SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

## Algoritmi strojnog učenja za klasifikaciju fragmenata datoteka

Petra Omrčen

Voditelj: dr.sc. Juraj Petrović

Zagreb, svibanj 2019.

**SADRŽAJ**

1. UVOD………………………………………………………………………………………………………….3
2. SPREMANJE DATOTEKA……………………………………………………………………………….4
3. KLASIFIKACIJA FRAGMENATA DATOTEKA…………………………………………………….6
4. ALGORITMI STROJNOG UČENJA…………………………………………………………………..8

4.1. Umjetne neuronske mreže………………………………………………………….8

4.2. Konvolucijske mreže…………………………………………………………………13

4.3. kNN…………………………………………………………………………………………..16

4.4. Algoritam slučajnih šuma………………………………………………………….19

1. IMPLEMENTACIJA……………………………………………………………………………………..23
2. ZAKLJUČAK………………………………………………………………………………………………..25
3. LITERATURA………………………………………………………………………………………………26
4. DODATAK………………………………………………………………………………………………….27
5. **UVOD**

U današnje vrijeme ljudi većinu bitnih stvari vezanih uz posao, privatan život i razne aktivnosti kojima se bave čuvaju na svojim računalima. To je dobar odabir zato što se brzo može pristupiti traženim podacima, ne zauzima se puno fizičkog prostora i sve je, može se reći, na jednome mjestu. Problem se javlja kada se dogodi određena promjena s računalom i više se ne može pristupiti stvarima koje su spremljene na njemu. Jedna od tih promjena može se dogoditi s tablicom koja nam govori kako i gdje su spremljene datoteke na disku. Datoteke se prije spremanja podijele na fragmente te se tako spremaju na disk i ne moraju nužno dijelovi iste datoteke biti jedni kraj drugih. Tada je potrebno naći način kako odrediti tipove tih fragmenata kako bi mogli pronaći određenu datoteku i pročitati njezin sadržaj u potpunosti.

Većina dosadašnjeg rada na identifikaciji fragmenta datoteka pokušala je riješiti problem klasifikacije pomoću kombinacije tehnika strojnog učenja i statističke analize. Istraživači obično podjele podatke s kojima rade u dvije skupine, “set za obuku” i “set za treniranje”. Datoteke iz skupa za obuku obrađuju se nekom vrstom statističke tehnike i rezultati se unose u algoritam strojnog učenja. Rezultati se koriste za izradu klasifikatora. Set za testiranje se zatim unosi u klasifikator.

Također dosta digitalnih forenzičkih istraživanja uključuje upravo analizu fragmenata datoteka.

U ovom seminarskom radu opisati ću način rada nekoliko algoritama strojnog učenja koji mogu poslužiti za određivanje tipova fragmenata odnosno za klasifikaciju fragmenata datoteka. Strojno učenje je danas jedno od najaktivnijih i najuzbudljivijih područja računarske znanosti. Ono omogućava računalima da uče na sličan način na koji ljudi to rade: stroj prikuplja znanje bazirano na prošlom iskustvu. Umjesto da mu se stalno mora ažurirati softverski kod, on je, kako vrijeme prolazi, sposoban samostalno poboljšati svoj rad.

**2. Spremanje datoteka**

Svi podaci na disku moraju biti organizirani u datoteke. Datoteka je niz bajtova pohranjenih u datotečnom sustavu pod korisničkim imenom.

One obično sadrže:

1) programe: .exe, .out, .bat, .sh, .dll, .jar

2) podatke: -dokumente(word, tekstualne datoteke, HTML,…)

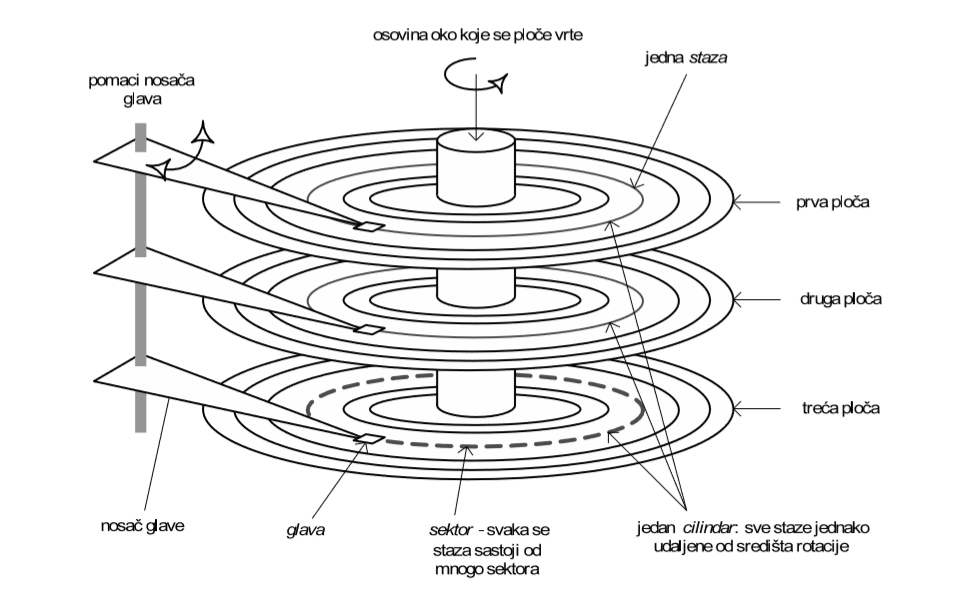
-multimediju(slike, video, muzika)

-postavke programa I sustava

3) ostalo: -privatne podatke OS-a

-privatne podatke datotečnog sustava

Datotečni sustav se brine za to kako su datoteke smještene na disku, kako doći do njih, koji dijelovi diska su slobodni… Disk se sastoji od određenog broja ploča, cilindara, staza, nosača glave i glave te sektora(podatkovne jedinice i u njih se upisuju dijelovi datoteka. Veličina sektora je 512 bajtova.)



Slika

Datotečnoj tablica sadrži:

* podatke koji definiraju disk, slobodni prostor (ponekad su ove informacije u zasebnim strukturama izvan tablice)
* opisnik datoteke: -ime datoteke

-direktorij gdje je datoteka smještena (u logičkoj org. diska)

-tip datoteke

-veličina datoteke

-vrijeme stvaranja, zadnje promjene, zadnjeg korištenja

-podaci o “vlasniku” (kojem korisniku pripada) – prava pristupa

-opis smještaja na disku (u kojim blokovima)

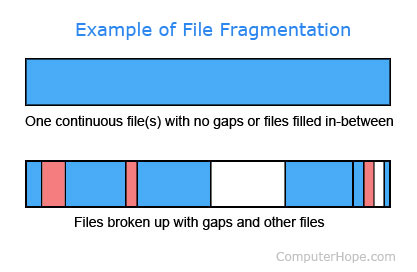
Bitno je znato što točno piše u opisniku. Pomoću njega određujemo koje će se datoteke čitati, brisati, mijenjati te sve ostale promjene vezane uz njih.

Datoteka se na disk ne sprema u jednom komadu već se razlomi na više dijelova i oni se spremaju u sektore. Idealno bi bilo kada bi bilo dovoljno mjesta tako da je datoteka spremljena na jednom mjestu odnosno svi sektori u koje je ona spremljena su jedni kraj drugih, ali to često nije slučaj te dolazi do fragmentacije.

**3.Klasifikacija fragmenata datoteka**

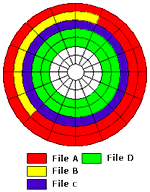
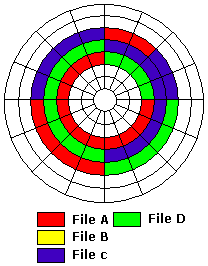
Fragmentacija

Fragmentacija datoteke je pojam koji opisuje grupu datoteka koje su razbacane po tvrdom disku umjesto da su na jednom kontinuiranom mjestu. Do toga dolazi kada se datoteke brišu s diska i ostaju rupe u koje se upisuju nove datoteke koje ne zauzimaju sav ponuđeni prostor nego samo dio. Tako nastaju sve manje rupe.



Slika

Na kraju, nova se datoteka neće moći kontinuirano upisati na disk već će se podijeliti u preostale manje rupe. Kad se to dogodi trebat će više vremena da se nađu svi dijelovi datoteke jer će glava na kojoj je čitač morati radit više okretaja te prolazit više staza da bi našla odgovarajući sektor u kojem se nalazi drugi dio tražene datoteke. Računalo znatno sporije izvršava zadatke kada se dogodi fragmentacija.



Slika Slika

Moguće je da se dogodi još veći problem, a to je da se opisnik datoteke poremeti, izgubi ili promjeni te onda ne znamo gdje su početak, sredina i kraj datoteke, pripadaju li određeni sektori uopće istoj datoteci, kojeg je tipa datoteka, koliko je velika datoteka te sva ostala pitanja na koja nam inače opisnik datoteka daje odgovor.

U tom slučaju morat ćemo na neki način klasificirati fragmente datoteka i odrediti njihov tip(format). Za to se danas koriste tehnike strojnog učenja i statističke analize.

**Tri načina određivanja tipova datoteka:**

1) EKSTENZIJA

Najbrži i najjednostavniji način određivanja tipa datoteke je prema ekstenziji koju ima. Najveća prednost ove metode je brzina detekcije formata. Nema potrebe za otvaranjem datoteke kako bi odredili tip. S druge stane ova metoda ima veliku ranjivost zato što se može vrlo lako zavarati promjenom imena ekstenzije.

2) MAGIČNI BAJTOVI

Druga metoda temelji se na magičnim bajtovima. Magični bajtovi su unaprijed definirani potpisi koji se mogu naći u zaglavlju datoteke. Ima nekoliko tisuća tipova datoteka za koje su magični bajtovi definirani i ima više lista magičnih bajtova koji nisu potpuno konzistentni. Provjera magičnih bajtova datoteka znatno je sporija metoda zato što se datoteka mora otvoriti, magični brojevi se trebaju iščitati i usporediti s nekim unaprijed definiranima. Jedan od glavnih nedostataka ove metode je nedostatak unaprijed definiranog standarda za programere, tako da se magični bajtovi ne koriste u svim vrstama datoteka. Štoviše, čarobni bajtovi rade samo na binarnim datotekama, a unaprijed definirani potpisi razlikuju se po duljini za različite tipove datoteka. Kada se priloži digitalni medij s datotekama izmijenjenog potpisa, forenzički softver će ukazati na prijevaru i sugerirati forenzičkom analitičaru pravi tip datoteke.

3) ISPITIVANJE SADRŽAJA DATOTEKE

Treća metoda temelji se na ispitivanju sadržaja datoteka i u tome nam pomažu algoritmi strojnog učenja koje ćemo obraditi u nastavku rada.

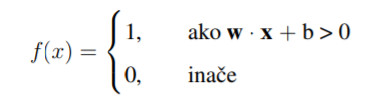
**4. Algoritmi strojnog učenja**

Opisati ću način rada četiri algoritama koji mogu poslužiti za klasifikaciju fragmenata datoteka. Kao ulaze uzimati će histograme bajtova(256).

**4.1. Umjetne neuronske mreže**

Umjetna neuronska mreža je niz algoritama koji nastoje prepoznati temeljne odnose u skupu podataka kroz proces koji oponaša način na koji ljudski mozak djeluje. Oponašaju način na koji naš mozak rješava probleme: uzima neke ulazne podatke/informacije, obrađuje ih i generira izlaz. Poput nas, oni uče prepoznati uzorke, ali to rade obukom(treniranjem) na označenim skupovima podataka. Neuronske mreže mogu se prilagoditi promjenama ulaza; tako mreža stvara najbolji mogući rezultat bez potrebe za redizajnom izlaznih kriterija.

Prvi model neuronske mreže temelji se na uočenom ponašanju neurona koje zahtjeva prekoračenje određenog praga podražaja prema neuronu kako bi generirao impuls. Inspirirano time konstruiran je umjetni neuron, nazvan **perceptron**. On je zamišljen kao binarni klasifikator. Perceptron prima niz ulaza, to jest signala, koji se množe sa težinskim faktorima koji definiraju unutarnje stanje perceptrona, na kraju se zbrajaju kako bi dali jedan izlaz. Na izlazu je definiran prag koji odlučuje kako će se izlazna vrijednost klasificirati.



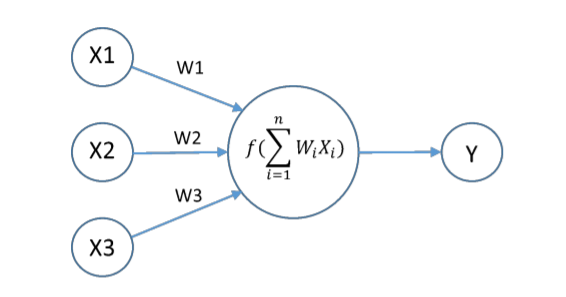
Slika

W – vektor težina

X – ulazni vektor

**Svaka neuronska mreža sadrži:**

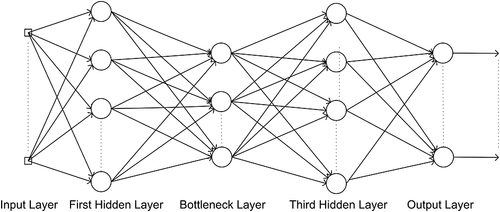
1. Neuron - Nešto što ima u sebi broj od 0 do 1 i to se naziva aktivacija. Aktivacija nam govori koliko je aktivan naš neuron. Što je on aktivniji, odnosno što je broj veći to je veća šansa da je upravo on naš traženi izlaz.



Slika

1. Ulazni sloj - Prvi sloj neuronske mreže. Ulazi neurona u ulaznom sloju su ulazni podaci problema; u našem slučaju histogrami bajtova.
2. Izlazni sloj - Posljednji sloj mreže i izlazi neurona koji se nalaze u njemu zapravo predstavljaju rješenje problema.
3. Skriveni slojevi - Svi ostali slojevi koji se nalaze između ulaznog i izlaznog. Može biti proizvoljan broj skrivenih slojeva. Ne vrijedi nužno da će mreža bolje raditi ako ima više slojeva.

Ulazni podaci na određeni način aktiviraju neurone u prvom sloju i taj uzorak aktivacije određuje ponašanje ostatka mreže i na kraju onaj “najaktivniji“ neuron predstavlja odabir koji je učinila naša neuronska mreža. Kada se govori o neuronskim mrežama najčešće se misli na unaprijeđene slojevite mreže. To su one kod kojih se izlaz prethodnog sloja dovodi na ulaz sljedećeg sloja. Svi neuroni su međusobno povezani, npr. svaki neuron iz prvog sloja je povezan sa svakim neuronom iz prvog skrivenog sloja itd.



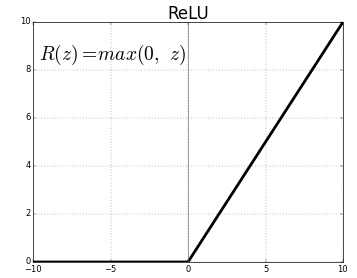
Slika

Zašto se ovakva struktura uopće ponaša inteligentno?

Ideja je da se problem koji promatramo podijeliti na manje dijelove i da svaki sloj određuje neke karakteristike koje vode do konačne odluke(u našem slučaju kojeg tipa je datoteka). Svaki sloj trebao bi biti sve specifičniji tako da predzadnji sloj može gotovo sigurno dati dobar rezultat.

**AKTIVACIJSKE FUKCIJE I TRENIRANJE MREŽE:**

Svakoj vezi između dva neurona dodjeljuje se određena težina(engl *weight*). Aktivacijske funkcije nam pomažu u određivanju tih težina. Naime, nakon svakog treniranja mreže na izlazu dobijemo izlazne neurone koji su aktivirani na određen način i te aktivacije čine vektor. Vektor se zatim uspoređuje s očekivanim vektorom i računa se pogreška. Nakon toga na temelju te pogreške mreža mijenja težine između neurona kako bi sljedeći put imam bolje rezultate. Za računanje se prije koristila sigmoid funkcija, a danas se koristi ReLu funckija(engl *rectified linear unit*) zbog lakšeg treniranja za duboke neuronske mreže.



AKTIVNO

f(a)=a

Slika

NEAKTIVNO

Može se dodat još jedna stvar ako ne želimo da nam se neuron aktivira na 0 nego npr. na 10. Tada uzimamo određeni (engl) “Bias“ i dodajemo ga u jednadžbu i on nam govori koliko velika ta suma mora biti prije nego što se taj neuron aktivira.

Za trenirane mreže mora se koristiti određen broj ulaznih podatak i povratna informacija o tome je li mreža napravila dobar posao odnosno treba nam vektor s kojim ćemo uspoređivati izlaz koji daje neuronska mreža. Nakon toga mreža se može testirati na podacima koje nikad nije vidjela i dati dobre rezultate zato što je preko treniranja naučila kako treba raditi.

Još bolja definicija NEURONA: to je funkcija koja prima izlazne vrijednosti od svih neurona iz prethodnog sloja i izbaci broj između 0 i 1

Cijela neuronska mreža je ustvari vrlo komplicirana funkcija.

**PREDNOSTI:**

* Neuronske mreže su fleksibilne i mogu se koristiti za probleme regresije i klasifikacije. U modelu se mogu koristiti bilo koji podaci koji se mogu numerički koristiti, jer je neuronska mreža matematički model s aproksimacijskim funkcijama.
* Neuronske mreže su dobre za modeliranje s nelinearnim podacima s velikim brojem ulaza; na primjer, slike.
* Pouzdana je za zadatke koji uključuju veliki broj značajki. Radi tako da se problem klasifikacije dijeli na slojevitu mrežu jednostavnijih elemenata. Kad je jednom trenirana, predviđanja koja daje su prilično brza.
* Neuronske mreže mogu se trenirati s bilo kojim brojem ulaza i slojeva.

**MANE:**

* Neuronske mreže su crne kutije, što znači da ne možemo znati koliko svaka nezavisna varijabla utječe na ovisne varijable.
* To je vrlo skupo i dugotrajno vježbanje s običnim procesorima.
* Neuronske mreže puno ovise o podacima za obuku. To dovodi do problema prekomjerne opremljenosti(engl overfitting) i generalizacije. Način rada više se oslanja na podatke o vježbanju i može se podesiti na podatke.

**Radovi u kojima se koristi ovaj algoritam:**

**File Type Identification for Digital Forensics**

Karampidis Konstantinos, Papadourakis Giorgos.

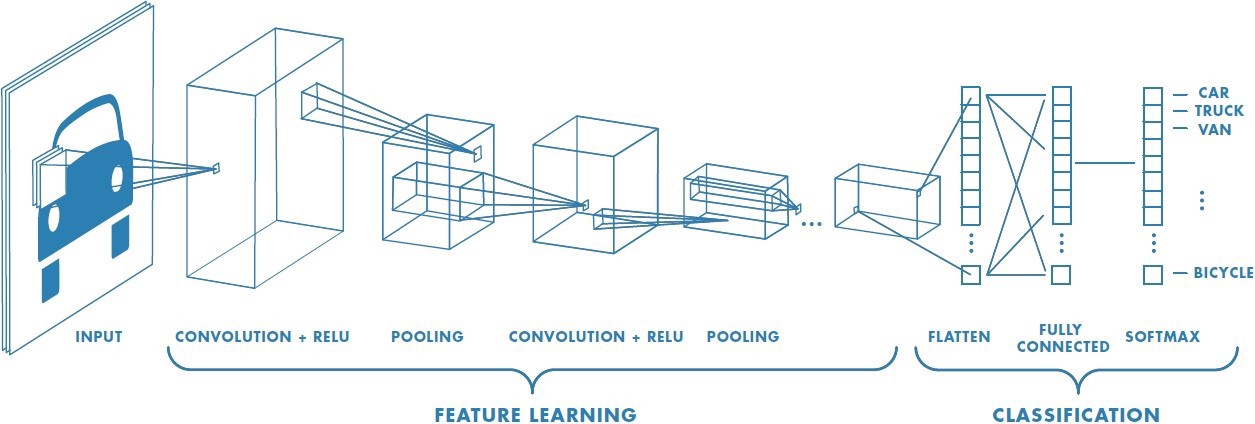
Conference: International Conference on Advanced Information Systems Engineering

Technological Educational Institute of Crete Department of Informatics Engineering

Heraklion Crete, Greece, 2016.<https://www.researchgate.net/publication/303823046_File_Type_Identification_for_Digital_Forensics>

**4.2. Konvolucijska mreža**

Konvolucijska neuronska mreža je nadogradnja nad običnim višeslojnim mrežama. Ona se sastoji, kao i obična mreža, od jednog ulaznog, jednog izlaznog i jednog ili više skrivenih slojeva. Kod konvolucijskih mreža specifični su konvolucijski slojevi i slojevi sažimanja te se oni izmjenjuju nekoliko puta te najčešće završavaju s jednim ili više potpuno povezanih slojeva koji služe za klasifikaciju.



Slika

**KONVOLUCIJSKI SLOJ**

Svaki konvolucijski sloj sastoji se od filtara koji sadrže težine koje je potrebno naučiti kako bi mreža davala dobre rezultate. Filtri su najčešće manjih dimenzija od ulaza, no uvijek su jednake dubine kao i ulaz. Potrebno je konvoluirat filtar s ulazom, rezultat konvolucije je dvodimenzionalna aktivacijska mapa koja predstavlja odziv filtra na svakoj prostornoj poziciji. Mreža će naučiti težine unutar filtra kako bi se filtar „aktivirao“ na mjestima gdje prepoznaje određena svojstva.

Kako bi bilo moguće definirati točnu veličinu izlaza konvolucijskog sloja potrebno je definirati i korak pomaka (engl *stride*). On definira koliko će se filtar pomicati po širini odnosno visini tijekom konvolucije.

Wy -širina izlaza

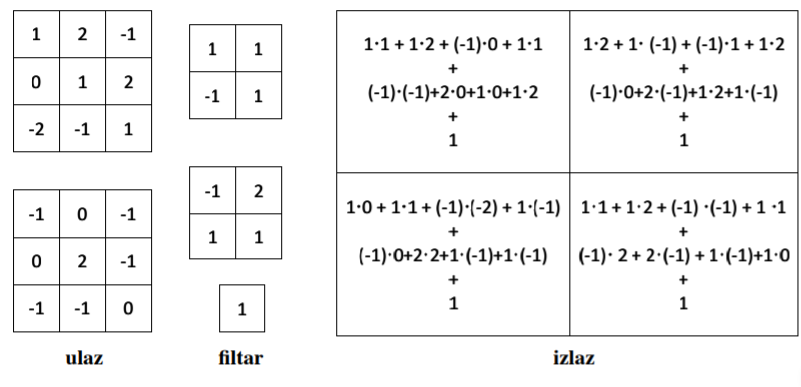
Wx -širina ulaza

S -korak pomaka filtra

F -veličina kvadratnog filtra

Primjer:

Definirane su dvije dvodimenzionalne matrice koje predstavljaju ulaz u konvolucijsku mrežu s dvije komponente. Ulaz se sastoji od dvije komponente tako da i u dubina filtra mora biti 2



Slika

Za izračunavanje izlaza konvolucijskog sloja, odnosno određivanje aktivacijske mape potrebno je provesti konvoluciju nad ulazom i filtrom. Konvolucija se provodi tako da odgovarajuće ulaze pomnožimo s težinama, dobivene vrijednosti se sumiraju i dodaje se prag. Pag je dodatna težina kod filtra.

Nakon što se izračunaju sve vrijednosti, kao i kod jednostavne neuronske mreže, dobivene vrijednosti potrebno je provesti kroz aktivacijsku funkciju.

**SLOJ SAŽIMANJA**

Slojevi sažimanja koriste se nakon nekoliko konvolucijskih slojeva s ciljem smanjivanja rezolucije mapi. Manji dio uzorka grupira se i obrađuje, te daje jednu vrijednost. Postoji nekoliko vrsta sažimanja; najčešći su sažimanje usrednjavanjem(grupirani podaci zamjenjuju se aritmetičkom sredinom grupiranih vrijednosti) i sažimanje maksimalnom vrijednosti(grupirane vrijednosti zamjenjuje se maksimalnom vrijednošću). Broj komponenata(dubina) prije sloja sažimanja i nakon sažimanja ostaje isti.

**UČENJE KONVOLUCIJSKE NEURONSKE MREŽE**

Algoritam propagacije pogreške unatrag.

Učenje mreže kreće s kraja prema početku. Prvo se računa greška posljednjeg sloja koja se propagira prema ranijim slojevima. Cilj učenja konvolucijskog sloja je naučiti težine koje se nalaze unutar filtra. Kako bi naučili težine potrebno je odrediti parcijalne derivacije greške po težinama.

**PREDNOSTI:**

* Najznačajnija prednost konvolucijske mreže je automatsko izdvajanje značajki za zadani zadatak.
* Ne zahtjevaju ogroman broj neurona koji bi prouzročili da treniranje traje jako dugo.

**MANE**:

* Učenje bez nadzora

Cilj strojnog učenja je imati strojeve koji mogu sami učiti, ali konvolucijske neuronske mreže još uvijek moraju biti nadzirane.

* Memorija

Konvolucijske mreže imaju problem zaboravljanja. One imaju tendenciju zaboravljanja prethodno uvježbanog zadatka kada ih pokušamo naučiti novom. To se događa zbog prepisivanja djelotvornosti veze ili težine.

* Rasuđivanje

CNN je slijepa za logiku i rasuđivanje zbog nedostatka reprezentacije znanja unutar sebe.

**Radovi u kojima se koristi ovaj algoritam:**

**File Fragment Type Identification with Convolutional Neural Networks**

Yanchao Wang, Zhongqian Su, Dayi Song

Conference: the 2018 International Conference

<https://www.researchgate.net/publication/327336441_File_Fragment_Type_Identification_with_Convolutional_Neural_Networks>

**4.3. kNN-K najbližih susjeda**

kNN je jedan od najjednostavnijih algoritama strojnog učenja koji se lako implementira i može se koristiti za klasifikaciju i regresiju. On spada u metode koje se zasnivaju na pronalaženju najsličnijih podataka podacima koje želimo klasificirati.

Algoritam KNN-a

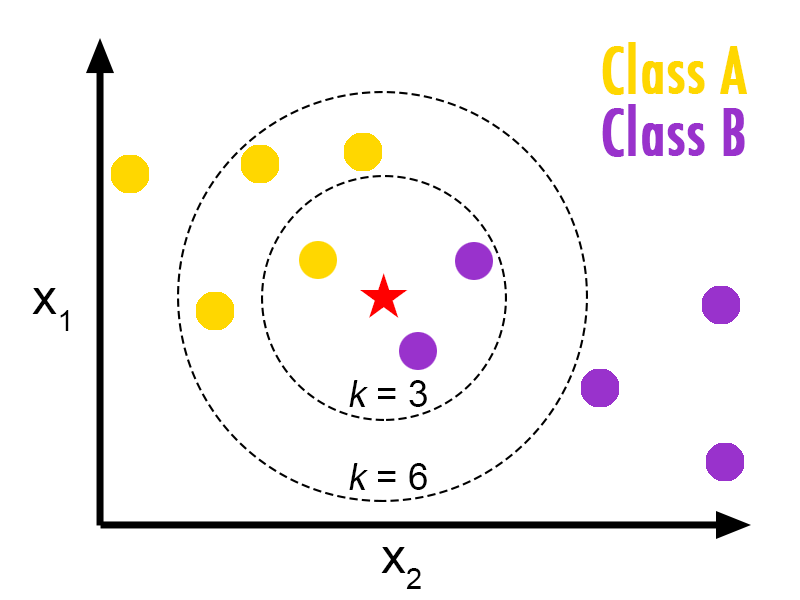
1. Odabire se pozitivan cijeli broj k, zajedno s novim uzorkom.
2. U našoj bazi podatak odabiremo k unosa koji su najbliži novom uzorku.
3. Nalazimo najčešću klasifikaciju tih unosa.
4. To je klasifikacija koju dajemo novom uzorku.

Udaljenost između dva podatka definira se kao Euklidska udaljenost.

N a početku dobijemo n vektora za treniranje, na primjeru slike ispod, to su žute i ljubičaste točkice i trebamo odrediti kojeg tipa je crvena zvjezdica. Odabiremo k najbližih susjeda.

U primjeru imamo dvije klase: ljubičasta i žuta.

Prvo odabiremo k=3:



Slika

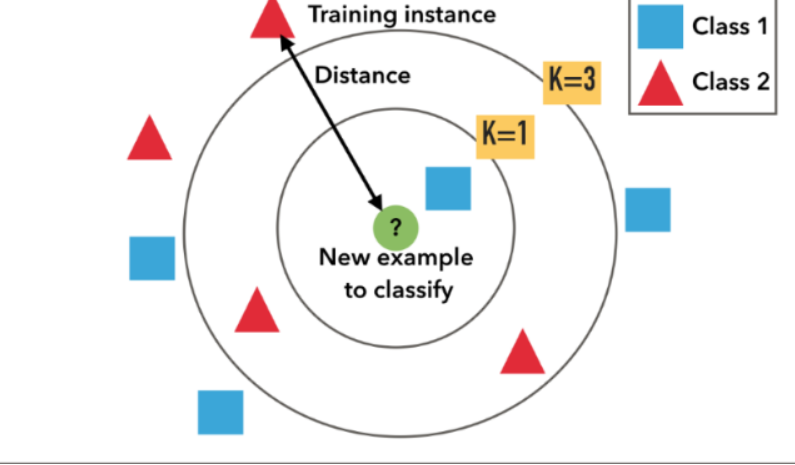
Prva kružnica predstavlja skupinu 3 najbliža susjeda, tamo se nalaze dvije ljubičaste i jedna crvena točkica. Ljubičastih je očito više te mi pretpostavljamo da je i zvjezdica ustvari ljubičasta točkica odnosno spada u tu klasu.

Zanimljivo je da ako uzmemo neki drugi k, npr. k=6, dobijemo drugačije rješenje. U našoj kružnici nalaze se četiri žute i samo dvije ljubičaste točkice i zaključak je da je naša zvjezdica ustvari žuta točkica.

Kako izabrati K?

Da bi odabrali K koji je ispravan za naše podatke, kNN algoritam pokrećemo nekoliko puta s različitim vrijednostima k i odabiremo onaj k koji smanjuje broj grešaka na koje nailazimo dok zadržavamo sposobnost algoritma da radi precizna predviđanja za podatke koje još nikad nije susreo.

* Kako smanjujemo vrijednost K prema 1, naša predviđanja postaju manje stabilna.
* Obrnuto ako povećavamo vrijednost K naša predviđanja postaju stabilnija zbog većinskog glasovanja, srednje vrijednosti…
* U jednom trenutku počinjemo dobivati sve više pogrešaka i tada znamo da smo previše povećali vrijednost K.
* K se može birati kao korijen iz ukupnog broja elemenata
* Treba se paziti da ne bude višekratnik broja klasa
* Ako imamo dvije klase bilo bi dobro odabrati čudan broj kako ne bi došlo do previše zabuna.



Slika

**PREDNOSTI:**

* Algoritam je jednostavan i lagan za implementiranje
* Nema potrebe za izgradnjom modela, podešavanjem parametara ili donošenjem dodatnih pretpostavki
* Algoritam je svestran, može se koristiti za klasifikaciju, regresiju i pretraživanje.

**MANE:**

* Algoritam postaje znatno sporiji s porastom broja primjeraka
* Potrebna velika memorija
* Osjetljiva na nevažne značajke

Zanimljivosti:

Ne postoji eksplicitna faza treniranja ili je vrlo minimalna, to također znači da je faza treninga prilično brza. Nedostatak generalizacije znači da kNN čuva sve podatke za treniranje. Točnije, svi (ili većina) podataka za treniranje potrebni su tijekom faze testiranja.

**Radovi u kojima se koristi ovaj algoritam:**

**Fast Content-Based File Type Identification**

Irfan Ahmed, Kyung-Suk Lhee, Hyun-Jung Shin and Man-Pyo Hong

<https://link.springer.com/content/pdf/10.1007/978-3-642-24212-0_5.pdf>

* 1. **Algoritam slučajnih šuma**

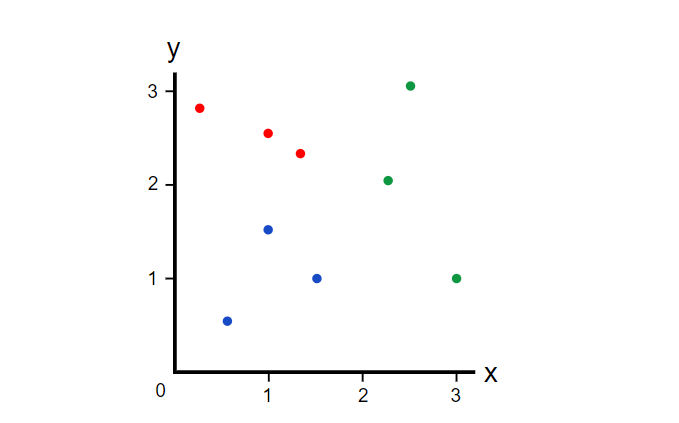
Slučajne šume ili šume s nasumičnim odabirom su metode strojnog učenja koje se koriste za klasifikaciju, regresiju i ostale zadatke koji zahtijevaju konstruiranje mnoštva stabala odlučivanja za vrijeme treniranja i kao izlaz daje neku klasu.

Slučajna šuma gradi mnoštvo stabala odlučivanja i spaja ih zajedno kako bi dobila točnije i stabilnije predviđanje.

**STABLA ODLUČIVANJA:**

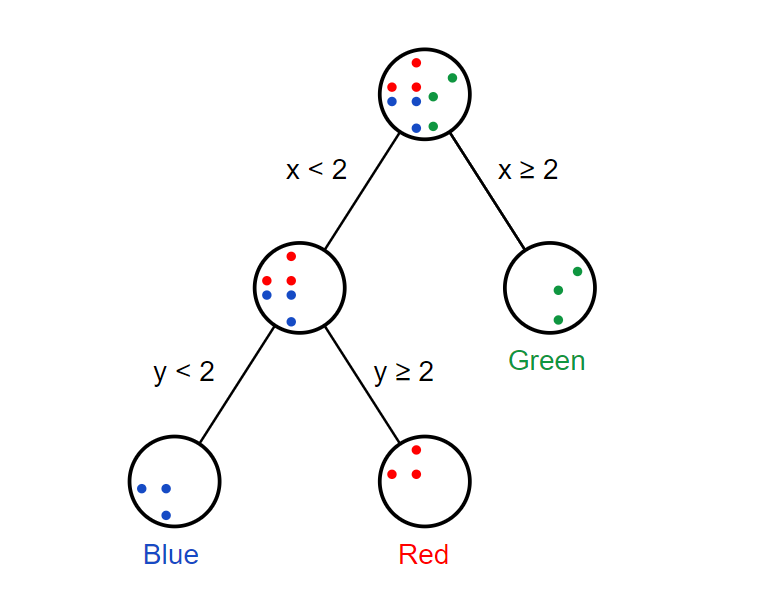
Stabla odlučivanja se koriste za dobivanje odgovora na pitanje: s obzirom na dani skup podataka kako bismo klasificirali nove uzorke?

Klasificirati novi ulazni podatak znači dodijeliti ga nekoj klasi. U našem slučaju za klasifikaciju fragmenatata datoteka naše klase će predstavljati tipove datoteka i mi ćemo naš ulazni podatak(histogram bajtova) dodijeliti nekoj od tih klasa, odnosno odredit ćemo njezin tip.



Slika

Najlakše je razumjeti rad algoritma na primjeru. Recimo da je ovo naš skup podataka. Prvo je potrebno odrediti čvor odlučivanja(podjelu) koji će odrediti npr. hoće li crvena boja biti lijevo od x=2. Dogodit će se grananje (slika 13). U desno listu biti će samo zelena boja, ali u lijevom listu su i dalje dvije boje, crvena i plava. Moramo opet odabrati čvor odlučivanja te će se dogoditi još jedno grananje. Dobar odabir bio bi y=2 zato što će se tako stvoriti nova dva lista i svaki će sadržavati po jednu boju.



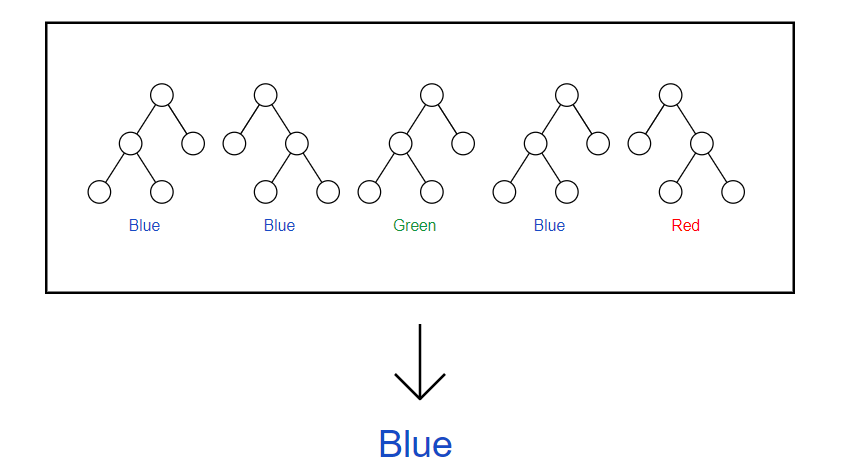
Slika

Ovo je primjer dobre podjele. Podjela je bolja što se više odvoje/podijele različite klase jedne od drugih; u ovom slučaju odvajanje boja. Također se može odrediti točno koliko je dobra podjela. Gini indeks je mjerni podatak koji mjeri koliko često bi slučajno odabrani element bio pogrešno identificiran. To znači da treba dati prednost atributu s nižim gini indeksom. Na temelju tih izračuna određuju se najbolje podjele i one se koriste u daljnjoj izgradnji stabla.

S podjelama prestajemo kada više ne možemo definirati razliku između elemenata u listu što drugim riječima znači da su svi elementi iste klase i da smo dobro podijelili naše ulazne podatke. Primjer toga nalazi se na slici 14. U svakom listu nalazi se samo jedna bolja i više nemamo na temelju čega raditi podjelu.

**SLUČAJNE ŠUME:**

Pakiranje(engl *bagging*): Treniramo stabla sa skupom podataka za treniranje. Zatim radimo predviđanja rezultata s određenim brojem individualnih stabala(parametar t). Svako stablo daje neko rješenje/izlaz. Ako se radi o klasi, npr. boje, uzimamo klasu koja je najviše zastupljena, a ako se radi o brojčanoj vrijednosti kao rezultatu onda uzimamo prosječnu vrijednost. Na slici vidimo da je većina stabala dala izlaz plave boje tako da zaključujemo da je to klasa u koju pripadaju ulazni podaci.



Slika

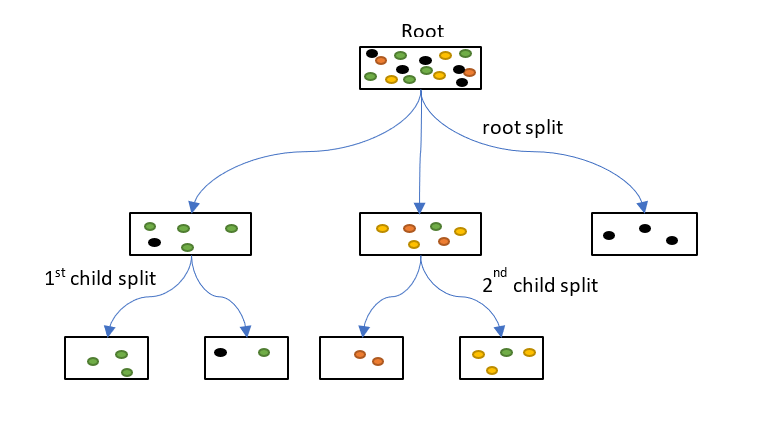
Konačno, slučajne šume jako su slične pakiranju samo što imaju jednu dodatnu stvar osim parametra t, a to je parametar koji kontrolira koliko značajki treba isprobati pri odabiru najbolje podjele. U normalnim uvjetima skupovi podataka imat će i stotine tisuća tih značajki.

**PREDNOSTI:**

* Velika prednost slučajne šume je u tome što se može koristiti i za klasifikacijske i za regresijske probleme, koji čine većinu postojećih sustava strojnog učenja.
* Lako se vizualiziraju i interpretiraju.
* Lako može snimiti linearne uzorke
* Zahtijeva manje obrade podataka od korisnika, na primjer, nema potrebe za normalizacijom stupaca

**MANE:**

* Jedan od velikih problema u strojnom učenju je preopterećenje(overfitting, algoritam strojnog učenja je treniran na skupu podataka za obuku, nakon toga primjenjuje se na novu skupinu podataka. Cilj je maksimalno povećati točnost predviđanja na novim podacima, a ne nužno i na podacima za treniranje. Ako se previše trudimo pronaći najbolje podatke za poduku postoji rizik da će se algoritam zapamtiti različite posebnosti treninga podatak umjesto da utvrdi opće pravilo predviđanja), ali to se neće dogoditi tako lako za klasifikator slučajnih šuma ako ima dovoljno stabala.
* Još jedno ograničenje je to da velik broj stabala može napraviti algoritam koji je prespor i neučinkovit za predviđanja u stvarnom vremenu. Uglavnom su svi algoritmi brzi za vježbanje, ali prilično spori za stvaranje predviđanja nakon što su obučeni.



Slika

Razlika slučajnih šuma i stabala odluka:

Ako unesemo skup podataka za treniranje s oznakama i značajkama u stablo odlučivanja, ono će formirati neki skup pravila koji će se koristiti za predviđanja. S druge strane algoritam slučajnih šuma nasumično odabire opažanja i značajke za izgradnju nekoliko stabala odluka i zatim računa prosjek rezultata.

**Radovi u kojima se koristi ovaj algoritam:**

**Fast Content-Based File Type Identification**

Irfan Ahmed, Kyung-Suk Lhee, Hyun-Jung Shin and Man-Pyo Hong

<https://link.springer.com/content/pdf/10.1007/978-3-642-24212-0_5.pdf>

1. **Implementacija**

U dodatku se nalaze cijele implementacije odabranih algoritama. Ovdje će biti objašnjenja bitnih dijelova koda za rješavanje zadanog problema; klasifikacije.

**5.1.**  **Neuronska mreža**

Sigmoid fukncija koja se poziva kako se bi izračunala greška koju neuronska mreža napravi na izlazu:  
def sigmoid(x):  
 return 1.0 / (1 + np.exp(-x))  
  
  
def sigmoid\_derivative(x):  
 return x \* (1.0 - x)  
  
U ovoj klasi neuronska mreža nasumično postavlja težine između neurona:

class NeuralNetwork:  
 def \_\_init\_\_(self, x, y):  
 self.input = x  
 self.weights1 = np.random.rand(self.input.shape[1], 4)  
 self.weights2 = np.random.rand(4, 1)  
 self.y = y  
 self.output = np.zeros(self.y.shape)

Postavljanje slojeva: def feedforward(self):  
 self.layer1 = sigmoid(np.dot(self.input, self.weights1))  
 self.output = sigmoid(np.dot(self.layer1, self.weights2))

Računanje greške i određivanje novih težina:  
 def backprop(self):  
 # application of the chain rule to find derivative of the loss function with respect to weights2 and weights1  
 d\_weights2 = np.dot(self.layer1.T, (2 \* (self.y - self.output) \* sigmoid\_derivative(self.output)))  
 d\_weights1 = np.dot(self.input.T, (np.dot(2 \* (self.y - self.output) \* sigmoid\_derivative(self.output),  
 self.weights2.T) \* sigmoid\_derivative(self.layer1)))  
  
 # update the weights with the derivative (slope) of the loss function  
 self.weights1 += d\_weights1  
 self.weights2 += d\_weights2

**5.2 . kNN**

Računanje Euklidske udaljenosti između podataka:

def euclideanDistance(instance1, instance2, length):  
 distance = 0  
 for x in range(length - 1):  
 distance += pow((instance1[x] - instance2[x]), 2)  
 return math.sqrt(distance)  
  
Određivanje susjeda:  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance) - 1  
 for x in range(len(trainingSet)):  
 dist = euclideanDistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
  
Brojanje koliko ima predstavnika koje klase, od ukupno k susjeda:  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
  
Računanje točnosti algotitma:  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct / float(len(testSet))) \* 100.0

* 1. **Slučajne šume**
  2. **Konvolucijske mreže**

Gradimo model neuronske mreže. Postavljena je veličina ulaznih podataka (256x1). Koriste se gotove funkcije koje rade konvoluciju i izvlačenje(pooling):

model = Sequential()  
 model.add(Conv1D(64, 3, activation='relu', input\_shape=(256, 1)))  
 model.add(Conv1D(64, 3, activation='relu'))  
 model.add(MaxPooling1D(3))  
 model.add(Conv1D(128, 3, activation='relu'))  
 model.add(Conv1D(128, 3, activation='relu'))  
 model.add(GlobalAveragePooling1D())  
 model.add(Dropout(0.5))   
 model.add(Dense(17, activation='sigmoid'))  
  
 model.compile(loss='binary\_crossentropy', optimizer='rmsprop',  
 metrics=['accuracy']

1. **ZAKLJUČAK**

U ovome radu najprije je na jednostavan način objašnjeno spremanje datoteka u memoriju i uvedeni su osnovni pojmovi koji su bitni za ovu temu. Objašnjeni su osnovni principi rada četiri algoritma strojnog učenja koji su pogodni za klasifikaciju fragmenata datoteka. Istaknute su mane i prednosti svakog od algoritama. Priložene su i implementacije pisane u jeziku Python svih algoritama s objašnjenima najvažnijih dijelova koda. Za isprobavanje i treniranje implementacija koristili smo veliku bazu podatak u kojoj se nalazile histogrami bajtova i točan tip datoteke. Pri izvođenju ovih implementacija najveću brzinu i točnost pokazala je konvolucijska mreža. Iako se ona najčešće koristi za klasifikaciju fotografija ovdje smo pokazali da se može koristiti i za druge svrhe.

1. **LITERATURA**

## Nehal Udyavar, A Beginner's Guide to Neural Networks: Part Two, Bias, activation functions, hidden layers, and building a more advanced feed-forward neural network architecture,27.5.2017.,<https://towardsdatascience.com/a-beginners-guide-to-neural-networks-part-two-bd503514c71a>

1. Nehal Udyavar, A Beginner's Guide to Neural Networks: Part One: The motivation behind neural networks, and the architecture behind the most basic one: the perceptron., 22.5.2017., <https://towardsdatascience.com/a-beginners-guide-to-neural-networks-b6be0d442fa4>
2. Budin L., Golub M., Jakobović D., Jelenković L. Operacijski sustavi, 3. Izdanje, Zagreb, Element d.o.o. 2013
3. Džomba K. Konvolucijske neuronske mreže, Diplomski rad, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, matematički odsjek, 2018
4. Kopljar D. Konvolucijske neuronske mreže, Završni rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva,2016.
5. Irfan Ahmed, Kyung-Suk Lhee, Hyun-Jung Shin, Man-Pyo Hong, Fast Content-Based File Type Identification <https://link.springer.com/chapter/10.1007/978-3-642-24212-0_5>
6. Rajat Harlalka, Choosing the Right Machine Learning Algorithm, 18.5.2018., <https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f>
7. Victor Zhou Random Forsest for Complite Beginners: The definitive guide to Random Forests and Decision Trees, 10.3.2019. <https://victorzhou.com/blog/intro-to-random-forests/>
8. Vassil Roussev, Candice Quates, File fragment encoding classification-An empirical approach, Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA, 2013, <https://doi.org/10.1016/j.diin.2013.06.008>
9. William C Calhoun, Drue Coles, Predicting the types of file fragments, Department of Mathematics, Computer Science and Statistics, Bloomsburg University of Pennsylvania, Bloomsburg, PA 17815, USA, 2008. <https://doi.org/10.1016/j.diin.2008.05.005>
10. Mehdi Chehel Amirani, Moshen Toorani, Feature-based Type Identification of File Fragments, 18.4.2012., [**https://doi.org/10.1002/sec.553**](https://doi.org/10.1002/sec.553)
11. Irfan Ahmed, Kyung-Suk Lhee, Hyun-Jung Shin, Man-Pyo Hong, Fast Content-Based File Type Identification, Part of the [IFIP Advances in Information and Communication Technology](https://link.springer.com/bookseries/6102) book series (IFIPAICT, volume 361), <https://link.springer.com/chapter/10.1007/978-3-642-24212-0_5>
12. Rajat Harlalka, Choosing the Right Machine Learning Algorithm, 2018., <https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f>
13. Victor Zhou, A Simpe Explanation of Gini Impurity: What Gini Impurity is(with Examples) and how it's used to train Decision Trees, 29.4.2019., <https://victorzhou.com/blog/gini-impurity/>
14. Tavish Sriva Stava, Intoduction to k\_nearest Neighbors: Simplified(with implementation in Python), 26.3.2018., <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>
15. James Loy, How to build your own Neural Network from scratch in Python, A beginner's guide to understanding the inner workings of Deep Learning, 14.5.2018., <https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6>
16. Strojno učenje, ferweb, <https://www.fer.unizg.hr/predmet/su?>
17. Jason Brownlee, Tutorial to Implement k.Nearest Neighbors in Python From Scratch, 12.9.2014., <https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/>
18. Jason Brownlee, Hot to Implement Random Forest From Scratch in Python, 14.11.2016., <https://machinelearningmastery.com/implement-random-forest-scratch-python/>
19. Getting started with the Keras Sequential model, Keras Documentation, <https://keras.io/getting-started/sequential-model-guide/>
20. **DODATAK**
21. **NEURONSKA MREŽA**

import numpy as np  
import csv  
import random  
import math  
import operator  
  
  
def sigmoid(x):  
 return 1.0 / (1 + np.exp(-x))  
  
  
def sigmoid\_derivative(x):  
 return x \* (1.0 - x)  
  
  
class NeuralNetwork:  
 def \_\_init\_\_(self, x, y):  
 self.input = x  
 self.weights1 = np.random.rand(self.input.shape[1], 4)  
 self.weights2 = np.random.rand(4, 1)  
 self.y = y  
 self.output = np.zeros(self.y.shape)  
  
 def feedforward(self):  
 self.layer1 = sigmoid(np.dot(self.input, self.weights1))  
 self.output = sigmoid(np.dot(self.layer1, self.weights2))  
  
 def backprop(self):  
 # application of the chain rule to find derivative of the loss function with respect to weights2 and weights1  
 d\_weights2 = np.dot(self.layer1.T, (2 \* (self.y - self.output) \* sigmoid\_derivative(self.output)))  
 d\_weights1 = np.dot(self.input.T, (np.dot(2 \* (self.y - self.output) \* sigmoid\_derivative(self.output),  
 self.weights2.T) \* sigmoid\_derivative(self.layer1)))  
  
 # update the weights with the derivative (slope) of the loss function  
 self.weights1 += d\_weights1  
 self.weights2 += d\_weights2  
  
  
def loadDataset(filename, X=[], Y=[]):  
 with open(filename, 'r') as csvfile:  
 lines = csv.reader(csvfile)  
 dataset = list(lines)  
 dataset.pop(0)  
 for x in range(len(dataset)):  
 for y in range(261):  
 dataset[x][y] = float(dataset[x][y])  
  
 np.append(X, dataset[x])  
 np.append(Y, dataset[x][261])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 X = []  
 Y = []  
 loadDataset('file\_type\_forenzika\_demo (3).csv', X, Y)  
  
 nn = NeuralNetwork(X, Y)  
  
 for i in range(1500):  
 nn.feedforward()  
 nn.backprop()  
  
 print(nn.output)

1. **KONVOLUCIJSKA MREŽA**

from keras.models import Sequential  
from keras.layers import Dense, Dropout  
from keras.optimizers import SGD  
from keras.callbacks import TensorBoard  
from keras.models import load\_model  
import csv  
import numpy as np  
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D  
  
def loadDataset(filename, split):  
 x\_train, x\_test, y\_train, y\_test = [], [], [], []  
  
 with open(filename, 'r') as csvfile:  
 lines = csv.reader(csvfile)  
 dataset = list(lines)  
 dataset.pop(0)  
  
 full\_size = len(dataset)  
  
 train\_size = int(full\_size\*split)  
  
 i = 0  
 for line in dataset:  
 sample = [float(x) for x in line[:-6]]  
 result = line[-1]  
  
 i += 1  
 if i < train\_size:  
 x\_train.append(sample)  
 y\_train.append(result)  
 else:  
 x\_test.append(sample)  
 y\_test.append(result)  
  
 y\_train = label\_processing(y\_train)  
 y\_test = label\_processing(y\_test)  
  
 x\_train = np.expand\_dims(np.asarray(x\_train), axis=2)  
 x\_test = np.expand\_dims(np.asarray(x\_test), axis=2)  
  
 return x\_train , x\_test, y\_train, y\_test  
  
def label\_processing(labels):  
 new\_labels = []  
 labels\_dict = get\_labels\_dict()  
 dict\_len = len(labels\_dict)  
  
 for label in labels:  
 vector = np.zeros(dict\_len)  
 vector[labels\_dict.get(label)] = 1  
 new\_labels.append(vector)  
  
 return np.asarray(new\_labels)  
  
def get\_labels\_dict():  
  
 switcher = {  
 ".swf": 0,  
 ".doc": 1,  
 ".ppt": 2,  
 ".csv": 3,  
 ".pdf": 4,  
 ".xls": 5,  
 ".html": 6,  
 ".txt": 7,  
 ".ps": 8,  
 ".jpg": 9,  
 ".rtf": 10,  
 ".gif": 11,  
 ".png": 12,  
 ".gz": 13,  
 ".xml": 14,  
 ".zip": 15,  
 ".enc": 16  
  
 }  
 return switcher  
  
def main(train=True):  
 if train:  
 model = Sequential()  
 model.add(Conv1D(64, 3, activation='relu', input\_shape=(256, 1)))  
 model.add(Conv1D(64, 3, activation='relu'))  
 model.add(MaxPooling1D(3))  
 model.add(Conv1D(128, 3, activation='relu'))  
 model.add(Conv1D(128, 3, activation='relu'))  
 model.add(GlobalAveragePooling1D())  
 model.add(Dropout(0.5))  
 model.add(Dense(17, activation='sigmoid'))  
  
 model.compile(loss='binary\_crossentropy',  
 optimizer='rmsprop',  
 metrics=['accuracy'])  
  
 print('Loading dataset...')  
 x\_train, x\_test, y\_train, y\_test = loadDataset('file\_type\_forenzika\_demo (3).csv', 0.8)  
  
 tensorboard = TensorBoard(log\_dir="logs/model1")  
  
 model.fit(x\_train, y\_train,  
 epochs=1,  
 batch\_size=128,  
 verbose=1,  
 callbacks=[tensorboard])  
  
 # save model  
 model.save('my\_model.h5')  
  
 else:  
 model = load\_model('my\_model.h5')  
  
 # za sve iz test  
 score, acc = model.evaluate(x\_test, y\_test, batch\_size=128)  
 print('Acc:', acc)  
  
 # za jedan iz test  
 model.predict(x\_test[0])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main(train=True)

1. **KNN**

import csv  
import random  
import math  
import operator  
  
  
def loadDataset(filename, split, trainingSet=[], testSet=[]):  
 with open(filename, 'r') as csvfile:  
 lines = csv.reader(csvfile)  
 dataset = list(lines)  
 dataset.pop(0)  
 for x in range(len(dataset) - 1):  
 for y in range(261):  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
  
  
def euclideanDistance(instance1, instance2, length):  
 distance = 0  
 for x in range(length - 1):  
 distance += pow((instance1[x] - instance2[x]), 2)  
 return math.sqrt(distance)  
  
  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance) - 1  
 for x in range(len(trainingSet)):  
 dist = euclideanDistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
  
  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
  
  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct / float(len(testSet))) \* 100.0  
  
  
def main():  
 # prepare data  
 trainingSet = []  
 testSet = []  
 split = 0.67  
 loadDataset('file\_type\_forenzika\_demo (3).csv', split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 # generate predictions  
 predictions = []  
 k = 3  
 for x in range(len(testSet)):  
 neighbors = getNeighbors(trainingSet, testSet[x], k)  
 result = getResponse(neighbors)  
 predictions.append(result)  
 print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))  
 accuracy = getAccuracy(testSet, predictions)  
 print('Accuracy: ' + repr(accuracy) + '%')  
  
  
main()

1. **ALGORITAM SLUČAJNIH ŠUMA**

from random import seed  
from random import randrange  
from csv import reader  
from math import sqrt  
  
  
# Load a CSV file  
def load\_csv(filename):  
 dataset = list()  
 with open(filename, 'r') as file:  
 csv\_reader = reader(file)  
 for row in csv\_reader:  
 if not row:  
 continue  
 dataset.append(row)  
 dataset.pop(0)  
 return dataset  
  
  
# Convert string column to float  
def str\_column\_to\_float(dataset, column):  
 for row in dataset:  
 row[column] = float(row[column].strip())  
  
  
# Convert string column to integer  
def str\_column\_to\_int(dataset, column):  
 class\_values = [row[column] for row in dataset]  
 unique = set(class\_values)  
 lookup = dict()  
 for i, value in enumerate(unique):  
 lookup[value] = i  
 for row in dataset:  
 row[column] = lookup[row[column]]  
 return lookup  
  
  
# Split a dataset into k folds  
def cross\_validation\_split(dataset, n\_folds):  
 dataset\_split = list()  
 dataset\_copy = list(dataset)  
 fold\_size = int(len(dataset) / n\_folds)  
 for i in range(n\_folds):  
 fold = list()  
 while len(fold) < fold\_size:  
 index = randrange(len(dataset\_copy))  
 fold.append(dataset\_copy.pop(index))  
 dataset\_split.append(fold)  
 return dataset\_split  
  
  
# Calculate accuracy percentage  
def accuracy\_metric(actual, predicted):  
 correct = 0  
 for i in range(len(actual)):  
 if actual[i] == predicted[i]:  
 correct += 1  
 return correct / float(len(actual)) \* 100.0  
  
  
# Evaluate an algorithm using a cross validation split  
def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):  
 folds = cross\_validation\_split(dataset, n\_folds)  
 scores = list()  
 for fold in folds:  
 train\_set = list(folds)  
 train\_set.remove(fold)  
 train\_set = sum(train\_set, [])  
 test\_set = list()  
 for row in fold:  
 row\_copy = list(row)  
 test\_set.append(row\_copy)  
 row\_copy[-1] = None  
 predicted = algorithm(train\_set, test\_set, \*args)  
 actual = [row[-1] for row in fold]  
 accuracy = accuracy\_metric(actual, predicted)  
 scores.append(accuracy)  
 return scores  
  
  
# Split a dataset based on an attribute and an attribute value  
def test\_split(index, value, dataset):  
 left, right = list(), list()  
 for row in dataset:  
 if row[index] < value:  
 left.append(row)  
 else:  
 right.append(row)  
 return left, right  
  
  
# Calculate the Gini index for a split dataset  
def gini\_index(groups, classes):  
 # count all samples at split point  
 n\_instances = float(sum([len(group) for group in groups]))  
 # sum weighted Gini index for each group  
 gini = 0.0  
 for group in groups:  
 size = float(len(group))  
 # avoid divide by zero  
 if size == 0:  
 continue  
 score = 0.0  
 # score the group based on the score for each class  
 for class\_val in classes:  
 p = [row[-1] for row in group].count(class\_val) / size  
 score += p \* p  
 # weight the group score by its relative size  
 gini += (1.0 - score) \* (size / n\_instances)  
 return gini  
  
  
# Select the best split point for a dataset  
def get\_split(dataset, n\_features):  
 class\_values = list(set(row[-1] for row in dataset))  
 b\_index, b\_value, b\_score, b\_groups = 999, 999, 999, None  
 features = list()  
 while len(features) < n\_features:  
 index = randrange(len(dataset[0]) - 1)  
 if index not in features:  
 features.append(index)  
 for index in features:  
 for row in dataset:  
 groups = test\_split(index, row[index], dataset)  
 gini = gini\_index(groups, class\_values)  
 if gini < b\_score:  
 b\_index, b\_value, b\_score, b\_groups = index, row[index], gini, groups  
 return {'index': b\_index, 'value': b\_value, 'groups': b\_groups}  
  
  
# Create a terminal node value  
def to\_terminal(group):  
 outcomes = [row[-1] for row in group]  
 return max(set(outcomes), key=outcomes.count)  
  
  
# Create child splits for a node or make terminal  
def split(node, max\_depth, min\_size, n\_features, depth):  
 left, right = node['groups']  
 del (node['groups'])  
 # check for a no split  
 if not left or not right:  
 node['left'] = node['right'] = to\_terminal(left + right)  
 return  
 # check for max depth  
 if depth >= max\_depth:  
 node['left'], node['right'] = to\_terminal(left), to\_terminal(right)  
 return  
 # process left child  
 if len(left) <= min\_size:  
 node['left'] = to\_terminal(left)  
 else:  
 node['left'] = get\_split(left, n\_features)  
 split(node['left'], max\_depth, min\_size, n\_features, depth + 1)  
 # process right child  
 if len(right) <= min\_size:  
 node['right'] = to\_terminal(right)  
 else:  
 node['right'] = get\_split(right, n\_features)  
 split(node['right'], max\_depth, min\_size, n\_features, depth + 1)  
  
  
# Build a decision tree  
def build\_tree(train, max\_depth, min\_size, n\_features):  
 root = get\_split(train, n\_features)  
 split(root, max\_depth, min\_size, n\_features, 1)  
 return root  
  
  
# Make a prediction with a decision tree  
def predict(node, row):  
 if row[node['index']] < node['value']:  
 if isinstance(node['left'], dict):  
 return predict(node['left'], row)  
 else:  
 return node['left']  
 else:  
 if isinstance(node['right'], dict):  
 return predict(node['right'], row)  
 else:  
 return node['right']  
  
  
# Create a random subsample from the dataset with replacement  
def subsample(dataset, ratio):  
 sample = list()  
 n\_sample = round(len(dataset) \* ratio)  
 while len(sample) < n\_sample:  
 index = randrange(len(dataset))  
 sample.append(dataset[index])  
 return sample  
  
  
# Make a prediction with a list of bagged trees  
def bagging\_predict(trees, row):  
 predictions = [predict(tree, row) for tree in trees]  
 return max(set(predictions), key=predictions.count)  
  
  
# Random Forest Algorithm  
def random\_forest(train, test, max\_depth, min\_size, sample\_size, n\_trees, n\_features):  
 trees = list()  
 for i in range(n\_trees):  
 sample = subsample(train, sample\_size)  
 tree = build\_tree(sample, max\_depth, min\_size, n\_features)  
 trees.append(tree)  
 predictions = [bagging\_predict(trees, row) for row in test]  
 return (predictions)  
  
  
# Test the random forest algorithm  
seed(2)  
# load and prepare data  
filename = 'file\_type\_forenzika\_demo (3).csv'  
dataset = load\_csv(filename)  
# convert string attributes to integers  
for i in range(0, len(dataset[0]) - 1):  
 str\_column\_to\_float(dataset, i)  
# convert class column to integers  
str\_column\_to\_int(dataset, len(dataset[0]) - 1)  
# evaluate algorithm  
n\_folds = 5  
max\_depth = 10  
min\_size = 1  
sample\_size = 1.0  
n\_features = int(sqrt(len(dataset[0]) - 1))  
for n\_trees in [1, 5, 10]:  
 scores = evaluate\_algorithm(dataset, random\_forest, n\_folds, max\_depth, min\_size, sample\_size, n\_trees, n\_features)  
 print('Trees: %d' % n\_trees)  
 print('Scores: %s' % scores)  
 print('Mean Accuracy: %.3f%%' % (sum(scores) / float(len(scores))))