SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Sustav za detekciju plagijata te određivanje autorstva izvornih kodova

Dino Rakipović

mentor: doc. dr. sc. Ante Đerek

Zagreb, lipanj 2017.

Sadržaj

Uvod

Plagijarizam ili postupak krađe tuđeg rada postaje sve veći problem u današnjem svijetu te ga pronalazimo akademskom(npr. eseji, znanstveni članci) i neakademskom(npr. knjige, pjesme) svijetu. Razlog tomu je što količina podataka na internetu, koji je postao dostupan velikom broju ljudi, raste eksponencijalno te je vrlo lagano ukrasti tuđi rad i predstaviti ga kao vlastiti. U ovom radu naglasak će biti na detekciji plagijata izvornih kodova. Cilj je izraditi sustav koji bi ubrzao i uvelike pomogao u detekciji plagijata među izvornim kodovima ljudima koji se brinu o programerskim natjecanjima, laboratorijskim vježbama itd.

Detekciji pristupamo iz dva kuta, jedan je određivanje autora izvornog koda tzv. deanonimizacija autorstva, a drugi određivanje sličnosti među parovima izvornih kodova. Određivanje sličnosti parova izvornih kodova je već opisano i implementirano u završnom radu autora ovog diplomskog rada, no ovdje predstavljam brže te arhitektualno ljepše rješenje istog problema. Za određivanje autorstva bitno je primjetiti da svaki autor ostavlja svoj jedinstveni otisak dok piše programiski kod, barem se tome nadamo. Kako bi autore razlikovali korištene su tehnike strojnog učenja u konkretnom slučaju klasifikator nasumične šume. Korisnik mora najprije predati skup za treniranje, koji se sastoji od izvornih kodova te točno označenih autora, klasifikatoru kako bi kasnije mogao utvrditi autorstvo na skupu koji želi provjeriti. Korisnika se prepoznaje po njegovom stilu programiranja tako da se izvorni kod pretvori u vektor brojeva od kojih je svaki broj nekakva unaprijed označena značajka(npr. ostavlja li autor novi red prije otvaranja vitičastih zagrada, koliko naredbi grananja koristi, itd.).

Dva pristupa su na kraju integrirana pod istim web sučeljem nazvanim Turtle. Ovo sučelje nudi detaljan uvid u slične parove izvornih kodova te boja slične dijelove jednakim bojama kako bi korisnik prije odlučio promatra li plagijat. Potrebno je naglasiti da je sustav trenutno implementiran za pronalazak plagijata jedino u programskom jeziku C++.

Također ispisuje autore za koje misli da imaju najveću vjerojatnost da su napisali promatrani kod. Utvrđivanje autorstva je veliki korak naprijed nad rješenjem napisanim za završni rad jer nam omogućuje detekciju plagijata među raznim akademskim godinama ukoliko se laboratorijski zadaci mijenjaju.

POGLAVLJE 1

Određivanje autorstva

U svijetu u kojem ne postoje unaprijed određena pravila pisanja programskog koda možemo pretpostaviti da svaki programer ostavlja svoj jedinstveni otisak dok programira. Cilj nam je kreirati klasifikator koji bi nam mogao odvojiti autore prema njihovom stilu programiranja. Ovakav klasifikator bi bilo moguće primjeniti na raznim open source projektima na kojima autori razvijaju kod anonimno te bi takav klasifikator mogao narušiti privatnost programera, ali ipak u ovom radu veći naglasak je na detekciji plagijata izvornih kodova te ovakav klasifikator koristimo nad laboratorijskim vježbama na fakultetima ili na nekim programerskim natjecanjima gdje autori predaju kod pod svojim imenom.

Za rješavanje ovog problema korišteno je strojno učenje. Strojno učenje je grana umjetne inteligencije koja se bavi algoritmima koji mogu učiti na i raditi predviđanja nad skupovima podataka. Kako bi mogli strojno učiti moramo imati skupove podataka s označenim kategorijama nad kojima algoritam uči. U ovom slučaju podaci su izvorni kodovi, a kategorije autori koji su ih napisali. Konkretno, korišten je klasifikator slučajne šume. Konfiguracija klasifikatora je detaljnije opisana u nastavku poglavlja. Klasifikacija je postupak u kojem određujemo kojoj kategoriji (od unaprijed određenih) novi podaci pripadaju. Algoritmi strojnog učenja uglavnom primaju ulazne podatke u obliku brojeva pa je potrebno izvorni kod pretvoriti u vektor brojeva u kojem će svaki broj biti neka od značajki. Te značajke su podijeljenje u leksičke, sintaksne i strukturalne. Što bolje značajke odaberemo algoritam će bolje moći odvajati kategorije tj. autore. Značajke dobijemo parsiranjem izvornog koda te ću postupak detaljnije opisati u nastavku poglavlja.

1.1 Izvlačenje značajki

Kao što je već spomenuto, kako bi algoritmi strojnog učenja radili potrebni su im brojčani podaci kao ulazi. Izvorni kod se u brojčani vektor značajki pretvara koristeći ideju prvi put opisanu u radu [1], a ideja je da se izvorni kod pretvori u vektor značajki sastavljen od tri dijela, leksičkog, sintaksnog i strukturnog. Leksičke i strukturalne značajke se dobiju izravno parsiranjem izvornog koda, dok nam je za sintaksne značajke potrebno apstraktno sintaksno stablo izvornog koda. Ovako definiran skup značajki je drugačiji za svaki pojedini programski jezik zbog različitosti među njima (npr. drugačije ključne riječi) te je potrebno napisati poseban parser za svaki od njih. U ovom radu naglasak je na programskom jeziku C++ te je izvlačenje značajki implementirano samo za njega.

U nastavku su detaljno opisana i objašnjena sva tri tipa značajki. Većina tih značajki preuzeta je iz [1] dok su neke ideja samog autora ovog rada. U većini značajki korištena je matematička operacija prirodnog logaritmiranja zbog svojstva da kako idemo prema većim vrijednostima ona sve manje i manje raste te dobro opisuje relativne razlike među značajkama.

1.1.1 Leksičke značajke

Leksičke značajke opisuju preferira li autor izvornog koda neke ključne riječi više od drugih(npr. for više od while), koristi li više funkcije ili piše monolitan kod, razne statistike(npr. prosječan broj parametara unutar funkcija), itd. Također izvorni kod se tokenizira te se računa frekvencija tako dobivenih tokena. *Tablica* 1 detaljno opisuje svaku od korištenih značajki.

1.1.2 Strukturne značajke

Strukturne značajke opisuju kakvu strukturu autor koristi dok piše izvorni kod, npr. koristi li tabove ili razmake na početku linije, piše li novu liniju prije nego otvori kontrolni blok, itd. *Tablica* 2 detaljno opisuje svaku od korištenih značajki.

1.1.3 Sintaksne značajke

Sintaksne značajke se dobiju kako je već spomenuto iz apstraktnog sintaksnog stabla izvornog koda. One su što se vremena tiče najskuplje jer kreacija apstraktnih sintaksnih stabala nije brza, no trebale bi dati odlične značajke koje bi uvelike pomogle u deanonimizaciji. Apstraktna sintaksna stabla su kreirana koristeći alat *joern* [2]. Ovaj alat nudi posebnu skriptu *joern-parse* koja parsira i vraća čvorove i bridove apstraktnog sintaksnog stabla.*Tablica* 3 detaljno opisuje svaku od korištenih značajki.

1.2 Selekcija značajki

Ovako kreirane značajke rezultiraju u ogromnim, rijetkim vektorima, čija veličina nekada doseže i stotine tisuća brojeva. Razlog tomu leži u definiciji značajki poput frekvencije tokena, frekvencije bigrama, itd. Rijetkost očitujemo u velikom broju nula unutar vektora. Rijetkost, također, može uzrokovati loš izabir idućeg čvora u klasifikatoru slučajne šume te s time i lošije rezultate. S velikim vektorima također dolazi i do puno sporijeg učenja klasifikatora jer će sva stabla odluka unutar slučajne šume imati više čvorova. Zbog svih navedenih razloga prije samog učenja klasifikatora napravljena je selekcija značajki koja odabire manji broj značajki koje sadrže dovoljno informacija da bi se klasifikator bolje i brže naučio. Tehnika selekcije značajki je mnogo pa ih ovdje neću detaljno opisivati nego ću se bazirati na algoritmima koji su korišteni za ovaj diplomski rad, a to su *ExtraTreeClassifier* [3] i *VarianceThreshold* [4]. Više o rezultatima sa i bez selekcije značajki u poglavlju 1.5

1.2.1 ExtraTreeClassifier

1.2.2 VarianceThreshold

1.3 Slučajna šuma

Slučajna šuma je klasifikator koji se sastoji od kolekcije nezavisnih stabala odlučivanja. Svako od stabala predstavlja jedan glas u većinskom donošenju odluke. Odluka se donosi zbrajanjem glasova te se odabire odluka s najvećim brojem glasova [5]. Slučajna šuma jer je u svojoj osnovi samo skup stabala vrlo dobro podnosi veliku dimenzi-

onalnost podataka (što za ovaj problem očekujemo) i ne očekuje linearnu odvojivost vektora značajki te je iz tih razloga odabrana kao korišteni algoritam.

Svako od N stabala odluke je izgrađeno nasumičnim uzorkovanjem s ponavljanjima skupa za treniranje tako da se uzorkuje podskup duljine N. Stabla se grade do maksimalne moguće dubine iako postoje instance algoritma u kojem se stabla podrezuju. U izgradnji stabla ponovno se slučajno odabire podskup značajki kojih ima M. Veličina tog podskupa je hiperparametar algoritma, u literaturi [6] se za klasifikacijski problem preporuča veličina od $\sqrt{M}$. Od tog podskupa treba odabrati najbolju značajku koja će biti iskorištena za idući čvor stabla. Odabir najbolje značajke uobičajeno se radi metodama Gini nečistoće ili uzajamnog sadržaja informacije.

1.3.1 Gini nečistoća

Gini nečistoća je mjera koliko često bi nasumično odabrana značajka iz nekog skupa bila krivo klasificirana ako bi ju se nasumično klasificiralo s obzirom na to kakva je razdioba značajki po razredima u podskupu svih značajki. Drugim riječima gini nečistoća je kriterij koji teži minimizaciji vjerojatnosti krive klasifikacije [7]. Računamo ju na sljedeći način [8]:

$$I_g(t) = 1 - \sum_{i=1}^{c} p(i|t)^2 \tag{1.1}$$

gdje je $p(i|t)$ broj značajki koje pripadaju klasi i za čvor t.

1.3.2 Uzajamni sadržaj informacije

Uzajamni sadržaj informacije je koncept baziran na entropiji. Entropija je definirana kao količina informacije koju nosi neka poruka te ju računamo:

$$H(t) = -\sum_{i} p(x_i) * log_2 p(x_i) \tag{1.2}$$

gdje su $p(x_i)$ vjerojatnosti svake od klasa.
Uzajamni sadržaj informacije definiran je kao:

$$I(X; Y_i) = H(X) - H(X|Y_i) \tag{1.3}$$

gdje je X klasa(autor), a Y_i i-ta značajka iz skupa. Intuitivno ga možemo zamisliti kao količinu informacije koju daje značajka i za klasu kojoj pripada.

1.4 Priklupljanje podataka

1.5 Rezultati i rasprava

Zaključak

Sažetak

Summary

Literatura

[1] A. Caliskan-Islam i dr. *De-anonymizing Programmers via Code Stylometry*. 2014. URL: https://www.princeton.edu/~aylinc/papers/caliskan-islam_deanonymizing.pdf.

[2] F. Yamaguchi. *Joern documentation*. 2017. URL: http://www.mlsec.org/joern/.

[3] *ExtreTreeClassificator documentation*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html.

[4] *VarianceThreshold documentation*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html.

[5] Nikola Bogunović. *Algoritmi strojnog učenja - 1, Strojno učenje*. predavanja, http://www.zemris.fer.hr/predmeti/kdisc/Algoritmi_1.ppt. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Hrvatska.

[6] T. Hastie, R. Tibshirani i J. Friedman. *The elements of statistical learning*. 2009.

[7] *DecisionTree*. predavanja, http://www.cse.msu.edu/~cse802/DecisionTrees.pdf. Michigan State University, USA.

[8] Sebastian Raschka. https://sebastianraschka.com/faq/docs/decision-tree-binary.html.