoBERTa dolazi od *robusno optimirani BERT pristup pred-treniranja* (engl. Robustly Optimized BERT Pretraining Approach), a ime BERT od *dvosmjerna enkoder reprezentacija dobivena primjenom transofrmera* (engl. Bidirectional Encoder Representations from Transformers).

Oba modela zasnovani su na istoj arhitekturi, koristeći *višeslojne dvosmjerne transformer enkodere* [238] koji su pred-treniran na velikoj količini teksta, točnije na *BooksCorpus* (800

milijuna riječi) [239] i *English Wikipedia* (2.5 bilijuna riječi).

Arhitektura se sastoji od 12 enkodera zasnovanih na transformerima. Svaki enkoder sastoji se od sloja zasnovanog na pažnji s više glava i potpuno povezanog sloja te sumiranja i normalizacije između navedenih slojeva i na izlazu iz enkodera. Na vrh ovih enkodera stavlja se potpuno povezana neuronska mreža i SoftMax sloj specijalizirani za zadatak za koji će se model primjenjivati. BERT je treniran istovremeno na dva zadatka: pogađanje skrivene riječi iz konteksta i pogađanje iduće rečenice. Razlika između BERT i RoBERTa modela je što BERT za vrijeme pred-procesiranja podatka maskiranje podataka čini samo jednom, a RoBERTa maskiranje vrši 10 puta čime se za svaku rečenicu dobije 10 različitih maskiranih verzija. RoBERTa ovim omogućuje korištenje različite maske za vrijeme svake epohe učenja.

Fasttext i RoBERTa korišteni su u sklopu ovog istraživanja.

4.2 Prikupljanje podataka

Kako bi se provelo istraživanje za ovu doktorsku disertaciju reprezentativni podatci su prikupljeni iz javno dostupnih repozitorija, slobodnog programskog kôda, dostupnih na platformi GitHub.

Za prikupljanje podataka napisana je programska skripta koristeći *PyGithub* * paket napisanog u programskom jeziku Python. Navedeni paket pruža jednostavno sučelje za komunikaciju s platformom GitHub.

Napisana skripta dohvaća 10 najpopularniji programskih repozitorija, koji imaju barem 5 zadataka označenih kao *zadatak dobar za početnike* (engl. Good First Issue). Kod razvoja programske potpore slobodno dostupnog kôda inženjeri, koji održavaju programski repozitorij, neke zadatke označe kao *zadatak koji je dobar za početnike*, tj. takvi zadatci ne zahtijevaju prethodno iskustvo rada na repozitoriju. Na takav način, inženjeri koji održavaju repozitorij olakšavaju novim inženjerima, bez prethodno iskustva, da se uključe u rad na repozitoriju. Prisutnost zadataka s navedenom oznakom implicira aktivan projekt. Navedeno ograničenje je uključeno kako bi se dohvatili aktivni projekti. S obzirom na to da je odlučeno semantičke značajke konstruirati koristeći GraphCodeBERT model, odbačeni su repozitoriji čiji primarni programski jezik nije podržan od strane navedenog modela. Dodatno, isključeni su repozitoriji s manje od 50 zvijezda i manje od 100 zadataka. Na ovaj način osigurano je dohvaćanje značajnih repozitorija s adekvatnom količinom podataka.

Tablica 4.1 prikazuje podatke prikupljene o svakom repozitoriju. Tablica 4.2 prikazuje podatke o zadacima koji su prikupljeni za svaki repozitorij, a Tablica 4.3 prikazuje podatke koji su prikupljeni za svaku *predaju promjene* izvornog programskog kôda za svaki repozitorij. Konačno, Tablica 4.4 prikazuje podatke prikupljene o svakoj datoteci izvornog programskog kôda

*`https://pypi.org/project/PyGithub/`

koja je ažurirana barem jednom predajom promjene kôda.

Tablica 4.1: Podatci prikupljeni o programskom repozitoriju

Ime	Opis
Ime	Ime repozitorija
Zvijezda	Broj zvijezda pridijeljen repozitoriju
Programski jezik	Glavni programski jezik repozitorija
Zadatci	Lista zadataka repozitorija
Predaja promjena kôda	Lista predaja promjena kôda repozitorija

Tablica 4.2: Podatci prikupljeni o svakom zadatku

Ime	Opis
Id	Jedinstveni identifikator
Broj	Jedinstveni identifikator unutar repozitorija
Naslov	Naslov radnog zadatka
Tijelo	Opis radnog zadatka
URL	URL do web stranice radnog zadatka
Oznake	Oznake pridijeljene radnom zadatku unutar repozitorija
Broj komentara	Broj popratnih komentara na radnom zadatku
Ima li integracijski zahtjev	Postoji li zahtjev za integraciju u glavnu programsku granu

Tablica 4.3: Podatci prikupljeni o svakoj predaji promjene izvornog programskog kôda

Ime	Opis
SHA	Jedinstveni identifikator
Poruka	Poruka uz predaju promjene izvornog programskog kôda
Datum	Vrijeme kada je predaja promjene kôda podnesena
Datoteke	Lista datoteka koje predaja promjene kôda ažurira

Koristeći opisani postupak prikupljanja podataka, prikupljeni su podatci 7 programskih repozitorija s ukupno 299773 zadataka i 153664 *predaja promjene kôda*. Tablica 4.5 prezentira abecedno sortiranu listu podataka o svakom repozitoriju, njegov glavni programski jezik, broj zadataka i broj *predaja promjene izvornog programskog kôda*. Slika 4.1 prikazuje distribuciju primarnih programskih jezika između repozitorija iz kojih su prikupljeni podatci.

Tablica 4.4: Podatci prikupljeni o svakoj verziji svake datoteke izvornog programskog kôda

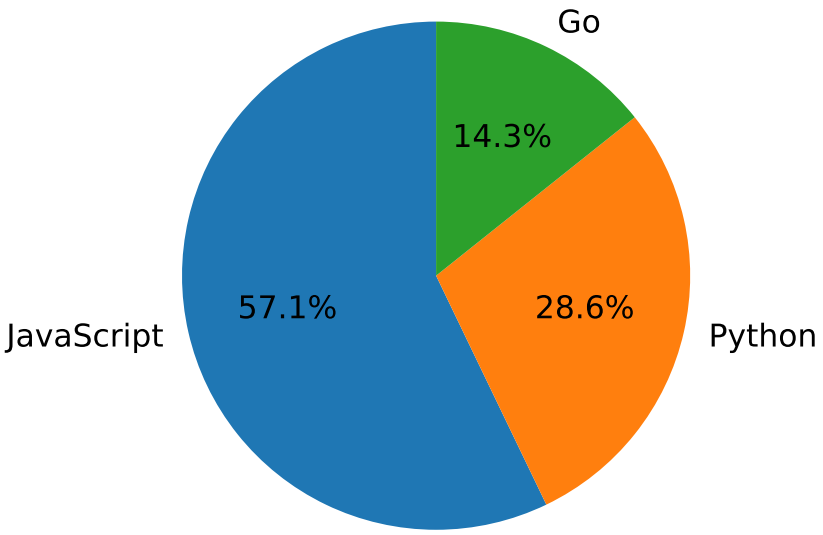
Ime	Opis
SHA	Jedinstveni identifikator datoteke tj. njene specifične verzije
Ime	Jedinstveno ime datoteke
Broj promjena	Broj promjena u datoteci temeljem zadnje predaje promjene
Broj dodanih linija	Broj dodanih linija kôda temeljem zadnje predaje promjene
Broj uklonjenih linija	Broj uklonjenih linija kôda temeljem zadnje predaje promjene
Sadržaj datoteke	Sadržaj datoteke nakon posljednje predaje promjene kôda
Kodiranje	Kodiranje korišteno za sadržaj datoteke

Tablica 4.5: Podatci prikupljeni o svakom programskom repozitoriju

Ime	Kratko ime	Glavni programski jezik	Broj zadataka	Broj predaja promjene kôda
facebook/react-native	react-native	JavaScript	34402	25473
huggingface/transformers	transformers	Python	19163	10789
kubernetes/kubernetes	kubernetes	Go	112661	28557
mui/material-ui	material-ui	JavaScript	33815	20518
nodejs/node	node	JavaScript	43374	37195
vercel/next.js	next.js	JavaScript	26399	12521
ytdl-org/youtube-dl	youtube-dl	Python	29959	18611

Nakon prikupljanja podataka svaka *predaja promjene izvornog programskog kôda* je analizirana, tj. poruka uz svako od njih je analizirana kako bi se pronašle veze sa zadatcima.

Zadatci koji su povezani s barem jednom *predajom promjene kôda* smatraju se *zadacima od interesa*. *Predaje promjena kôda* koje sadrže vezu na barem jedan zadatak smatraju se *predajama od interesa*. Pri analizi poruka pronađene su veze na zadatke koji više ne postoje jer su obrisani s platforme GitHub. *Predaje promjene kôda* koje sadrže takve zadatke u svojim porukama su uklonjene iz podatkovnog skupa *predaja od interesa* jer nije moguće doći do jednog ili više zadataka s kojim su povezane. Analizom svih *predaja promjena programskog kôda* otkrivene su datoteke izvornog programskog kôda koje su mijenjane isključivo s *zadacima od interesa*. Takve datoteke smatraju se *datotekama od interesa*. Zadržane su samo datoteke s ekstenzijama *.py*, *.php*, *.js*, *.java*, *.go* i *.rb*, jer one označavaju programske jezike podržane od strane GraphCodeBERT modela koji je korišten za izgradnju semantičkih značajki. Radi se o nešto širem skupu programskih jezika nego što je skup glavnih programskih jezika repozitorija iz kojih su podatci prikupljeni. Iako je jedan glavni jezik repozitorija, repozitorij može sadržavati i datoteke napisane u drugim programskim jezicima. Zato je skup ekstenzija nešto širi. S obzirom na to da su dohvaćene datoteke izvornog programskog jezika kodirane koristeći base64 kodiranje, sve verzije svih datoteka su prvo dekodirane. Potom su uklonjeni svi komentari ko-



Slika 4.1: Glavni programski jezici repozitorija iz kojih su prikupljeni podatci

risteći *pyparsing* [†] programski paket napisan u programskom jeziku Python. Nakon uklanjanja komentara ispostavilo se da ponekad uzastopne verzije datoteka izvornog programskog kôda nemaju nikakve razlike. Ovo nastaje ako *predaja promjene kôda* ažurira samo sadržaj komentara unutar datoteke. Kod pojava identičnih uzastopnih verzija datoteka, kasnija verzija datoteke je isključena iz razmatranja.

Tablica 4.6 prikazuje konačan broj IOI, COI i FOI za svaki od repozitorija. Iz podataka prikazanih u Tablici 4.5 i Tablici 4.6 vidljivo je kako neki repozitoriji, poput *node* projekta, sistematično povezuju *predaje promjena kôda* sa zadacima, dok drugi, poput *kubernetes* projekta, to ne rade, pa je vrlo teško pronaći veze između njihovih zadataka i *predaja promjena kôda*.

4.3 Klasifikacija zadataka

U ovom poglavlju opisan je postupak izgradnje podatkovnog skupa za klasifikaciju zadataka i postupak izgradnje modela za klasifikaciju zadataka.

Slika 4.2 prikazuje općeniti pregled postupka klasifikacije zadataka. U gornjem dijelu slike prikazani su zadatci koji se sastoje od naslova i opisa. Koristeći naslove i opise zadataka te ručno određene oznake zadataka, razvija se model za klasifikaciju zadataka. U sklopu ovog istraživanja razmatrano je nekoliko metoda za klasifikaciju zadataka: *jednostavna heuristika zasnovana na ključnim riječima* (engl. Simple Keyword Heuristic, skr. KWM), *unaprijeđena*

[†] `https://pypi.org/project/pyparsing/`

Tablica 4.6: Podatci od interesa za svaki programski repozitorij

Repozitorij	Broj IOI	Broj COI	Broj FOI
react-native	363	4443	549
transformers	3039	5567	2338
kubernetes	94	10381	195
material-ui	5408	7716	10872
node	13875	15397	15189
next.js	5501	5676	8464
youtube-dl	148	6075	72

heuristika zasnovana na ključnim riječima (engl. Improved Keyword Heuristic, skr. IKWM), *FastText* model i *RoBERTa* model. Slika prikazuje korištenje razvijenih modela kako bi se označili zadatci. Iako prikazane veze pokazuju od modela na zadatke korištene za razvoj modela to nema smisla već se model treba primijeniti na zadatke koji nisu korišteni prilikom njegove izgradnje. Dodijeljena oznaka, tj. dali je zadatak *zahtjev za popravkom pogreške* ili ne, prikazana je na slici s malim zelenim ili crvenim kvadratom. Zeleni kvadrat označava zadatke koji nisu bili *zahtjev za popravkom pogreške*, a crveni kvadrati označavaju zadatke koji su *zahtjev za popravkom pogreške*. U donjem dijelu slike prikazane su veze između označenih zadataka i povezanih *predaja promjene kôda*. Predaje promjene kôda mijenjaju izvorni programski kôd unutar žute, plave i crvene datoteke. Prikaz ovih datoteka olakšava razumijevanje kasnijeg postupka izgradnje podatkovnog skupa za predviđanje pogrešaka programske potpore te kako se iz oznaka zadataka dobivaju konačne oznake za predviđanje pogrešaka programske potpore.

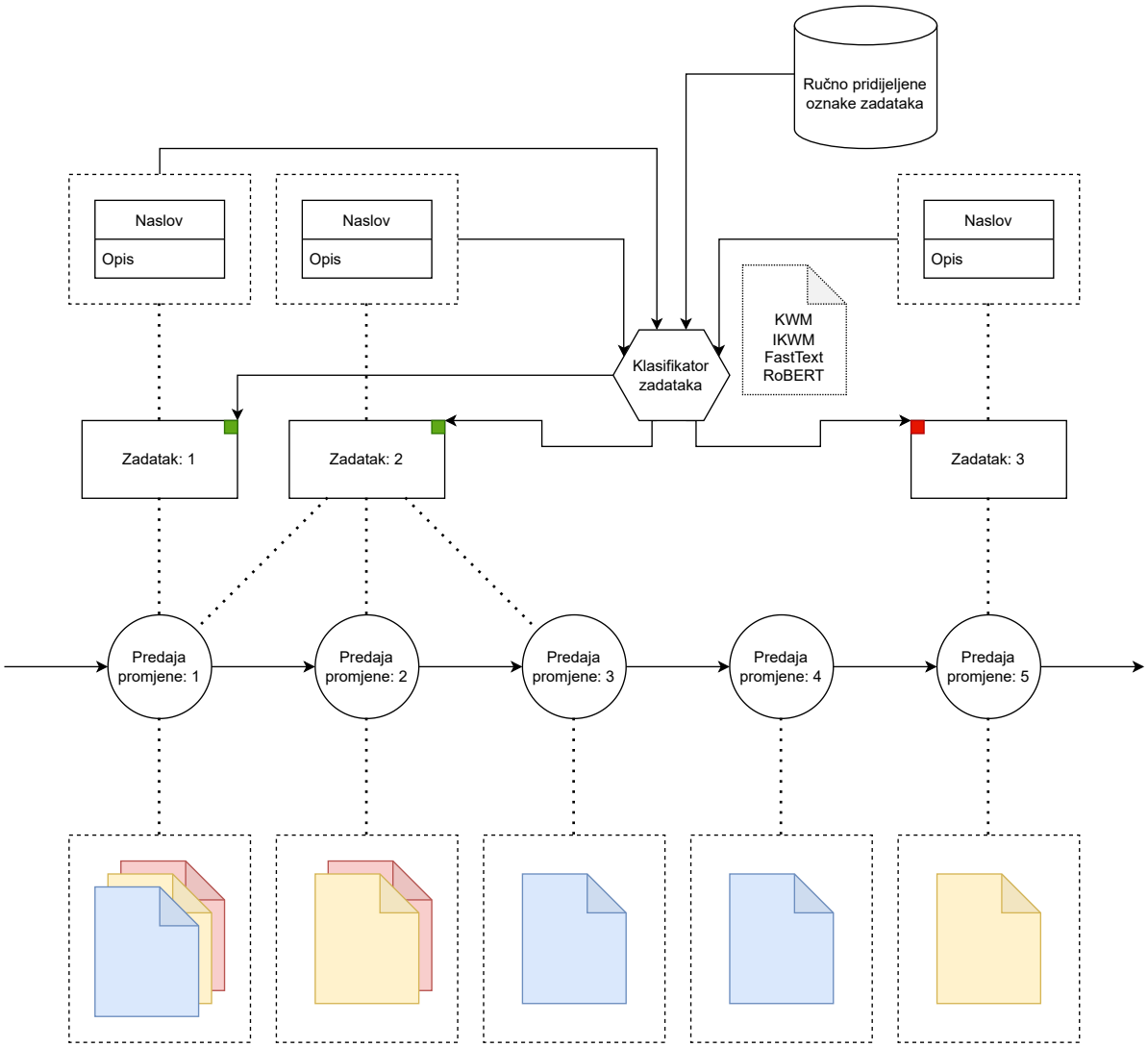
U poglavlju 4.3.1 opisan je postupak izgradnje podatkovnog skupa za klasifikaciju zadataka, a u poglavlju 4.3.2 opisani su modeli korišteni za klasifikaciju zadataka.

4.3.1 Izgradnja podatkovnog skupa

Nakon određivanja IOI, ručno je označeno barem 1000 primjera IOI iz svakog od prikupljenih repozitorija. Ako je u repozitoriju bilo manje od 1000 IOI tada su svi označeni, a ako je bilo više od 1000, odabrano je prvih 1000 poredanih po broju zadatka i potom su dodani IOI koji utječu na FOI koje ažurira prvih 1000 IOI.

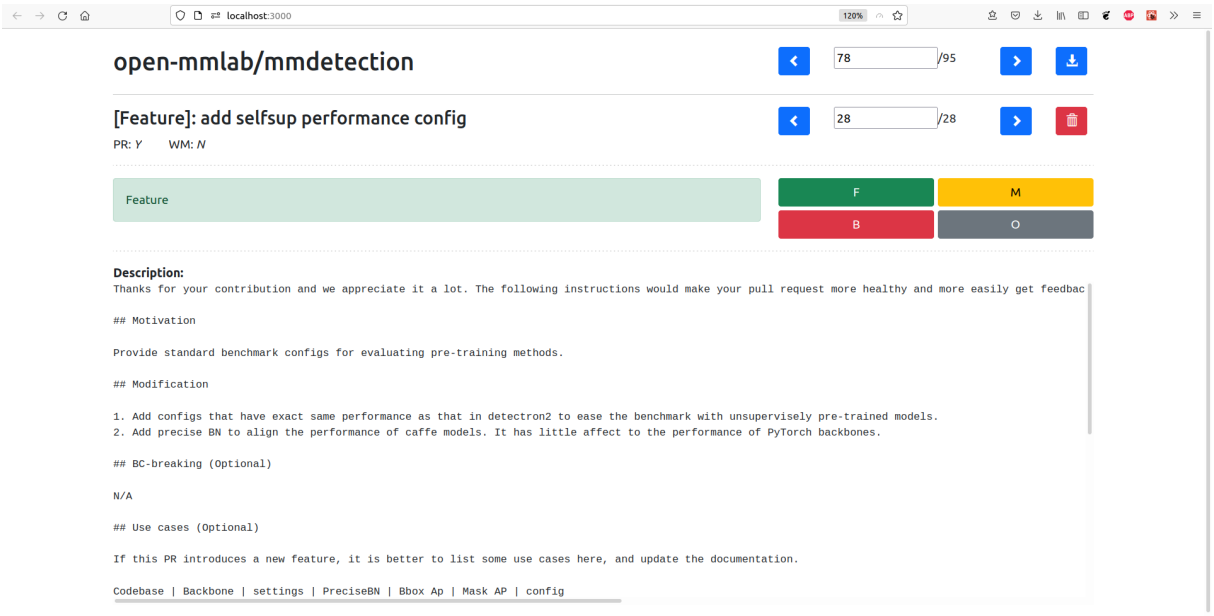
Za potrebe pregledavanja i označavanja IOI razvijena je React [‡] *mrežna aplikacija* (engl. Web Application). U aplikaciju je moguće učitati prikupljene podatke. Aplikacija prikazuje *ime repozitorija*, *naslov zadatka*, *opis zadatka*, *oznake* koje su pridijeljene zadatku i dali je temeljem razmatranog zadatka podnesen *integracijski zahtjev* (engl. Merge Request). Aplikacija

[‡] `https://reactjs.org/`



Slika 4.2: Pregled klasifikacije zadataka

omogućuje *navigaciju* između repozitorija i između zadataka unutar repozitorija. Nadalje, aplikacija omogućuje *označavanje zadataka* i *preuzimanje podataka* u istom formatu u kojem ih i prihvaća. Slika 4.3 prikazuje grafičko sučelje aplikacije.



Slika 4.3: Grafičko sučelje aplikacije za označavanje zadataka

Aplikacija omogućuje označavanje zadataka s jednim od četiri različita razreda tj. s jednom od četiri različite oznake. *Zahtjev za novom funkcionalnošću* (engl. Feature Request), *zahtjev za izmjenom postojeće funkcionalnosti* (engl. Modification Request), *zahtjev za popravkom pogreške* (engl. Bug Request) i *ostalo* (engl. Other). *Ostalo* označava zadatke koji su zapravo pitanja, pokretanje rasprave, komentari, ažuriranje dokumentacije ili bilo što drugo što ne pripada u ni jednu od preostalih skupina zahtjeva. Iako je za potrebe ovog istraživanja dovoljno razlikovati samo između *zahtjeva za popravkom pogreške* i svih ostalih, aplikacija je napravljena općenitije od toga, imajući na umu da bi detaljnije razlikovanje zadataka moglo biti od interesa drugim istraživačima. U ovom istraživanju zahtjevi označeni kao *zahtjev za novom funkcionalnošću*, *zahtjev za izmjenom postojeće funkcionalnosti* i kao *preostali zahtjevi* tretiraju se kao zahtjev koji nije za popravak pogreške programske potpore.

Sadržaj konstruiranih podatkovnih skupova za klasifikaciju zadataka prikazan je u Tablici 4.7. Označeni zahtjevi od interesa označeni su s LIOI (engl. Labeled Issues of Interest).

Iako su iz repozitorija prikupljene oznake zadataka iz sustava GitHub, one nisu korištene za automatsko označavanje zadataka. Naime, navedene oznake su proizvoljno definirane i nisu obavezne, pa se značajno razlikuju od projekta do projekta. Dodatno, ove oznake često ne označavaju vrstu zahtjeva na koji se zadatak odnosi već dio programske potpore u kojem je potrebno napraviti promjene kako bi se zadatak adresirao.

Tablica 4.7: Podatkovni skupovi za klasifikaciju zadataka za svaki repozitorij

Repozitorij	Broj primjera	Broj zahtjeva za popravak pogreške	Broj preostalih zahtjeva
react-native	363	93	270
transformers	1013	278	735
kubernetes	94	36	58
material-ui	1102	296	806
node	1012	301	711
next.js	1062	328	734
youtube-dl	148	55	93

4.3.2 Modeli za klasifikaciju zadataka

U ovom poglavlju opisani su modeli korišteni za klasifikaciju zadataka. Prvo je opisana *heuristika zasnovana na ključnim riječima*, potom *unaprijeđena heuristika zasnovana na ključnim riječima*, potom *FastText* model i konačno *RoBERTa* model.

Heuristika zasnovana na ključnim riječima

Kao bazni model za klasifikaciju zadatka korištena je *heuristika zasnovana na ključnim riječima*. Heuristika se zasniva na analizi naslova i opisa zadatka, u potrazi za ključnim riječima *bug* ili *fix*, ne razlikujući velika i mala slova.

Unaprijeđena heuristika zasnovana na ključnim riječima

Za *unaprijeđenu heuristiku zasnovanu na ključnim riječima* potrebno je odrediti ključne riječi koje impliciraju da je zadatak *zahtjev za popravkom pogreške*. Prvo se naslov i opis zadatka transformiraju u isključivo mala slova. Potom se uklone svi interpunkcijski znakovi i na svaku riječ se primjeni *snowball stemmer* (engl. Snowball Stemmer) [240]. Naslov i opis zadatka se pregledavaju riječ po riječ, uzimajući u obzir samo one riječi koje se isključivo sastoje od abecednih znakova i dulje su od dva slova. Za svaku riječ se broji broj njenih pojavljivanja (*tc*) i broj zadataka u kojima se pojavljuje (*dc*). Pri pregledavanju zadataka računa se ukupan broj zadataka označenih kao *zahtjev za popravkom pogreške* (*bCnt*) i ukupan broj preostalih zahtjeva (*oCnt*). Za svaku riječ koja se pojavila u zadatku označenom kao *zahtjev za popravkom pogreške* računa se *metrika važnosti zahtjeva za popravak pogreške* (engl. Bug Importance Score) prikazana jednadžbom 4.1.

$$log(tc/dc) * (dc/bCnt) \tag{4.1}$$

Na sličan način, za svaku riječ koja se pojavljuje u zadatku koji nije označen kao *zahtjev za popravkom pogreške* računa se *metrika važnosti preostalih zahtjeva* (engl. Other Importance

Score) prikazana jednadžbom 4.2.

$$log(tc/dc) * (dc/oCnt) \tag{4.2}$$

Za kraj, za svaku riječ se oduzme *metrika važnosti preostalih zahtjeva* od *metrike važnosti zahtjeva za popravak pogreške*. Potom se riječi sortiraju koristeći dobivenu razliku. Dobivena lista riječi na jednom kraju sadrži riječi koje impliciraju *zahtjeva za popravkom pogreške*, a na drugom kraju riječi koje impliciraju drukčiju vrstu zahtjeva. Slika 4.4 prikazuje *oblak riječi* (engl. Word Cloud) koje najviše impliciraju *zahtjev za popravkom pogreške*, a Slika 4.5 prikazuje *oblak riječi* koje najviše impliciraju drugu vrstu zahtjeva. Opisana procedura određivanja ključnih riječi prikazana je Algoritmom 1.

Slika 4.4: Vizualizacija riječi koje impliciraju zahtjev za popravkom pogreške

Korištenjem ekspertnog znanja odabran je podskup ključnih riječi s vrha dobivene liste. Odabrane riječi se koriste za klasifikaciju zadataka. Odabrano je 8 riječi: *bug*, *error*, *fix*, *issue*, *line*, *out*, *not* i *test*.

Odabran podskup je malen jer bi veći podskup riječi rezultirao s prevelikim brojem kombinacija riječi koje bi bilo potrebno provjeriti u unaprijeđenoj heuristici za klasifikaciju zadataka. Naime, za svaki repozitorij traži se kombinacija najboljih riječi za klasifikaciju zadataka korištenjem naslova i kombinacija najboljih riječi za klasifikaciju zadataka korištenjem opisa zadatka. Dakle, potrebno je provjeriti $(2^8)^2 = 65536$ kombinacija ključnih riječi. Konačno određene kombinacije ključnih riječi i prethodno opisano pred-procesiranje teksta čine unaprijeđenu heuristiku za klasifikaciju zadataka korištenjem ključnih riječi.

Algorithm 1 Procedura određivanja ključnih riječi

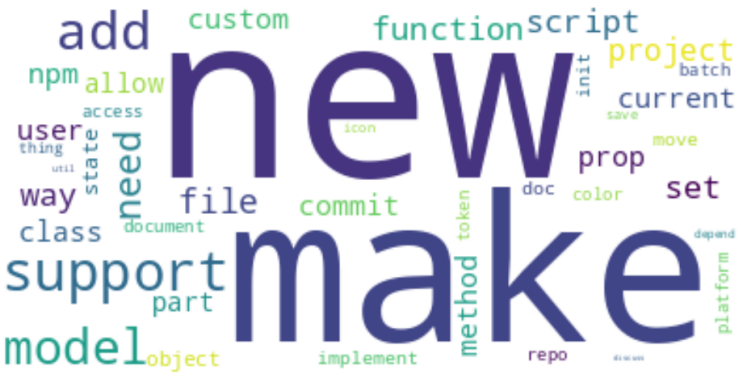
```
Ulazni podaci:
 Zadatci: issues
Rezultati
 Sortirana lista važnih riječi: scores
Procedura
bCnt = 0
oCnt = 0
bugWC = map()
otherWC = map()
words = set()

for issue in issues do
    description = cleanText(issue["description"])
    description = description.lower()
    description = removePunctuations(description)
    description = applySBStemmer(description)
    for unique word in description do
        if len(word) >= 3 and word.isalpha()  then
            words.add(word)
            cnt = description.count(word)
            if issue["type"]=="Bug" then
                bugWC[word].tc += cnt
                bugWC[word].dc++
            else if issue["type"]=="Feature" then
                otherWC[word].tc += cnt
                otherWC[word].dc++
            end if
        end if
    end for
    if issue["type"]=="Bug" then
        bCnt = bCnt + 1
    else
        oCnt = oCnt + 1
    end if
end for

for word in bugWC do
    tc = bugWC[word].tc
    dc = bugWC[word].dc
    bugWC[word].sc = log(tc/dc)*(dc/bCnt)
end for
for word in otherWC do
    tc = otherWC[word].tc
    dc = otherWC[word].dc
    otherWC[word].sc = log(tc/dc)*(dc/oCnt)
end for

scores = list()
for word in words do
    sc = bugWC[word].sc - otherWC[word].sc
    scores.add([sc, word])
end for

return  sort(scores)
```



Slika 4.5: Vizualizacija riječi koje impliciraju drugu vrstu zahtjeva

FastText

Korištena je implementacija *FastText* modela dostupna u sklopu *fasttext* [§] paketa dostupnog u programskom jeziku Python. Prije primjene modela, naslov i opis zadatka spojeni su u jedan tekst. Tekst je pretvoren u mala slova. Uklonjeni su HTML tagovi, svi ne-alfanumerički znakovi i uzastopni razmaci, a svi hiperlinkovi su zamijenjeni s riječi *link*.

Za svaki repozitorij treniran je zasebni FastText model. Označeni IOI od repozitorija korišteni su kao *skup za testiranje*, a označeni IOI od preostalih repozitorija kao *skup za treniranje* i *skup za validaciju*. Praktički, radi se o označavanju zadataka između projekata. Zadatci preostalih repozitorija podijeljeni su 80% u *skup za treniranje* i 20% u *skup za validaciju*. Implementacija modela iz korištenog programskog paketa nudi mogućnost automatske optimizacije hiperparametara. Modelu je dozvoljeno 15 minuta za optimizaciju hiperparametara, trenirajući ga na Intel(R) Core(TM) i7-7700 @ 3.60GHz CPU. Po isteku vremena, model se ponovno trenira koristeći najbolje pronađene hiperparametre.

RoBERTa

Zadnji i najnapredniji model korišten u ovom istraživanju je *RoBERTa* model [236]. Korištena je implementacija RoBERTa modela dostupna u sklopu *huggingface transformer* [¶] paketa dostupnog u programskom jeziku Python. Prije primjene modela, naslov i opis zadatka spojeni su

[§] `https://fasttext.cc/docs/en/python-module.html`
[¶] `https://huggingface.co/docs/transformers/model_doc/roberta`

u jedan tekst. Uklonjeni su HTML tagovi i uzastopni razmaci, a svi hiperlinkovi su zamijenjeni riječi *link*. Pred-procesiran text je tokeniziran koristeći tokenizator RoBERTa modela.

Za svaki repozitorij treniran je posebni RoBERTa model. Označeni IOI od tog repozitorija korišteni su kao *skup za testiranje* modela, a označeni IOI od preostalih repozitorija kao *skup za treniranje* i *skup za validaciju*. Od zadataka preostalih repozitorija 80% je korišteno kao *skup za treniranje* i 20 % kao *skup za validaciju*. Model prima 512 ulaznih tokena. Ulazni tekstovi kraći od toga se nadopunjuju do željene duljine, a oni dulji od toga se skraćuju na 512 tokena. Model je treniran 6 epoha s *malim grupama* (engl. Mini-Batch) od 4 primjera. Za optimizaciju svih parametara modela je korišten AdamW optimizator [241] sa stopom učenja $2 \cdot 10^{-5}$. *Faktor propadanja težina* (engl. Weight Decay Factor) AdamW optimizatora postavljen je na 0.001 za sve parametre modela, osim *pristranosti* (engl. Bias). U slučaju *pristranosti*, *faktor propadanja težina* postavljen je na nula. *Akumulacija gradijenata* (engl. Gradient Accumulation) postavljena je na 4 koraka i koristi se *pamćenje gradijentnih kontrolnih točki* (engl. Gradient Checkpointing) [242] kako bi se smanjila memorija potrebna za treniranje modela. Dodatno, kako bi se smanjilo vrijeme treniranja i dodatno smanjila potrebna memorija, treniranje se vrši korištenjem *pola preciznosti* (*FP16*). Nakon svake epohe, model se vrednuje na setu za validaciju i ako je bolji od prethodnih verzija pohranjuje. Na kraju treniranja učitava se najbolja verzija modela s obzirom na rezultate postignute na skupu za validaciju te koristi kao konačni istrenirani model.

4.4 Predviđanje pogrešaka programske potpore

U ovom poglavlju objašnjena je konstrukcija skupova za predviđanje pogrešaka programske potpore temeljem prikupljenih podataka i temljem rezultata klasifikacije zadataka. Podatkovni skupovi izgrađeni su temeljem podataka prikupljenih iz 7 navedenih programskih repozitorija, ručno označenih zadataka i oznaka zadataka dobivenih korištenjem modela za klasifikaciju zadataka. Modeli korišteni za predviđanje pogrešaka programske potpore su modeli često korišteni u istraživanjima u ovom području. Konkretno, *logistička regresija* (engl. Logistic Regression, skr. LR), *stablo odluke* (engl. Decision Tree, skr. DTC), *naivan Bayes* (engl. Naive Bayes, skr. NB) i *k najbližih susjeda* (engl. K-Nearest Neighbors, skr. KNN).

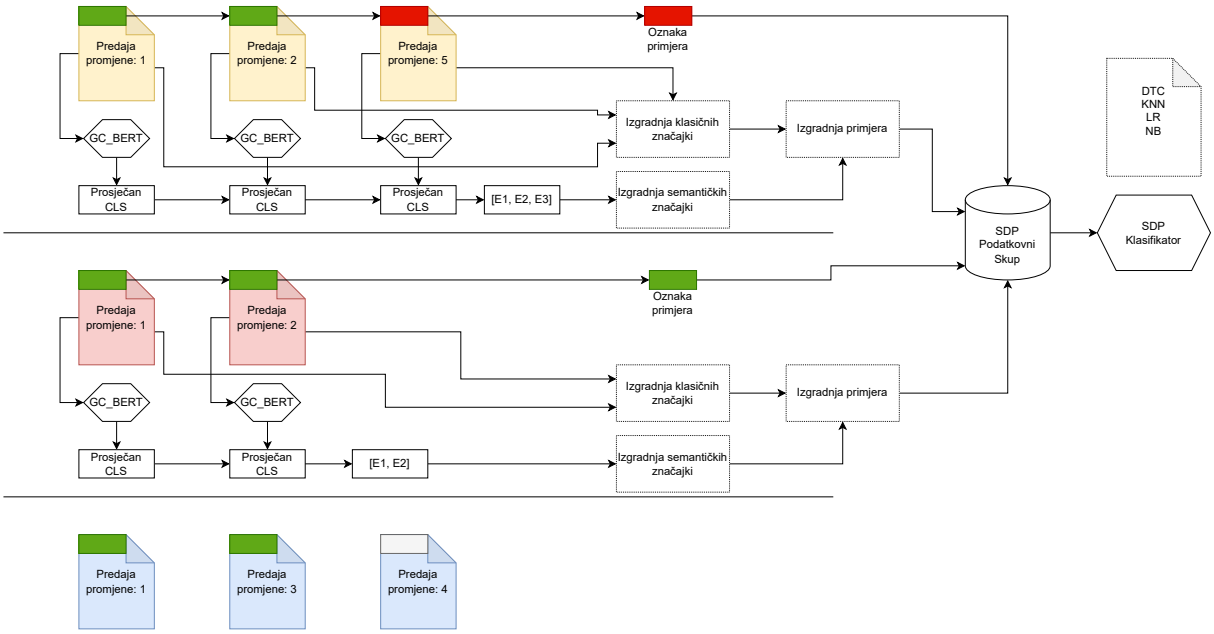
Poglavlje 4.4.1 opisuje postupak izgradnje podatkovnog skupa za predviđanje pogrešaka programske potpore, a poglavlje 4.4.2 opisuje modele korištene za predviđanje pogrešaka programske potpore.

4.4.1 Izgradnja podatkovnog skupa

Zadatci označeni ručnim označavanjem smatraju se *označenim zadacima od interesa* (skr. LIOI). Za svaki *označeni zadatak od interesa* određuju se *datoteke izvornog programskog kôd od interesa* na koje utječu kroz *predaje promjena programskog kôda* povezane s LIOI. Ove datoteke izvornog programskog kôda smatraju se *označenim datotekama od interesa* (skr. LFOI).

Pri izgradnji podatkovnog skupa za predviđanje pogrešaka programske potpore svaka LFOI rezultira jednim primjerom u izgrađenom podatkovnom skupu. Oznaka tog primjera temelji se na zadacima asociranim s LFOI preko COI. Ako je i jedan od ovih zadataka označen kao *zahtjev za popravkom pogreške*, tada se primjer smatra primjerom programskog modula sklonog pogreškama. Ako ni jedan zahtjev nije označen kao *zahtjev za popravak pogreške*, tada se primjer programskog modula smatra nesklon pogreškama. Ova oznaka primjera može se odrediti koristeći ručno pridijeljene oznake zadataka ili one dobivene primjenom klasifikacijskog modela. Programski modul opisan je s tri različita tipa značajki. Koriste se osnove *značajke složenosti kôda*, *semantičke značajke* i *procesne značajke*. Za značajke, ili bolje rečeno značajku, složenosti kôda koristi se prosječan *broj linija kôda* (engl. Lines of Code) svih verzija LFOI. Kao procesne značajke koristi se *broj predaja promjene kôda* koje ažuriraju LFOI, *ukupan broj zadataka povezanih s tim predajama promjene kôda* i *prosječan broj zadataka povezanih s tim predajama promjene kôda*. Semantičke značajke korištene za opis programskih modula su najkompleksnije. Svaka verzija LFOI se preslikava u vektorski prostor korištenjem GraphCodeBERT modela. S obzirom na to da je ulaz modela ograničen na 512 tokena izvorne programske datoteke se kodiraju u dijelovima od 512 tokena s 256 tokena preklapanja između uzastopnih dijelova. Za svaki preslikani dio uzima se vektor CLS tokena koji bi trebao obuhvatiti semantiku cjelokupnog dijela. Potom se računa prosječni CLS token s obzirom na sve preslikane dijelove *verzije datoteke* (engl. File Version) izvornog programskog kôda. Konačne semantičke značajke su *prosječan CLS token svih verzija datoteke izvornog programskog kôda* i *suma svih razlika CLS tokena uzastopnih verzija datoteke izvornog programskog kôda*. Prosjek svih *verzija datoteke* trebao bi opisivati sadržaj datoteke, a suma svih razlika trebala bi opisivati kako se datoteka izvornog programskog kôda mijenjala kroz vrijeme. Opisane značajke složenosti kôda, semantičke značajke i procesne značajke čine skup konačnih značajki koje opisuju programski modul tj. primjer u skupu za predviđanje pogrešaka programske potpore. S obzirom na to da se značajke grade temeljem datoteka izvornog programskog kôda u ovom istraživanju predviđanje se radi na razini datoteke.

Slika 4.6 prikazuje opisani proces, a Tablica 4.8 prikazuje podatke o konstruiranim skupovima za predviđanje pogrešaka programske potpore.



Slika 4.6: Pregled izgradnje skupa za predviđanje pogrešaka programske potpore

Tablica 4.8: Podatkovni skupovi za predviđanje pogrešaka programske potpore izgrađeni za svaki repozitorij

Repozitorij	Broj primjera	Broj primjera sklonih pogreškama	Broj primjera nesklonih pogreškama
react-native	549	114	435
transformers	851	312	539
kubernetes	195	91	104
material-ui	2097	1190	907
node	2188	454	1734
next.js	1484	517	967
youtube-dl	72	24	48

4.4.2 Modeli za predviđanje pogrešaka programske potpore

U sklopu ovog istraživanja, za predviđanje pogrešaka programske potpore korišteni su modeli logistička regresija, stablo odluke, naivan Bayes i k-najbližih susjeda, trenirani pomoću izgrađenih podatkovnih skupova. Nisu korišteni napredniji modeli jer cilj istraživanja nije razvoj naprednijeg modela nego analiza utjecaja klasifikacije zadataka na kvalitetu predviđanja pogrešaka programske potpore.

Treniranje korištenih modela ponovljeno je 30 puta za svaki podatkovni skup kako bi se ublažio utjecaj stohastičke naravi procedure treniranja na rezultate vrednovanja modela. Za svaki repozitorij i svaku metodu klasifikacije zadataka, dobiveni podatkovni skup dijeli se tako da se 80 % skupa koristi za treniranje modela, a 20 % skupa za testiranje modela. Navedena podjela radi se nasumično pri svakom od 30 ponavljanja. Oznake u skupu za treniranje određene

su temeljem rezultata automatske klasifikacije zadataka, a oznake u skupu za testiranje uvijek su određene temeljem rezultata ručnog označavanja zadataka.

4.5 Dobiveni rezultati

U ovom poglavlju prezentirani za rezultati analize utjecaja klasifikacije zadataka na podatkovne skupove i modele za predviđanje pogrešaka programske potpore.

Za svaki od modela za klasifikaciju zadataka i svaki od repozitorija, izložena je postignuta preciznost, odziv i F1 mjera. Rezultati su prikazani u Tablici 4.9.

Potom je prikazan utjecaj klasifikacije zadataka na dobiveni podatkovni skup za predviđanje pogrešaka programske potpore. Utjecaj je opisan matricom zabune klasificiranih zadataka u usporedbi s ručno dodijeljenim oznakama zadataka. Matrica zabune prikazuje četiri vrijednosti TP, TN, FP i FN. TP označava broj primjera koji su trebali biti označeni kao *zahtjev za popravak pogreške* i uistinu jesu. TN označava broj primjera koji su trebali biti označeni kao ostala vrsta zahtjeva i uistinu jesu. FP označava broj primjera koji su označeni kao *zahtjev za popravak pogreške*, a to nisu. Konačno, FN označava broj zahtjeva koji su označeni kao ostala vrsta zahtjeva, a zapravo su *zahtjev za popravak pogreške*. Stvarna oznaka zadatka smatra se oznaka pridijeljena pri ručnom označavanju zadataka. Rezultati su prikazani u Tablici 4.10.

Konačno prezentirani su rezultati predviđanja pogrešaka programske potpore modela logistička regresija, stablo odluke, naivan Bayes i k najbližih susjeda treniranih na izgrađenim skupovima za predviđanje pogrešaka programske potpore. Dodatno, modeli su trenirani na skupovima za predviđanje pogrešaka programske potpore dobivenim korištenjem ručno određenih oznaka zadataka. Za svaki od navedenih modela, svaku od metoda klasifikacije zadataka i svaki repozitorij, prikazana je preciznost, odziv i MCC mjera. Rezultati su prikazani u Tablici 4.11.

Za svaki model klasifikacije zadataka, na razmatranim repozitorijima, navedeni su minimalan, srednji i maksimalan udio lažnih pozitiva i lažnih negativa u izgrađenim podatkovnim skupovima za predviđanje pogrešaka programske potpore. Udio lažnih pozitiva računa se dijeljenjem broja lažnih pozitiva i ukupnog broja primjera. Na sličan način, udio lažnih negativa računa se dijeljenjem broja lažnih negativa i ukupnog broja primjera. Minimalna vrijednost udjela lažnih pozitiva je najmanji udio lažnih pozitiva razmatrajući izgrađene podatkovne skupove za predviđanje pogrešaka programske potpore između različitih repozitorija. Maksimalna vrijednost udjela lažnih pozitiva je najveći udio lažnih pozitiva razmatrajući izgrađene podatkovne skupove za predviđanje pogrešaka programske potpore između različitih repozitorija. Prosječna vrijednost udjela lažnih pozitiva je prosjek svih udjela lažnih pozitiva između različitih repozitorija. Na sličan način računa se minimalna, srednja i maksimalna vrijednost za lažne negative.

U poglavlju 4.5.1 analizirani su rezultati primjene *heuristike zasnovane na ključnim riječima*. U poglavlju 4.5.2 analizirani su rezultati primjene *unaprijeđene heuristike zasnovane na ključnim riječima*. U poglavlju 4.5.3 analizirani su rezultati primjene *FastText* modela. U poglavlju 4.5.4 analizirani su rezultati primjene *RoBERTa* modela. Konačno, poglavlje 4.5.5 sažeti su i dodatno prokomentirani rezultati istraživanja.

4.5.1 Heuristika zasnovana na ključnim riječima

Heuristika zasnovana na ključnim riječima u suštini je ne-nadzirani pristup, tako da je moguće koristiti čitav skup podataka za vrednovanje modela. Ključne riječi traže se u naslovu i opisu zadatka. Iz rezultata prezentiranih u Tablici 4.10 izračunato je da izgrađeni podatkovni skupovi za predviđanje pogrešaka programske potpore sadrže od 13.2093% do 58.3333% lažnih pozitiva i od 0.0000% do 3.4335% lažnih negativa. U prosjeku sadrže 38.4181% lažnih pozitiva i 1.7780% lažnih negativa. Iz navedenih rezultata vrednovanja pristupa zaključeno je da je pristup sklon lažno pozitivnim rezultatima.

4.5.2 Unaprijeđena heuristika zasnovana na ključnim riječima

Za svaki repozitorij primijenjena je *unaprijeđena heuristika zasnovana na ključnim riječima*. Iz rezultata prezentiranih u Tablici 4.10 izračunato je da izgrađeni podatkovni skupovi za predviđanje pogrešaka programske potpore sadrže od 14.9261% do 46.0838% lažnih pozitiva i 0.0000% do 5.1979% lažnih negativa. U prosjeku sadrže 30.7992% lažnih pozitiva i 1.9316% lažnih negativa. Usporedbom rezultata vrednovanja s onima *heuristike zasnovane na ključnim riječima*, vidljiv je pad u količini lažnih pozitiva s 38.4181% na 30.7992%, ali i blagi porast lažnih negativa s 1.7780% na 1.9316% u izgrađenim podatkovnim skupovima za predviđanje pogrešaka.

4.5.3 FastText

Za svaki repozitorij treniran je zasebni *FastText* model za klasifikaciju zadataka. Iz rezultata prezentiranih u Tablici 4.10 izračunato je da izgrađeni podatkovni skupovi za predviđanje pogrešaka programske potpore sadrže od 0.0000% do 19.4444% lažnih pozitiva i od 6.0109% do 18.4615% lažnih negativa. U prosjeku sadrže 7.4104% lažnih pozitiva i 9.7246% lažnih negativa. Usporedbom rezultata vrednovanja s onima prethodnih metoda, vidi se značajan pad broja lažnih pozitiva, ali uz porast broja lažnih negativa. Ipak, kada se gleda ukupna količina šuma u izgrađenim podatkovnim skupovima, ona je smanjena. Kod IKWM metode, u prosjeku 32.7308% podatkovnog skupa za predviđanje pogrešaka programske potpore je bilo je krivo označeno, dok je kod FastText modela u prosjeku 17.1350% podatkovnog skupa za predviđanje pogrešaka programske potpore krivo označeno.

4.5.4 RoBERTa

Za svaki repozitorij treniran je zasebni *RoBERTa* model za klasifikaciju zadataka. Iz rezultata prezentiranih u Tablici 4.10 izračunato je da izgrađeni podatkovni skupovi za predviđanje pogrešaka programske potpore sadrže od 0.5464% do 23.6111% lažnih pozitiva i od 0.5875% do 11.8598% lažnih negativa. U prosjeku sadrže 9.3369% lažnih pozitiva i 5.0272% lažnih negativa.

Kada se ovi rezultati usporede s KWM i IKWM modelima, vidljiv je značajan pad u broju lažnih pozitiva i porast u broju lažnih negativa. Kada se ovi rezultati usporede s onima FastText modela, vidljiv je malen porast u broju lažnih pozitiva, ali značajan pad u broju lažnih negativa. Ukupno gledano izgrađeni podatkovni skupovi za predviđanje pogrešaka programske potpore sadrže u prosjeku 14.3641% krivo označenih primjera.

Analizom rezultata vrednovanja modela za predviđanje pogrešaka programske potpore prijavljenih u Tablici 4.11 moguće je zaključiti da oni bolje rade na podatkovnim skupovima nastalim korištenjem RoBERTa modela za klasifikaciju zadataka. Iz pozitivnih vrijednosti MCC metrike jasno je da modeli rade bolje od nasumičnog pogađanja.

Dobiveni rezultati vrednovanja modela za predviđanje pogrešaka programske potpore dodatno su statistički validirani usporedbom distribucija MCC metrika modela treniranih na podatkovnim skupovima dobivenim klasifikacijom zadataka primjenom RoBERTa modela i primjenom preostalih modela za klasifikaciju zadataka.

Pri usporedbi distribucija rezultata vrednovanja prvo se nad svakom provodi test normalnosti (uz p = 0.05). Ako se ne uspije odbaciti nulta hipoteza za obje distribucije metrika, pretpostavlja se da obje dolaze iz normalnih distribucija i uspoređuju se korištenjem Student t-testa (uz p = 0.05). Ako se barem jedna nulta hipoteza testova normalnosti odbaci, tada se za usporedbu koristi Mann-Whitney U test (uz p = 0.05). Ako se nulta hipoteza Student t-testa, odnosno Mann-Whitney U testa, odbaci, zaključak je da se distribucije razlikuju na statistički značajan način.

Za svaki repozitorij uspoređeni su rezultati vrednovanja modela predviđanja pogrešaka programske potpore razvijenih na podatkovnim skupovima dobivenim primjenom RoBERTa modela s rezultatima vrednovanja istih modela na podatkovnim skupovima dobivenim primjenom preostale 3 metode klasifikacije zadataka. S obzirom na to da se razmatra 4 modela za predviđanje pogrešaka programske potpore i da je prikupljeno 7 repozitorija vrši se $3 * 4 * 7 = 84$ usporedbe rezultata vrednovanja.

Modeli trenirani na skupovima za predviđanje pogrešaka programske potpore dobivenim primjenom RoBERTa modela na klasifikaciju zadataka imali su superiorne rezultate u 65 od 84 usporedbe, a rezultati su bili superiorni i statistički značajni u 55 od 84 usporedbe, odnosno u 65.4761% usporedbi.

Dodatno, iz podataka prikazanih u Tablici 4.11 uspoređeni su rezultati vrednovanja modela

za predviđanje pogrešaka programske potpore treniranih na skupovima dobivenim primjenom RoBERTa modela na klasifikaciju zadataka i rezultati vrednovanja modela za predviđanje pogrešaka programske potpore treniranih na skupovima dobivenim korištenjem ručno dodijeljenih oznaka zadatka.

Ponovno je korištena ista procedura statističke potvrde uz iste p vrijednosti. Od 28 usporedbi, u 17 usporedbi ne postoji nikakva statistički značajna razlika između distribucija rezultata vrednovanja modela za predviđanje pogrešaka programske potpore.

Korištenjem t-SNEa [243] metode vizualizirana je klasifikacija zadataka različitih repozitorija primjenom RoBERTa modela. Za svaki repozitorij, za oznake zadataka korišten je RoBERTa modela s najboljim rezultatima vrednovanja. Crvene točke predstavljaju zadatke koji su označeni *kao zahtjev za popravak pogreške*, a plave točke predstavljaju preostale zahtjeve. Vizualizacija prikazuju kako su modeli naučili razlikovati zadatke označene kao *zahtjev za popravkom pogreške* od preostalih zadataka. Slika 4.7 prikazuje opisanu vizualizaciju.

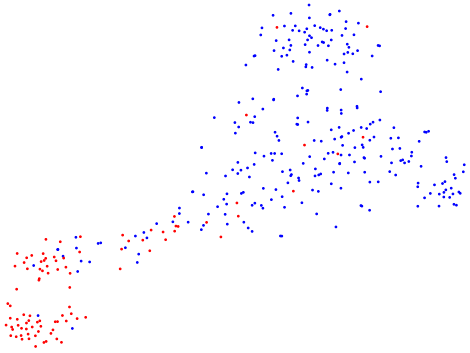
4.5.5 Sažetak rezultata

Svi rezultati istraživanja prikazani su u Tablici 4.9, Tablici 4.10 i Tablici 4.11. Tablica 4.9 prikazuje rezultate vrednovanja modela za klasifikaciju zadataka. Tablica 4.10 prikazuje utjecaj klasifikacije zadataka na kvalitetu podatkovnih skupova za predviđanje pogrešaka programske potpore. Konačno, Tablica 4.11 prikazuje rezultate vrednovanja modela za predviđanje pogrešaka programske potpore treniranih na dobivenim podatkovnim skupovima.

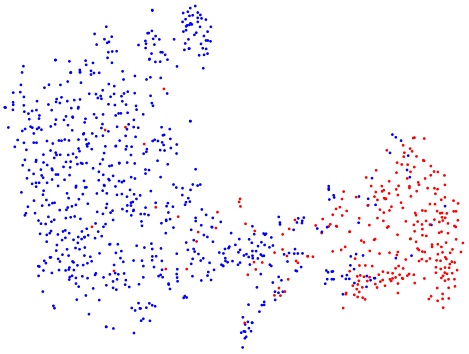
Podatkovni skupovi za predviđanje pogrešaka programske potpore, izgrađeni primjenom KWM metode za klasifikaciju zadataka, imali su u prosjeku 38.4181% lažnih pozitiva i 1.7780% lažnih negativa, odnosno imali su u prosjeku ukupno 40.1961% krivo označenih primjera. Metoda IKWM rezultirala je podatkovnim skupovima koji su u prosjeku imali 30.7992% lažnih pozitiva i 1.9316% lažnih negativa, odnosno imali su u prosjeku ukupno 32.7308% krivo označenih primjera. Primjena modela FastText rezultirala je podatkovnim skupovima koji su u prosjeku imali 7.4104% lažnih pozitiva i 9.7246% lažnih negativa, odnosno imali su u prosjeku ukupno 17.1350% krivo označenih primjera. Konačno, primjena modela RoBERTa rezultirala je podatkovnim skupovima koji su u prosjeku imali 9.3369% lažnih pozitiva i 5.0272% lažnih negativa, odnosno u prosjeku su imali 14.3641% krivo označenih primjera.

Kim et al. [58] postavili su granicu prihvatljive količine šuma u podatkovnom skupu na 20% FP i 20% FN. Po rezultatima njihova istraživanja, nakon navedene granice dolazi do ozbiljne degradacije rada modela za predviđanje pogrešaka programske potpore. Pandey et al. [70] su bili stroži i postavili su granicu prihvatljive količine šuma u podatkovnom skupu na 10% krivo označenih primjera. Po rezultatima njihova istraživanja, nakon navedene granice dolazi do ozbiljne degradacije rada modela.

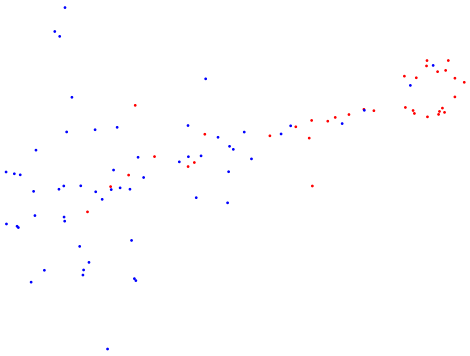
Podatkovni skupovi nastali primjenom KWM metode imaju preveliku količinu šuma s obzi-



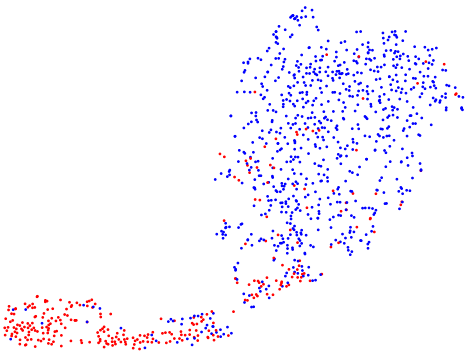
(a) facebook/react-native



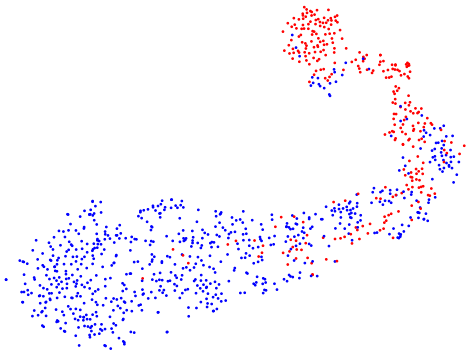
(b) huggingface/transformers



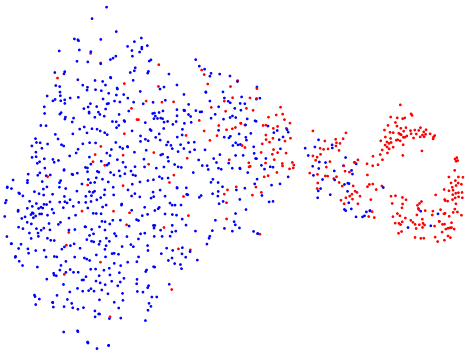
(c) kubernetes/kubernetes



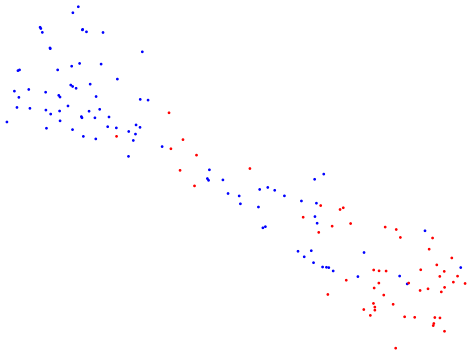
(d) mui/material-ui



(e) nodejs/node



(f) vercel/next.js



(g) ytdl-org/youtube-dl

Slika 4.7: t-SNE vizualizacija RoBERTa vektorizacije

Tablica 4.9: Rezultati klasifikacije zadataka

Repozitorij	Model	Preciznost	Odziv	F1
react-native	KWM	0.5600	0.7527	0.6422
react-native	IKWM	0.5600	0.7527	0.6422
react-native	FastText	0.5200	0.8387	0.6420
react-native	RoBERTa	0.8795	0.7850	**0.8296**
transformers	KWM	0.4080	0.9173	0.5648
transformers	IKWM	0.4188	0.9460	0.5806
transformers	FastText	0.5288	0.9245	0.6728
transformers	RoBERTa	0.7278	0.9425	**0.8213**
kubernetes	KWM	0.4853	0.9167	0.6346
kubernetes	IKWM	0.7143	0.8333	0.7692
kubernetes	FastText	0.6482	0.9722	**0.7778**
kubernetes	RoBERTa	0.8696	0.5556	0.6780
material-ui	KWM	0.4965	0.7095	0.5842
material-ui	IKWM	0.4942	0.7162	0.5848
material-ui	FastText	0.6201	0.6892	0.6528
material-ui	RoBERTa	0.7467	0.7669	**0.7567**

Repozitorij	Model	Preciznost	Odziv	F1
node	KWM	0.4517	0.5748	0.5059
node	IKWM	0.4695	0.8439	0.6033
node	FastText	0.4745	0.8638	0.6125
node	RoBERTa	0.6434	0.8870	**0.7458**
next.js	KWM	0.6559	0.8018	0.7215
next.js	IKWM	0.6559	0.8018	0.7215
next.js	FastText	0.5351	0.8842	0.6667
next.js	RoBERTa	0.8577	0.6250	**0.7231**
youtube-dl	KWM	0.3939	0.9455	0.5562
youtube-dl	IKWM	0.5177	0.8000	0.6286
youtube-dl	FastText	0.5313	0.9273	0.6755
youtube-dl	RoBERTa	0.6076	0.8727	**0.7164**

Tablica 4.10: Utjecaj klasifikacije zadataka na podatkovne skupove za predviđanje pogrešaka programske potpore

Repozitorij	Model	TP	TN	FP	FN
react-native	KWM	100	238	197	14
react-native	IKWM	100	238	197	14
react-native	FastText	112	182	253	2
react-native	RoBERTa	100	432	**3**	**14**
transformers	KWM	308	110	429	4
transformers	IKWM	310	107	432	2
transformers	FastText	309	389	150	3
transformers	RoBERTa	307	470	**69**	**5**
kubernetes	KWM	91	29	75	0
kubernetes	IKWM	48	69	35	43
kubernetes	FastText	91	58	46	0
kubernetes	RoBERTa	73	94	**10**	**18**
material-ui	KWM	1118	630	277	72
material-ui	IKWM	1118	626	281	72
material-ui	FastText	1081	594	313	109
material-ui	RoBERTa	1082	829	**78**	**108**

Repozitorij	Model	TP	TN	FP	FN
node	KWM	427	571	1163	27
node	IKWM	435	516	1218	19
node	FastText	397	774	960	57
node	RoBERTa	418	1320	**414**	**36**
next.js	KWM	467	678	289	50
next.js	IKWM	467	678	289	50
next.js	FastText	484	451	516	33
next.js	RoBERTa	341	888	**79**	**176**
youtube-dl	KWM	23	6	42	1
youtube-dl	IKWM	21	24	24	3
youtube-dl	FastText	22	23	25	2
youtube-dl	RoBERTa	21	31	**17**	**3**

rom na granice postavljene od strane navedenih istraživača. IKWM rezultira boljim podatkovnim skupovima, ali i dalje ne zadovoljavaju postavljene granice. Podatkovni skupovi dobiveni primjenom FastText modela zadovoljavaju granice postavljene od Kim et al. [58], ali ne i one

od Pandey et al. [70] . Podatkovni skupovi dobiveni primjenom RoBERTa modela imaju još manje šuma, ali i dalje ne zadovoljavaju uvjete od Pandey et al. [70] .

Iz dobivenih rezultata vrednovanja, vidljivo je da primjena RoBERTa modela na klasifikaciju zadataka producira statistički značajno bolje modele za predviđanje pogrešaka programske potpore u 55 od 84 slučaja. U 17 od 28 slučaja ne postoji statistički značajna razlika između rezultata vrednovanja modela razvijenih korištenjem podatkovnih skupova dobivenih primjenom RoBERTa modela i podatkovnih skupova dobivenih ručno pridijeljenim oznakama zadataka.

4.6 Rizici valjanosti istraživanja

U ovom poglavlju kratko su navedeni rizici valjanosti istraživanja koji su mogli utjecati na rezultate istraživanja te koraci poduzeti za smanjenje njihova utjecaja.

U poglavlju 4.6.1 opisan je rizik koji proizlazi iz korištenih javno dostupnih podatkovnih skupova i programskih repozitorija. U poglavlju 4.6.2 opisan je rizik koji je posljedica ručnog označavanja zadataka. U poglavlju 4.6.3 opisan je rizik koji je posljedica metode konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore. U poglavlju 4.6.4 opisan je rizik korištenja zadataka isključivo napisanih na engleskom jeziku. Konačno, u poglavlju 4.6.5 opisan je rizik koji je posljedica načina razvoja modela za predviđanje pogrešaka programske potpore.

4.6.1 Podatkovni skupovi i javno dostupni repozitoriji otvorenog programskog kôda

Javno dostupni repozitoriji otvorenog programskog kôda korišteni za prikupljanje podataka za analizu utjecaja klasifikacije zadataka na kvalitetu podatkovnih skupova i rad modela za predviđanje pogrešaka programske potpore. Moguće je da prikupljeni repozitoriji ne predstavljaju reprezentativan podatkovni skup za različite vrste programske potpore i time utječu na dobivene rezultate. Kako bi se smanjio ovaj rizik korišteno je više različitih repozitorija.

4.6.2 Ručno označavanje zadataka

Ručno označavanje zadataka provedeno je u sklopu analize utjecaja klasifikacije zadataka na kvalitetu podatkovnih skupova i rad modela za predviđanje pogrešaka programske potpore. S obzirom na to da autor ove doktorske disertacije ne posjeduje znanje o svim detaljima repozitorija iz kojih su prikupljeni podatci moguće je da su neki od zadataka krivo označeni. Kako bi se smanjio rizik krivog označavanja, uz autora disertacije još dvije osobe su neovisno označile iste zadatke te su konačne oznake zadataka određene uzimajući većinski dodijeljenu oznaku

za svaki od zadataka. Dodatno, podatkovni skupovi napravljeni u sklopu istraživanja javno su dostupni kako bi ih drugi istraživači mogli koristiti, ali i pronaći potencijalne pogreške u njima.

4.6.3 Metoda konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore

Metoda konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore primijenjena u sklopu ovog istraživanja nije jedina moguća. Na primjer, da je izgrađen podatkovni skup za JIT predviđanje pogrešaka programske potpore, metoda izgradnje i izgrađeni podatkovni skup bili bi drukčiji. Isto tako, da su korištene drukčije značajke za opis programskih modula izgrađeni podatkovni skup bio bi drukčiji. Kako bi se smanjio utjecaj ovog rizika u istraživanju su korištene različite vrste značajki. Korištene su klasične značajke složenosti kôda, semantičke značajke i procesne značajke.

4.6.4 Zadatci napisani na engleskom jeziku

Zadatci napisani na engleskom jeziku korišteni su pri analizi utjecaja klasifikacije zadataka na rad predviđanja pogrešaka programske potpore. Moguće je da je ovim unesena jezična pristranost u podatkovni skup. Ipak, dodatne mjere smanjenja ovog rizika nisu poduzete jer je korištenje engleskog jezika standard u svijetu pri razvoju programske potpore.

4.6.5 Razvoju modela za predviđanje pogrešaka programske potpore

Pri razvoju modela za predviđanje pogrešaka programske potpore u većini slučajeva korišteni su hiperparametri modela kako su zadani od strane korištenih programskih paketa. Zbog ograničenja resursa nije provođena iscrpna optimizacija hiperparametara modela, pa je moguće da neki od modela ne postižu najbolje rezultate koje bi mogli. Kako bi se smanjio utjecaj ovog rizika korišten je velik broj različitih modela.

Tablica 4.11: Rezultati vrednovanja modela za predviđanje pogrešaka programske potpore

Repozitorij	Model	Izvor podatkovnog skupa	Preciznost	Odziv	MCC	Repository	Model	Dataset Source	Precision	Recall	MCC
react-native	DTC	Ručno	0.5763	0.6020	*0.4723*	material-ui	KNN	Ručno	0.8524	0.8329	*0.6405*
react-native	DTC	KWM	0.2601	0.6729	0.1271	material-ui	KNN	KWM	0.7446	0.8674	0.5037
react-native	DTC	IKWM	0.2337	0.6455	0.1076	material-ui	KNN	IKWM	0.7318	0.8390	0.4506
react-native	DTC	FastText	0.2627	0.8026	0.1589	material-ui	KNN	FastText	0.7145	0.8508	0.4379
react-native	DTC	RoBERTa	0.6089	0.5028	**0.4460**	material-ui	KNN	RoBERTa	0.8496	0.7920	**0.6091**
react-native	NB	Ručno	0.3499	0.2622	*0.1427*	material-ui	LR	Ručno	0.8337	0.8160	*0.5985*
react-native	NB	KWM	0.2741	0.3686	0.0400	material-ui	LR	KWM	0.7279	0.8890	0.4913
react-native	NB	IKWM	0.2395	0.3547	0.0020	material-ui	LR	IKWM	0.7278	0.8878	0.4836
react-native	NB	FastText	0.4385	0.4022	**0.2603**	material-ui	LR	FastText	0.7041	0.8682	0.4310
react-native	NB	RoBERTa	0.4439	0.2275	0.1973	material-ui	LR	RoBERTa	0.8341	0.8080	**0.6010**
react-native	KNN	Ručno	0.5103	0.2395	*0.2378*	node	DTC	Ručno	0.6252	0.6288	*0.5281*
react-native	KNN	KWM	0.2430	0.5987	0.0791	node	DTC	KWM	0.2569	0.8887	0.1886
react-native	KNN	IKWM	0.2097	0.5802	0.0444	node	DTC	IKWM	0.2502	0.9066	0.1838
react-native	KNN	FastText	0.2602	0.8045	0.1534	node	DTC	FastText	0.2830	0.8312	0.2302
react-native	KNN	RoBERTa	0.6071	0.2038	**0.2614**	node	DTC	RoBERTa	0.4212	0.7515	**0.3985**
react-native	LR	Ručno	0.7321	0.4586	*0.4947*	node	NB	Ručno	0.4350	0.3214	*0.2381*
react-native	LR	KWM	0.2810	0.7119	0.1794	node	NB	KWM	0.3177	0.3287	0.1403
react-native	LR	IKWM	0.2584	0.7079	0.1732	node	NB	IKWM	0.3170	0.3272	0.1340
react-native	LR	FastText	0.2838	0.8582	0.2256	node	NB	FastText	0.2979	0.2493	0.1012
react-native	LR	RoBERTa	0.7464	0.3750	**0.4489**	node	NB	RoBERTa	0.3595	0.2528	**0.1508**
transformers	DTC	Ručno	0.5914	0.5918	*0.3548*	node	KNN	Ručno	0.6198	0.2572	*0.3103*
transformers	DTC	KWM	0.4006	0.9185	0.1517	node	KNN	KWM	0.2275	0.8750	0.0894
transformers	DTC	IKWM	0.3989	0.9095	0.1410	node	KNN	IKWM	0.2356	0.8378	0.1166
transformers	DTC	FastText	0.5520	0.7879	**0.3914**	node	KNN	FastText	0.2585	0.7597	0.1547
transformers	DTC	RoBERTa	0.5630	0.6790	0.3655	node	KNN	RoBERTa	0.3732	0.6294	**0.2940**
transformers	NB	Ručno	0.4367	0.8260	*0.2193*	node	LR	Ručno	0.7836	0.6124	*0.6239*
transformers	NB	KWM	0.5306	0.4047	0.2110	node	LR	KWM	0.2491	0.9577	0.1957
transformers	NB	IKWM	0.5154	0.4223	0.1994	node	LR	IKWM	0.2372	0.9774	0.1771
transformers	NB	FastText	0.4953	0.8196	0.3119	node	LR	FastText	0.2822	0.9315	0.2692
transformers	NB	RoBERTa	0.4189	0.8375	0.1841	node	LR	RoBERTa	0.5044	0.8051	**0.5113**
transformers	KNN	Ručno	0.6478	0.4600	*0.3454*	next.js	DTC	Ručno	0.5088	0.5184	*0.2493*
transformers	KNN	KWM	0.3864	0.9868	0.1470	next.js	DTC	KWM	0.4540	0.6394	0.2128
transformers	KNN	IKWM	0.3960	0.9343	0.1459	next.js	DTC	IKWM	0.4406	0.6523	0.2250
transformers	KNN	FastText	0.5039	0.6977	0.2763	next.js	DTC	FastText	0.3965	0.7487	0.1434
transformers	KNN	RoBERTa	0.5608	0.6121	**0.3298**	next.js	DTC	RoBERTa	0.5312	0.4608	**0.2548**
transformers	LR	Ručno	0.7203	0.5551	*0.4594*	next.js	NB	Ručno	0.5466	0.2492	*0.1799*
transformers	LR	KWM	0.3776	0.9786	0.0848	next.js	NB	KWM	0.5335	0.2471	0.1681
transformers	LR	IKWM	0.3779	0.9831	0.0888	next.js	NB	IKWM	0.5044	0.2417	0.1581
transformers	LR	FastText	0.5718	0.7802	0.4138	next.js	NB	FastText	0.5051	0.2760	0.1601
transformers	LR	RoBERTa	0.6328	0.6723	**0.4422**	next.js	NB	RoBERTa	0.4725	0.2496	0.1241
kubernetes	DTC	Ručno	0.6642	0.6596	*0.3924*	next.js	KNN	Ručno	0.5662	0.3825	*0.2537*
kubernetes	DTC	KWM	0.4780	0.8868	0.1110	next.js	KNN	KWM	0.4692	0.5970	**0.2222**
kubernetes	DTC	IKWM	0.4576	0.4343	-0.0168	next.js	KNN	IKWM	0.4263	0.6143	0.1901
kubernetes	DTC	FastText	0.5404	0.7725	0.1919	next.js	KNN	FastText	0.3885	0.7173	0.1159
kubernetes	DTC	RoBERTa	0.5912	0.5702	**0.2692**	next.js	KNN	RoBERTa	0.5237	0.3870	0.2172
kubernetes	NB	Ručno	0.6274	0.0910	*0.1445*	next.js	LR	Ručno	0.6838	0.4059	*0.3588*
kubernetes	NB	KWM	0.3975	0.5053	-0.1280	next.js	LR	KWM	0.4995	0.6074	0.2675
kubernetes	NB	IKWM	0.4378	0.1175	0.0317	next.js	LR	IKWM	0.4947	0.6313	0.2947
kubernetes	NB	FastText	0.6615	0.1571	**0.1156**	next.js	LR	FastText	0.3880	0.8951	0.1710
kubernetes	NB	RoBERTa	0.4937	0.0640	0.1035	next.js	LR	RoBERTa	0.7242	0.3040	**0.3259**
kubernetes	KNN	Ručno	0.5490	0.5585	*0.1901*	youtube-dl	DTC	Ručno	0.5965	0.5634	*0.3843*
kubernetes	KNN	KWM	0.4566	0.9554	0.0392	youtube-dl	DTC	KWM	0.3191	0.8577	-0.0544
kubernetes	KNN	IKWM	0.4578	0.3423	-0.0121	youtube-dl	DTC	IKWM	0.3911	0.6868	0.0938
kubernetes	KNN	FastText	0.5416	0.7626	0.1913	youtube-dl	DTC	FastText	0.3047	0.6308	-0.0118
kubernetes	KNN	RoBERTa	0.5384	0.4889	0.1631	youtube-dl	DTC	RoBERTa	0.4089	0.6491	**0.1668**
kubernetes	LR	Ručno	0.7220	0.7640	*0.5288*	youtube-dl	NB	Ručno	0.7002	0.6763	*0.5471*
kubernetes	LR	KWM	0.4764	0.9927	0.1851	youtube-dl	NB	KWM	0.6457	0.8594	0.5890
kubernetes	LR	IKWM	0.4368	0.3088	-0.0453	youtube-dl	NB	IKWM	0.7746	0.7180	**0.6254**
kubernetes	LR	FastText	0.5578	0.9057	0.3059	youtube-dl	NB	FastText	0.6760	0.7227	0.5438
kubernetes	LR	RoBERTa	0.7354	0.6480	**0.4755**	youtube-dl	NB	RoBERTa	0.6718	0.6416	0.4774
material-ui	DTC	Ručno	0.8198	0.8288	*0.5889*	youtube-dl	KNN	Ručno	0.4589	0.2176	*0.1651*
material-ui	DTC	KWM	0.7215	0.8460	0.4426	youtube-dl	KNN	KWM	0.3283	0.9889	-0.0218
material-ui	DTC	IKWM	0.7264	0.8539	0.4521	youtube-dl	KNN	IKWM	0.3750	0.6847	0.0695
material-ui	DTC	FastText	0.7041	0.8145	0.3910	youtube-dl	KNN	FastText	0.3095	0.6487	-0.0106
material-ui	DTC	RoBERTa	0.8089	0.7906	**0.5512**	youtube-dl	KNN	RoBERTa	0.4520	0.6784	**0.2366**
material-ui	NB	Ručno	0.7649	0.3652	*0.2460*	youtube-dl	LR	Ručno	0.9200	0.8024	*0.7923*
material-ui	NB	KWM	0.7154	0.2831	0.1631	youtube-dl	LR	KWM	0.3194	0.9514	-0.0924
material-ui	NB	IKWM	0.7135	0.2838	0.1572	youtube-dl	LR	IKWM	0.4951	0.9739	0.4390
material-ui	NB	FastText	0.7074	0.2679	0.1526	youtube-dl	LR	FastText	0.4151	0.9627	0.3420
material-ui	NB	RoBERTa	0.7465	0.3224	**0.2135**	youtube-dl	LR	RoBERTa	0.6104	0.8648	**0.5410**

Poglavlje 5

Predviđanje pogrešaka programske potpore zasnovano na otkrivanju anomalija

Predviđanje pogrešaka programske potpore najčešće se tretira kao problem binarne klasifikacije. Mnogi podatkovni skupovi koji se koriste za razvoj modela pate od problema neuravnoteženosti razreda [28, 29, 30, 31, 32, 33, 34, 35], sadržavajući daleko manje primjera koji su označeni kao skloni pogreškama nego primjera koji su označeni kao neskloni pogreškama [21].

Iz iskustva autora, pogreške programske potpore obično su posljedica nedostatka razumijevanja programskog kôda na kojem se radi, stalno mijenjajućih ili čak međusobno konfliktnih zahtjeva i neočekivanih prekida za vrijeme razvoja programske potpore. Svi navedeni uzroci mogli bi se smatrati anomalijama u normalnom procesu razvoja programske potpore. Tretiranje pogrešaka programske potpore poput anomalija daje objašnjenje za problem neuravnoteženosti razreda, gdje je daleko više primjera programskih modula nesklonih pogreškama nego onih koji su skloni pogreškama. Anomalije su po definiciji rijetke pojave koje odstupaju od onog što je normalno očekivano. Anomalije se mogu podijeliti u tri tipa: *točkaste anomalije*, tj. pojedinačne primjere koji značajno odstupaju od ostatka populacije, *kontekstualne anomalije*, tj. primjere koji su anomalije samo u nekom podatkovnom kontekstu (npr. temperatura 0 stupnjeva Celzijusa nije anomalija ako se izmjeri u Zagrebu u Siječnju, ali jeste anomalija ako se izmjeri u Splitu u Srpnju) i *kolektivne anomalije*, tj. grupe primjera koji su međusobno slični, ali toliko odstupaju od ostatka populacije da se ne mogu smatrati normalnim pojavama. Tretiranjem pogrešaka programske potpore poput anomalija zaobilazi se problem neuravnoteženosti razreda.

U sklopu ove disertacije razvijen je model za predviđanje pogrešaka programske potpore tretirajući ih kao anomalije. Pri istraživanju za ovu doktorsku disertaciju pronađen je samo jedan prethodni rad koji je problemu predviđanja pogrešaka programske potpore pristupio tre-

tirajući ih kao anomalije [169], a rezultati rada autora ove disertacije, objavljeni u članku [37], inspirirali su druge istraživače da razmotre ovakav pristup problemu predviđanja pogrešaka programske potpore [172, 173, 174, 175].

Detaljnije govoreći, u sklopu ove disertacije napravljen je novi model koji problemu predviđanja pogrešaka programske potpore unutar projekta pristupa tretirajući ga kao problem otkrivanja anomalija. Razvijeni model naziva se REPD što je skraćenica za *vjerojatnosnu distribuciju grešaka rekonstrukcije* (engl. Reconstruction Error Probability Distribution, skr. REPD). REPD je moguće primijeniti za otkrivanje točkastih i kolektivnih anomalija, ali ne i kontekstnih anomalija jer model nema pojam konteksta. Razvijeni model je vrednovan koristeći pet podatkovnih skupova s klasičnim značajkama uspoređujući ga s pet modela: *logistička regresija* (engl. Logistic Regression, skr. LR), *stablo odluke* (engl. Decision Tree, skr. DTC), *naivan Bayes* (engl. Naive Bayes, skr. NB), *k najbližih susjeda* (engl. K-Nearest Neighbors, skr. KNN) i Hybrid SMOTE-Enseble (skr. HSME). Prva četiri modela su standardni modeli za klasifikaciju dok je Hybrid SMOTE-Ensemble [195] jedan od trenutno najboljih (engl. State-of-the-art) modela. Navedeni modeli su odabrani zbog njihovog čestog korištenja u području predviđanja pogrešaka programske potpore [30, 56, 89, 125, 126, 128, 244, 245, 246] i lako dostupnih implementacija. Dodatno, predloženi REPD model je vrednovan na 24 podatkovna skupa zasnovana na semantičkim značajkama izgrađenim primjenom autoenkoder modela na leksičke jedinke izvornog programskog kôda i uspoređen s prethodno navedenim modelima. Za potrebe vrednovanja korištena je F1 mjera. Rezultati vrednovanja su statistički validirani pokazujući superioran rad predloženog modela, poboljšavajući F1 mjeru i do 7.12%. Konačno, analizirana je otpornost REPD modela na problem neuravnoteženosti razreda korištenjem metoda pod-uzorkovanja i ponovnog uzorkovanja.

U nastavku ovog poglavlja detaljno je opisan razvoj i vrednovanje REPD modela za predviđanje pogrešaka programske potpore koji se zasniva na ideji otkrivanja anomalija.

U poglavlju 5.1 opisan je način izgradnje i korištenja predloženog REPD modela. U poglavlju 5.2 opisani su podatkovni skupovi korišteni u istraživanju. U poglavlju 5.3 korišten je postupak vrednovanja predloženog modela. Konačno, u poglavlju 5.4 predstavljeni su rizici valjanosti provedenog istraživanja i opisan je način smanjenja istih.

5.1 Predloženi model: REPD

U ovom poglavlju opisan je predloženi REPD model za predviđanje pogrešaka programske potpore. Model se zasniva na arhitekturi autoenkoder neuronske mreže koja se pokazala poprilično uspješna u otkrivanju anomalija [247].

Autoenkoder je vrsta potpuno povezane unaprijedne neuronske mreže kod koje su ulazni sloj i izlazni sloj iste veličine. Mreža uči, na svom izlazu, rekonstruirati ulazni primjer što

je bolje moguće, a u skrivenim slojevima radi podatkovnu kompresiju. Kompresija se postiže definiranjem skrivenog sloja koji je manji od ulaznog tj. izlaznog sloja. Tako je mreža prisiljena birati koje informacije propagira prema izlazu kako bi postigla što bolju rekonstrukciju. Osnovna pretpostavka REPD modela je da će autoenkoder naučen rekonstruirati primjere programskih modula nesklonih pogreškama imati različite distribucije grešaka rekonstrukcije programskih modula sklonih pogreškama i programskih modula nesklonih pogreškama. Kako bi se kvantificirala razlika u distribucijama grešaka rekonstrukcije, za svaku od distribucija procjenjuje se funkcija vjerojatnosti gustoće. Greška rekonstrukcije neviđenog primjera imat će vjerojatnost pripadanja distribuciji grešaka programskih modula nesklonih pogreškama tj. distribuciji grešaka programskih modula sklonih pogreškama. Temeljem tih vjerojatnosti primjer se klasificira kao nesklon pogreškama ili kao sklon pogreškama ovisno o tome kojoj distribuciji je veća vjerojatnost pripadanja. Dakle, izlaz autoenkoder mreže nisu nove značajke već samo međurezultat koji se koristi za izračun greške rekonstrukcije, a primjer se klasificira temeljem vjerojatnosti pripadnosti greške rekonstrukcije različitim distribucijama.

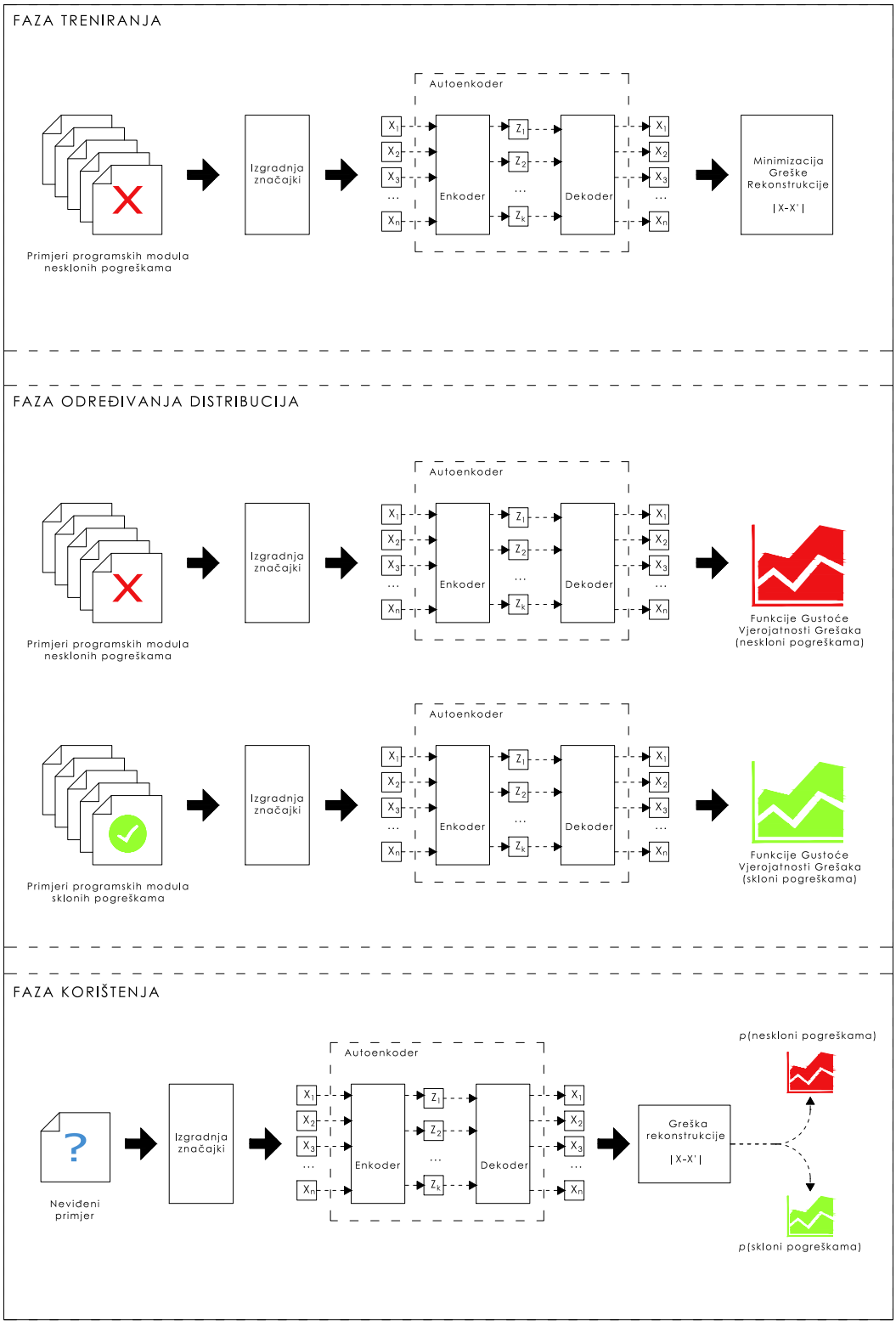
Slika 5.1 prikazuje pregled korištenja opisanog modela. Korištenje je podijeljeno u tri faze: *fazu treniranja*, *fazu određivanja distribucija* i *fazu korištenja* razvijenog modela.

Za vrijeme faze treniranja autoenkoder se uči rekonstruirati primjere programskih modula nesklonih pogreškama. Za vrijeme faze određivanja distribucija, određuju se distribucije koje najbolje opisuju dobivene skupove grešaka rekonstrukcija programskih modula nesklonih pogreškama i programskih modula sklonih pogreškama. Konačno, u fazi korištenja dobiveni model se primjenjuje u svrhu predviđanja pogrešaka programske potpore. Svaka od navedenih faza je detaljnije opisana u idućim poglavljima.

U poglavlju 5.1.1 detaljno je opisana faza treniranja modela. U poglavlju 5.1.2 detaljno je opisana faza određivanja distribucija. Konačno, u poglavlju 5.1.3 detaljno je opisana faza korištenja modela.

5.1.1 Faza treniranja

Prvo je potrebno osigurati značajke koje opisuju programske module za koje se vrši predviđanje. REPD zahtjeva numeričke značajke bez obzira opisivale one složenost kôda ili proces izgradnje programske potpore. Primjeri za učenje se dijele na one koji opisuju programske module nesklone pogreškama i one koji opisuju programske module sklone pogreškama. Potom se trenira autoenkoder mreža da što bolje rekonstruira primjere koji opisuju programske module nesklone pogreškama. Autoenkoder korišten u sklopu ovog istraživanja se sastoji od 3 sloja: ulaznog, skrivenog i izlaznog sloja. Pri tom su težine ulaznog i izlaznog sloja jednake. Ulazni i izlazni sloj iste su veličine kao i broj značajki koje se korite za opis programskih modula, a skriveni sloj je pola veličine tih slojeva. Autoenkoder prvo kodira ulazni primjer iz ulaznog u skriveni sloj i potom iz skrivenog u izlazni sloj. Na ovaj način se gradi rekonstrukcija ulaznih podataka



Slika 5.1: Faze REPD modela

za svaki primjer. Moguće je koristiti i drugačiji autoenkoder. Cilj je istrenirati autoenkoder da rekonstrukcije budu što sličnije ulaznim podacima.

Detaljnije govoreći, autoenkoder se sastoji od *enkodera* i *dekodera*. Enkoder radi prijelaz $\phi : \mathscr{X} \to \mathscr{F}$ iz ulaznog podatkovnog prostora $\mathscr{X}$ u podatkovni prostor skrivenih reprezentacija. Dekoder radi prijelaz $\mathscr{F}$ i $\psi : \mathscr{F} \to \mathscr{X}$ iz podatkovnog prostora skrivenih reprezentacija u izvorni podatkovni prostor. Enkoder i dekoder uče funkcije rekonstrukcije na način prikazan jednadžbom 5.1.

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi(\phi(X))\|^2. \tag{5.1}$$

Gdje su ϕ i ψ definirani jednadžbom 5.2 i jednadžbom 5.3.

$$\phi(\mathbf{x}) = \mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{5.2}$$

$$\psi(\mathbf{z}) = \mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}'), \tag{5.3}$$

$\mathbf{W}, \mathbf{W}'$ predstavljaju matrice težina, a $\mathbf{b}, \mathbf{b}'$ pristranosti slojeva. Učenjem ovih parametara minimizira se kvadratna pogreška prikazana jednadžbom 5.4.

$$\begin{aligned} \mathscr{L}(\mathbf{x}, \mathbf{x}') &= \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')\|^2 \\ &= \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2. \end{aligned} \tag{5.4}$$

Dakle, računa se korijen srednje kvadratne pogreške (engl. Root Mean Square Error, skr. RMSE) između izvornog ulaza i njegove rekonstrukcije. Parametri autoenkoder inicijaliziraju se primjenom Xavier inicijalizacije [248] i uče primjenom gradijentnog spusta koristeći optimizator Adam [249]. Po završetku treniranja autoenkodera, koristi se za izračun grešaka rekonstrukcije programskih modula nesklonih pogreškama i izračun grešaka rekonstrukcije programskih modula sklonih pogreškama.

Kako je prethodno rečeno, mogle su se koristiti drugačije vrste autoenkodera ili autoenkoderi drugačije arhitekture. Navedeni autoenkoder i njegova arhitektura odabrani su zbog svoje jednostavnosti i brzine.

5.1.2 Faza određivanja distribucija

U fazi određivanja distribucija, određuju se funkcije gustoće vjerojatnosti grešaka rekonstrukcija za programske module sklone pogreškama i programske module nesklone pogreškama. Ista procedura se primjenjuje na greške rekonstrukcija programskih modula nesklonih pogreškama i greške rekonstrukcija programskih modula sklonih pogreškama. Pretpostavka je da greške

dolaze iz jedne od idućih distribucija: normalne [250], lognormal [251], eksponencijalna Weibullova kontinuirana slučajna varijabla [252], pareto [253], gamma [254] ili beta [254]. Za svaku distribuciju određuju se parametri s kojima najbolje opisuje skup grešaka rekonstrukcije. Potom se koristi Kolmogorov-Smirnov test [204] kako bi se odredilo koja od distribucija najbolje opisuje dobiveni skup grešaka rekonstrukcija.

Detaljnije govoreći, za svaku od navedenih distribucija parametri se određuju primjenom procjene najveće vjerojatnosti (engl. Maximum Likelihood Estimation, skr. MLE) uz pomoć programskog paketa *SciPy* * [255] dostupnog u programskom jeziku Python. MLE određuje parametre distribucije tako da povećava vjerojatnost da je dani skup uzorkovan iz razmatrane distribucije. Nakon određivanja parametara svake distribucije, KS test mjeri koliko dobro distribucija opisuje dobiveni skup. KS testom računa se empirijska distribucijska funkcija prikazana jednadžbom 5.5, gdje $X_1, \ldots, X_n$ označava uzorkovane primjere (greške rekonstrukcije) te je $I_{[-\infty,x]}(X_i) = 1$ ako je $X_i \leq x$, a u suprotnom 0.

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[-\infty,x]}(X_i), \tag{5.5}$$

Empirijska distribucija uspoređuje se s distribucijom F koristeći Kolmogorov-Smirnov statistiku (gdje manja vrijednost znači bolji rezultat) prikazanu jednadžbom 5.6.

$$D_n = \sup_x |F_n(x) - F(x)|. \tag{5.6}$$

Vjerojatnosna distribucija s najmanjom vrijednošću KS statistike D_n koristi se za opis grešaka rekonstrukcije.

Nakon primjene navedene procedure na greške programskih modula nesklonih pogreškama i greške programskih modula sklonih pogreškama dobivaju se dvije distribucije svaka od kojih opisuje jedan od navedenih skupova grešaka rekonstrukcije. Algoritam 2 prikazuje opisanu proceduru određivanja distribucija.

5.1.3 Faza korištenja

U fazi korištenja dobiveni model se koristi za predviđanje pogrešaka programske potpore do sada neviđenih programskih modula. Na neviđeni primjer prvo se primjeni istrenirati autoenkoder kako bi se dobila greška rekonstrukcije. Potom se gleda vjerojatnost dobivanja takove greške rekonstrukcije iz određene distribucije grešaka rekonstrukcija programskih modula nesklonih pogreškama i iz određene distribucije grešaka rekonstrukcija programskih modula sklonih pogreškama. Primjer je klasificiran kao primjer programskog modula nesklonog pogreškama ako je vjerojatnost dolaska njegove greške rekonstrukcije iz distribucije grešaka rekonstruk-

*`https://scipy.org/`

Algorithm 2 REPD model: određivanje distribucije

Ulazni podaci
Primjeri: X_defective, X_non_defective
Oznake primjera: y
Data coding model: autoencoder

Rezultati
Distribucija grešaka programskih modula nesklonih pogreškama: non_def_dist
Distribucija grešaka programskih modula sklonih pogreškama: def_dist

Procedura

```
non_def_errors =
        autoencoder.reconstruction_error(X_non_defective)
def_errors =
        autoencoder.reconstruction_error(X_defective)
non_def_distributions = ∅
def_distributions = ∅
for distribution ← potential_distributions do
   non_def_distributions = non_def_distributions ∪
           distribution.new_instance().fit(non_def_errors)
   def_distributions = def_distributions ∪
           distribution.new_instance().fit(def_errors)
end for
non_def_dist = KS_test_get_best(non_def_distributions)
def_dist = KS_test_get_best(def_distributions)
```

cija programskih modula nesklonih pogreškama veća nego vjerojatnost dolaženja iz distribucije grešaka rekonstrukcija programskih modula sklonih pogreškama što prikazuje jednadžba 5.7

$$PDF_{defective}(error) > PDF_{non-defective}(error). \tag{5.7}$$

Opisana procedura primjene modela prikazan je Algoritmom 3.

Algorithm 3 REPD model: primjena

Ulazni podaci
Neoznačeni primjer: X

Rezultati
Oznaka primjera: non_defective *ili* defective

Procedure
p_nd = non_def_dist.probability_density_function(X)
p_d = def_dist.probability_density_function(X)
if p_nd > p_d **then**
  return non_defective
else
  return defective
end if

5.2 Podatkovni skupovi korišteni za vrednovanje modela

U ovom poglavlju opisani su podatkovni skupovi korišteni za vrednovanje modela. Korišteno je 5 podatkovnih skupova s klasičnim značajkama i 24 izgrađena podatkovna skupa zasnovana na semantičkim značajkama.

U poglavlju 5.2.1 opisani su podatkovni skupovi zasnovani na klasičnim značajkama, a u poglavlju 5.2.2 opisani su podatkovni skupovi zasnovani na semantičkim značajkama.

5.2.1 Podatkovni skupovi s klasičnim značajkama

Pet podatkovnih skupova zasnovanih na klasičnim značajkama preuzeti su iz PROMISE repozitorija [72] i opisuju programsku potporu NASA proizvoda. Konkretno korišteni su podatkovni skupovi CM1, JM1, KC1, KC2 i PC1. Primjeri programskih modula u navedenim skupovima opisani su s 21 značajkom. Za opis programskih modula korištene su Halstead i McCabe značajke. Više informacija o značajkama navedenih podatkovnih skupova dostupno je u Dodatku 6.

Osnovne informacije o preuzetim podatkovnim skupovima prikazane su u Tablici 5.1.

Tablica 5.1: Informacije o podatkovnim skupovima s klasičnim značajkama

Podatkovni skup	Programski jezik	Broj primjera	Broj programskih modula sklonih pogreškama	Udio programskih modula sklonih pogreškama
CM1	C	498	49	9.84%
JM1	C	10885	2106	19.35 %
KC1	C++	2109	326	15.46%
KC2	C++	522	105	20.11 %
PC1	C	1109	77	6.94 %

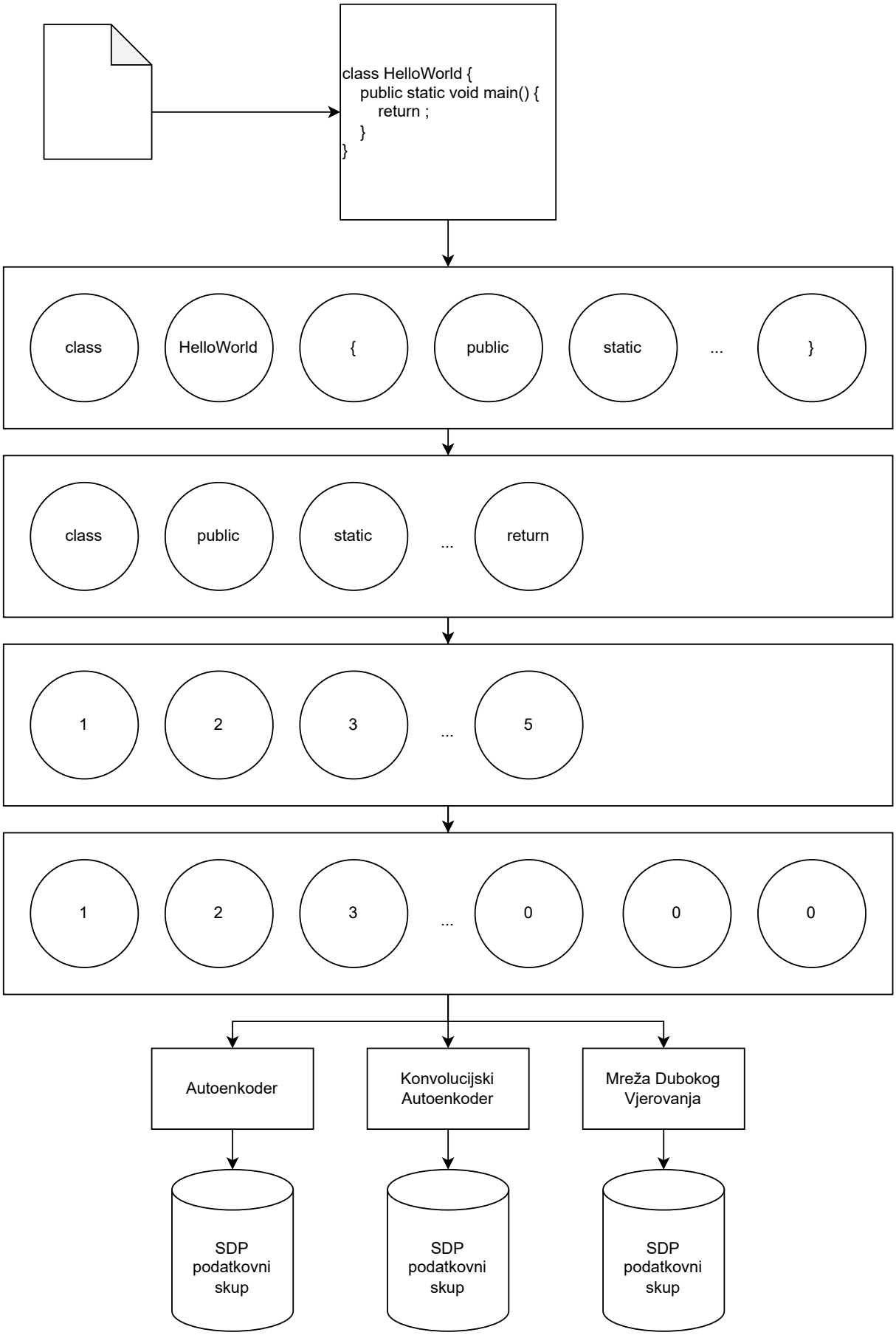
5.2.2 Podatkovni skupovi s semantičkim značajkama

Za potrebe dodatnog vrednovanja modela, preuzeti su podatkovni skupovi: ant *v*1.5, ant *v*1.6, camel *v*1.2, camel *v*1.4, log4j *v*1.1, log4j *v*1.2, poi *v*2.0, i poi *v*2.5 [256]. Navedeni podatkovni skupovi, u preuzetom izdanju, zasnovani su na klasičnim značajkama. Za svaki skup, tj. svaku specifičnu verziju svakog podatkovnog skupa preuzet je i izvorni programski kôd temeljem kojeg je podatkovni skup izgrađen. Svi navedeni podatkovni skupovi izgrađeni su temeljem programske potpore napisane u programskom jeziku Java. S obzirom na to da su PROMISE podatkovni skupovi zasnovani na programskoj potpori napisanoj u programskom jeziku C ili C++ ovim se doprinosi širini vrednovanja predloženog modela.

Koristeći preuzete podatke izgrađeni su semantički podatkovni skupovi primjenom iduće procedure. Za svaki programski razred u preuzetim podatkovnim skupovima provedeno je parsiranje izvornog programskog kôda. Kôd je leksičkom analizom podijeljen u niz leksičkih jedinki. Potom su zadržane samo leksičke jedinke koje predstavljaju ključne riječi. Potom je svaka leksička jedinka zamijenjena cijelim brojem koji je određen za predstavljanje ključne riječi pojedine jedinke. Kako bi nizovi svih leksičkih jedinki bili jednaki, kraći nizovi su prošireni nulama. Na konstruirane vektore primijenjene su tri različite vrste neuronskih mreža kako bi se izgradili različiti podatkovni skupovi zasnovani na semantičkim značajkama. Korištene su *mreža dubokog vjerovanja*, *duboki autoenkoder* i *konvolucijski autoenkoder*. Primjenom navedenih neuronskih mreža izvorni vektori su sažeti na vektore duljine 100. Slika 5.2 prikazuje opisani proces izgradnje podatkovnih skupova zasnovanih na semantičkim značajkama.

S obzirom na 8 početnih podatkovnih skupova i 3 korištene neuronske mreže, ovom procedurom izgrađeno je $8 * 3 = 24$ semantička podatkovna skupa. S obzirom na to da su navedene mreže stohastički modeli, tj. njihovo treniranje neće uvijek rezultirati istim konačnim modelom, navedeni postupak zapravo je ponovljen 30 puta za svaku od mreža i svaki podatkovni skup čime je izgrađeno $24 * 30 = 720$ podatkovnih skupova. Na ovaj način nastoji se minimizirati efekt potencijalno loše pojedinačne optimizacije na rezultate vrednovanja modela. Pri prezentaciji rezultata vrednovanja prikazani su prosječni rezultati za svaku grupu ponavljanja.

Informacije o izgrađenim podatkovnim skupovima prikazane su u Tablici 5.2.



Slika 5.2: Proces izgradnje podatkovnih skupova sa semantičkim značajkama

Tablica 5.2: Informacije o podatkovnim skupovima zasnovanim na semantičkim značajkama

Podatkovni skup	Broj primjera	Broj programskih modula sklonih pogreškama	Udio programskih modula sklonih pogreškama
ant *v*1.5	292	32	10.96%
ant *v*1.6	350	92	26.29%
camel *v*1.2	595	216	36.30%
camel *v*1.4	846	145	17.14%
log4j *v*1.1	104	37	35.58%
log4j *v*1.2	194	186	95.88%
poi *v*2.0	309	37	11.97%
poi *v*2.5	380	248	62.26%

5.3 Vrednovanje modela

U ovom poglavlju opisan je postupak i rezultati vrednovanja REPD modela. Model je vrednovan u tri scenarija. U prvom slučaju vrednovan je na podatkovnim skupovima zasnovanim na klasičnim značajkama, potom je vrednovan na podatkovnim skupovima zasnovanim na semantičkim značajkama i konačno, napravljena je analiza otpornosti modela na problem neuravnoteženih razreda.

U poglavlju 5.3.1 opisan je postupak i rezultati vrednovanja modela na podatkovnim skupovima zasnovanim na klasičnim značajkama. U poglavlju 5.3.2 opisan je postupak i rezultati vrednovanja modela na podatkovnim skupovima zasnovanim na semantičkim značajkama. Konačno, u poglavlju 5.3.3 opisan je postupak analize otpornosti modela na problem neuravnoteženih razreda i postignuti rezultati.

5.3.1 Rad modela na podatkovnim skupovima zasnovanim na klasičnim značajkama

U ovom poglavlju opisan je postupak vrednovanja modela i postignuti rezultati na podatkovnim skupovima zasnovanim na klasičnim značajkama. Rezultati vrednovanja modela uspoređeni su s rezultatima vrednovanja 5 alternativnih modela: *logistička regresija* (engl. Logistic Regression, skr. LR), *stablo odluke* (engl. Decision Tree, skr. DTC), *naivan Bayes* (engl. Naive Bayes, skr. NB), *k najbližih susjeda* (engl. K-Nearest Neighbors, skr. KNN) i Hybrid SMOTE-Enseble (skr. HSME). Predloženi model REPD i svaki od alternativnih modela trenirani su 100 puta. Kao glava metrika kvalitete rada modela korištena je F1 mjera. Za podatkovne skupove na kojima REPD postiže najveću prosječnu vrijednost F1 mjere provedena je statistička analiza usporedbe s alternativnim modelima.

Skup rezultata vrednovanja svakog od modela tretira se kao skup uzoraka nasumične varijable. Nad svakim skupom prvo se provodi test normalnosti [205] tj. dali je skup uzorkovan iz normalne distribucije. Korištena p vrijednost je 0.001 što je dosta niska granica čime se omogućuje dosta liberalno određivanje uzorka normalne distribucije. Ako se odredi da oba skupa

pripadaju normalnim distribucijama tada se uspoređuju primjenom t-testa [206] uz p vrijednost 0.05. Konačno, ako se t-testom odredi razlika između srednjih vrijednosti distribucija tada se ta razlika kvantificira korištenjem Choenove d vrijednosti [208]. Za d vrijednost koja mjeri veličinu efekta određene su iduće granice:

- *trivijalan* ako $d\ vrijednost < 0.2$,
- *malen* ako $0.2 \leq d\ vrijednost < 0.5$,
- *umjeren* ako $0.5 \leq d\ vrijednost < 0.8$,
- *velik* ako $0.8 \leq d\ vrijednost$.

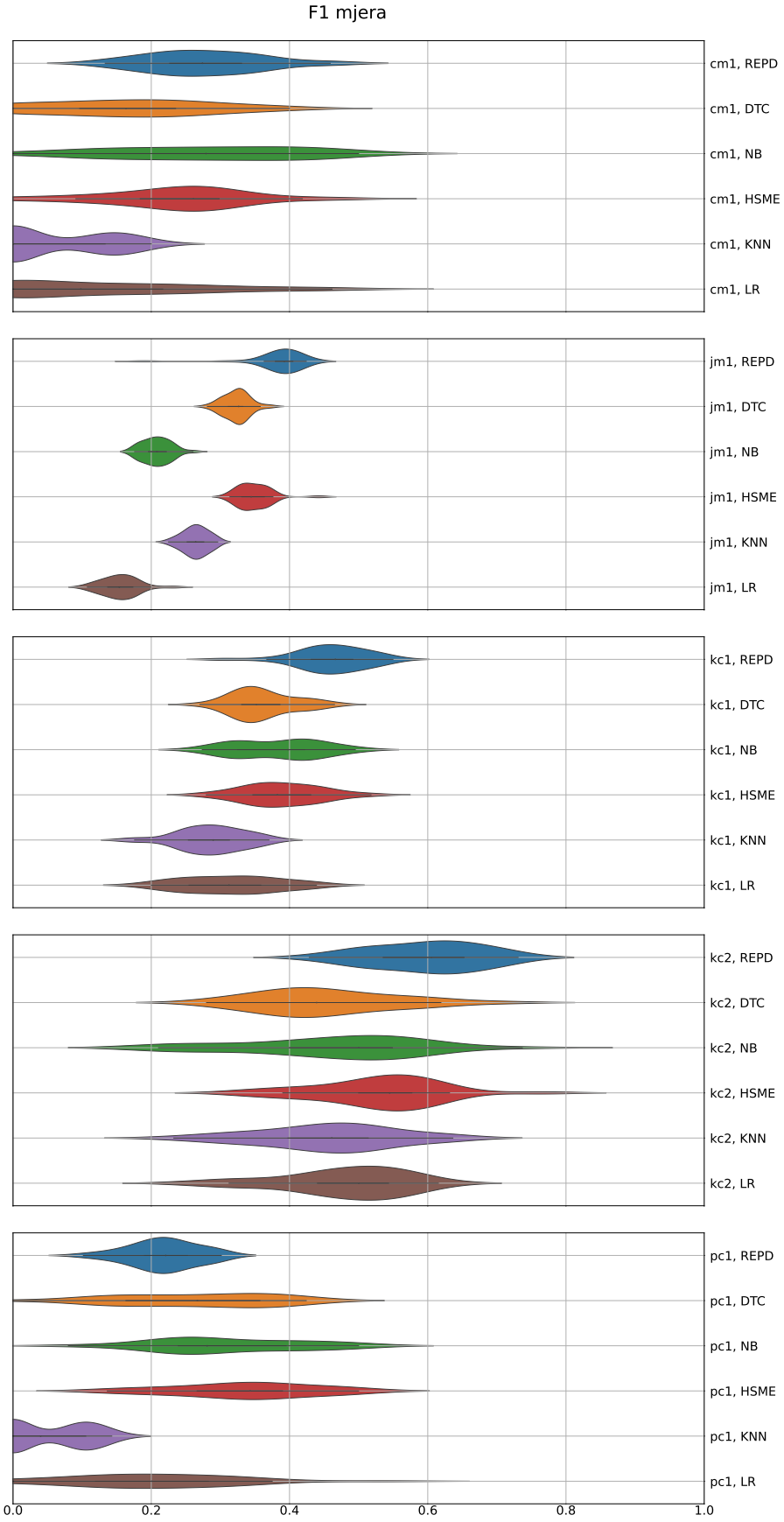
Slika 5.3 prikazuje distribucije F1 mjera postignutih u sklopu vrednovanja modela na podatkovnim skupovima zasnovanim na klasičnim značajkama.

Pregledom Slike 5.3 lako je utvrditi kako predloženi REPD model ima najbolju tj. najveću vrijednost F1 mjere na 4 od 5 podatkovnih skupova. Na ovim podatkovnim skupovima provedena je opisana statistička potvrda rezultata vrednovanja. Rezultati statističke potvrde izneseni su u Tablici 5.3 za podatkovni skup CM1, Tablici 5.4 za podatkovni skup JM1, Tablici 5.5 za podatkovni skup KC1 i Tablici 5.6 za podatkovni skup KC2.

U navedenim tablicama, prvi stupac (*Model*) navodi model s kojim se uspoređuje predloženi model REPD. Drugi stupac (*Normalna distribucija*) navodi dali je distribucija rezultata vrednovanja modela normalna distribucija. Treći stupac (*p vrijednost testa normalnosti*) navodi dobiveni p vrijednost primjenom testa normalnosti. Ako distribucija rezultata vrednovanja modela nije normalna daljnji stupci su prazni jer se ne provodi daljnja statistička usporedba rezultata. Četvrti stupac (*Različite distribucije*) navodi jeli određena razlika srednjih vrijednosti distribucija rezultata vrednovanja modela primjenom t-testa. Peti stupac (*p vrijednost t-testa*) navodi dobivenu p vrijednost t-testa. Šesti stupac (*d vrijednost*) navodi Cohenovu d vrijednost i konačno zadnji stupac (*interpretacija efekta*) navodi interpretaciju dobivene d vrijednosti, tj. koliko je razlika rada modela statistički bitna.

Iz izloženih rezultata vidljivo je da REPD u većini slučajeva postiže najbolje rezultate, pri čemu je razlika u radu modela od statistički velike važnosti. Iznos F1 mjere REPD modela veći je od iznosa F1 mjere ostalih modela za 0.16% do 7.12% ovisno o podatkovnom skupu. Većim F1 vrijednostima prvenstveno doprinosi veća vrijednost odziva modela. REPD na navedenim podatkovnim skupovima postiže odziv od 0.4838 do 0.6971. Praktično gledano, REPD je vrlo efikasan u otkrivanju pogrešaka programske potpore i u većini slučajeva označava više od pola datoteka koje ih sadrže. Kod podatkovnog skupa PC1 na kojem REPD ne radi dobro, vjerojatno nije zadovoljena osnovna pretpostavka modela, tj. distribucije programskih modula nesklonih pogreškama i programskih modula sklonih pogreškama nemaju značajnih razlika.

Slika 5.4 prikazuje distribucije odziva, a Slika 5.5 prikazuje distribucije preciznosti modela dobivene vrednovanjem modela na podatkovnim skupovima zasnovanim na klasičnim značajkama. Preglednom navedenih slika vidljiva je sklonost modela prema većem odzivu na štetu



Slika 5.3: F1 mjere vrednovanja modela na podatkovnim skupovima

Tablica 5.3: REPD usporedba na CM1 podatkovnom skupu

Model	Normalna distribucija	p vrijednost testa normalnosti	Različite distribucije	p vrijednost t-testa	d vrijednost	interpretacija efekta
NB	da	0.2904	ne	9.5810e-01		
LR	da	0.1327	da	6.4000e-06	1.2817	velik
KNN	ne	0.0000				
DTC	da	0.4691	da	1.0882e-04	1.0725	velik
HSME	da	0.4470	ne	9.5736e-02		

Tablica 5.4: REPD usporedba na JM1 podatkovnom skupu

Model	Normalna distribucija	p vrijednost testa normalnosti	Različite distribucije	p vrijednost t-testa	d vrijednost	interpretacija efekta
NB	da	0.6869	da	2.7569e-28	5.3013	velik
LR	da	0.1304	da	8.1836e-33	6.4732	velik
KNN	da	0.8308	da	1.3613e-20	3.6776	velik
DTC	da	0.6142	da	1.6015e-09	1.8477	velik
HSME	ne	0.0000				

Tablica 5.5: REPD usporedba na KC1 podatkovnom skupu

Model	Normalna distribucija	p vrijednost testa normalnosti	Različite distribucije	p vrijednost t-testa	d vrijednost	interpretacija efekta
NB	da	0.1638	da	2.4800e-06	1.3489	velik
LR	da	0.3268	da	5.3244e-13	2.3860	velik
KNN	da	0.5832	da	1.3882e-19	3.4918	velik
DTC	da	0.4055	da	7.7310e-11	2.0500	velik
HSME	da	0.8862	da	3.8844e-06	1.3172	velik

Tablica 5.6: REPD usporedba na KC2 podatkovnom skupu

Model	Normalna distribucija	p vrijednost testa normalnosti	Različite distribucije	p vrijednost t-testa	d vrijednost	interpretacija efekta
NB	da	0.5634	da	7.8509e-06	1.2671	velik
LR	da	0.1290	da	1.0428e-06	1.4095	velik
KNN	da	0.8052	da	9.2545e-08	1.5755	velik
DTC	da	0.3227	da	8.1248e-08	1.5843	velik
HSME	da	0.4648	da	3.0167e-03	0.7995	umjeren

preciznosti. To znači da predloženi model pronalazi više pogrešaka programske potpore, ali i da je skloniji lažnim uzbunama.

5.3.2 Rad modela na podatkovnim skupovima zasnovanim na semantičkim značajkama

REPD je dodatno vrednovan koristeći izgrađene podatkovne skupove zasnovane na semantičkim značajkama. Uz REPD vrednovani su i prethodno navedeni alternativni modeli te je provedena usporedba dobivenih rezultata vrednovanja.

Kako je navedeno u poglavlju 5.2.2 za svaki od izvornih podatkovnih skupova i svaki od modela korištenih za izgradnju značajki zapravo je napravljeno 30 podatkovnih skupova kako bi se izbjegao efekt potencijalno lošeg pojedinačnog učenja modela korištenih za izgradnju značajki. Ovim postupkom je dobiveno $24 * 30 = 720$ podatkovnih skupova. Za svaki od tih podatkovnih skupova modeli za predviđanje pogrešaka programske potpore su trenirani 30 puta, ponovno kako bi se reducirao stohastički efekt. To znači da je za svaki model provedeno $720 * 30 = 21600$ treniranja tj. da je ukupno provedeno $5 * 21600 = 108000$ treniranja modela za predviđanje pogrešaka programske potpore. Rezultati vrednovanja modela su grupirani po podatkovnom skupu, modelu za izgradnju značajki i modelu za predviđanje pogrešaka programske potpore. Kao glavna metrika kvalitete modela koristi se F1 mjera.

Na Slici 5.6 prikazane su F1 mjere, na Slici 5.7 vrijednosti preciznosti, a na Slici 5.8 vrijednosti odziva postignute na podatkovnim skupovima zasnovanim na *ant* podatkovnom skupu.

Na Slici 5.9 prikazane su F1 mjere, na Slici 5.10 vrijednosti preciznosti, a na Slici 5.11 vrijednosti odziva postignute na podatkovnim skupovima zasnovanim na *camel* podatkovnom skupu.

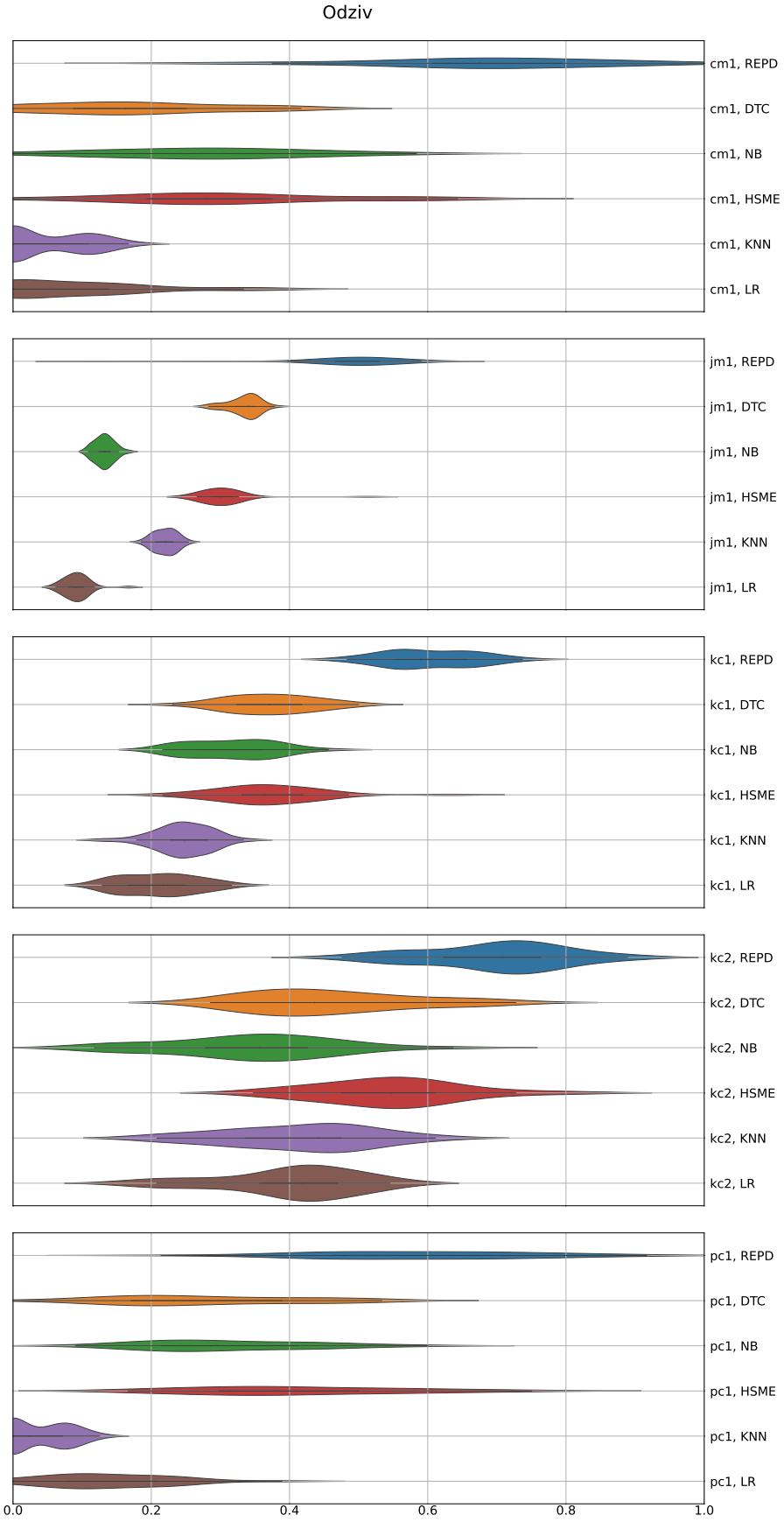
Na Slici 5.12 prikazane su F1 mjere, na Slici 5.13 vrijednosti preciznosti, a na Slici 5.14 vrijednosti odziva postignute na podatkovnim skupovima zasnovanim na *log4j* podatkovnom skupu.

Na Slici 5.15 prikazane su F1 mjere, na Slici 5.16 vrijednosti preciznosti, a na Slici 5.17 vrijednosti odziva postignute na podatkovnim skupovima zasnovanim na *poi* podatkovnom skupu.

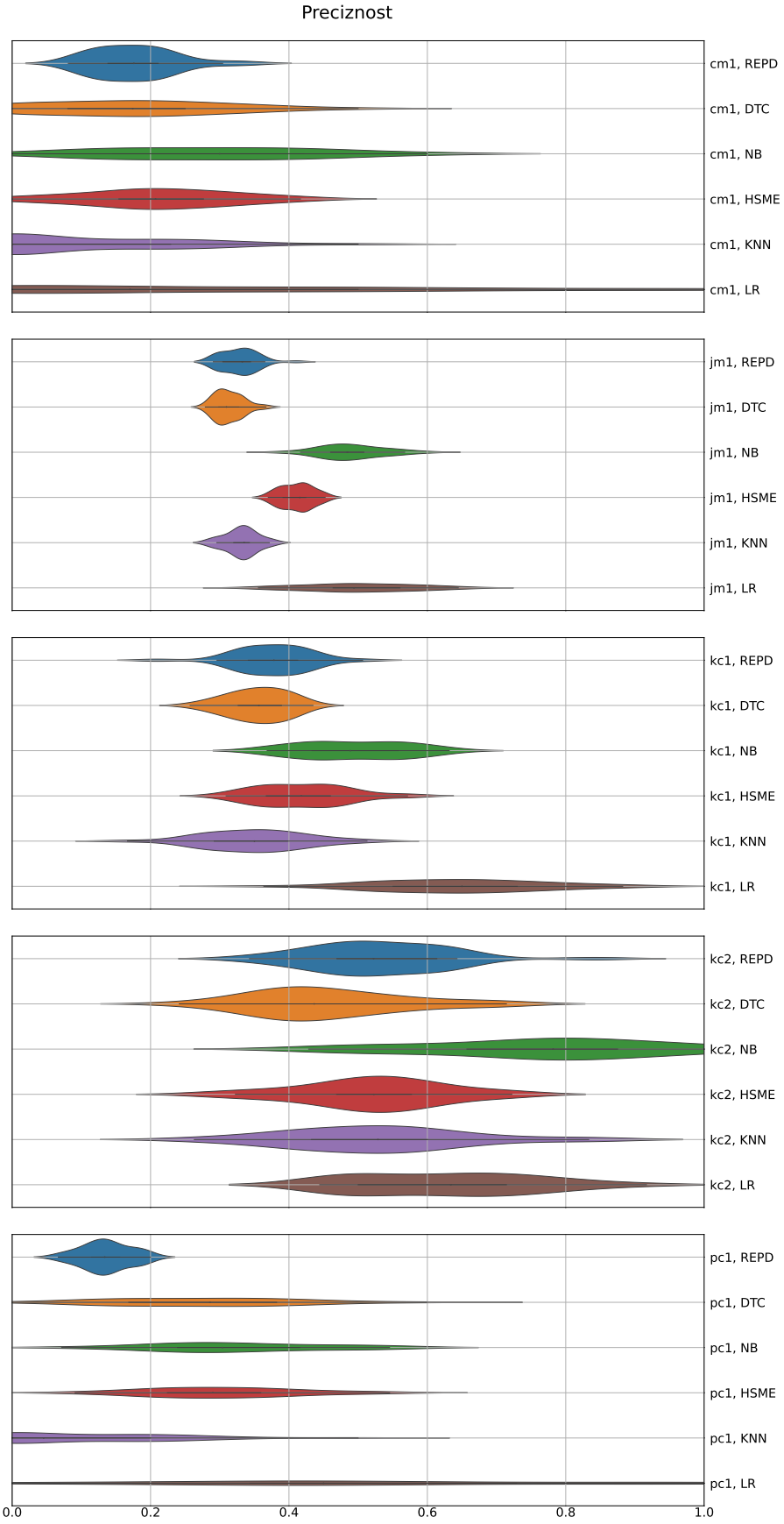
Na svim slikama korištene su oznake *dbn* za duboke mreže vjerovanja, *da* za duboki autoenkoder i *ca* za konvolucijski autoenkoder.

Usporedbom prosjeka postignutih F1 mjera na *ant v*1.5 vidljivo je da je REPD drugi najbolji model na podatkovnim skupovima gdje su značajke izgrađene primjenom konvolucijskog i dubokog autoenkodera te najbolji model na podatkovnim skupovima gdje su značajke izgrađene primjenom duboke mreže vjerovanja.

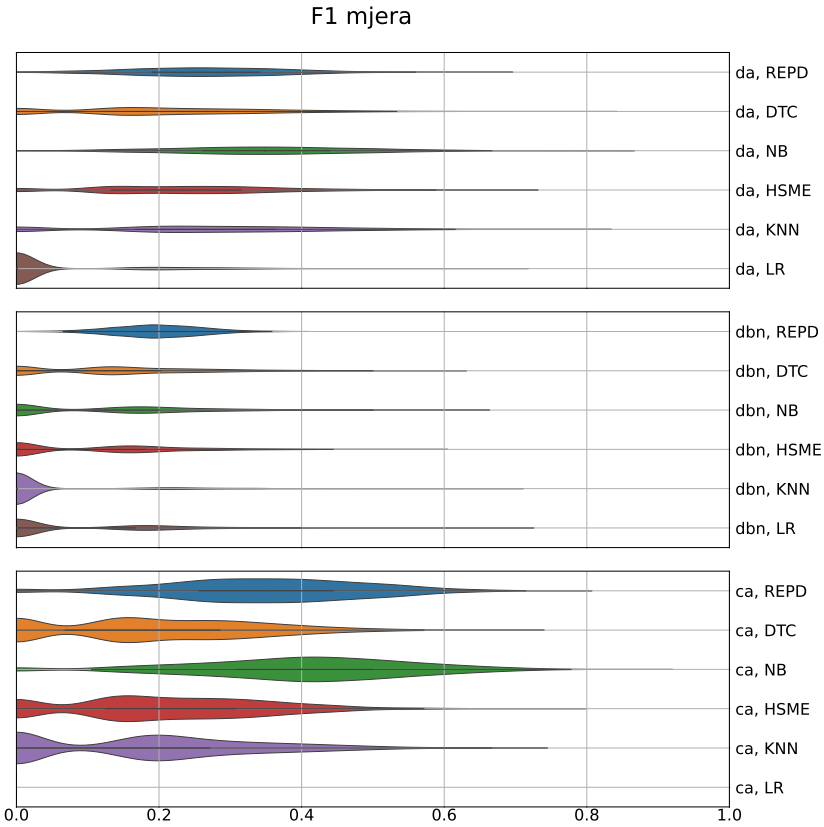
Usporedbom prosjeka postignutih F1 mjera na *ant v*1.6 vidljivo je da je REPD najbolji model na podatkovnim skupovima gdje su značajke izgrađene primjenom konvolucijskog auto-



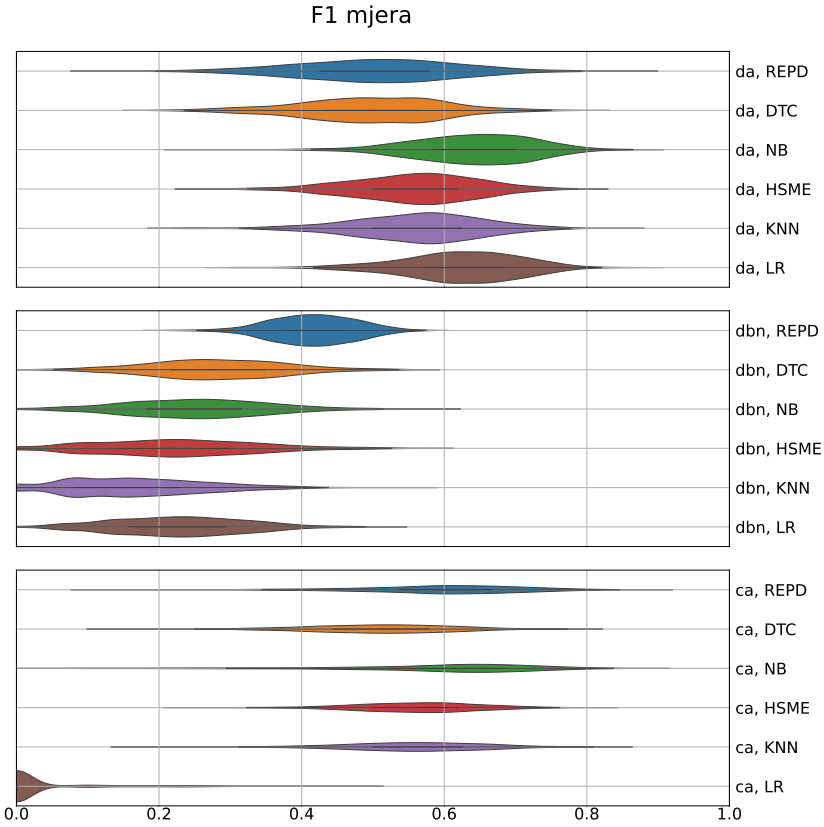
Slika 5.4: Odziv različitih modela na različitim podatkovnim skupovima



Slika 5.5: Preciznost različitih modela na različitim podatkovnim skupovima

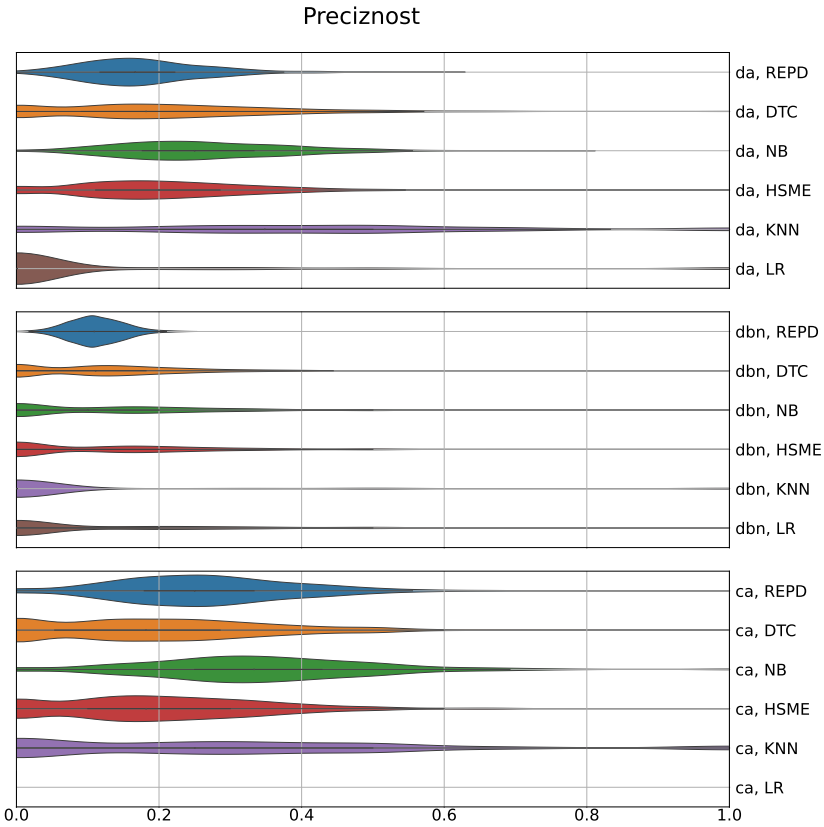


(a) ant *v*1.5

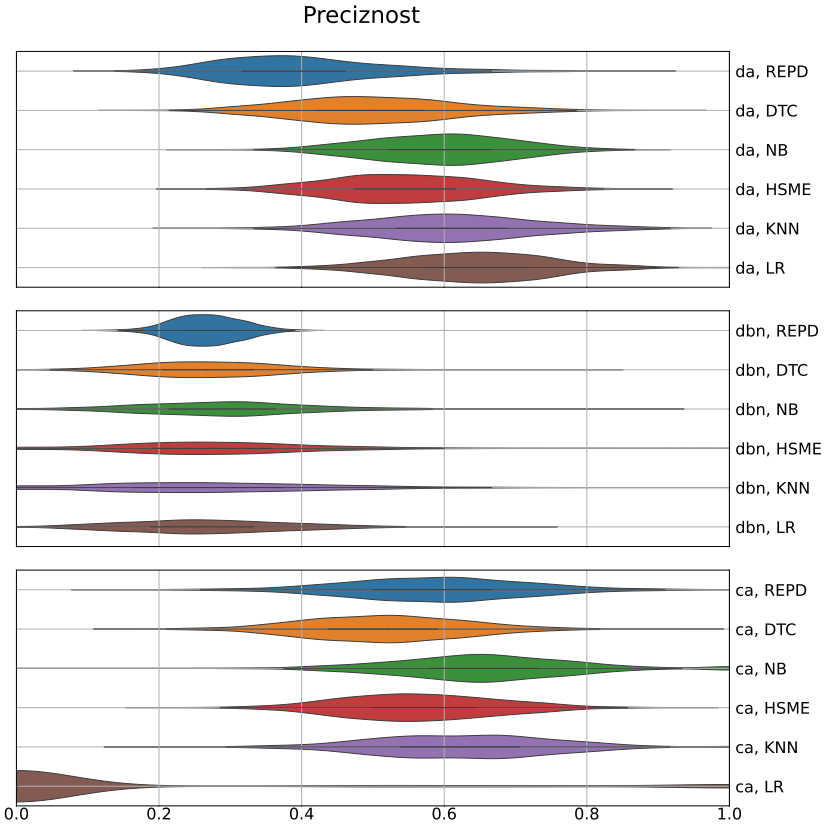


(b) ant *v*1.6

Slika 5.6: F1 mjere modela na ant podatkovnim skupovima zasnovanim na semantičkim značajkama

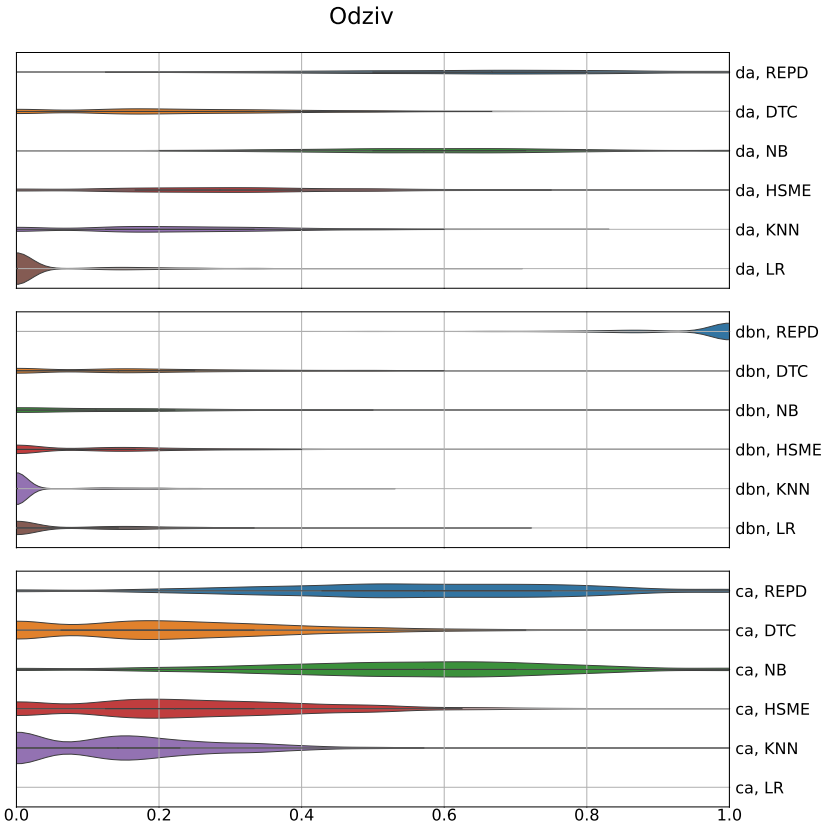


(a) ant *v*1.5

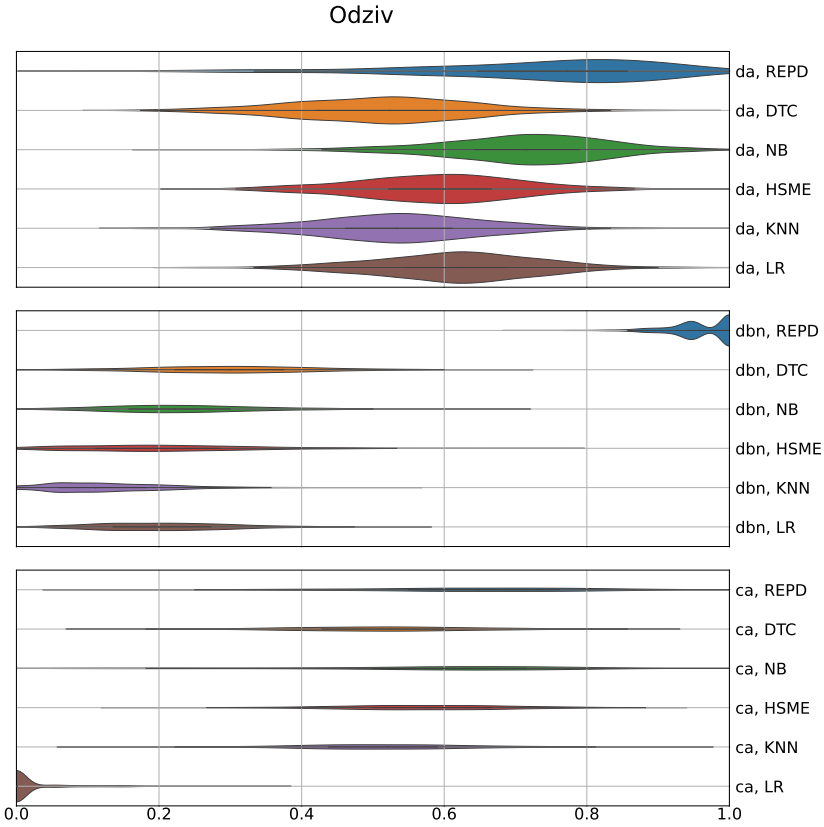


(b) ant *v*1.6

Slika 5.7: Preciznost modela na ant podatkovnim skupovima zasnovanim na semantičkim značajkama



(a) ant *v*1.5



(b) ant *v*1.6

Slika 5.8: Odziv modela na ant podatkovnim skupovima zasnovanim na semantičkim značajkama

enkodera i duboke mreže vjerovanja, dok na podatkovnim skupovima izgrađenim primjenom dubokog autoenkodera ne radi dobro.

Usporedbom prosjeka postignutih F1 mjera na *camel v*1.2 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom dubokog autoenkodera i duboke mreže vjerovanja, a treći najbolji model na podatkovnim skupovima izgrađenim primjenom konvolucijskog autoenkodera.

Usporedbom prosjeka postignutih F1 mjera na *camel v*1.4 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom konvolucijskog autoenkodera i duboke mreže vjerovanja, a drugi najbolji model na podatkovnim skupovima izgrađenim primjenom dubokog autoenkodera.

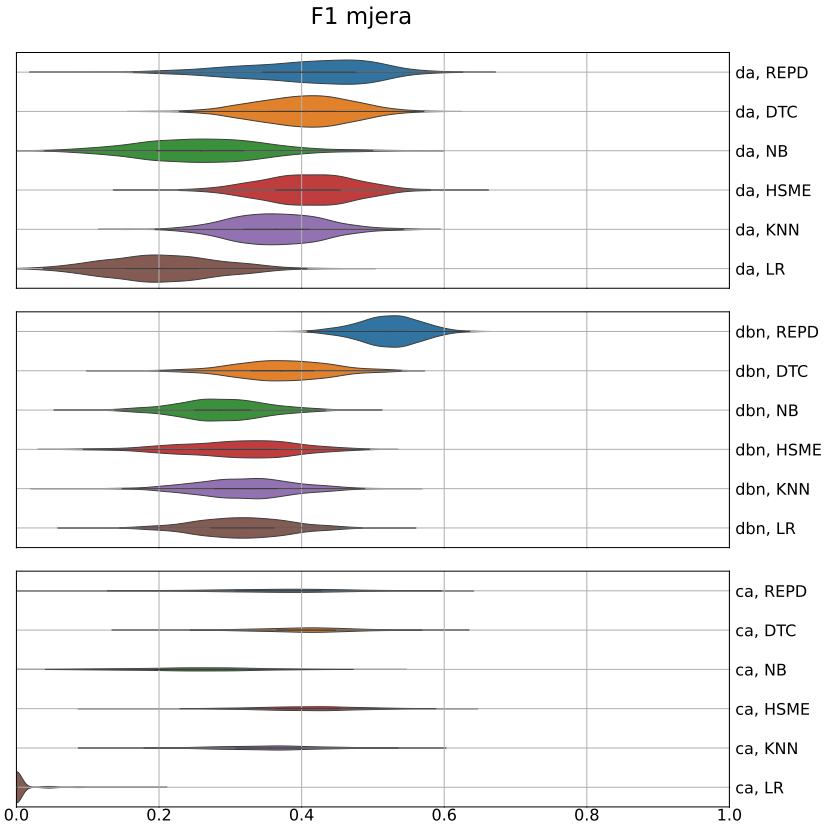
Usporedbom prosjeka postignutih F1 mjera na *log4j v*1.1 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom duboke mreže vjerovanja i drugi najbolji model na podatkovnim skupovima izgrađenim primjenom konvolucijskog autoenkodera, ali je najlošiji model na podatkovnim skupovima izgrađenim primjenom dubokog autoenkodera.

Usporedbom prosjeka postignutih F1 mjera na *log4j v*1.2 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom duboke mreže vjerovanja, ali je najlošiji model na podatkovnim skupovima izgrađenim primjenom konvolucijskog i dubokog autoenkodera. Razlog lošeg rada modela, u ovom slučaju, vjerojatno leži u činjenici da ovaj podatkovni skup sadrži malen broj programskih modula nesklonih pogreškama, pa autoenkoder koji se koristi za izračun grešaka rekonstrukcije nema dovoljno podataka iz kojih bi naučio.

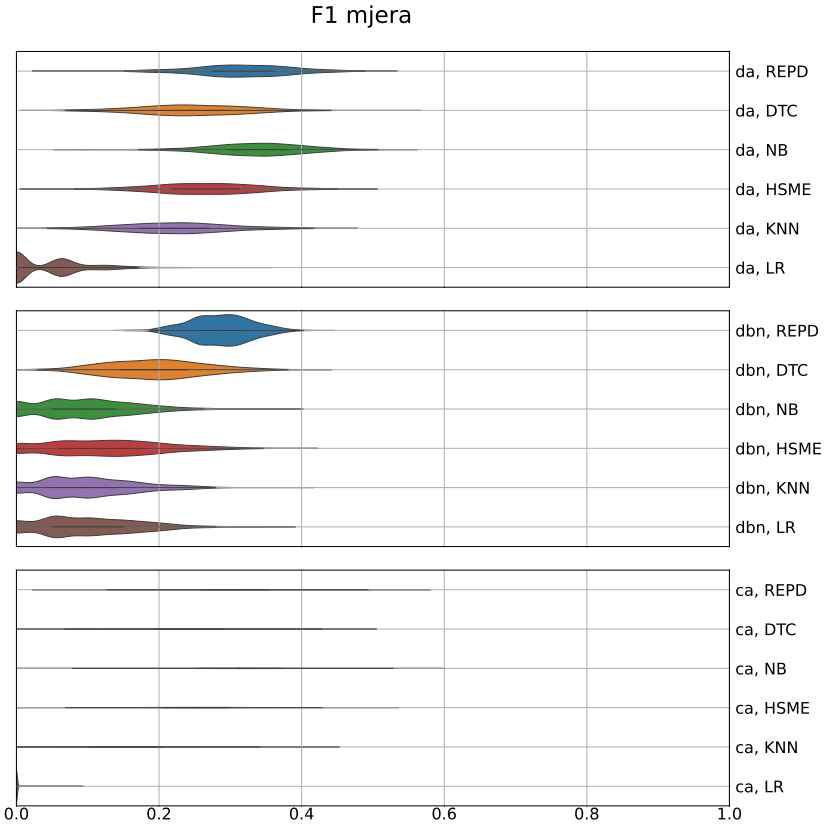
Usporedbom prosjeka postignutih F1 mjera na *poi v*2.0 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom duboke mreže vjerovanja i drugi najbolji model na podatkovnim skupovima izgrađenim primjenom konvolucijskog i dubokog autoenkodera.

Usporedbom prosjeka postignutih F1 mjera na *poi v*2.5 vidljivo je da je REPD najbolji model na podatkovnim skupovima izgrađenim primjenom duboke mreže vjerovanja, ali je loš na podatkovnim skupovima izgrađenim primjenom konvolucijskog i dubokog autoenkodera. Ponovno, ovaj podatkovni skup sadrži malen broj primjera programskih modula nesklonih pogreškama što onemogućuje autoenkoder koji se koristi za izračun grešaka rekonstrukcije da dobro nauči rekonstruirati primjere.

Ukupno gledano REPD je najbolji model u 11 od 24 slučaja i drugi najbolji u 6 od 24 slučaja. Očigledno, model nije uvijek najbolji, ali je izuzetno dobar model za predviđanje pogrešaka programske potpore. Model ponovno pokazuje visoke vrijednosti odziva što ga čini efikasnim sustavom za otkrivanje programskih modula sklonih pogreškama. Još važnije, dobiveni rezultati pokazuju da je problem predviđanja pogrešaka programske potpore moguće tretirati kao problem otkrivanja anomalija.

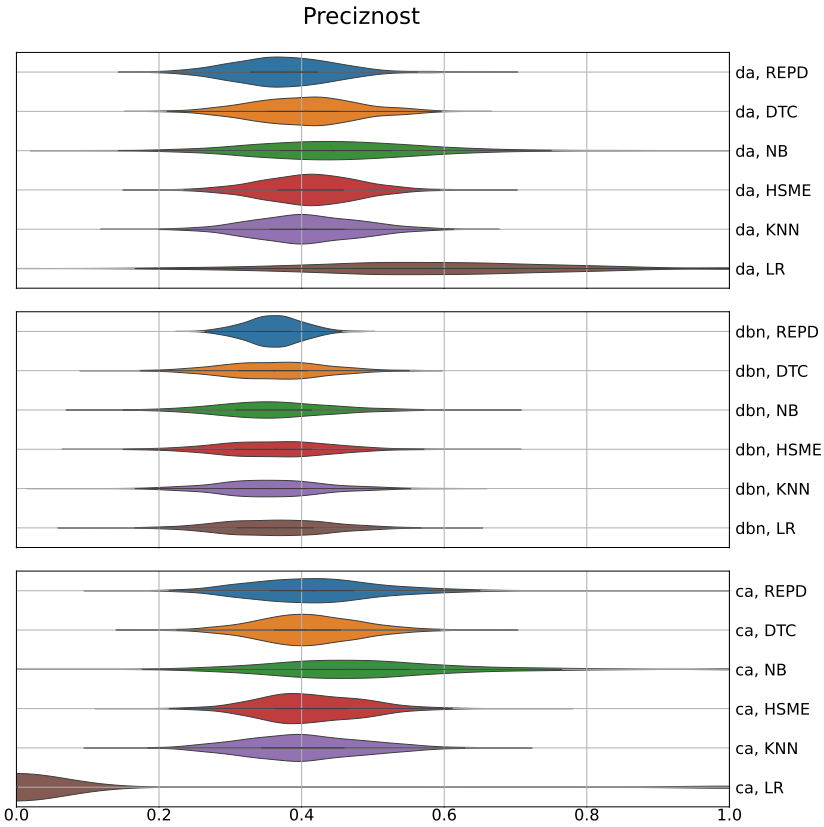


(a) camel *v*1.2

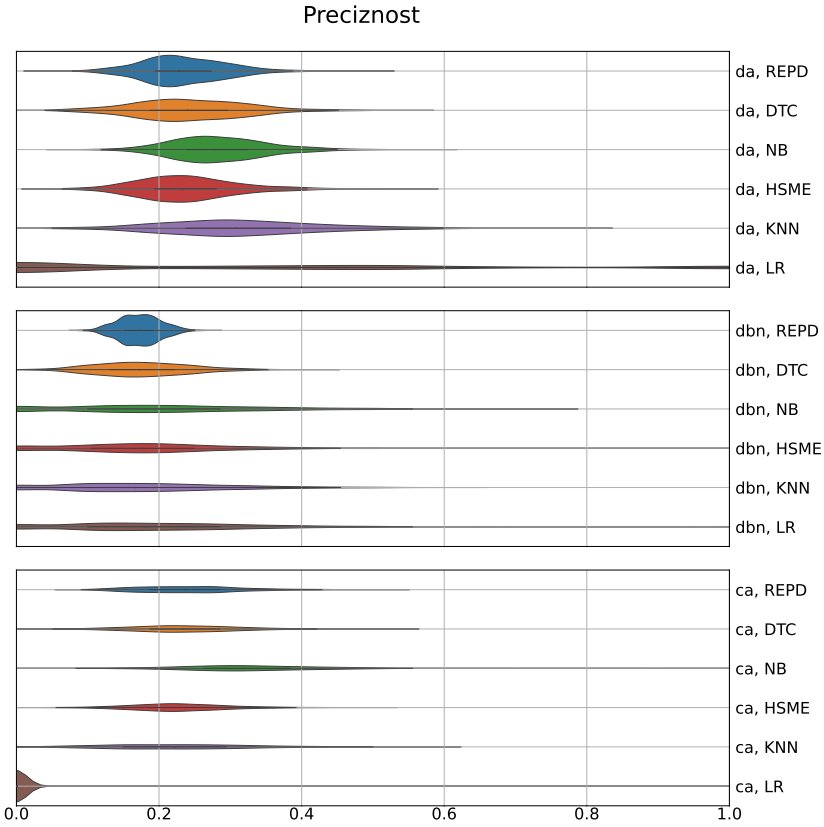


(b) camel *v*1.4

Slika 5.9: F1 mjera modela na camel podatkovnim skupovima zasnovanim na semantičkim značajkama

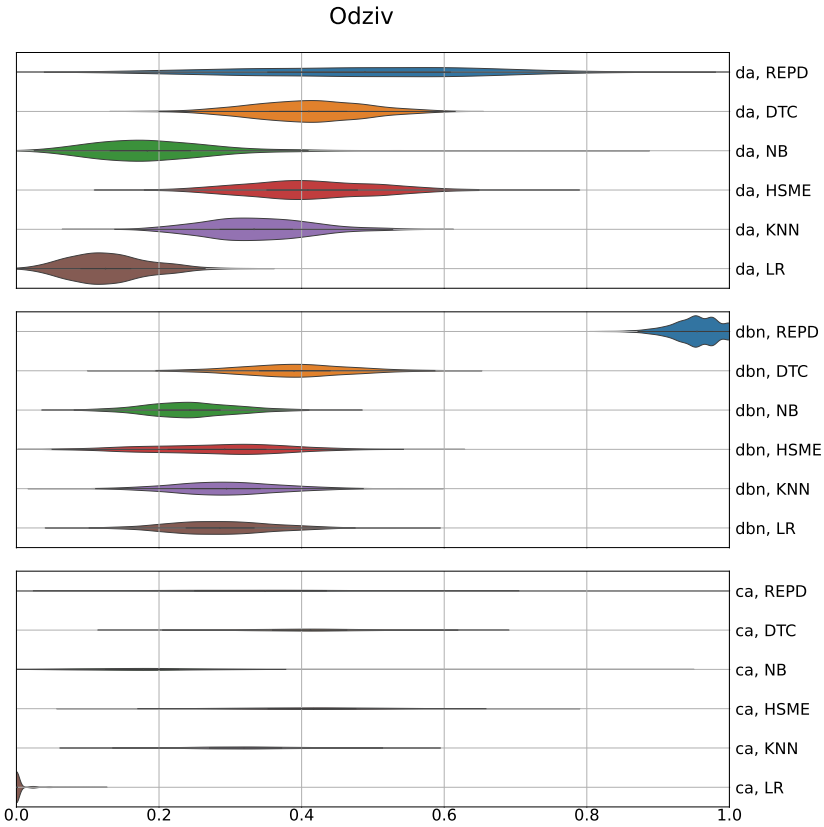


(a) camel *v*1.2

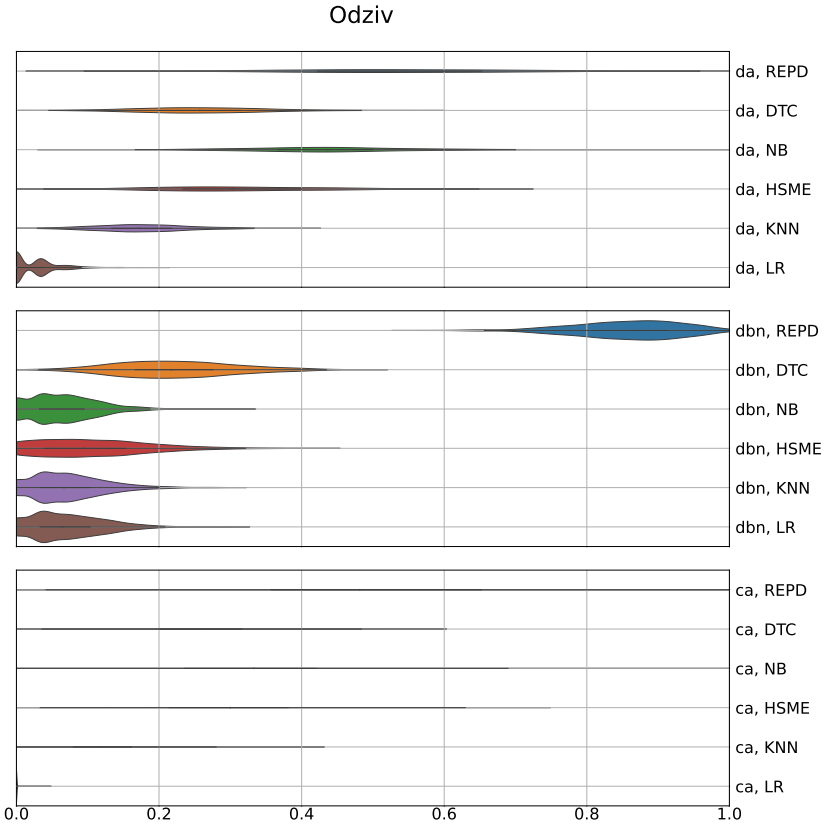


(b) camel *v*1.4

Slika 5.10: Preciznost modela na camel podatkovnim skupovima zasnovanim na semantičkim značajkama

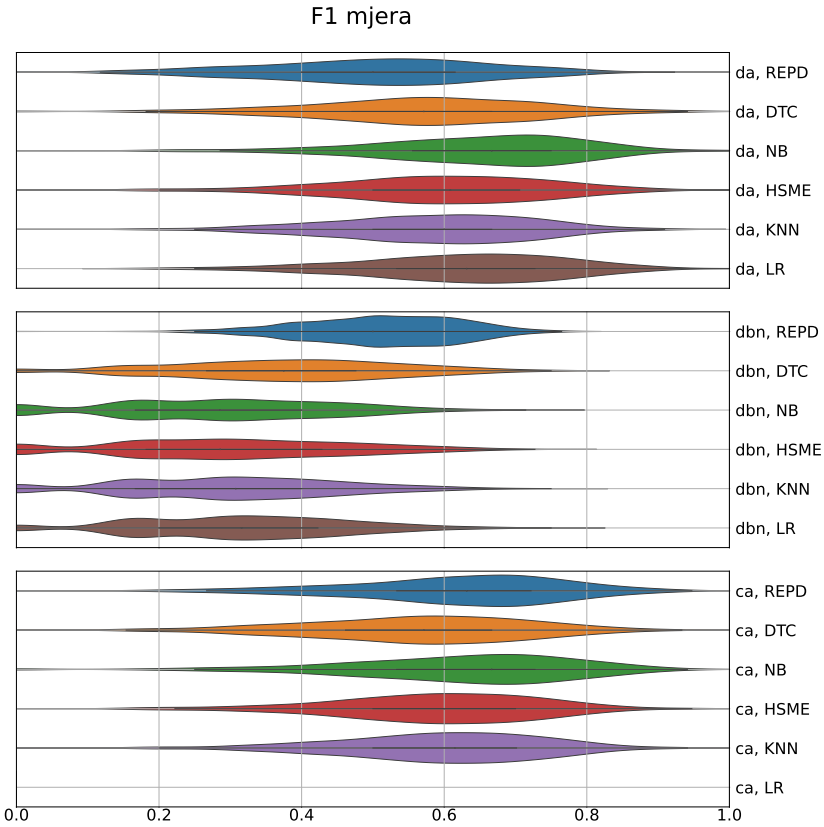


(a) camel *v*1.2

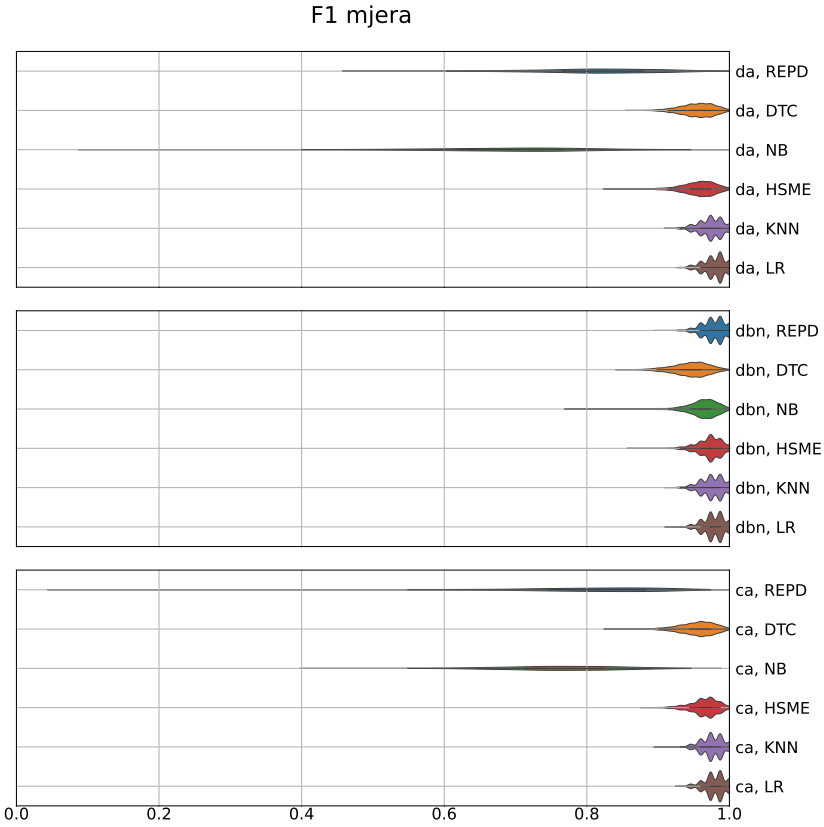


(b) camel *v*1.4

Slika 5.11: Odziv modela na camel podatkovnim skupovima zasnovanim na semantičkim značajkama

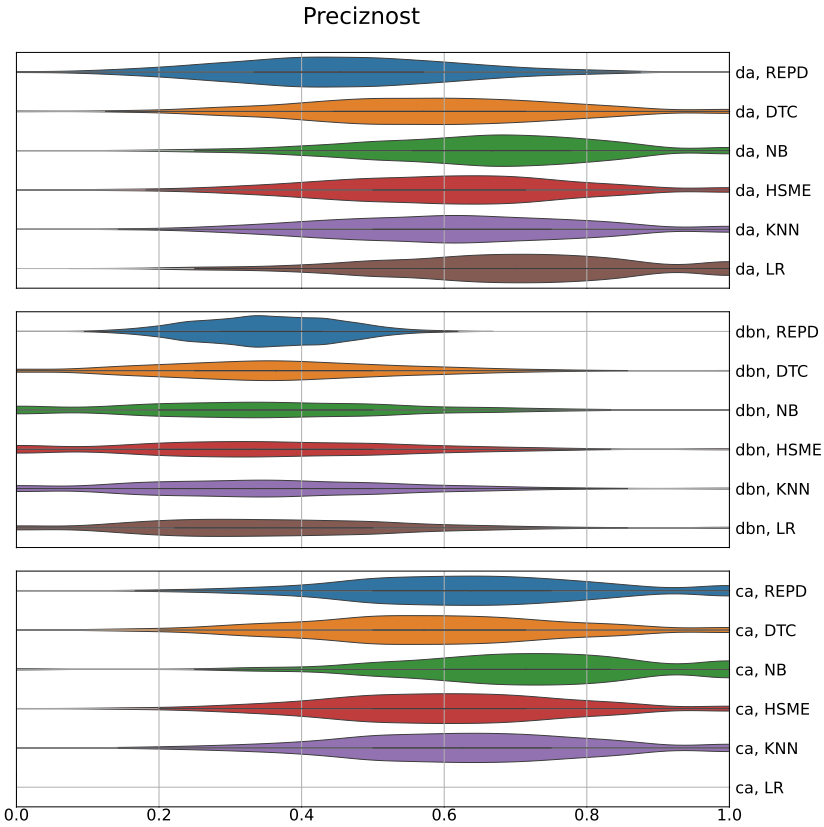


(a) log4j *v*1.1

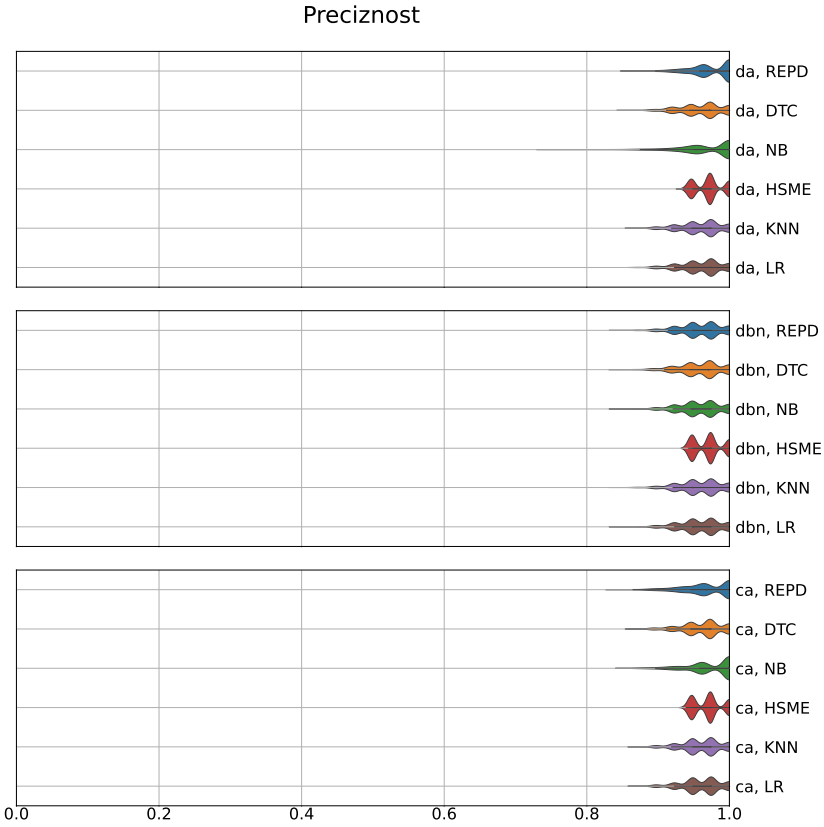


(b) log4j *v*1.2

Slika 5.12: F1 mjera modela na log4j podatkovnim skupovima zasnovanim na semantičkim značajkama

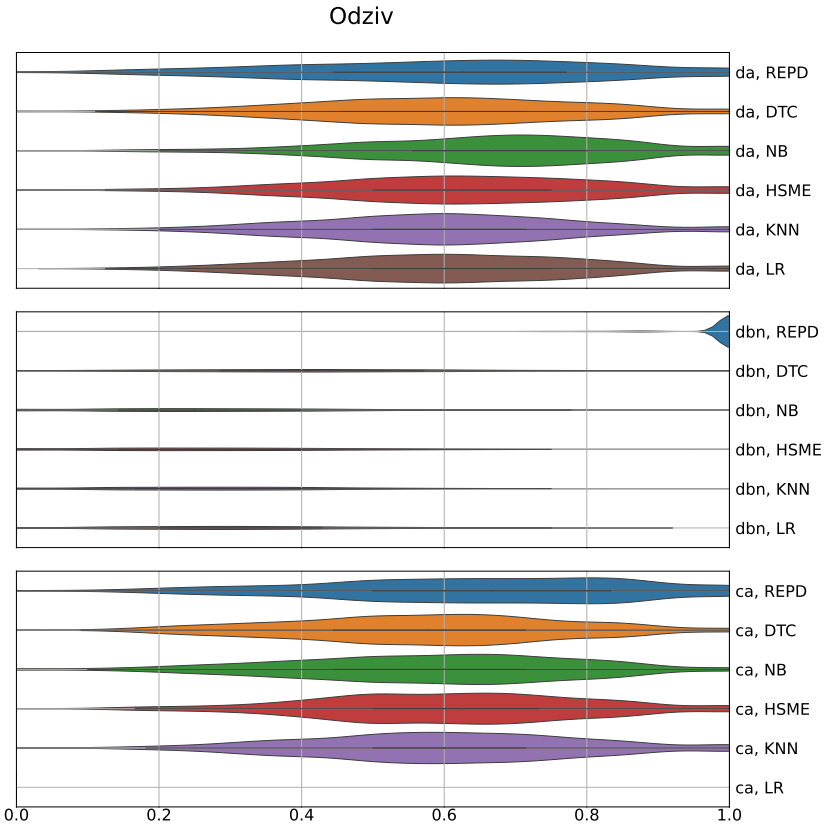


(a) log4j *v*1.1

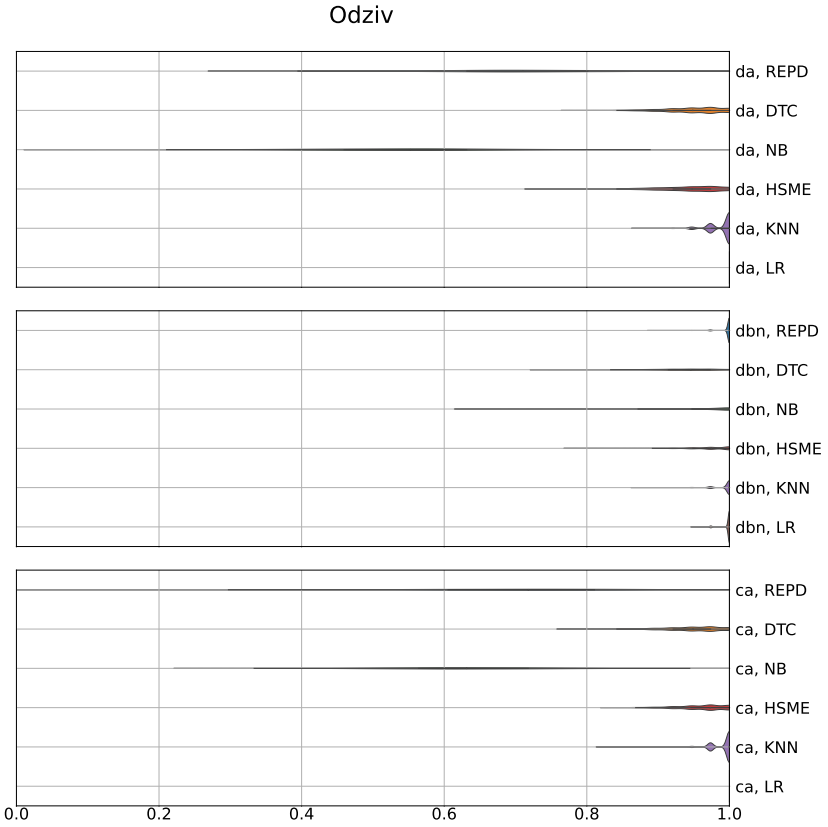


(b) log4j *v*1.2

Slika 5.13: Preciznost modela na log4j podatkovnim skupovima zasnovanim na semantičkim značajkama

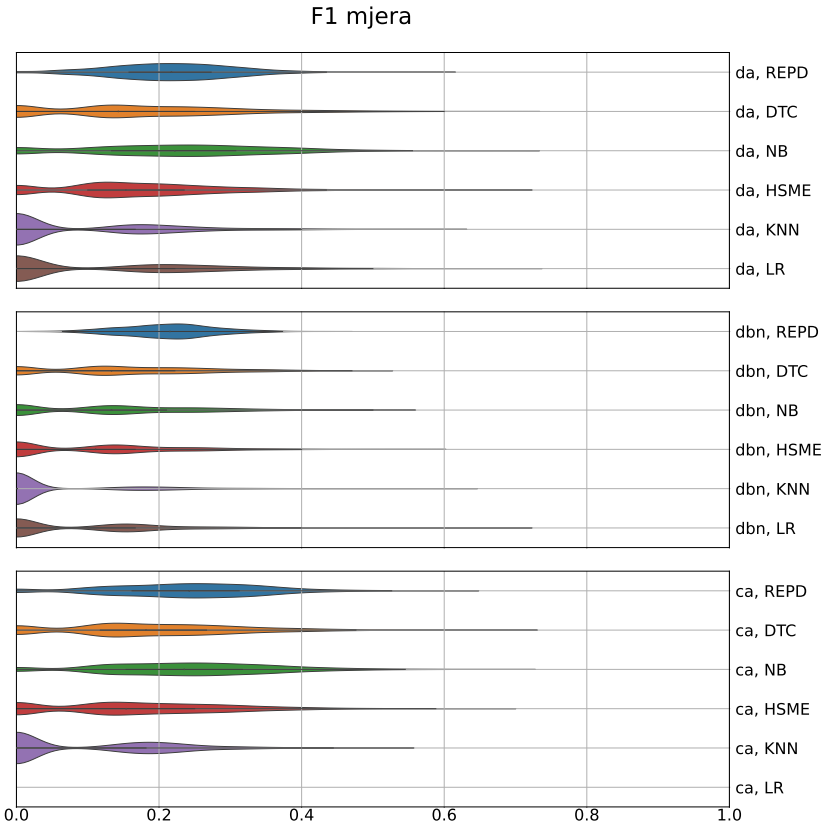


(a) log4j *v*1.1

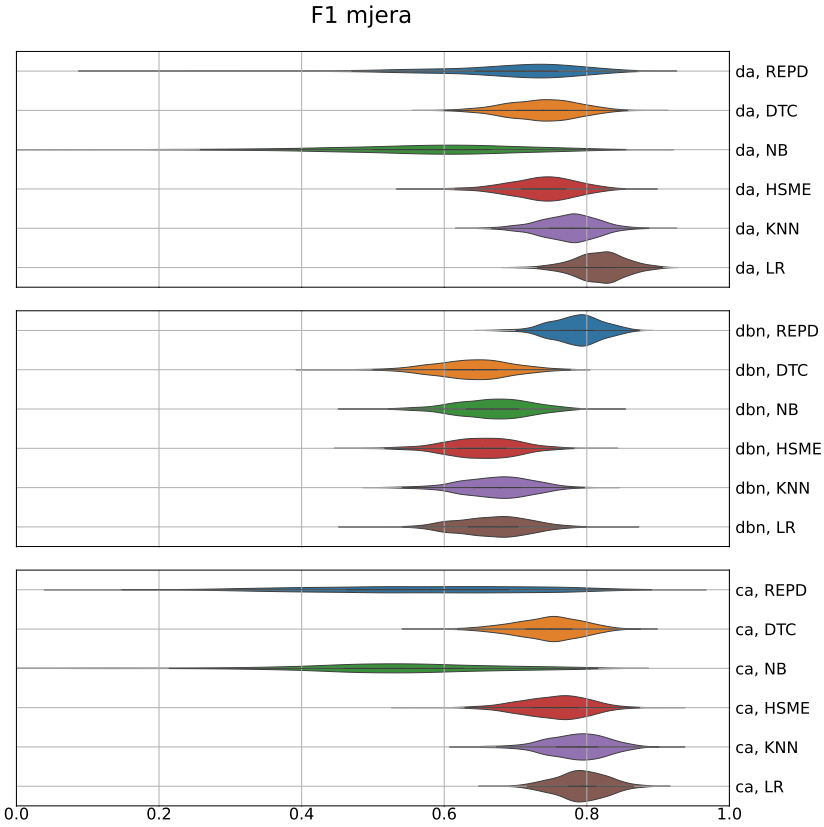


(b) log4j *v*1.2

Slika 5.14: Odziv modela na log4j podatkovnim skupovima zasnovanim na semantičkim značajkama

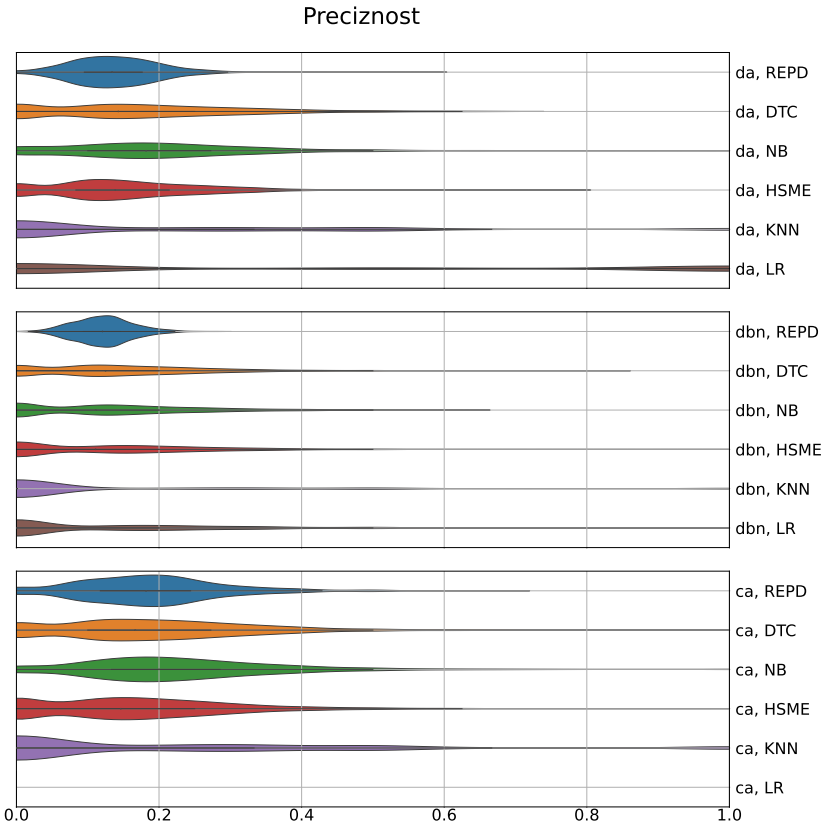


(a) poi *v*2.0

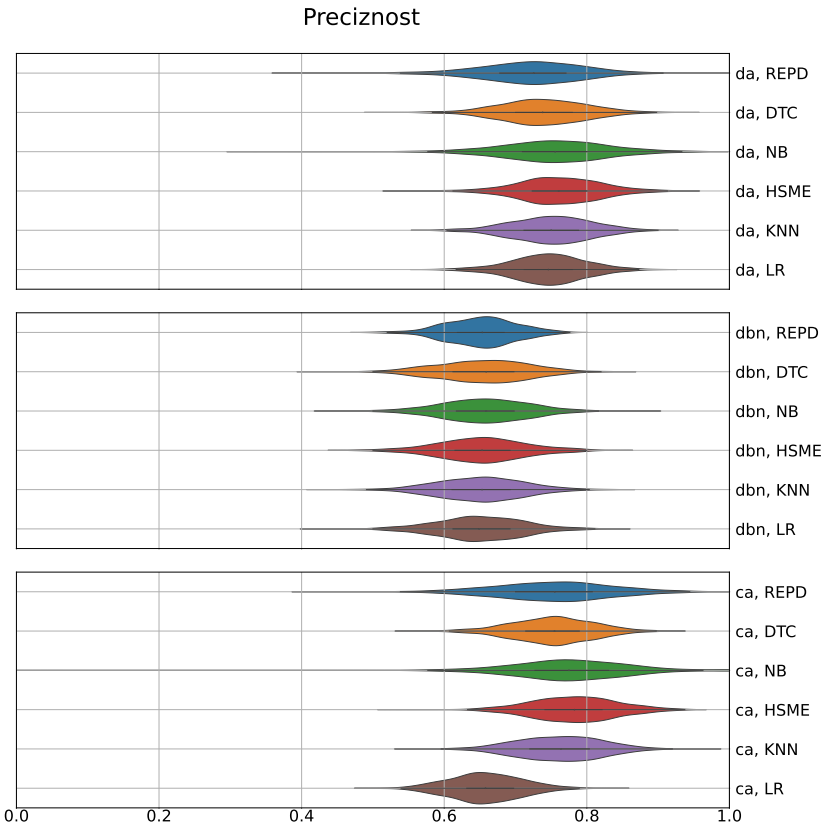


(b) poi *v*2.5

Slika 5.15: F1 mjera modela na poi podatkovnim skupovima zasnovanim na semantičkim značajkama

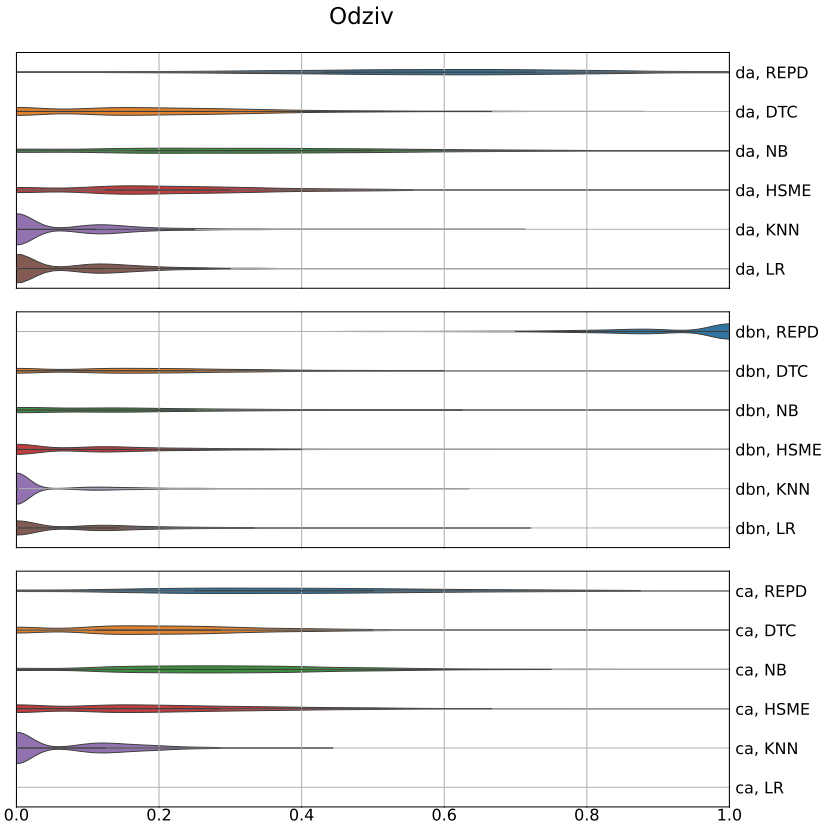


(a) poi *v*2.0

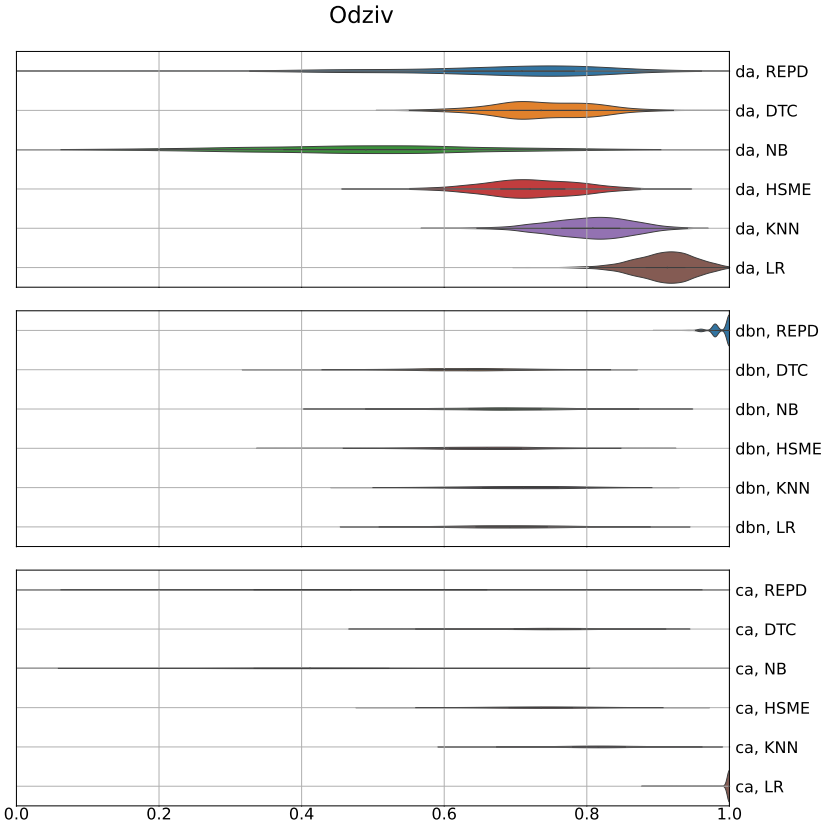


(b) poi *v*2.5

Slika 5.16: Preciznost modela na poi podatkovnim skupovima zasnovanim na semantičkim značajkama



(a) poi *v*2.0



(b) poi *v*2.5

Slika 5.17: Odziv modela na poi podatkovnim skupovima zasnovanim na semantičkim značajkama

5.3.3 Otpornost modela na problem neuravnoteženih razreda

U sklopu istraživanja provedena je i analiza otpornosti predloženog modela na problem neuravnoteženih razreda. U sklopu analize korišteni su podatkovni skupovi JM1, KC1 i KC2 jer u izvornom stanju imaju više od 20% primjera koji predstavljaju programske module sklone pogreškama. Navedeni podatkovni skupovi odabrani su kako bi se mogla promatrati promjena rada modela s promjenom neuravnoteženosti podatkovnog skupa. CM1 i PC1 podatkovni skupovi, u svom izvornom izdanju, već imaju veliku neuravnoteženost razreda što ne ostavlja puno prostora za daljnje promjene.

Udio primjera koji predstavljaju programske module sklone pogreškama mijenja se od 20% do 5% udjela takvih primjera s korakom od 1%. Navedena promjena vrši se na dva načina pod-uzorkovanjem primjera programskih modula sklonih pogreškama čime se smanjuje njihov broj i ponovnim uzorkovanjem programskih modula nesklonih pogreškama čime se smanjuje udio primjera programskih modula sklonih pogreškama u ukupnom podatkovnom skupu.

Nakon ažuriranja podatkovnih skupova, treniraju se modeli za predviđanje pogrešaka programske potpore koji su i do sada korišteni u istraživanju. Prikupljeni rezultati vrednovanja se vizualiziraju i analizira se ponašanje modela s obzirom na neuravnoteženost razreda u podatkovnom skupu. Mjera F1 ponovno se koristi kao glavni indikator kvalitete rada modela.

Važno je napomenuti da pod-uzorkovanje uklanja primjere iz podatkovnog skupa, tj. smanjuje ukupnu količinu raspoloživih informacija za učenje, dok ponovno uzorkovanje ne uklanja informacije iz podatkovnog skupa.

U poglavlju 5.3.3 opisano je korištenje pod-uzorkovanja, a u poglavlju 5.3.3 opisano je korištenje ponovnog uzorkovanja.

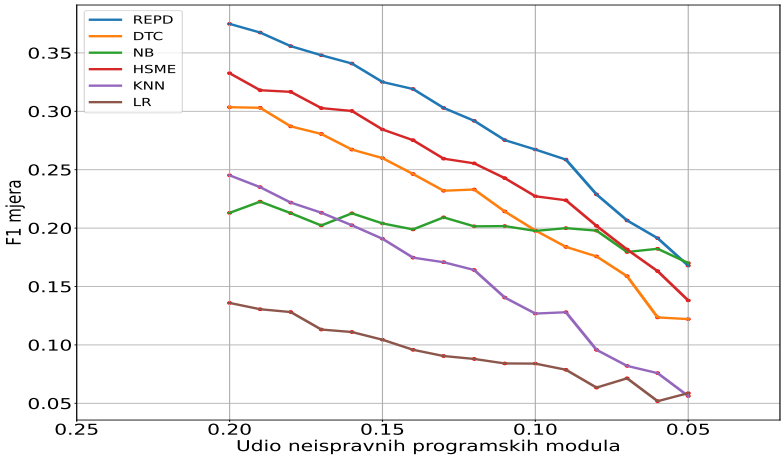
Pod-uzorkovanje

U ovom poglavlju opisano je korištenje pod-uzorkovanja za analizu otpornosti modela na problem neuravnoteženih razreda. Udio primjera programskih modula sklonih pogreškama smanjuje se s 20% na 5% s korakom od 1%. Za svaki postotak 30 puta se nasumično odabiru primjeri programskih modula sklonih pogreškama koji će se ukloniti iz podatkovnog skupa te se na dobivenom podatkovnom skupu treniraju modeli korišteni u istraživanju.

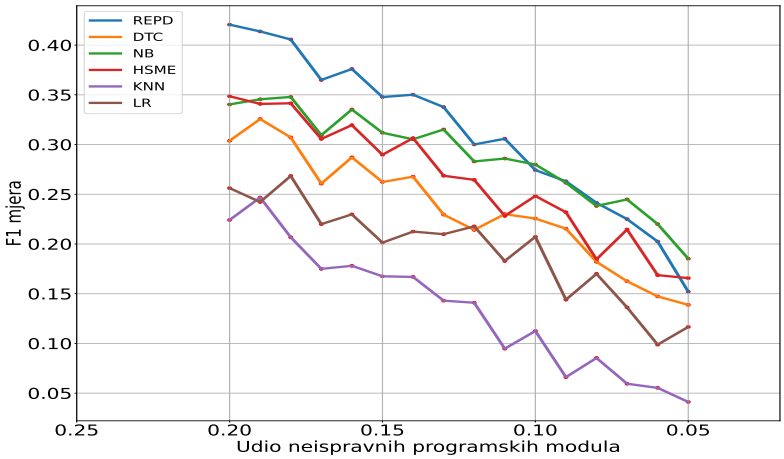
Slika 5.18 prikazuje rezultate vrednovanja modela pri promjeni udjela primjera programskih modula sklonih pogreškama. Vidljivo je kako kvaliteta rada svih modela opada što je i očekivano s obzirom na to da se smanjuje ukupno dostupna količina informacija za učenje.

Ponovno uzorkovanje

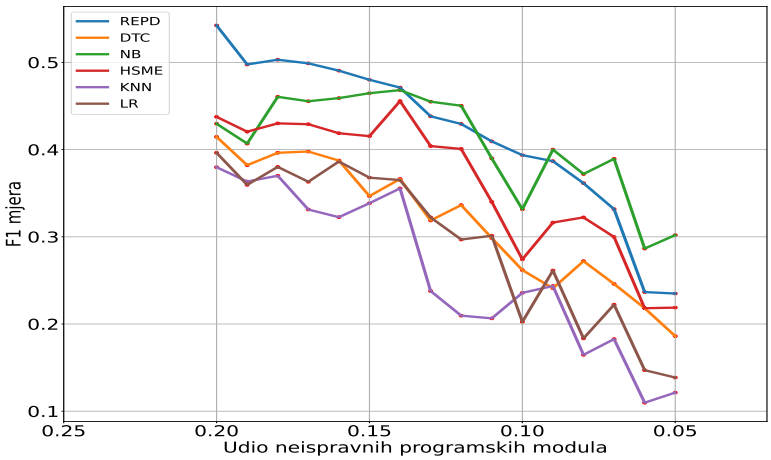
U ovom poglavlju opisano je korištenje ponovnog uzorkovanja za analizu otpornosti modela na problem neuravnoteženih razreda. Udio primjera programskih modula sklonih pogreškama



(a) JM1



(b) KC1



(c) KC2

Slika 5.18: F1 mjere pri pod-uzorkovanju

smanjuje se s 20% na 5% s korakom od 1%. Za svaki postotak 30 puta se nasumično odabiru primjeri programskih modula nesklonih pogreškama koji će se ponovno dodati u podatkovni skup te se na dobivenom podatkovnom skupu treniraju modeli korišteni u istraživanju.

Slika 5.19 prikazuje rezultate vrednovanja modela pri promjeni udjela primjera programskih modula sklonih pogreškama. Na podatkovnim skupovima dobivenim ažuriranjem JM1 većina modela radi stabilno. Jedini model čija kvaliteta rada vidno opada je k najbližih susjeda, što nije iznenađujuće s obzirom da se ponovnim uzorkovanjem mijenjaju susjedstva modela. REPD je najbolji model u svim koracima vrednovanja. Na podatkovnim skupovima dobivenim ažuriranjem KC1 k najbližih susjeda i logistička regresija imaju značajan pad u kvaliteti rada. Za KNN model objašnjenje je isto kao i kod podatkovnih skupova dobivenih ažuriranjem JM1. Kod logističke regresije jedno moguće objašnjenje bila bi promjena funkcije pogreške uslijed ponovnog uzorkovanja primjera. REPD je najbolji model u svim koracima vrednovanja. Na podatkovnim skupovima dobivenim ažuriranjem KC2 rezultati vrednovanja modela dosta osciliraju. Razlog oscilacije vjerojatno leži u maloj veličini korištenog podatkovnog skupa. REPD je najbolji model u svim koracima vrednovanja.

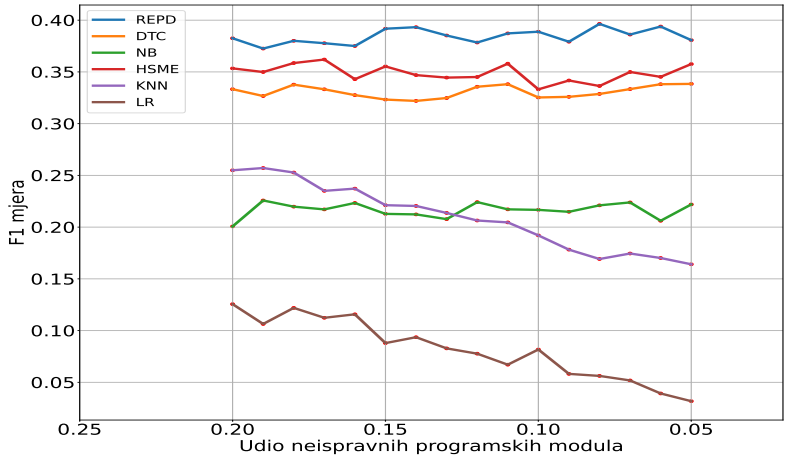
5.4 Rizici valjanosti istraživanja

U ovom poglavlju kratko su navedeni rizici valjanosti istraživanja koji su mogli utjecati na rezultate istraživanja te koraci poduzeti za smanjenje njihova utjecaja.

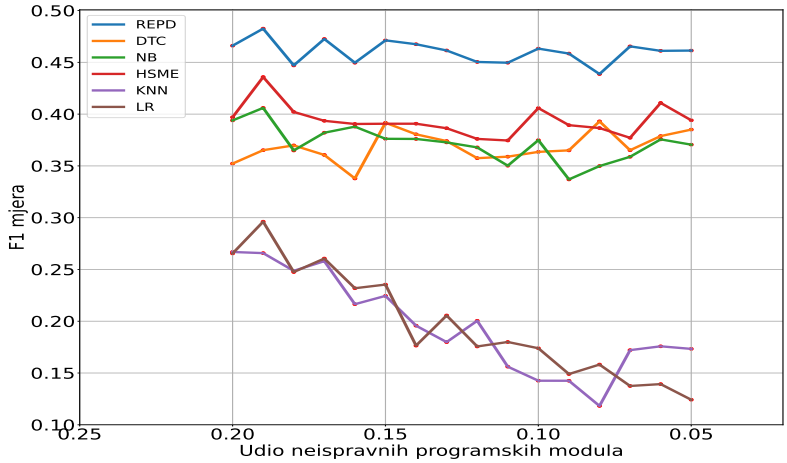
U poglavlju 5.4.1 opisan je rizik koji proizlazi iz korištenih javno dostupnih podatkovnih skupova i programskih repozitorija. U poglavlju 5.4.2 opisan je rizik koji je posljedica metode konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore. Konačno, u poglavlju 5.4.3 opisan je rizik koji je posljedica načina razvoja modela za predviđanje pogrešaka programske potpore.

5.4.1 Podatkovni skupovi i javno dostupni repozitoriji otvorenog programskog kôda

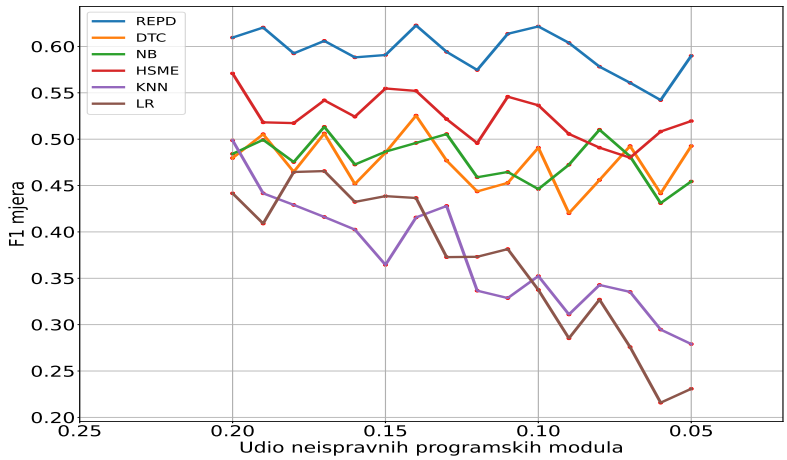
Korišteni podatkovni skupovi preuzeti su iz PROMISE programa i javno dostupnih repozitorija otvorenog programskog kôda. Moguće je da korišteni podatkovni skupovi i repozitoriji ne predstavljaju reprezentativan podatkovni skup za različite vrste programske potpore i time utječu na dobivene rezultate. Kako bi se smanjio ovaj rizik korišteno je više različitih izvora podataka. Dodatno, obzirom da su ovi podatkovni skupovi izrađeni automatskim prikupljanjem podatka iz repozitorija programske potpore moguće je da sadrže šum. Šum o ovim podatkovnim skupovima bio bi posljedica prikupljanja podataka korištenjem jednostavne heuristike zasnovane na ključnim riječima.



(a) JM1



(b) KC1



(c) KC2

Slika 5.19: F1 mjere pri ponovnom uzorkovanju

5.4.2 Metoda konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore

Metoda konstrukcije podatkovnih skupova za predviđanje pogrešaka programske potpore primijenjena u sklopu ovog istraživanja nije jedina moguća. Način izgradnje semantičkih značajki mogao je biti drugačiji. Kako bi se smanjio utjecaj ovog rizika korišteno je više podatkovnih skupova uasnovanih na klasičnim značajkama i više podatkovnih skupova zasnovanih na semantičkim značajkama.

5.4.3 Razvoju modela za predviđanje pogrešaka programske potpore

Pri razvoju modela za predviđanje pogrešaka programske potpore u većini slučajeva korišteni su hiperparametri modela kako su zadani od strane korištenih programskih paketa. Zbog ograničenja resursa nije provođena iscrpna optimizacija hiperparametara modela, pa je moguće da neki od modela ne postižu najbolje rezultate koje bi mogli. Kako bi se smanjio utjecaj ovog rizika korišten je velik broj različitih modela.

Poglavlje 6

Zaključak

Značaj programske potpore u današnjem svijetu je neosporan. Računala su dio sve većeg broja stvari, od osobnih računala do modernih automobila, pametnih tv prijemnika pa čak i dizala u zgradama. Široka rasprostranjenost i velik značaj programske potpore diktiraju i velike napore u osiguranju njene kvalitete.

Predviđanje pogrešaka programske potpore jedna je od aktivnosti kojom inženjeri nastoje poboljšati kvalitetu programske potpore. Predviđanje pogrešaka programske potpore teži otkrivanju pogrešaka u programskoj potpori prije nego se one manifestiraju u njenom radu. Procjenom rizičnosti izvornog programskog kôda dobiva se dojam o njegovoj kvaliteti, ali i bolji uvid kako rasporediti ograničene resurse i u što kraće vremena adresirati što veći broj pogrešaka u razvijenoj programskoj potpori.

Razvoj modela za predviđanje pogrešaka programske potpore temelji se na posebno izgrađenim podatkovnim skupovima. Ti podatkovni skupovi se grade temeljem podataka prikupljenih iz sustava za upravljanje verzijama kôda i sustava za praćenje zadataka. S obzirom na to da je prikupljanje podataka automatizirano, a oslanja se na oznake zadataka i mogućnost uparivanja zadataka s predajama promjena programskog kôda, vrlo je vjerojatno da dobiveni podatkovni skupovi za razvoj modela za predviđanje pogrešaka programske potpore sadrže šum tj. pogreške u podacima. Ovaj šum utječe na kvalitetu razvijenih modela. Dodatan problem pri razvoju modela za otkrivanje pogrešaka programske potpore je i problem neuravnoteženosti razreda u izgrađenim podatkovnim skupovima. Naime, podatkovni skupovi za predviđanje pogrešaka programske potpore učestalo sadrže daleko više primjera programskih modula nesklonih pogreškama nego programskih modula sklonih pogreškama.

U ovoj disertaciji analiziran je utjecaj krivo označenih zadataka u sustavima na praćenje zadataka na kvalitetu skupova za razvoj modela za predviđanje pogrešaka programske potpore te utjecaj na rad razvijenih modela [36]. Dodatno, predložen je model za predviđanje pogrešaka programske potpore zasnovan na ideji otkrivanja anomalija [37] čime se inherentno rješava problem neuravnoteženosti razreda.

Prezentirana je analiza utjecaja klasifikacije zadataka na kvalitetu podatkovnih skupova i rad modela za predviđanje pogrešaka programske potpore. Za potrebe analize prikupljeni su podatci 7 javno dostupnih repozitorija otvorenog programskog kôda. Koristeći prikupljene podatke izgrađeni su podatkovni skupovi za klasifikaciju zadataka i podatkovni skupovi za predviđanje pogrešaka programske potpore. Razmotrene su različite metode klasifikacije zadataka: heuristika zasnovana na ključnim riječima, unaprijeđena heuristika zasnovana na ključnim riječima, FastText model i RoBERTa model. Prikupljeni rezultati pokazuju da korištenje RoBERTa modela rezultira najboljim podatkovnim skupovima za predviđanje pogrešaka programske potpore. Ti podatkovni skupovi, u prosjeku sadrže 14.3641 % krivo označenih primjera. U 65 od 84 slučaja modeli trenirani na podatkovnim skupovima dobivenim korištenjem RoBERTa modela ostvaruju bolje rezultate vrednovanja nego isti modeli trenirani na podatkovnim skupovima dobivenim korištenjem drugih pristupa klasifikacije zadataka. Od 65 slučajeva 55 pokazuje statistički značajnu razliku. Nadalje, u 17 od 28 slučajeva ne postoji statistički značajna razlika u rezultatima vrednovanja modela treniranih na podatkovnim skupovima dobivenim korištenjem RoBERTa modela i podatkovnim skupovima dobivenim ručnim označavanjem zadataka. Napravljeni podatkovni skupovi su javno objavljeni i uz rezultate analize predstavljaju znanstvene doprinose autora. Cijelo istraživanje je objavljeno u obliku znanstvenog članka [36], a programski kôd napravljen je javno dostupnim [38].

Osim toga prezentiran je novi pristup predviđanju pogrešaka programske potpore zasnovan na ideji otkrivanja anomalija čime se inherentno rješava problem neuravnoteženosti razreda. Anomalije u ovom slučaju predstavljaju primjere programskih modula sklonih pogreškama. Predstavljeni pristup naziva se REPD (engl. Reconstruction Error Probability Distribution, skr. REPD) što je skraćenica za vjerojatnosnu distribuciju grešaka rekonstrukcije. Rad modela uspoređen je s radom pet alternativnih modela na 5 podatkovnih skupova koji koriste klasične značajke složenosti programskog kôda i na 24 podatkovna skupa koji koriste semantičke značajke izgrađene primjenom modela na leksičke jedinke izvornog programskog kôda. Rezultati vrednovanja REPD modela pokazuju superiornost njegova rada povećavajući iznos ostvarene F1 mjere za 7.12 %. Za kraj analiziran je utjecaj neuravnoteženosti razreda na rad REPD modela. Metoda izgradnje semantičkih značajki primjenom autoenkoder modela na leksičke jedinke izvornog programskog kôda i REPD metoda za predviđanje pogrešaka programske potpore zasnovana na ideji otkrivanja anomalija predstavljaju znanstvene doprinose autora i objavljeni su u obliku znanstvenog članka [37], a programski kôd napravljen je javno dostupnim [39].

Rezultati predstavljenu u ovoj disertaciji predstavljaju plod rada u sklopu doktorskog istraživanja autora. Dobiveni rezultati pokazuju utjecaj kvalitete klasifikacije zadataka na kvalitetu podatkovnih skupova i rad modela predviđanja pogrešaka programske potpore te pokazuju da je problemu predviđanja pogrešaka programske potpore moguće pristupiti tretirajući ga kao problem otkrivanja anomalija. Predloženi model REPD postiže rezultate sumjerljive s rezultatima

drugih modela u području.

Sav programski kôd i podatkovni skupovi razvijeni u sklopu provedenog istraživanja javno su dostupni. Nada autora je da će time motivirati druge znanstveno-istraživačke skupine iz područja predviđanja pogrešaka programske potpore na daljnja istraživanja.

Dodatak A

PROMISE

U Tablici 6.1 navedene su značajke PROMISE projekata korištenih u provedenom istraživanju. Radi se o klasičnim značajkama za predviđanje pogrešaka programske potpore. U prvom stupcu navedena je oznaka značajke, a u drugom stupcu opis značajke.

Tablica 6.1: Značajke PROMISE projekata

Oznaka značajke	Opis značajke
loc	McCabeov broj linija kôda
v(g)	McCabeova ciklometrijska složenost
ev(g)	McCabeova esencijalna složenost
iv(g)	McCabeiva složenost dizajna
n	Halsteadov ukupan broj operatora i operanada
v	Halsteadov volumen
l	Halsteadova duljina programa
d	Halsteadova težina
i	Halsteadova inteligencija
e	Halsteadov trud
b	Halsteadova metrika
t	Halsteadova vremenska procjena
lOCode	Halsteadov broj linija
lOComment	Halsteadov broj linija komentara
lOBlank	Halsteadov broj praznih linija
lOCodeAndComment	Halsteadov broj linija i linija komentara
uniq_Op	Broj jedinstvenih operatora
uniq_Opnd	Broj jedinstvenih operanada
total_Op	Ukupan broj operatora
total_Opnd	Ukupan broj operanada
branchCount	Broj grana u grafu izvršavanja

Literatura

[1]Goldstine, H. H., Goldstine, A., "The electronic numerical integrator and computer (eniac)", IEEE Ann. Hist. Comput., Vol. 18, No. 1, mar 1996, str. 10–16, dostupno na: https://doi.org/10.1109/85.476557

[2]Chomsky, N., "Three models for the description of language", IRE Transactions on Information Theory, Vol. 2, No. 3, 1956, str. 113-124.

[3]Gustafson, J. L., Moore's Law. Boston, MA: Springer US, 2011, str. 1177–1184, dostupno na: https://doi.org/10.1007/978-0-387-09766-4_81

[4]Walter, C., "Kryder's law.", Scientific American, Vol. 293 2, 2005, str. 32-3.

[5]Sandeep Kumar, S. S. R., Software Fault Prediction - A Road Map. Springer Singapore, 2018.

[6]de Loisy, N., Transportation and the Belt and Road Initiative: A Paradigm Shift (color Edition). Supply Chain Management Outsource Limited, 2019, dostupno na: https://books.google.hr/books?id=3TxKzQEACAAJ

[7]Tanenbaum, A. S., Bos, H., Modern Operating Systems, 4th ed. Boston, MA: Pearson, 2014.

[8]Ghaffarian, S. M., Shahriari, H. R., "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey", ACM Comput. Surv., Vol. 50, No. 4, aug 2017, dostupno na: https://doi.org/10.1145/3092566

[9]Thota, M. K., Shajin, F. H., Rajesh, P., "Survey on software defect prediction techniques", International Journal of Applied Science and Engineering, Vol. 17, December 2020, str. 331–344, dostupno na: https://doi.org/10.6703/IJASE.202012_17(4).331

[10]Krasner, H., "The cost of poor software quality in the us: A 2020 report.", Proc. Consortium Inf. Softw. QualityTM (CISQTM), 2021.

[11]“The global talent crunch”, https://www.kornferry.com/content/dam/kornferry/docs/pdfs/KF-Future-of-Work-Talent-Crunch-Report.pdf, accessed: 23.03.2023.

[12]Busechian, S., Ivanov, V., Rogers, A., Sirazitdinov, I., Succi, G., Tormasov, A., Yi, J., “Understanding the impact of pair programming on the minds of developers”, in 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER), 2018, str. 85-88.

[13]Qazi, A., Shahzadi, S., Humayun, M., “A comparative study of software inspection techniques for quality perspective”, International Journal of Modern Education and Computer Science (IJMECS), Vol. 10, 10 2016, str. 9-16.

[14]Fronza, I., Hellas, A., Ihantola, P., Mikkonen, T., “Code reviews, software inspections, and code walkthroughs: Systematic mapping study of research topics”, in Software Quality: Quality Intelligence in Software and Systems Engineering. Springer International Publishing, Dec. 2019, str. 121–133, dostupno na: https://doi.org/10.1007/978-3-030-35510-4_8

[15]Heiser, J. E., “An overview of software testing”, in 1997 IEEE Autotestcon Proceedings AUTOTESTCON ’97. IEEE Systems Readiness Technology Conference. Systems Readiness Supporting Global Needs and Awareness in the 21st Century, 1997, str. 204-211.

[16]Lewis, W. E., Software Testing and Continuous Quality Improvement, Third Edition, 2nd ed. Boston, MA, USA: Auerbach Publications, 2008.

[17]Wahono, R. S., “A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks”, Journal of Software Engineering, Vol. 1, 2007, str. 1-16.

[18]Hryszko, J., Madeyski, L., “Assessment of the software defect prediction cost effectiveness in an industrial project”, in Software Engineering: Challenges and Solutions. Springer, 2017, str. 77–90.

[19]Zhang, H., Gong, L., Versteeg, S., “Predicting bug-fixing time: an empirical study of commercial software projects”, in 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013, str. 1042–1051.

[20]Jiang, Y., Cukic, B., Ma, Y., “Techniques for evaluating fault prediction models”, Empirical Software Engineering, Vol. 13, No. 5, Aug. 2008, str. 561–595, dostupno na: https://doi.org/10.1007/s10664-008-9079-3

[21]Li, Z., Jing, X.-Y., Zhu, X., “Progress on approaches to software defect prediction”, IET Software, Vol. 12, 02 2018, str. 161-175.

[22]Abaei, G., Selamat, A., “A survey on software fault detection based on different prediction approaches”, Vietnam Journal of Computer Science, Vol. 1, No. 2, May 2014, str. 79-95, dostupno na: https://doi.org/10.1007/s40595-013-0008-z

[23]Catal, C., Diri, B., “A systematic review of software fault prediction studies”, Expert Systems with Applications, Vol. 36, No. 4, 2009, str. 7346-7354, dostupno na: https://www.sciencedirect.com/science/article/pii/S0957417408007215

[24]Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.-G., “Is it a bug or an enhancement? a text-based approach to classify change requests”, in Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, ser. CASCON ’08. New York, NY, USA: Association for Computing Machinery, 2008, dostupno na: https://doi.org/10.1145/1463788.1463819

[25]Herzig, K., Just, S., Zeller, A., “It’s not a bug, it’s a feature: How misclassification impacts bug prediction”, in 2013 35th International Conference on Software Engineering (ICSE), 2013, str. 392-401.

[26]Pandey, N., Sanyal, D. K., Hudait, A., Sen, A., “Automated classification of software issue reports using machine learning techniques: an empirical study”, Innovations in Systems and Software Engineering, Vol. 13, No. 4, Dec 2017, str. 279-297, dostupno na: https://doi.org/10.1007/s11334-017-0294-1

[27]Ohira, M., Yoshiyuki, H., Yamatani, Y., “A case study on the misclassification of software performance issues in an issue tracking system”, in 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), 2016, str. 1-6.

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

[29]Zhang, H., Zhang, X., “Comments on "data mining static code attributes to learn defect predictors"”, IEEE Transactions on Software Engineering, Vol. 33, No. 9, 2007, str. 635-637.

[30]Tan, M., Tan, L., Dara, S., Mayeux, C., “Online defect prediction for imbalanced data”, in Proceedings of the 37th International Conference on Software Engineering - Volume 2, ser. ICSE ’15. Piscataway, NJ, USA: IEEE Press, 2015, str. 99–108, dostupno na: http://dl.acm.org/citation.cfm?id=2819009.2819026

[31]Bennin, K. E., Keung, J., Phannachitta, P., Monden, A., Mensah, S., “Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect

prediction", IEEE Transactions on Software Engineering, Vol. 44, No. 6, June 2018, str. 534-550.

[32]Raeder, T., Forman, G., Chawla, N. V., Learning from Imbalanced Data: Evaluation Matters. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, str. 315–331, dostupno na: https://doi.org/10.1007/978-3-642-23166-7_12

[33]Yu, X., Liu, J., Yang, Z., Jia, X., Ling, Q., Ye, S., "Learning from imbalanced data for predicting the number of software defects", in 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Oct 2017, str. 78-89.

[34]He, H., Garcia, E. A., "Learning from imbalanced data", IEEE Transactions on Knowledge and Data Engineering, Vol. 21, No. 9, 2009, str. 1263-1284.

[35]Goyal, S., "Handling class-imbalance with knn (neighbourhood) under-sampling for software defect prediction", Artificial Intelligence Review, Vol. 55, No. 3, Mar 2022, str. 2023-2064, dostupno na: https://doi.org/10.1007/s10462-021-10044-w

[36]Afric, P., Vukadin, D., Silic, M., Delac, G., "Empirical study: How issue classification influences software defect prediction", IEEE Access, Vol. 11, 2023, str. 11 732-11 748.

[37]Afric, P., Sikic, L., Kurdija, A. S., Silic, M., "Repd: Source code defect prediction as anomaly detection", Journal of Systems and Software, Vol. 168, 2020, str. 110641, dostupno na: https://www.sciencedirect.com/science/article/pii/S0164121220301138

[38]Afri ć, P., "How-issue-classification-influences-software-defect-prediction", https://github.com/pa1511/Empirical-Study-How-Issue-Classification-Influences-Software-Defect-Prediction accessed: 22.03.2023.

[39]Afri ć, P., "Repd model-source code", https://github.com/pa1511/REPD, accessed: 22.03.2023.

[40]Akiyama, F., "An example of software system debugging.", in IFIP Congress (1), Freiman, C. V., Griffith, J. E., Rosenfeld, J. L., (ur.). North-Holland, 1971, str. 353-359, dostupno na: http://dblp.uni-trier.de/db/conf/ifip/ifip71-1.html#Akiyama71

[41]Zimmermann, T., Premraj, R., Zeller, A., "Predicting defects for eclipse", in Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), 2007, str. 9-9.

[42]Cubranic, D., Murphy, G., "Hipikat: recommending pertinent software development artifacts", in 25th International Conference on Software Engineering, 2003. Proceedings., 2003, str. 408-418.

[43]Fischer, M., Pinzger, M., Gall, H., “Populating a release history database from version control and bug tracking systems”, in International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., 2003, str. 23-32.

[44]undefinedliwerski, J., Zimmermann, T., Zeller, A., “When do changes induce fixes?”, SIGSOFT Softw. Eng. Notes, Vol. 30, No. 4, may 2005, str. 1–5, dostupno na: https://doi.org/10.1145/1082983.1083147

[45]Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B., “The misuse of the nasa metrics data program data sets for automated software defect prediction”, in 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 2011, str. 96-103.

[46]Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., Riquelme, J. C., “Preliminary comparison of techniques for dealing with imbalance in software defect prediction”, in Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ser. EASE ’14. New York, NY, USA: Association for Computing Machinery, 2014, dostupno na: https://doi.org/10.1145/2601248.2601294

[47]Mahmood, Z., Bowes, D., Lane, P. C. R., Hall, T., “What is the impact of imbalance on software defect prediction performance?”, in Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, ser. PROMISE ’15. New York, NY, USA: Association for Computing Machinery, 2015, dostupno na: https://doi.org/10.1145/2810146.2810150

[48]Kabir, M. A., Keung, J. W., Bennin, K. E., Zhang, M., “Assessing the significant impact of concept drift in software defect prediction”, in 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 1, 2019, str. 53-58.

[49]Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., Matsumoto, K., “The impact of mislabelling on the performance and interpretation of defect prediction models”, in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, 2015, str. 812-823.

[50]Xu, Z., Liu, J., Yang, Z., An, G., Jia, X., “The impact of feature selection on defect prediction performance: An empirical comparison”, in 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016, str. 309-320.

[51]Liu, S., Chen, X., Liu, W., Chen, J., Gu, Q., Chen, D., “Fecar: A feature selection framework for software defect prediction”, in 2014 IEEE 38th Annual Computer Software and Applications Conference, 2014, str. 426-435.

[52]Liu, W., Liu, S., Gu, Q., Chen, X., Chen, D., “Fecs: A cluster based feature selection method for software fault prediction with noises”, in 2015 IEEE 39th Annual Computer Software and Applications Conference, Vol. 2, 2015, str. 276-281.

[53]Xu, Z., Xuan, J., Liu, J., Cui, X., “Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering”, in 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 1, 2016, str. 370-381.

[54]Shivaji, S., Whitehead, E. J., Akella, R., Kim, S., “Reducing features to improve bug prediction”, in 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, str. 600-604.

[55]Kakkar, M., Jain, S., “Feature selection in software defect prediction: A comparative study”, in 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), 2016, str. 658-663.

[56]Menzies, T., Greenwald, J., Frank, A., “Data mining static code attributes to learn defect predictors”, IEEE Transactions on Software Engineering, Vol. 33, No. 1, 2007, str. 2-13.

[57]Nam, J., Pan, S. J., Kim, S., “Transfer defect learning”, in 2013 35th International Conference on Software Engineering (ICSE), 2013, str. 382-391.

[58]Kim, S., Zhang, H., Wu, R., Gong, L., “Dealing with noise in defect prediction”, in 2011 33rd International Conference on Software Engineering (ICSE), 2011, str. 481-490.

[59]Wu, R., Zhang, H., Kim, S., Cheung, S.-C., “Relink: Recovering links between bugs and changes”, in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE ’11. New York, NY, USA: Association for Computing Machinery, 2011, str. 15–25, dostupno na: https://doi.org/10.1145/2025113.2025120

[60]Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., Devanbu, P., “Fair and balanced? bias in bug-fix datasets”, in Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ser. ESEC/FSE ’09. New York, NY, USA: Association for Computing Machinery, 2009, str. 121–130, dostupno na: https://doi.org/10.1145/1595696.1595716

[61]Bachmann, A., Bernstein, A., “Software process data quality and characteristics: a historical view on open and closed source projects”, in IWPSE-Evol ’09, 2009.

[62]Mockus, Votta, “Identifying reasons for software changes using historic databases”, in Proceedings 2000 International Conference on Software Maintenance, 2000, str. 120-130.

[63]Rahman, F., Posnett, D., Herraiz, I., Devanbu, P., “Sample size vs. bias in defect prediction”, in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, str. 147–157, dostupno na: https://doi.org/10.1145/2491411.2491418

[64]Nguyen, A. T., Nguyen, T. T., Nguyen, H. A., Nguyen, T. N., “Multi-layered approach for recovering links between bug reports and fixes”, in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ser. FSE ’12. New York, NY, USA: Association for Computing Machinery, 2012, dostupno na: https://doi.org/10.1145/2393596.2393671

[65]Khan, S., Azmain, M., Niloy, N., Kabir, A., “Impact of label noise and efficacy of noise filters in software defect prediction”, in International Conference on Software Engineering and Knowledge Engineering, 07 2020.

[66]Jing, X.-Y., Wu, F., Dong, X., Xu, B., “An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems”, IEEE Transactions on Software Engineering, Vol. 43, No. 4, 2017, str. 321-339.

[67]Malhotra, R., Khanna, M., “An empirical study for software change prediction using imbalanced data”, Empirical Software Engineering, Vol. 22, No. 6, Dec 2017, str. 2806-2851, dostupno na: https://doi.org/10.1007/s10664-016-9488-7

[68]Bennin, K. E., Keung, J., Monden, A., Kamei, Y., Ubayashi, N., “Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models”, in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Vol. 1, 2016, str. 154-163.

[69]Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., Folleco, A., “An empirical study of the classification performance of learners on imbalanced and noisy software quality data”, Information Sciences, Vol. 259, 2014, str. 571-595, dostupno na: https://www.sciencedirect.com/science/article/pii/S0020025511000065

[70]Pandey, S. K., Tripathi, A. K., “An empirical study toward dealing with noise and class imbalance issues in software defect prediction”, Soft Computing, Vol. 25, No. 21, Nov 2021, str. 13 465-13 492, dostupno na: https://doi.org/10.1007/s00500-021-06096-3

[71]Shepperd, M., Song, Q., Sun, Z., Mair, C., “Data quality: Some comments on the nasa software defect datasets”, IEEE Transactions on Software Engineering, Vol. 39, No. 9, 2013, str. 1208-1215.

[72]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.

[73]Mauša, G., Grbac, T. G., Baši ć, B. D., “A systematic data collection procedure for software defect prediction”, Computer Science and Information Systems, Vol. 13, No. 1, 2016, str. 173-197.

[74]Ferenc, R., Tóth, Z., Ladányi, G., Siket, I., Gyimóthy, T., “A public unified bug dataset for java”, in Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, ser. PROMISE’18. New York, NY, USA: Association for Computing Machinery, 2018, str. 12–21, dostupno na: https://doi.org/10.1145/3273934.3273936

[75]Ferenc, R., Tóth, Z., Ladányi, G., Siket, I., Gyimóthy, T., “A public unified bug dataset for java and its assessment regarding metrics and bug prediction”, Software Quality Journal, Vol. 28, No. 4, Dec 2020, str. 1447-1506, dostupno na: https://doi.org/10.1007/s11219-020-09515-0

[76]McCabe, T., “A complexity measure”, IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, 1976, str. 308-320.

[77]Halstead, M. H., Elements of Software Science (Operating and Programming Systems Series). USA: Elsevier Science Inc., 1977.

[78]Hitz, M., Montazeri, B., “Chidamber and kemerer’s metrics suite: a measurement theory perspective”, IEEE Transactions on Software Engineering, Vol. 22, No. 4, 1996, str. 267-271.

[79]Harrison, R., Counsell, S. J., Nithi, R. V., “An evaluation of the mood set of object-oriented software metrics”, IEEE Transactions on Software Engineering, Vol. 24, No. 6, 1998, str. 491-496.

[80]Lorenz, M., Kidd, J., Object-Oriented Software Metrics: A Practical Guide. USA: Prentice-Hall, Inc., 1994.

[81]Briand, L., Devanbu, P., Melo, W., “An investigation into coupling measures for c++”, in Proceedings of the (19th) International Conference on Software Engineering, 1997, str. 412-421.

[82]Li, W., Henry, S., “Object-oriented metrics that predict maintainability”, Journal of Systems and Software, Vol. 23, No. 2, 1993, str. 111-122, object-Oriented Software, dostupno na: https://www.sciencedirect.com/science/article/pii/016412129390077B

[83]Bansiya, J., Davis, C. G., “A hierarchical model for object-oriented design quality assessment”, IEEE Transactions on Software Engineering, Vol. 28, No. 1, 2002, str. 4-17.

[84]Misra, S., Akman, I., “Measuring complexity of object oriented programs”, in Computational Science and Its Applications – ICCSA 2008. Springer Berlin Heidelberg, 2008, str. 652–667, dostupno na: https://doi.org/10.1007/978-3-540-69848-7_53

[85]Yacoub, S. M., Ammar, H. H., Robinson, T., “Dynamic metrics for object oriented designs”, in Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), 1999, str. 50-61.

[86]Arisholm, E., Briand, L. C., Foyen, A., “Dynamic coupling measurement for object-oriented software”, IEEE Transactions on Software Engineering, Vol. 30, No. 8, 2004, str. 491-506.

[87]Áine Mitchell, Power, J. F., “A study of the influence of coverage on the relationship between static and dynamic coupling metrics”, Science of Computer Programming, Vol. 59, No. 1, 2006, str. 4-25, special Issue on Principles and Practices of Programming in Java (PPPJ 2004), dostupno na: https://www.sciencedirect.com/science/article/pii/S0167642305000821

[88]Tahir, A., MacDonell, S. G., “A systematic mapping study on dynamic metrics and software quality”, in 2012 28th IEEE International Conference on Software Maintenance (ICSM), 2012, str. 326-335.

[89]Wang, S., Liu, T., Tan, L., “Automatically learning semantic features for defect prediction”, in Proceedings of the 38th International Conference on Software Engineering, ser. ICSE ’16. New York, NY, USA: Association for Computing Machinery, 2016, str. 297–308, dostupno na: https://doi.org/10.1145/2884781.2884804

[90]Wang, S., Liu, T., Nam, J., Tan, L., “Deep semantic feature learning for software defect prediction”, IEEE Transactions on Software Engineering, Vol. 46, No. 12, 2020, str. 1267-1293.

[91]Huo, X., Yang, Y., Li, M., Zhan, D.-C., “Learning semantic features for software defect prediction by code comments embedding”, in 2018 IEEE International Conference on Data Mining (ICDM), 2018, str. 1049-1054.

[92]Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C. B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M., "Graphcodebert: Pre-training code representations with data flow", CoRR, Vol. abs/2009.08366, 2020, dostupno na: https://arxiv.org/abs/2009.08366

[93]Jiang, T., Tan, L., Kim, S., "Personalized defect prediction", in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Nov 2013, str. 279-289.

[94]Pinzger, M., Nagappan, N., Murphy, B., "Can developer-module networks predict failures?", in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16. ACM Press, 2008, dostupno na: https://doi.org/10.1145/1453101.1453105

[95]Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P., "Don't touch my code! examining the effects of ownership on software quality", in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, 2011, str. 4–14, dostupno na: https://doi.org/10.1145/2025113.2025119

[96]Matsumoto, S., Kamei, Y., Monden, A., Matsumoto, K.-i., Nakamura, M., "An analysis of developer metrics for fault prediction", in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ser. PROMISE '10. New York, NY, USA: Association for Computing Machinery, 2010, dostupno na: https://doi.org/10.1145/1868328.1868356

[97]Madeyski, L., Jureczko, M., "Which process metrics can significantly improve defect prediction models? an empirical study", Software Quality Journal, Vol. 23, No. 3, Jun. 2014, str. 393–422, dostupno na: https://doi.org/10.1007/s11219-014-9241-7

[98]Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., Murphy, B., "Change bursts as defect predictors", in 2010 IEEE 21st International Symposium on Software Reliability Engineering, 2010, str. 309-318.

[99]Nagappan, N., Ball, T., "Use of relative code churn measures to predict system defect density", in Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., 2005, str. 284-292.

[100]Šiki ć, L., Afrić, P., Kurdija, A. S., ŠIlić, M., "Improving software defect prediction by aggregated change metrics", IEEE Access, Vol. 9, 2021, str. 19 391-19 411.

[101]Moser, R., Pedrycz, W., Succi, G., "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction", in Proceedings of the 13th international conference on Software engineering - ICSE '08. ACM Press, 2008, dostupno na: https://doi.org/10.1145/1368088.1368114

[102]Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A., Catal, C., "Empirical analysis of change metrics for software fault prediction", Computers & Electrical Engineering, Vol. 67, 2018, str. 15-24, dostupno na: https://www.sciencedirect.com/science/article/pii/S0045790617336121

[103]Rhmann, W., Pandey, B., Ansari, G., Pandey, D., "Software fault prediction based on change metrics using hybrid algorithms: An empirical study", Journal of King Saud University - Computer and Information Sciences, Vol. 32, No. 4, 2020, str. 419-424, emerging Software Systems, dostupno na: https://www.sciencedirect.com/science/article/pii/S1319157818313077

[104]Meneely, A., Williams, L. A., Snipes, W., Osborne, J. A., "Predicting failures with developer networks and social network analysis", in SIGSOFT FSE, 2008.

[105]Bacchelli, A., D'Ambros, M., Lanza, M., "Are popular classes more defect prone?", in Fundamental Approaches to Software Engineering, Rosenblum, D. S., Taentzer, G., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, str. 59–73.

[106]Byun, J., Rhew, S., Hwang, M., Sugumara, V., Park, S., Park, S., "Metrics for measuring the consistencies of requirements with objectives and constraints", Requirements Engineering, Vol. 19, No. 1, Mar 2014, str. 89-104, dostupno na: https://doi.org/10.1007/s00766-013-0180-9

[107]Premraj, R., Herzig, K., "Network versus code metrics to predict defects: A replication study", in 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, str. 215-224.

[108]Mnkandla, E., Mpofu, B., "Software defect prediction using process metrics elasticsearch engine case study", in 2016 International Conference on Advances in Computing and Communication Engineering (ICACCE), 2016, str. 254-260.

[109]Jureczko, M., Madeyski, L., "A review of process metrics in defect prediction studies", Methods of Applied Computer Science, Vol. 5, 01 2011, str. 133-145.

[110]Ramadhina, S., Bahawares, R. B., Hermadi, I., Suroso, A. I., Rodoni, A., Arkeman, Y., "Software defect prediction using process metrics systematic literature review: Dataset

and granularity level", in 2021 9th International Conference on Cyber and IT Service Management (CITSM), 2021, str. 1-7.

[111]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, May 2010, str. 375–407, dostupno na: https://doi.org/10.1007/s10515-010-0069-5

[112]Nagappan, N., Ball, T., Zeller, A., "Mining metrics to predict component failures", in Proceedings of the 28th International Conference on Software Engineering, ser. ICSE '06. New York, NY, USA: Association for Computing Machinery, 2006, str. 452–461, dostupno na: https://doi.org/10.1145/1134285.1134349

[113]Graves, T. L., Karr, A. F., Marron, J. S., Siy, H., "Predicting fault incidence using software change history", IEEE Transactions on Software Engineering, Vol. 26, No. 7, July 2000, str. 653-661.

[114]Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L., "Early prediction of software component reliability", in 2008 ACM/IEEE 30th International Conference on Software Engineering, May 2008, str. 111-120.

[115]Zhang, H., Zhang, X., Gu, M., "Predicting defective software components from code complexity measures", in 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), Dec 2007, str. 93-96.

[116]Calikli, G., Tosun, A., Bener, A., Celik, M., "The effect of granularity level on software defect prediction", in 2009 24th International Symposium on Computer and Information Sciences, 2009, str. 531-536.

[117]Yan, M., Fang, Y., Lo, D., Xia, X., Zhang, X., "File-level defect prediction: Unsupervised vs. supervised models", in 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, str. 344-353.

[118]Bettenburg, N., Nagappan, M., Hassan, A. E., "Think locally, act globally: Improving defect and effort prediction models", in 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), June 2012, str. 60-69.

[119]Kim, S., Zimmermann, T., Whitehead Jr., E. J., Zeller, A., "Predicting faults from cached history", in 29th International Conference on Software Engineering (ICSE'07), 2007, str. 489-498.

[120]Ostrand, T. J., Weyuker, E. J., Bell, R. M., “Predicting the location and number of faults in large software systems”, IEEE Transactions on Software Engineering, Vol. 31, No. 4, April 2005, str. 340-355.

[121]Zhang, H., “An investigation of the relationships between lines of code and defects”, in 2009 IEEE International Conference on Software Maintenance, Sep. 2009, str. 274-283.

[122]Scanniello, G., Gravino, C., Marcus, A., Menzies, T., “Class level fault prediction using software clustering”, in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, str. 640-645.

[123]Patil, S., Rao, A. N., Bindu, C. S., “Class level software fault prediction using step wise linear regression”, International Journal of Engineering & Technology, Vol. 7, No. 4, Sep. 2018, str. 2552, dostupno na: https://doi.org/10.14419/ijet.v7i2.17.14881

[124]Shippey, T., Hall, T., Counsell, S., Bowes, D., “So you need more method level datasets for your software defect prediction? voilà!”, in Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ser. ESEM ’16. New York, NY, USA: Association for Computing Machinery, 2016, dostupno na: https://doi.org/10.1145/2961111.2962620

[125]Giger, E., D’Ambros, M., Pinzger, M., Gall, H. C., “Method-level bug prediction”, in Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2012, str. 171-180.

[126]Koru, A. G., Liu, H., “An investigation of the effect of module size on defect prediction using static measures”, SIGSOFT Softw. Eng. Notes, Vol. 30, No. 4, May 2005, str. 1–5, dostupno na: http://doi.acm.org/10.1145/1082983.1083172

[127]Tessema, H. D., Abebe, S. L., “Enhancing just-in-time defect prediction using change request-based metrics”, in 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021, str. 511-515.

[128]Kim, S., Whitehead,, E. J., Zhang, Y., “Classifying software changes: Clean or buggy?”, IEEE Transactions on Software Engineering, Vol. 34, No. 2, 2008, str. 181-196.

[129]Hassan, A. E., “Predicting faults using the complexity of code changes”, in 2009 IEEE 31st International Conference on Software Engineering, May 2009, str. 78-88.

[130]Prechelt, L., Pepper, A., “Why software repositories are not used for defect-insertion circumstance analysis more often: A case study”, Information and Software Technology, Vol. 56, No. 10, 2014, str. 1377 - 1389, dostupno na: http://www.sciencedirect.com/science/article/pii/S0950584914001049

[131]Kim, S., Zimmermann, T., Pan, K., Jr. Whitehead, E. J., "Automatic identification of bug-introducing changes", in 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), Sep. 2006, str. 81-90.

[132]Qing, H., Biwen, L., Beijun, S., Xia, Y., "Cross-project software defect prediction using feature-based transfer learning", in Proceedings of the 7th Asia-Pacific Symposium on Internetware, ser. Internetware '15. New York, NY, USA: Association for Computing Machinery, 2015, str. 74–82, dostupno na: https://doi.org/10.1145/2875913.2875944

[133]Erturk, E., Sezer, E. A., "A comparison of some soft computing methods for software fault prediction", Expert Systems with Applications, Vol. 42, No. 4, 2015, str. 1872-1879, dostupno na: https://www.sciencedirect.com/science/article/pii/S0957417414006496

[134]Catal, C., "Software fault prediction: A literature review and current trends", Expert Systems with Applications, Vol. 38, No. 4, 2011, str. 4626-4636, dostupno na: https://www.sciencedirect.com/science/article/pii/S0957417410011681

[135]Li, M., Zhang, H., Wu, R., Zhou, Z.-H., "Sample-based software defect prediction with active and semi-supervised learning", Automated Software Engineering, Vol. 19, No. 2, Jul. 2011, str. 201–230, dostupno na: https://doi.org/10.1007/s10515-011-0092-1

[136]Bhutamapuram, U. S., Sadam, R., "With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique", Journal of King Saud University - Computer and Information Sciences, Vol. 34, No. 10, Part A, 2022, str. 8675-8691, dostupno na: https://www.sciencedirect.com/science/article/pii/S1319157821002615

[137]Malohtra, R., Yadav, H. S., "An improved cnn-based architecture for within-project software defect prediction", in Soft Computing and Signal Processing, Reddy, V. S., Prasad, V. K., Wang, J., Reddy, K. T. V., (ur.). Singapore: Springer Singapore, 2021, str. 335–349.

[138]Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D. A., "Within-project defect prediction of infrastructure-as-code using product and process metrics", IEEE Transactions on Software Engineering, Vol. 48, No. 6, 2022, str. 2086-2104.

[139]Liang, M., Li, D., Xu, B., Zhao, D., Yu, X., Xiang, J., "Within-project software aging defect prediction based on active learning", in 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2021, str. 1-8.

[140]Šiki ć, L., Kurdija, A. S., Vladimir, K., Šilić, M., "Graph neural network for source code defect prediction", IEEE Access, Vol. 10, 2022, str. 10 402-10 415.

[141]Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B., "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process", in Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ser. ESEC/FSE '09. New York, NY, USA: Association for Computing Machinery, 2009, str. 91–100, dostupno na: https://doi.org/10.1145/1595696.1595713

[142]He, Z., Shu, F., Yang, Y., Li, M., Wang, Q., "An investigation on the feasibility of cross-project defect prediction", Automated Software Engineering, Vol. 19, No. 2, Jul. 2011, str. 167–199, dostupno na: https://doi.org/10.1007/s10515-011-0090-3

[143]Ma, Y., Luo, G., Zeng, X., Chen, A., "Transfer learning for cross-company software defect prediction", Information and Software Technology, Vol. 54, No. 3, 2012, str. 248-256, dostupno na: https://www.sciencedirect.com/science/article/pii/S0950584911001996

[144]Peters, F., Menzies, T., Marcus, A., "Better cross company defect prediction", in 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, May 2013, dostupno na: https://doi.org/10.1109/msr.2013.6624057

[145]Pan, S. J., Yang, Q., "A survey on transfer learning", IEEE Transactions on Knowledge and Data Engineering, Vol. 22, No. 10, 2010, str. 1345-1359.

[146]Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., Xu, B., "How far we have progressed in the journey? an examination of cross-project defect prediction", ACM Trans. Softw. Eng. Methodol., Vol. 27, No. 1, Apr. 2018, dostupno na: https://doi.org/10.1145/3183339

[147]Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S., "Multi-objective cross-project defect prediction", in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, str. 252-261.

[148]Panichella, A., Oliveto, R., De Lucia, A., "Cross-project defect prediction models: L'union fait la force", in 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, str. 164-173.

[149]Watanabe, S., Kaiya, H., Kaijiri, K., "Adapting a fault prediction model to allow inter languagereuse", in Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, ser. PROMISE '08. New York, NY, USA: Association for Computing Machinery, 2008, str. 19–24, dostupno na: https://doi.org/10.1145/1370788.1370794

[150]Chen, J., Wang, X., Cai, S., Xu, J., Chen, J., Chen, H., "A software defect prediction method with metric compensation based on feature selection and transfer learning", Frontiers of Information Technology & Electronic Engineering, Vol. 23, No. 5, May 2022, str. 715-731, dostupno na: https://doi.org/10.1631/FITEE.2100468

[151]Turhan, B., Menzies, T., Bener, A. B., Di Stefano, J., "On the relative value of cross-company and within-company data for defect prediction", Empirical Software Engineering, Vol. 14, No. 5, Oct 2009, str. 540-578, dostupno na: https://doi.org/10.1007/s10664-008-9103-7

[152]Porto, F., Minku, L., Mendes, E., Simao, A., "A systematic study of cross-project defect prediction with meta-learning", dostupno na: https://arxiv.org/abs/1802.06025 2018.

[153]Pan, S. J., Tsang, I. W., Kwok, J. T., Yang, Q., "Domain adaptation via transfer component analysis", IEEE Transactions on Neural Networks, Vol. 22, No. 2, 2011, str. 199-210.

[154]Pal, S., "Generative adversarial network-based cross-project fault prediction", CoRR, Vol. abs/2105.07207, 2021, dostupno na: https://arxiv.org/abs/2105.07207

[155]Xia, X., Lo, D., Pan, S. J., Nagappan, N., Wang, X., "Hydra: Massively compositional model for cross-project defect prediction", IEEE Transactions on Software Engineering, Vol. 42, No. 10, 2016, str. 977-998.

[156]Jin, C., "Cross-project software defect prediction based on domain adaptation learning and optimization", Expert Systems with Applications, Vol. 171, 2021, str. 114637, dostupno na: https://www.sciencedirect.com/science/article/pii/S0957417421000786

[157]Ding, S., Yu, J., Qi, B., Huang, H., "An overview on twin support vector machines", Artificial Intelligence Review, Vol. 42, No. 2, Mar. 2012, str. 245–252, dostupno na: https://doi.org/10.1007/s10462-012-9336-0

[158]Sun, J., Feng, B., Xu, W., "Particle swarm optimization with particles having quantum behavior", in Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Vol. 1, 2004, str. 325-331 Vol.1.

[159]Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L., "Heterogeneous defect prediction", IEEE Transactions on Software Engineering, Vol. 44, No. 9, 2018, str. 874-896.

[160]Chen, X., Mu, Y., Liu, K., Cui, Z., Ni, C., "Revisiting heterogeneous defect prediction methods: How far are we?", Information and Software Technology, Vol. 130, 2021, str. 106441, dostupno na: https://www.sciencedirect.com/science/article/pii/S0950584920301944

[161]Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., Hassan, A. E., “Studying just-in-time defect prediction using cross-project models”, Empirical Software Engineering, Vol. 21, No. 5, Sep. 2015, str. 2072–2106, dostupno na: https://doi.org/10.1007/s10664-015-9400-x

[162]Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., Ubayashi, N., “An empirical study of just-in-time defect prediction using cross-project models”, in Proceedings of the 11th Working Conference on Mining Software Repositories, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, str. 172–181, dostupno na: https://doi.org/10.1145/2597073.2597075

[163]Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., Leung, H., “Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models”, in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, str. 157–168, dostupno na: https://doi.org/10.1145/2950290.2950353

[164]Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J., “Deep learning for just-in-time defect prediction”, in 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015, str. 17-26.

[165]Pascarella, L., Palomba, F., Bacchelli, A., “Fine-grained just-in-time defect prediction”, Journal of Systems and Software, Vol. 150, 2019, str. 22-36, dostupno na: https://www.sciencedirect.com/science/article/pii/S0164121218302656

[166]Huang, Q., Xia, X., Lo, D., “Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction”, in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, str. 159-170.

[167]Yan, Z., Chen, X., Guo, P., “Software defect prediction using fuzzy support vector regression”, in Advances in Neural Networks - ISNN 2010. Springer Berlin Heidelberg, 2010, str. 17–24, dostupno na: https://doi.org/10.1007/978-3-642-13318-3_3

[168]Bell, R. M., Ostrand, T. J., Weyuker, E. J., “Looking for bugs in all the right places”, in Proceedings of the 2006 International Symposium on Software Testing and Analysis, ser. ISSTA ’06. New York, NY, USA: Association for Computing Machinery, 2006, str. 61–72, dostupno na: https://doi.org/10.1145/1146238.1146246

[169]Neela, K. N., Ali, S. A., Ami, A. S., Gias, A. U., “Modeling software defects as anomalies: A case study on promise repository”, JSW, Vol. 12, 2017, str. 759-772.

[170]Koh, K., Univariate Normal Distribution. Dordrecht: Springer Netherlands, 2014, str. 6817–6819, dostupno na: https://doi.org/10.1007/978-94-007-0753-5_3109

[171]Flury, B., The Multivariate Normal Distribution. New York, NY: Springer New York, 1997, str. 171–207, dostupno na: https://doi.org/10.1007/978-1-4757-2765-4_3

[172]Zhang, S., Jiang, S., Yan, Y., "A software defect prediction approach based on bigan anomaly detection", Scientific Programming, Vol. 2022, Apr 2022, str. 5024399, dostupno na: https://doi.org/10.1155/2022/5024399

[173]Moussa, R., Azar, D., Sarro, F., "Investigating the use of one-class support vector machine for software defect prediction", dostupno na: https://arxiv.org/abs/2202.12074 2022.

[174]Lomio, F., Pascarella, L., Palomba, F., Lenarduzzi, V., "Regularity or anomaly? on the use of anomaly detection for fine-grained jit defect prediction", in 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2022, str. 270-273.

[175]Akimova., E. N., Bersenev., A. V., Bersenev., A. Y., Cheshkov., A. Y., Deikov., A. Y., Deikov., A. V., Kobylkin., K. S., Konygin., A. V., Mezentsev., I. P., Misilov., V. E., "A-index: Semantic-based anomaly index for source code", in Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE,, INSTICC. SciTePress, 2022, str. 259-266.

[176]Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., Abraham, A., "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools", Engineering Applications of Artificial Intelligence, Vol. 111, 2022, str. 104773, dostupno na: https://www.sciencedirect.com/science/article/pii/S0952197622000616

[177]Khan, M. A., Elmitwally, N. S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M., Khan, F., "Software defect prediction using artificial neural networks: A systematic literature review", Scientific Programming, Vol. 2022, May 2022, str. 2117339, dostupno na: https://doi.org/10.1155/2022/2117339

[178]Akimova, E., Bersenev, A., Deikov, A., Kobylkin, K., Konygin, A., Mezentsev, I., Misilov, V., "A survey on software defect prediction using deep learning", Mathematics, Vol. 9, 05 2021, str. 1180.

[179]Yang, Y., Xia, X., Lo, D., Bi, T., Grundy, J., Yang, X., "Predictive models in software engineering: Challenges and opportunities", ACM Trans. Softw. Eng. Methodol., Vol. 31, No. 3, apr 2022, dostupno na: https://doi.org/10.1145/3503509

[180]Alsaeedi, A., Khan, M., "Software defect prediction using supervised machine learning and ensemble techniques: A comparative study", Journal of Software Engineering and Applications, Vol. 12, 01 2019, str. 85-100.

[181]Matloob, F., Aftab, S., Ahmad, M., Khan, M. A., Fatima, A., Iqbal, M., Alruwaili, W. M., Elmitwally, N. S., "Software defect prediction using supervised machine learning techniques: A systematic literature review", Intelligent Automation & Soft Computing, Vol. 29, No. 2, 2021, str. 403–421, dostupno na: http://www.techscience.com/iasc/v29n2/42941

[182]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, Nov 2012, str. 1276-1304.

[183]Jing, X.-Y., Ying, S., Zhang, Z.-W., Wu, S.-S., Liu, J., "Dictionary learning based software defect prediction", in Proceedings of the 36th International Conference on Software Engineering, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, str. 414–423, dostupno na: https://doi.org/10.1145/2568225.2568320

[184]Wang, H., Zhuang, W., Zhang, X., "Software defect prediction based on gated hierarchical lstms", IEEE Transactions on Reliability, Vol. 70, No. 2, 2021, str. 711-727.

[185]Wang, K., Liu, L., Yuan, C., Wang, Z., "Software defect prediction model based on lasso–svm", Neural Computing and Applications, Vol. 33, No. 14, Jul 2021, str. 8249-8259, dostupno na: https://doi.org/10.1007/s00521-020-04960-1

[186]Goyal, S., "Effective software defect prediction using support vector machines (svms)", International Journal of System Assurance Engineering and Management, Vol. 13, No. 2, Apr 2022, str. 681-696, dostupno na: https://doi.org/10.1007/s13198-021-01326-1

[187]Pan, C., Lu, M., Xu, B., "An empirical study on software defect prediction using codebert model", Applied Sciences, Vol. 11, No. 11, 2021, dostupno na: https://www.mdpi.com/2076-3417/11/11/4793

[188]Khurma, R. A., Alsawalqah, H., Aljarah, I., Elaziz, M. A., Damaševi čius, R., "An enhanced evolutionary software defect prediction method using island moth flame optimization", Mathematics, Vol. 9, No. 15, 2021, dostupno na: https://www.mdpi.com/2227-7390/9/15/1722

[189]Lu, H., Cukic, B., Culp, M., "A semi-supervised approach to software defect prediction", in 2014 IEEE 38th Annual Computer Software and Applications Conference, 2014, str. 416-425.

[190]Wu, F., Jing, X.-Y., Dong, X., Cao, J., Xu, M., Zhang, H., Ying, S., Xu, B., "Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution", in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017, str. 195-197.

[191]Li, N., Shepperd, M., Guo, Y., "A systematic review of unsupervised learning techniques for software defect prediction", Information and Software Technology, Vol. 122, 2020, str. 106287, dostupno na: https://www.sciencedirect.com/science/article/pii/S0950584920300379

[192]Fu, W., Menzies, T., "Revisiting unsupervised learning for defect prediction", in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, str. 72–83, dostupno na: http://doi.acm.org/10.1145/3106237.3106257

[193]Nam, J., Kim, S., "Clami: Defect prediction on unlabeled datasets (t)", in 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, str. 452-463.

[194]Zhang, F., Zheng, Q., Zou, Y., Hassan, A. E., "Cross-project defect prediction using a connectivity-based unsupervised classifier", in 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, str. 309-320.

[195]Alsawalqah, H., Faris, H., Aljarah, I., Alnemer, L., Alhindawi, N., "Hybrid smote-ensemble approach for software defect prediction", in Software Engineering Trends and Techniques in Intelligent Systems, Silhavy, R., Silhavy, P., Prokopova, Z., Senkerik, R., Kominkova Oplatkova, Z., (ur.). Cham: Springer International Publishing, 2017, str. 355–366.

[196]Khatri, Y., Singh, S. K., "Cross project defect prediction: a comprehensive survey with its swot analysis", Innovations in Systems and Software Engineering, Vol. 18, No. 2, Jun 2022, str. 263-281, dostupno na: https://doi.org/10.1007/s11334-020-00380-5

[197]Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S., Soomro, T. R., "Software defect prediction using ensemble learning: A systematic literature review", IEEE Access, Vol. 9, 2021, str. 98 754-98 771.

[198]Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., "Smote: Synthetic minority over-sampling technique", J. Artif. Int. Res., Vol. 16, No. 1, Jun. 2002, str. 321–357.

[199]Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., "CodeBERT: A pre-trained model for programming and natural languages", in Findings of the Association for Computational Linguistics: EMNLP 2020. Online: Association for Computational Linguistics, Nov. 2020, str. 1536–1547, dostupno na: https://aclanthology.org/2020.findings-emnlp.139

[200]Wang, T., Zhang, Z., Jing, X., Liu, Y., "Non-negative sparse-based semiboost for software defect prediction", Software Testing, Verification and Reliability, Vol. 26, No. 7, 2016, str. 498-515, dostupno na: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1610

[201]Zhang, Z.-W., Jing, X.-Y., Wang, T.-J., "Label propagation based semi-supervised learning for software defect prediction", Automated Software Engineering, Vol. 24, No. 1, Mar 2017, str. 47-69, dostupno na: https://doi.org/10.1007/s10515-016-0194-x

[202]Meng, F., Cheng, W., Wang, J., "Semi-supervised software defect prediction model based on tri-training", KSII Transactions on Internet and Information Systems, Vol. 15, No. 11, November 2021, str. 4028-4042.

[203]Chicco, D., Tötsch, N., Jurman, G., "The matthews correlation coefficient (mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation", BioData Mining, Vol. 14, No. 1, Feb 2021, str. 13, dostupno na: https://doi.org/10.1186/s13040-021-00244-z

[204]Dodge, Y., Kolmogorov–Smirnov Test. New York, NY: Springer New York, 2008, str. 283–287, dostupno na: https://doi.org/10.1007/978-0-387-32833-1_214

[205]Thode, H. C., Normality Tests. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, str. 999–1000, dostupno na: https://doi.org/10.1007/978-3-642-04898-2_423

[206]Haynes, W., Student's t-Test. New York, NY: Springer New York, 2013, str. 2023–2025, dostupno na: https://doi.org/10.1007/978-1-4419-9863-7_1184

[207]Dodge, Y., Mann–Whitney Test. New York, NY: Springer New York, 2008, str. 327–329, dostupno na: https://doi.org/10.1007/978-0-387-32833-1_243

[208]Hill, C. R., Thompson, B., Computing and Interpreting Effect Sizes. Dordrecht: Springer Netherlands, 2005, str. 175–196, dostupno na: https://doi.org/10.1007/1-4020-2456-8_5

[209]Wang, J., Zhang, X., Chen, L., “How well do pre-trained contextual language representations recommend labels for github issues?”, Knowledge-Based Systems, Vol. 232, 2021, str. 107476, dostupno na: https://www.sciencedirect.com/science/article/pii/S0950705121007383

[210]Herbold, S., Trautsch, A., Trautsch, F., “On the feasibility of automated issue type prediction”, CoRR, Vol. abs/2003.05357, 2020, dostupno na: https://arxiv.org/abs/2003.05357

[211]Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S., “Predicting issue types on github”, Science of Computer Programming, Vol. 205, 2021, str. 102598, dostupno na: https://www.sciencedirect.com/science/article/pii/S0167642320302069

[212]Siddiq, M. L., Santos, J. C. S., “Bert-based github issue report classification”, in 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE), 2022, str. 33-36.

[213]Li, Z., Pan, M., Pei, Y., Zhang, T., Wang, L., Li, X., “Deeplabel: Automated issue classification for issue tracking systems”, in Proceedings of the 13th Asia-Pacific Symposium on Internetware, ser. Internetware ’22. New York, NY, USA: Association for Computing Machinery, 2022, str. 231–241, dostupno na: https://doi.org/10.1145/3545258.3545276

[214]Bharadwaj, S., Kadam, T., “Github issue classification using bert-style models”, in 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE), 2022, str. 40-43.

[215]Colavito, G., Lanubile, F., Novielli, N., “Issue report classification using pre-trained language models”, in 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE), 2022, str. 29-32.

[216]Catolino, G., Palomba, F., Zaidman, A., Ferrucci, F., “Not all bugs are the same: Understanding, characterizing, and classifying bug types”, Journal of Systems and Software, Vol. 152, 2019, str. 165-181, dostupno na: https://www.sciencedirect.com/science/article/pii/S0164121219300536

[217]Rathnayake, R., Kumara, B., Ekanayake, E., “Cnn - based priority prediction of bug reports”, in 2021 International Conference on Decision Aid Sciences and Application (DASA), 2021, str. 299-303.

[218]Sushentsev, D., Khvorov, A., Vasiliev, R., Golubev, Y., Bryksin, T., "Dapstep: Deep assignee prediction for stack trace error representation", CoRR, Vol. abs/2201.05256, 2022, dostupno na: https://arxiv.org/abs/2201.05256

[219]Meng, F., Wang, X., Wang, J., Wang, P., "Automatic classification of bug reports based on multiple text information and reports' intention", in Theoretical Aspects of Software Engineering. Springer International Publishing, 2022, str. 131–147, dostupno na: https://doi.org/10.1007%2F978-3-031-10363-6_9

[220]Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T., "What makes a good bug report?", in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. SIGSOFT '08/FSE-16. New York, NY, USA: Association for Computing Machinery, 2008, str. 308–318, dostupno na: https://doi.org/10.1145/1453101.1453146

[221]Fang, F., Wu, J., Li, Y., Ye, X., Aljedaani, W., Mkaouer, M. W., "On the classification of bug reports to improve bug localization", Soft Computing, Vol. 25, No. 11, Jun 2021, str. 7307-7323, dostupno na: https://doi.org/10.1007/s00500-021-05689-2

[222]Zhu, Y., Pan, M., Pei, Y., Zhang, T., "A bug or a suggestion? an automatic way to label issues", dostupno na: https://arxiv.org/abs/1909.00934 2019.

[223]Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., "Bert: Pre-training of deep bidirectional transformers for language understanding", dostupno na: https://arxiv.org/abs/1810.04805 2018.

[224]Manning, C. D., Schutze, H., Foundations of statistical natural language processing. MIT Press, 1999.

[225]Šuman, S., "Overview of natural language processing and machine translation methods", Zbornik Veleučilišta u Rijeci / Journal of the Polytechnic of Rijeka, Vol. 9, No. 1, 2021, str. 371-384.

[226]Wermter, S., Riloff, E., Scheler, G., Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing. Springer Berlin, Heidelberg, 09 1996.

[227]Mikolov, T., Chen, K., Corrado, G., Dean, J., "Efficient estimation of word representations in vector space", dostupno na: https://arxiv.org/abs/1301.3781 2013.

[228]Kalchbrenner, N., Grefenstette, E., Blunsom, P., "A convolutional neural network for modelling sentences", dostupno na: https://arxiv.org/abs/1404.2188 2014.

[229]Zhang, X., Zhao, J., LeCun, Y., "Character-level convolutional networks for text classification", dostupno na: https://arxiv.org/abs/1509.01626 2015.

[230]Shen, D., Zhang, Y., Henao, R., Su, Q., Carin, L., "Deconvolutional latent-variable model for text sequence matching", in AAAI Conference on Artificial Intelligence, 2017.

[231]Liu, P., Qiu, X., Huang, X., "Recurrent neural network for text classification with multi-task learning", dostupno na: https://arxiv.org/abs/1605.05101 2016.

[232]Seo, M., Min, S., Farhadi, A., Hajishirzi, H., "Neural speed reading via skim-rnn", dostupno na: https://arxiv.org/abs/1711.02085 2017.

[233]Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., Hovy, E., "Hierarchical attention networks for document classification", in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. San Diego, California: Association for Computational Linguistics, Jun. 2016, str. 1480–1489, dostupno na: https://aclanthology.org/N16-1174

[234]Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B., Bengio, Y., "A structured self-attentive sentence embedding", dostupno na: https://arxiv.org/abs/1703.03130 2017.

[235]Sun, C., Qiu, X., Xu, Y., Huang, X., "How to fine-tune bert for text classification?", dostupno na: https://arxiv.org/abs/1905.05583 2019.

[236]Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., "Roberta: A robustly optimized bert pretraining approach", dostupno na: https://arxiv.org/abs/1907.11692 2019.

[237]Joulin, A., Grave, E., Bojanowski, P., Mikolov, T., "Bag of tricks for efficient text classification", dostupno na: https://arxiv.org/abs/1607.01759 2016.

[238]Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I., "Attention is all you need", dostupno na: https://arxiv.org/abs/1706.03762 2017.

[239]Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., Fidler, S., "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books", dostupno na: https://arxiv.org/abs/1506.06724 2015.

[240]Porter, M. F., "An algorithm for suffix stripping", Program, Vol. 14, No. 3, Jan 1980, str. 130-137, dostupno na: https://doi.org/10.1108/eb046814

[241]Loshchilov, I., Hutter, F., "Decoupled weight decay regularization", dostupno na: https://arxiv.org/abs/1711.05101 2017.

[242]Feng, J., Huang, D., “Optimal gradient checkpoint search for arbitrary computation graphs”, dostupno na: https://arxiv.org/abs/1808.00079 2018.

[243]van der Maaten, L., Hinton, G., “Visualizing data using t-sne”, Journal of Machine Learning Research, Vol. 9, No. 86, 2008, str. 2579–2605, dostupno na: http://jmlr.org/papers/v9/vandermaaten08a.html

[244]Denaro, G., “Estimating software fault-proneness for tuning testing activities”, in Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium, 2000, str. 704-706.

[245]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, Apr 2005, str. 183–218, dostupno na: https://doi.org/10.1007/s10664-004-6191-x

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

[247]Sakurada, M., Yairi, T., “Anomaly detection using autoencoders with nonlinear dimensionality reduction”, in Proceedings of the MLSDA 2014 2Nd Workshop on Machine Learning for Sensory Data Analysis, ser. MLSDA’14. New York, NY, USA: ACM, 2014, str. 4:4–4:11, dostupno na: http://doi.acm.org/10.1145/2689746.2689747

[248]Glorot, X., Bengio, Y., “Understanding the difficulty of training deep feedforward neural networks”, Journal of Machine Learning Research - Proceedings Track, Vol. 9, 01 2010, str. 249-256.

[249]Kingma, D., Ba, J., “Adam: A method for stochastic optimization”, International Conference on Learning Representations, 12 2014.

[250]Singh, V. P., Normal Distribution. Dordrecht: Springer Netherlands, 1998, str. 56–67, dostupno na: https://doi.org/10.1007/978-94-017-1431-0_5

[251]Hart, P. E., Lognormal Distribution. London: Palgrave Macmillan UK, 1990, str. 145–147, dostupno na: https://doi.org/10.1007/978-1-349-20570-7_20

[252]Pal, M., Ali, M., Woo, J., “Exponentiated weibull distribution”, Statistica, Vol. 66, 01 2003, str. 139-147.

[253]Arnold, B. C., Pareto Distribution. American Cancer Society, 2015, str. 1-10, dostupno na: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat01100.pub2

[254]Dubey, S. D., “Compound gamma, beta and f distributions”, Metrika, Vol. 16, No. 1, Dec 1970, str. 27–31, dostupno na: https://doi.org/10.1007/BF02613934

[255]Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Contributors, S. . ., “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”, Nature Methods, Vol. 17, 2020, str. 261–272.

[256]“Defect prediction - traditional features dataset source”, https://github.com/klainfo/DefectData, accessed: 22.03.2023.

Popis algoritama

Popis slika

Popis tablica

Životopis

Petar Afrić rođen je 1993. godine u Splitu u Hrvatskoj gdje je kasnije završio je III. gimnaziju. Preddiplomski studij programskog inženjerstva završio je 2015. godine, a diplomski studij računalne znanosti 2018. godine u Zagrebu na Fakultetu Elektrotehnike i Računarstva, Sveučilišta u Zagrebu. Po završetku diplomskog studija bio je dobitnik nagrade *Magna Cum Laude* kao jedan od najboljih studenata svoje generacije. Od Kolovoza 2015. do Svibnja 2018. godine radio je kao programski inženjer za *Sophion Bioscience A/S* od čega je prvih godinu dana proveo u Ballerupu u Danskoj. Od Ožujka 2018 do Ožujka 2021 bio je zaposlen kao zavodski suradnik Fakulteta Elektrotehnike i Računarstva, Sveučilišta u Zagrebu i radio na projektu *Istraživanje i razvoj naprednog sustava za upravljanje pametnim elektroenergetskim i komunikacijskim mrežama - Gdi Ensemble OperOSS*. Za vrijeme rada na fakultetu sudjelovao je u izvođenju nekoliko kolegija te bio voditelj projekta *W4 Scheduler* koji se bavio razvojem sustava za automatsku raspodjelu telekomunikacijskih radnika. Konačno od Ožujka 2021 zaposlen je kao glavni tehnički direktor u kompaniji *DataBlast d.o.o*. Za vrijeme doktorskog studija na Fakultetu Elektrotehnike i Računarstva, Sveučilišta u Zagrebu bavio se predviđanjem pogrešaka programske potpore. Objavio je radove u časopisima *Journal of Systems and Software* i *IEEE Access* te na konferenciji *IEEE International Conference on Software Quality, Reliability and Security*. Općenito, njegovi znanstveni interesi su u području automatskog predviđanja pogrešaka u izvornom programskom kôdu, području osiguranja kvalitete izvornog programskog kôda, području optimizacijskih algoritama i području strojnog učenja.

Popis objavljenih djela

Radovi u časopisima

1.Afri ć, P., Vukadin, D., Šilić, M. i Delač, G., "Empirical Study: How Issue Classification Influences Software Defect Prediction", IEEE Access, Vol. 11, Veljača 2023, str. 11732-11748

2.Šiki ć, L., Afrić, P., Kurdija, A. S. i ŠIlić, M. , "Improving Software Defect Prediction by Aggregated Change Metrics", IEEE Access, Vol. 9, January 2021, str. 19391-19411

3.Afri ć, P., Šikić, L., Kurdija, A. S., Šilić, M., "REPD: Source code defect prediction as anomaly detection", Journal of Systems and Software, Vol. 168, Listopad 2020

Radovi na konferencijama

1.Požek, M., Šiki ć, L., Afrić, P., Kurdija, A., Klemo, V., Delač, G., Šilić, M. "Performance of Common Classifiers on node2vec Network Representations", Proceedings of the International Conference on Computers in Technical Systems MIPRO 2019 Opatija, Opatija, Hrvatska, svibanj 2019., str. 1071-1076.

2.Tadi ć, L., Šikić, L., Afrić, P., Kurdija, A., Klemo, V., Delač, G., Šilić, M. "Analysis and Comparison of Exact and Approximate Bin Packing Algorithms", Proceedings of the International Conference on Computers in Technical Systems MIPRO 2019 Opatija, Opatija, Hrvatska, svibanj 2019., str. 1059-1064.

3.Afri ć, P., Kurdija, A., Šikić, L., Šilić, M., Delač, G., Vladimir, K., Srbljić, S., "Population-Based Variable Neighborhood Descent for Discrete Optimization", Proceedings of the International Conference on AI and Mobile Services, San Diego, SAD, lipanj 2019., str. 1-12.

4.Afri ć, P., Kurdija, A., Šikić, L., Šilić, M., Delač, G., Vladimir, K., Srbljić, S., "GRASP Method for Vehicle Routing with Delivery Place Selection", Proceedings of the International Conference on AI and Mobile Services, San Diego, SAD, lipanj 2019., str. 72-83.

5.Afri ć, P., Šikić, L., Kurdija, A., Delač, G., Šilić, M., "REPD: Source Code Defect Prediction As Anomaly Detection", Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofija, Bugarska, srpanj 2019., str. 227-234.

6.Šiki ć, L., Janković, J., Afrić, P., Šilić, M., Ilić, Ž., Pandžić, H., Živić, M., Džanko, M., "A comparison of application layer communication protocols in IoT-enabled smart grid", Proceedings of the 2020 International Symposium ELMAR, Zadar, Croatia, rujan 2020., str. 83-86.

7.Kurdija, A., Afri ć, P., Šikić, L., Plejić, B., Šilić, M., Delač, G., Vladimir, K., Srbljić, S., "Building Vector Representations for Candidates and Projects in a CV Recommender System", Proceedings of the Artificial Intelligence and Mobile Services – AIMS 2020, Honolulu, SAD, rujan 2020., str. 17-29

8.Kurdija, A., Afri ć, P., Šikić, L., Plejić, B., Šilić, M., Delač, G., Vladimir, K., Srbljić, S., "Candidate Classification and Skill Recommendation in a CV Recommender System", Proceedings of the Artificial Intelligence and Mobile Services – AIMS 2020, Honolulu, SAD, rujan 2020., str. 30-44.

9.Jankovi ć, J., Šikić, L., Afrić, P., Šilić, M., Ilić, Ž., Pandžić, H., Živić, M. and Džanko, M., "Empirical study: IoT-based microgrid", Proceedings of the 3rd International Colloquium

on Intelligent Grid Metrology (SMAGRIMET), Cavtat, Hrvatska, listopad 2020., str. 1-6.

Biography

Petar Afrić was born in 1993 in Split, Croatia where he later finished the III. gymnasium. He completed his undergraduate studies in software engineering in 2015, and his graduate studies in computer science in 2018 at the University of Zagreb, Faculty of Electrical Engineering and Computing located in Zagreb. After graduating, he was awarded the *Magna Cum Laude* award as one of the best students of his generation. From August 2015 to May 2018, he worked as a software developer for *Sophion Bioscience A/S*, the first year of which he spent in Ballerup, Denmark. From March 2018 to March 2021, he was employed as a research associate at the University of Zagreb, Faculty of Electrical Engineering and Computing, and worked on the *Istraživanje i razvoj naprednog sustava za upravljanje pametnim elektroenergetskim i komunikacijskim mrežama - Gdi Ensemble OperOSS* project. During his time at the university, he participated in the execution of several courses and was the lead on the *W4 Scheduler* project, which dealt with the development of a system for automatic telecommunication workforce assignment. Finally, since March 2021, he has been employed as the chief technical officer at DataBlast *d.o.o.*. During his PhD at the faculty, he focused his research efforts on the field of software defect prediction. He has published papers in the *Journal of Systems and Software* and in *IEEE Access* and at the *IEEE International Conference on Software Quality, Reliability and Security*. In general, his research interests are in the field of software defect prediction, the field of source code quality assurance, the field of optimization algorithms and the field of machine learning.