ALGORITAM SLUČAJNIH ŠUMA U OPENCV BIBLIOTECI

Lazor, Denis

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:526137

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ALGORITAM SLUČAJNIH ŠUMA U OPENCV BIBLIOTECI

Završni rad

Denis Lazor

Osijek, 2019.

SADRŽAJ

1. UVOD 1
1.1. Zadatak završnog rada 1
2. ALGORITAM SLUČAJNIH ŠUMA 2
3. OPENCV BIBLIOTEKA 7
3.1. Implementacija algoritma slučajnih šuma 7
4. PRIMJENA NA PROBLEM KLASIFIKACIJE STANICA RAKA DOJKE 11
5. ZAKLJUČAK 16
PRILOZI 17
LITERATURA 19
SAŽETAK 20
ABSTRACT 21
ŽIVOTOPIS 22

1. UVOD

Strojno učenje je relativno novi pojam. Još je u ranom stadiju razvitka i ima jako veliki prostor za napredak. Strojno učenje je jedna od grana umjetne inteligencije. Zasnovano je na ideji da računala mogu obavljati specifične zadatke bez da su programirani za to, već uče iz dobivenih podataka i primjenjuju to znanje u rješavanju različitih zadataka. Ti zadatci se obično dijele na klasifikacijske i regresijske. Klasifikacijski zadatci se odnose na klasifikaciju podataka u određene kategorije odakle i dolazi naziv. Najčešće se radi o prepoznavanju različitih objekata na slici, prepoznavanju govora, rukopisa i sličnih problema. Regresija se odnosi na predviđanje kretanja kontinuiranih podataka poput cijena dionica, potrošnje električne energije i slično. U ovom radu će biti obrađen problem klasifikacije pomoću algoritma slučajnih šuma implementiranog u OpenCV[1] biblioteci. [1]

1.1. Zadatak završnog rada

Opisati teorijsku podlogu algoritma slučajnih šuma i njegovu implementaciju u OpenCV biblioteci. Potrebno je demonstrirati rad navedenog algoritma na problemu klasifikacije implementirajući ga pomoću OpenCV biblioteke pri tome koristeći Python programski jezik.

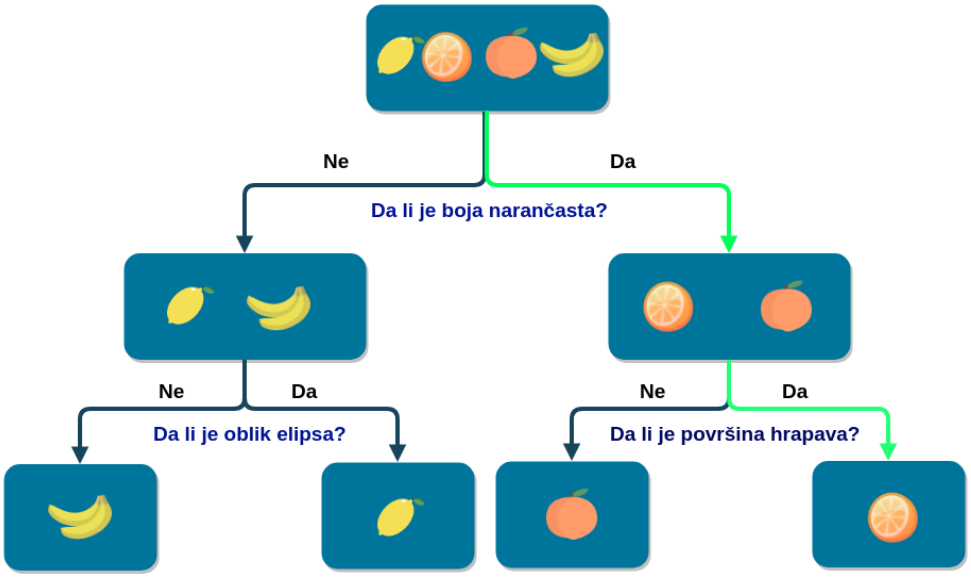
[1] engl. *Open Source Computer Vision Library*

2. ALGORITAM SLUČAJNIH ŠUMA

Naziv „šuma” dolazi od činjenice da se ovaj algoritam sastoji od više takozvanih stabala odluke. Stablo odluke predstavlja način kojim računalo dolazi do zaključka. Sve kreće od korijenskog čvora. Nakon što su algoritmu predani podatci s kojima će raditi, danji postupak se svodi na postavljanje pravih pitanja. Odgovori na pitanja moraju biti u obliku točno ili netočno to jest 0 ili 1. Tako da kod svakog pitanja svaki od čvorova stvara svoja 2 nova podčvora. Postavljanjem dovoljnog broja pravih pitanja dolazi se do konačnog čvora u stablu. Odabir pravih pitanja, to jest odabir atributa koji će se testirati u pojedinom čvoru vrši se izračunom informacijske dobiti. Informacijska dobit predstavlja očekivanu redukciju u entropiji uzrokovanu podjelom čvora. Veća redukcija znači veću informacijsku dobit. Dakle podjelom čvorova po nekom atributu dobivamo dodatnu informaciju o uzorku i time se smanjuje entropija.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i \tag{2-1}$$

Izraz (2-1) predstavlja formulu za entropiju . *S* predstavlja početni skup, *p* predstavlja frekvenciju pojavljivanja pojedine kategorije u tom skupu dok *i* predstavlja broj kategorija [2]. Uzmimo za primjer sljedeće stablo:



Sl. 2.1. *Binarno stablo odluke*

Zadatak algoritma je klasificirati voće u jednu od 4 kategorije. Kategorije su limun, naranča, marelica i banana. Također pretpostavimo da imamo spremnih 100 uzoraka za treniranje gdje svaki uzorak predstavlja neko voće i da svaki uzorak ima 3 atributa, a to su boja, oblik i tekstura. Neka se limun pojavljuje 20 puta, banana 30, naranča 40 i marelica 10 puta. Tada možemo izračunati entropiju korijenskog čvora koji ima ulogu roditeljskog čvora na sljedeći način:

$E(roditelj) = - (2/10) \log_2(2/10) - (3/10) \log_2(3/10) - (4/10) \log_2(4/10) - (1/10) \log_2(1/10) = 1,85$

Entropija ima vrijednost više od 1 što je uobičajeno za podatkovni skup sa više od 2 klase. Sada će s obzirom izbora atributa za podjelu djeca korijenskog čvora imati različite entropije. Računaju se padovi entropije za sve atribute i odabire se onaj koji daje najveću informacijsku dobit. Podjela po boji će uzrokovati da 50 uzoraka ostaje u jednom čvoru i 50 u drugom novonastalom čvoru s obzirom da imamo 50 uzoraka sa atributom boje jednakom žuta , a 50 uzoraka s vrijednosti narančasta. Slijedi izračun entropija pojedinih čvorova.

$E(žuta) = -(20/50) \log_2(20/50) - (30/50)\log_2(30/50) = 0,97$

$E(narančasta) = -(40/50) \log_2(40/50) - (10/50) \log_2(10/50) = 0,72$

Ukupna težinska entropija novonastalih čvorova je tada:

$E(boja) = (50/100)*0.97 + (50/100)*0.72 = 0,85$

Podjela po teksturi će uzrokovati podjelu u omjeru 60/40 gdje se u čvoru sa 60 uzoraka nalaze limuni i naranče sa hrapavom teksturom, a u drugom banane i marelice s glatkom teksturom. Slijedi izračun entropija pojedinih čvorova:

$E(hrapava) = -(20/60) \log_2(20/60) - (40/60)\log_2(40/60) = 0,92$

$E(glatka) = -(10/40) \log_2(10/40) - (30/40)\log_2(30/40) = 0,81$

Ukupna težinska entropija novonastalih čvorova je tada:

$E(tekstura) = (60/100)*0.92 + (40/100)*0.81 = 0,88$

Ukoliko pretpostavimo da su limun, naranča i marelica eliptičnog oblika, a banana polukružnog oblika tada će podjela po obliku uzrokovati podjelu u omjeru 70/30. Slijedi izračun entropija pojedinih čvorova:

$E(elipsa) = -(20/70) \log_2(20/70) - (40/70)\log_2(40/70) - (10/70)\log_2(10/70) = 1,38$

E(polukrug) = -(30/30)$\log_2$(30/30) = 0

Ukupna težinska entropija novonastalih čvorova je tada:

E(oblik) = (70/100) *1.38 + (30/100)*0 = 0,97

Nastale informacijske dobiti su:

I(roditelj, boja) =E(roditelj) - E(boja) = 1.85- 0.85 = 1

I(roditelj, tekstura) = E(roditelj) - E(tekstura) = 1.85- 0.88 = 0,97

I(roditelj, oblik) = E(roditelj) - E(oblik) = 1.85- 0.97 = 0,88

Iz izračuna entropija je vidljivo da podjela po boji daje najveću informacijsku dobit zbog čega je taj atribut izabran za podjelu što je vidljivo na slici 2.1.

Na isti način se dijele novonastali čvorovi dok entropija ne dostigne vrijednost 0. Na slici 2.1. je vidljivo da nakon dijeljenja po atributima oblika i teksture svi novonastali čvorovi imaju entropiju 0 jer imaju samo jednu klasu to jest informacijska dobit je jednaka trenutnoj entropiji. Ti čvorovi se nazivaju listovi dok zelena boja predstavlja put zaključivanja. Zadnji čvor predstavlja konačni zaključak.

Umjesto metode informacijske dobiti može se koristiti i Gini indeks metoda koja daje približno iste rezultate i radi na sličan način, ali ne koristi logaritamske funkcije što dovodi do malo bržeg algoritma.

Općeniti algoritam izgradnja stabla odluke bi bio sljedeći:

Korak 1. Izračunati entropiju korijenskog čvora

Korak 2. Izračunati informacijsku dobit za sve atribute i odabrati onaj koji daje najveću

Korak 3. Izgraditi sljedeću razinu stabla gdje je prethodno odabrani atribut sada korijen stabla

Korak 4. Ponavljati korake 1 do 3 dok entropija ne dostigne vrijednost 0

Tako stablo uči postavljati pitanja koja će najbrže i najpreciznije riješiti problem. Nakon treniranja možemo mu predati podatke istog formata i tipa bez rezultata i tražiti stablo da predvidi rezultat.

Rezultat takvog jednog stabla koje je trenirano na jedan način i s jednom skupinom podataka najčešće neće biti baš precizan ako stablo ima malu dubinu, a ako ima veliku podložno je šumovima u podatcima (*overfitting).* Tu dolazimo do ideje slučajnih šuma. Ideja je da imamo više

takvih stabala koji uče iz nasumično odabranih podataka, odakle i naziv „slučajna”, da bi stabla bila što raznovrsnija. Također stabla mogu imati veću dubinu jer se povećanjem broja stabala smanjuje utjecaj šumova, to jest kompenzira se *overfitting* zasebnih stabala odluke. Isto tako zbog velike dubine stabla, to jest kompleksnosti, riješen je problem nedovoljno dobre interpretacije veza između varijabli unutar korištenog skupa podataka (*underfitting).* Velika otpornost na *underfitting* i *overfitting* je jedna od najvećih prednosti algoritma slučajnih šuma.

Slučajna šuma je zapravo kolekcija stabala odluke. Razlika je što se ne uzimaju svi uzorci i atributi za treniranje pojedinih stabala već određen broj nasumično odabranih atributa i uzoraka. Kada se radi o klasifikaciji najčešće se za broj nasumično odabranih atributa uzima vrijednost korijena iz ukupnog broja atributa.

Slučajne šume koriste odvajajuću (engl. *bagging*) metodu koja za izgradnju stabala odluke uzima nasumičan podskup uzoraka iz trening skupa podataka. Otprilike trećina uzoraka tog podskupa je izostavljena pri izgradnji stabala. Takvi uzorci zovu se odvajajući (engl. *out-of-bag)* uzorci. Ti uzorci se koriste za testiranje stabala koji nisu vidjeli te uzorke. Glasovi takvih stabala se pamte i koriste za izračunavanje odvajajuće generalizacijske ocijene (engl. o*ut-of-bag score).* Time nema potrebe za unakrsnom validacijom.

Općeniti algoritam slučajne šume glasi ovako:

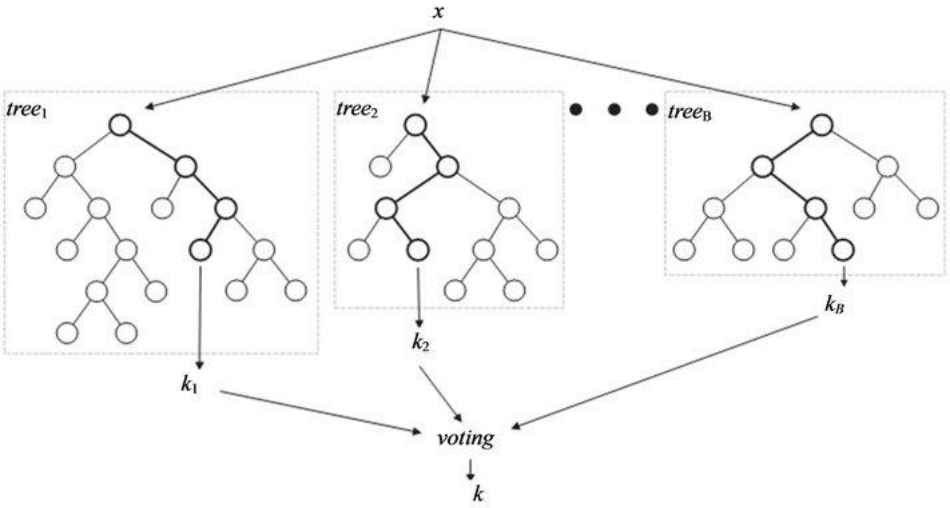
Korak 1. Odabrati broj stabala *n, b*roja atributa iz kojih će se generirati stabla *m* i broj uzoraka za treniranje *N*.

Korak 2. Odabrati N uzoraka iz trening skupa koristeći slučajni odabir s ponavljanjem (*engl. bootstrap sample*).

Korak 3. Izgraditi stablo koristeći prethodno odabrane podatke i veličinu *m* za trenutni čvor. Od *m* atributa uzeti onaj koji daje najveću informacijsku dobit.

Korak 4. Ponavljati korak 2 i 3 dok se ne izgradi *n* stabala.

Korak 5. Naći kategoriju koju je za rezultat dalo najveći broj stabala i tu kategoriju dati kao rezultat algoritma. [3]



Sl. 2.2. *Opći prikaz slučajne šume* [4]

Izgradnja pojedinih stabala se i dalje svodi na izračun informacijske dobiti samo što sada stablo ne radi sa svim uzorcima i atributima.

3. OPENCV BIBLIOTEKA

OpenCV je biblioteka otvorenog koda koja sadrži više stotina algoritama vezanih za računalni vid. Biblioteka se sastoji od modula. Svaki od modula definira zasebno područje imenovanja (*engl. Namespace*) i može koristiti definicije klasa, funkcija, varijabli i konstanti iz drugih modula. *Core functionality* modul sadrži definicije osnovnih funkcija i podatkovnih struktura korištenih od svih drugih modula. *Image Processing* modul se koristi za obradu slika pomoću različitih filtra, geometrijskih transformacija, histograma i ostalih alata. *Video Analysis* je modul *za* video analizu koji sadrži algoritme za estimaciju kretanja, praćenje objekata i izdvajanje pozadine. *Camera Calibration and 3D Reconstruction* modul sadrži algoritme za kalibraciju kamere, estimaciju položaja objekata, višedimenzionalnu geometriju i 3D rekonstrukciju. *2D Features Framework* modul za detekciju značajnih dijelova slike i njihovo kodiranje u vektore pomoću deskriptora.. *Object Detection* je modul za detekciju objekata predefiniranih klasa. *High-level GUI* modul pruža jednostavna korisnička grafička sučelja za brzo i jednostavno testiranje funkcionalnosti napisanog programa. Uz glavne module biblioteka sadrži i nekoliko pomoćnih.[5]

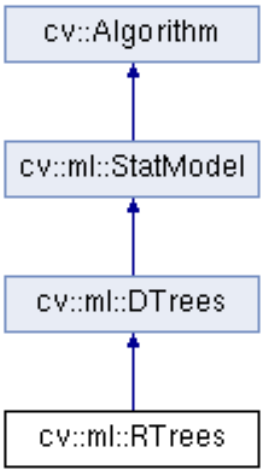
Biblioteka je napisana u C++ programskom jeziku, ali ima sučelje i u programskim jezicima Python i Java te u Matlab-u. Od operacijskih sustava podržava Windows, Linux, Android, iOS, MacOS, Maemo, BlackBerry 10, i različite BSD Unix distribucije.

OpenCV zajednica ima preko 47000 članova, a broj skidanja prelazi 18 milijuna.

Verzija koja će biti korištena u svrhu demonstracije u ovom radu je 4.1.0. [6]

3.1. Implementacija algoritma slučajnih šuma

Kao što je ranije spomenuto OpenCV biblioteka je napisana u C++ programskom jeziku. OpenCV biblioteka se bazira na klasama i nasljeđivanju. Temeljna klasa koju sve klase algoritama moraju naslijediti je klasa *Algorithm.* Tu klasu nasljeđuju klase algoritama za različite primjene, jedna od tih klasa je klasa statističkih modela koja čini matematičku bazu algoritama strojnog učenja. Algoritam koji nas zanima zove se stablo odluke s klasom pod imenom *DTrees* u OpenCV biblioteci. Iz te klase je izvedena i konačna klasa koja je tema ovog rada, a to je klasa koja implementira sam algoritam slučajnih šuma (*RTrees*). Vizualan prikaz strukture klasa i nasljeđivanja se nalazi na slici 3.1.



Sl. 3.1. *Struktura nasljeđivanja u OpenCV-u* [7]

Klasa slučajne šume osim što sadrži svoje metode također nasljeđuje i metode od sve tri roditeljske klase. Kako bi dobili ideju o primjeni tih metoda proći ćemo kroz one najosnovnije.

Stvaranje klasifikatora stabla odluke u programskom jeziku Python postižemo sljedećom naredbom:

DTreeClassificator = cv.ml.DTrees_create( )

Kako bi novostvoreni klasifikator radio potrebno je postaviti atribute klasifikatora, to jest njegove parametre. To se postiže pomoću funkcija za postavljanje atributa na sljedeći način:

DTreeClassificator.setIME_ATRIBUTA(vrijednost_atributa)

S obzirom da algoritam slučajne šume sam stvara stabla odluke, nema potrebe ručno stvarati zasebna stabla već možemo odmah stvoriti klasifikator slučajne šume:

RTreeClassificator = cv.ml.RTrees_create()

Također sva stabla će biti stvorena prema istim parametrima. S obzirom da klasa slučajne šume ima pristup atributima klase stabla odluke, parametre vezane za stablo odluke i parametre vezane za sam algoritam slučajnih šuma možemo postaviti preko klasifikatora slučajne šume na sljedeći način:

RTreeClassificato.setIME_ATRIBUTA(vrijednost_atributa)

Te parametre ćemo podijeliti na one karakteristične za klasu *DTrees* i na one karakteristične za klasu *RTrees*.

Parametri vezani za klasu stabla odluke su sljedeći:

MaxCategories parametar se koristi u slučaju većeg broja kategorija. U prethodnom poglavlju je pokazano kako se računa najbolja podjela čvorova. Račun sadrži dosta logaritama koji usporavaju rad algoritma. Veliki broj kategorija znači veliki broj logaritama što dovodi do jako sporog algoritma. Tada se kategorije razvrstavaju u skupine kako bi se dobila suboptimalna podjela čvora i ubrzao rad algoritma. Proces se provodi ukoliko je broj kategorija čvora veći od vrijednosti postavljenog parametra.

MaxDepth predstavlja maksimalnu dubinu stabla. U koliko ograničimo dubinu stabla moguće je da će stablo prestati dijeliti čvorove i prije nego što dostigne entropiju vrijednosti 0. Tako se smanjuje preciznost, ali sprječava *overfitting.* S obzirom da je slučajna šuma otporna na *overfitting,* nema potrebe ograničavati dubinu stabla.

MinSampleCount predstavlja minimalan broj uzoraka koji mora biti prisutan u čvoru kako bi se dalje dijelio. Svrha ovo parametra je sprječavanje pojave *overfittinga.* Slično kao i u slučaju s *MaxCategories* parametrom to nije potrebno. Želimo sniziti entropiju do vrijednosti 0, a ograničavanje dubine stabla to može spriječiti. S obzirom da je slučajna šuma prilično otporna na šumove, obično je vrijednost ovog parametra u slučaju klasifikacije postavljena na vrijednost 2.

Priors parametar predstavlja matricu težina kategorija koje predviđamo. Ukoliko se neka kategorija u podatcima pojavljuje jako mali broj, algoritam ima tendenciju *overfittinga* nad tom klasom i često će manjinsku kategoriju pogrešno klasificirati jer tretiranje svih klasa jednako dovodi do boljih performansi algoritma. Tada se manjinskoj kategoriji uvodi veća težina.

Surrogates parametar se koristi u slučaju nepotpunih podataka tako što zamjenjuje nedostajuće atribute prosječnom vrijednošću tog atributa. Nedostatak podataka jako utječe na preciznost stabla dok kod slučajne šume nedostatak podataka ne stvara probleme zbog velike raznovrsnosti stabala.

Parametri vezani za klasu slučajne šume su sljedeći:

ActiveVarCount parametar predstavlja ranije pojašnjenu veličinu označenom slovom *m.* Predstavlja broj nasumično odabranih atributa koji će se koristiti pri odabiru najbolje podjele čvora s obzirom na informacijsku dobit pojedinih atributa. Najčešće je taj parametar jednak korijenu ukupnog broja atributa.

TermCriteria parametar predstavlja kriterij zaustavljanja trening algoritma. Dva su moguća kriterija, a to su postizanje dovoljne preciznosti ili kada se dosegne dovoljan broj istreniranih stabala odluke. Veći broj stabala odluke smanjuje mogući *overffiting* i povećava preciznost algoritma. Najčešće nakon određenog broja istreniranih stabala preciznost prestaje značajnije rasti tako da treniranje dodatnih stabala samo usporava cijeli algoritam. [7]

Kako bi istrenirali model koristimo *StatModel* klasu. Statistički modeli su zapravo veze između varijabli opisane pomoću matematičkih alata. Model se trenira na sljedeći način:

RTreeClassifier.train(trening_uzorci, nacin_uzimanja_uzoraka, trening_rezultati)

Trening uzorci su dio ukupnih uzoraka koji se koriste za trening klasifikatora. Trening rezultati predstavljaju kategorije u koje treba klasificirati pojedine trening uzorke. Te kategoriju su obično tipa *string* i treba ih kodirati u *int* tip s kojim algoritam može raditi. Način uzimanja uzoraka može biti u obliku stupaca (COL_SAMPLE) i u obliku redova (ROW_SAMPLE) izvornog podatkovnog seta.[9]

Klasa slučajne šume nasljeđuje i mnoge druge metode. Prikazane metode služe za stvaranje slučajne šume i pokretanje njenog treninga. Podatke treba od nekuda učitati ili stvoriti i optimizirati ih kako bi algoritam mogao raditi s njima. Podatci se dobivaju obradom velikog broja slika, videa, audio datoteka ili drugih izvora podataka. Iz njih se vade potrebni podatci i spremaju u tekstualnom obliku pogodnom za korištenje. Ti podatci se predstavljaju u obliku uzoraka gdje se uzorci sastoje od različitih atributa. Zbog različitih smetnji pri obradi pojavit će se neprecizni i nepotpuni uzorci kojih se treba riješiti kako se ne bi doveo dodatan šum u podatke što bi moglo utjecati na krajnje performanse algoritma. Algoritam slučajnih šuma nije osjetljiv na nepotpune podatke.

4. PRIMJENA NA PROBLEM KLASIFIKACIJE STANICA RAKA DOJKE

Prvi korak pri izradi projekta je priprema podataka koje ćemo koristiti. U svrhu ovog projekta skinuti su podatci u *csv* formatu preuzeti sa UCIML[10] baze podataka. Datoteka sadrži 32 stupaca i 570 redaka. Prvi i drugi stupac redom sadrže identifikacijski broj pacijenta i dijagnozu koja se sastoji od dvije kategorije. Kategorija „M" predstavlja dijagnozu malignog tumora to jest raka dok kategorija „B" predstavlja benigni tumor. Ostalih 30 stupaca sadrži razne informacije o strukturi stanica koje treba klasificirati u prethodno navedene kategorije. Prvi redak predstavlja nazive stupaca, dok ostali redci počevši od trećeg stupca predstavljaju uzorke.

Nakon pripreme podataka slijedi pisanje programskog koda. U tu svrhu će biti korištena Python 3.7 verzija programskog jezika i PyCharm 2019.1 verzija razvojnog okruženja na Ubuntu 19.04 distribuciji operacijskog sustava Linux.

Kako bi koristili prednosti različitih biblioteka prvo ih moramo učitati u projekt. Korišten je apsolutni način učitavanja.

```
from numpy import int32, float32

from cv2.ml import RTrees_create, ROW_SAMPLE
from cv2 import TermCriteria_MAX_ITER, TermCriteria_EPS

from pandas import read_csv

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Verzije korištenih biblioteka su sljedeće: NumPy 1.16.4, OpenCV 4.1.0.25, Pandas 0.24.2, Sickit-learn 0.21.2 .

Nadalje potrebno je učitati lokalno spremljenu *csv* datoteku što se postiže sljedećom linijom koda:

```
data_file = open('/home/wolfie/Desktop/Završni/RFA-py/breast_cancer_dataset.csv')
```

Čitanje datoteke i popunjavanje matrice uzoraka i matrice odziva:

```
samples, response = load_data(data_file)
```

Gdje je *load_data* korisnički definirana funkcija gdje se prvo podatci čitaju iz predane datoteke pomoću *read_csv* funkcije iz Pandas biblioteke koja kao argumente prima datoteku, tip podataka

u kojem će učitani podatci biti spremljeni i znak kojim su razdijeljeni podatci u datoteci. U ovom slučaju su to prethodno učitana *csv* datoteka, None što predstavlja da će funkcija sama prepoznati tip podataka i tako ih i spremiti te znak zareza budući da se radi o *csv* datoteci. Podatci se spremaju u obliku specifičnom za Pandas biblioteku (*pandas DataFrame)* koji automatski odvaja zaglavlje i numerira uzorke i dolazi s funkcijom *iloc* za izdvajanje pojedinih redaka i stupaca. Početni indeks stupaca i redaka je nula. Tako smo u matricu uzoraka *samples* učitali sve retke i 30 stupaca s informacijama o strukturi stanice, a u matricu odziva r*esponse* sve retke i samo drugi stupac koji sadrži već objašnjene kategorije gdje su zaglavlje i redni brojevi uzoraka izbačeni kao i stupac s identifikacijskim brojevima pacijenata.

```
def load_data(data_file):
    data = read_csv(data_file, dtype=None, delimiter=',')
    samples = data.iloc[:, 2:31].values
    response = data.iloc[:, 1].values,
    return samples, response
```

Slučajna šuma ne može raditi s kategoričnim vrijednostima stoga vrijednosti matrice odziva moramo prebaciti u brojevne vrijednosti. To se lako postiže pomoću *LabelEncoder* klase i *fit_transform* funkcije iz Sickit-learn biblioteke. Također, slučajna šuma u OpenCV biblioteci je jako osjetljiva na ulazni tip podataka koji moraju biti u 32-bitnom formatu što se lako može postići korištenjem NumPy biblioteke koja omogućava lako manipuliranje matricama u Python programskom jeziku. Kodiranje matrice odziva i prebacivanje matrica u 32-bitni format se postiže sljedećim naredbama:

```
le = LabelEncoder()

samples = float32(samples)
response = int32(le.fit_transform(response))
```

Podjela podataka na trening i test podatke se postiže pomoću *train_test_split* funkcije iz Sickit-learn biblioteke. Kao argumente prima matricu uzoraka, matricu odziva i postotak koji odlazi na test ili trening podatke u obliku decimalne vrijednosti. U ovom slučaju 20 posto podataka će otići na testiranje, a 80 posto na trening:

```
samples_train, samples_test, response_train, response_test = train_test_split(samples, response,
test_size=0.2)
```

Funkcija je sadržavala i argument *random_state* koja osigurava da se uvijek uzmu isti nasumični uzorci prilikom optimizacije parametra metodom pokušaja i pogrešaka.

Sada kada smo završili s pripremom podataka potrebno je stvoriti model slučajne šume i postaviti parametre koje želimo dok će ostali ostati na zadanim vrijednostima:

```
rf_classifier = RTrees_create()

set_params(rf_classifier, 0, 20)
```

Gdje je *set_params* korisnički definirana funkcija:

```
def set_params(classifier, active_var, min_sample):
    classifier.setActiveVarCount(active_var)
    classifier.setMinSampleCount(min_sample)
    classifier.setTermCriteria((TermCriteria_MAX_ITER | TermCriteria_EPS, 500, 0.001))
```

Funkcija postavlja parametre slučajne šume na predane joj argumente. Značenje pojedinih parametra je objašnjeno u prethodnom poglavlju. Parametri stabla odluke vezanih za unakrsnu validaciju nisu potrebni budući da ona nije potrebna za algoritam slučajnih šuma. *MaxCategories* parametar također nije potreban jer u ovom modelu postoji samo dvije kategorije odziva. Računanje važnosti pojedinih varijabli nije omogućeno budući da neće biti korišteno. Stvaranje surogat čvorova je također onemogućeno zato što su podatci potpuni. *MaxDepth* parametar je ostavljen na zadanoj vrijednosti koja je *INT_MAX* što označava da će dubina stabla biti maksimalno moguća. Najčešće, kao što je i slučaj s ovim modelom, to daje najbolje rezultate. Pomoću *TermCriteria* parametra povećan je maksimalan broj stabala s 50 na 500 budući da stabla u modelu rastu do maksimalne moguće veličine što može dovesti do *overfitting-a, a* povećanje broja stabala ga kompenzira. Veličina podskupova svojstava koji se koriste za izračun najbolje podijele čvorova je postavljena na 0 što je znak algoritmu da za tu veličinu uzme korijen iz ukupnog broj svojstava tog čvora. Minimalan broj uzoraka koji mora imati čvor da bi se dalje dijelio je postavljen na vrijednost 2 kako bi stablo raslo do maksimalno moguće veličine.

Sljedeći korak je trening. On se obavlja pomoću *train* funkcije iz OpenCV biblioteke koja kao argumente prima matricu uzoraka, oblik uzoraka (red ili stupac) i matricu odziva:

```
rf_classifier.train(samples_train, ROW_SAMPLE, response_train)
```

Nakon što je model istreniran možemo ga testirati tako što ćemo *predict* funkciji predati uzorke za testiranje, a ona će vratiti poredanu matricu odziva (*tuple).*

var, results = rf_classifier.predict(samples_test)

Nakon provedenog testiranja modela možemo izračunati njegovu preciznost koristeći funkciju *accuracy_score* iz Sickit-learn biblioteke koja vraća omjer uspješnih klasifikacija i ukupnog broja klasifikacija:

performance = accuracy_score(response_test, results)

Kako bi vizualno usporedili dobivene i tražene klasifikacije moramo dekodirati kodiranu testnu matricu odziva tako što ćemo na već napravljenom *LabelEncoderu* pozvati funkciju *inverse_transform*:

response_test = le.inverse_transform(response_test)

Slijedi ispis rezultata na ekran:

print_results(response_test, results, performance)

Gdje je *print_results* korisnički definirana funkcija:

```
def print_results(response, results, performance):
    results = ['B' if y == 0 else 'M' for y in results]
    correct_pred= int(round(performance*len(response)))
    incorrect_pred= len(response)-correct_pred
    for x, y in zip(response, results):
        print(" Real value:", x, "  Predicted value:", y)
    print("\n", "Correct predictions:", correct_pred "  Incorrect predictions:", incorrect_pred)
    print("\n", "Algorithm accuracy is:", round(performance*100, 2), "%")
```

Funkcija prvo prolazi kroz dobivenu matricu odziva i mijenja brojčane vrijednosti s oznakama kategorija. Zatim izračunava točne klasifikacije kao umnožak preciznosti i ukupnog broja klasifikacija u matrici odziva te tu vrijednost zaokružuje i pretvara u cjelobrojni tip podataka kako bi se riješili točke i decimalne nule. Netočne klasifikacije su tada jednake razlici ukupnog broja klasifikacija u matrici odziva i točnih klasifikacija. Nadalje za svaki član unutar matrica traženih i dobivenih odziva ispisuje njihove vrijednosti kako bi ih mogli usporediti. Na kraju ispisuje broj uspješnih i neuspješnih klasifikacija i preciznost modela u obliku postotka zaokruženog na dvije decimale.

```
Real value: B   Predicted value: B
Real value: M   Predicted value: M
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: M   Predicted value: M
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: B   Predicted value: B
Real value: M   Predicted value: M
Real value: M   Predicted value: M

Correct predictions: 113   Incorrect predictions: 1

Algorithm accuracy is: 99.12 %

Process finished with exit code 0
```

SL. 4.1. *Dio ispisa print_results funkcije*

Ne postavljajući *random_state* parametar model je testiran na različitim podskupovima podataka. Najniža dobivena preciznost je bila 92,11 % dok je najveća bila 99,12 % kao što vidimo na slici 4.1., pri čemu je od 114 klasifikacija jedna bila neuspješna.

5. ZAKLJUČAK

OpenCV biblioteka osim što ima brojne module za strojno učenje ima i jako velik repozitorij modula i funkcija za obradu slika i videozapisa tako da ne zahtijeva unaprijed obrađene slike i videozapise u obliku gotovih podataka već se mogu obraditi pomoću modula i funkcija iz biblioteke i tako pripremiti za daljnje korištenje. Budući da je fokus ovog rada na algoritam strojnog učenja, korišteni su već obrađeni podatci. Također sve je bilo moguće riješiti pomoću samo OpenCV i NumPy biblioteke, ali je ovaj rad ograničen jedino implementacijom algoritma koji je morao biti u OpenCV biblioteci te su stoga za pripremu podataka i obradu rezultata korištene pomoćne biblioteke koje posjeduju neke naprednije i jednostavnije funkcije za tu svrhu. Do optimalnih parametara slučajne šume koji su dobiveni metodom pokušaja i pogrešaka mogle se doći i nekim drugim metodama koje bi taj zadatak obavile brže i preciznije, ali većina tih metoda bi zahtijevale omotavanje (engl. *Wrapping*) modula u obliku modula neke druge biblioteke, poput *GridSearchCV* metode iz Sickit-learn biblioteke. Ali i bez korištenja tih metoda algoritam je postigao prilično dobre rezultate s preciznošću koja se kreće između 92,11 i 99,12 % ovisno o uzetom podskupu podataka. Cilj je naravno da su oscilacije što manje, a preciznost što bliže gornjoj granici od 100 %. Iako je trenutno jako teško, možda i nemoguće, napraviti algoritam koji će obavljati takve zadatke sa 100-postotnom sigurnošću, pokazalo se da je umjetna inteligencija u velikom broju slučajeva dosta preciznija od ljudi.

PRILOZI

Programski kod:

```
from numpy import int32, float32

from cv2.ml import RTrees_create, ROW_SAMPLE
from cv2 import TermCriteria_MAX_ITER, TermCriteria_EPS

from pandas import read_csv

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score



def load_data(data_file):
    data = read_csv(data_file, dtype=None, delimiter=',')
    samples = data.iloc[:, 2:31].values
    response = data.iloc[:, 1].values
    return samples, response


def set_params(classifier, active_var, min_sample):
    classifier.setActiveVarCount(active_var)
    classifier.setMinSampleCount(min_sample)
    classifier.setTermCriteria((TermCriteria_MAX_ITER | TermCriteria_EPS, 500, 0.001))

def print_results(response, results, performance):
    results = ['B' if y == 0 else 'M' for y in results]
    correct_pred = int(round(performance*len(response)))
    incorrect_pred = len(response)-correct_pred
    for x, y in zip(response, results):
        print(" Real value:", x, "  Predicted value:", y)
```

```
    print("\n", "Correct predictions:", correct_pred, "  Incorrect predictions:", incorrect_pred)
    print("\n", "Algorithm accuracy is:", round(performance*100, 2), "%")

def main():
    data_file = open('/home/wolfie/Desktop/Završni/RFA-py/breast_cancer_dataset.csv')
    samples, response = load_data(data_file)
    le = LabelEncoder()

    samples = float32(samples)
    response = int32(le.fit_transform(response))

    samples_train, samples_test, response_train, response_test = train_test_split(samples,
    response, test_size=0.2)

    rf_classifier = RTrees_create()

    set_params(rf_classifier, 0, 2)

    rf_classifier.train(samples_train, ROW_SAMPLE, response_train)

    var, results = rf_classifier.predict(samples_test)

    performance = accuracy_score(response_test, results)

    response_test = le.inverse_transform(response_test)

    print_results(response_test, results, performance)

if __name__ == "__main__":
    main()
```

LITERATURA

[1] „Strojno učenje" https://www.sas.com/en_us/insights/analytics/machine-learning.html lipanj, 2019.

[2] „Slika formule za entropiju" dostupno na https://www.scirp.org/html/6-9101686/f799e10c-50bd-48ec-9344-49d767083be5.jpg rujan, 2019.

[3] „Algoritam stabla odluke, algoritam slučajne šume, ID3 algoritam" https://bib.irb.hr/datoteka/145853.Upotreba_stabla_odlucivanja_u_testiranju_znanja_pomocu_kviza.pdf rujan, 2019.

[4] „Slika primjera slučajne šume" dostupno na https://victorzhou.com/blog/intro-to-random-forests/ lipanj, 2019.

[5] „Opis OpenCV biblioteke" https://docs.opencv.org/4.1.0/d1/dfb/intro.html rujan, 2019.

[6] „OpenCV" https://en.wikipedia.org/wiki/OpenCV lipanj, 2019.

[7] „Metode *RTrees* klase u OpenCV biblioteci", „Slika strukture nasljeđivanja u OpenCV-u" dostupno na https://docs.opencv.org/4.1.0/d0/d65/classcv_1_1ml_1_1RTrees.html lipanj, 2019.

[8] „Metode *DTrees* klase u OpenCV biblioteci" https://docs.opencv.org/4.1.0/d8/d89/classcv_1_1ml_1_1DTrees.html lipanj, 2019.

[9] „Metode *StatModel* klase u OpenCV biblioteci" https://docs.opencv.org/3.4.5/db/d7d/classcv_1_1ml_1_1StatModel.html lipanj, 2019.

[10] „UCIML baza podataka za strojno učenje kalifornijskog sveučilišta u Irvinu" https://www.kaggle.com/uciml/breast-cancer-wisconsin-data srpanj, 2019.

SAŽETAK

U ovom radu je obrađena teorijska podloga algoritma slučajnih šuma i njegova implementacija u OpenCV biblioteci. Primjena teorijske podloge je prikazana na primjeru klasifikacije stanica tumora dojke. U tu svrhu, uz ostale pomoćne biblioteke, korištena je OpenCV biblioteka te Python programski jezik i UCIML baza podataka. Nakon kreiranja slučajne šume i postavljanja njenih parametara sljedeći korak je bio učenje. Učenje je provedeno na 455 uzoraka gdje je svako stablo odluke unutra slučajne šume učeno koristeći nasumično odabran podskup uzoraka. Testiranje je provedeno na skupu od 114 uzoraka s maksimalnom preciznošću od 113 uspješnih klasifikacija i minimalnom preciznošću od 105 uspješnih klasifikacije ovisno o nasumično odabranom podskupu uzoraka.

Ključne riječi: strojno učenje, OpenCV, slučajna šuma, klasifikacija, Python, UCIML

ABSTRACT

Title: Random forest algorithm in OpenCV library

In this paper the theoretical background of the random forest algorithm and its implementation in OpenCV library was elaborated. Application of the theoretical background was presented on the example of breast tumor cells classification. In addition to other support libraries, the OpenCV library was used with the Python programming language, and the UCIML database. After a random forest object was created and its parameters were set, the next step was learning. Learning was carried out on 455 samples where each decision tree inside a random forest was taught using a randomly selected subset of samples. Testing was performed on a set of 114 samples with a maximum precision of 113 successful classifications and a minimum precision of 105 successful classifications depending on a randomly selected subset of samples.

Key words: machine learning, OpenCV, random forest, classification, Python, UCIML

ŽIVOTOPIS

Denis Lazor rođen je 5. srpnja 1997. u Ćupriji. U Slatini je završio osnovnu školu te je nakon toga upisao Opću gimnaziju. Završetkom gimnazije upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Trenutačno pohađa treću godinu Preddiplomskog sveučilišnog studija računarstva.