SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5421

Dubinska analiza podataka u radnom okviru Apache Spark pomoću knjižnice MLlib

Lucia Penić

Zagreb, lipanj 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Veliku zahvalnost izražavam svojoj obitelji na svoj brizi, pažnji, ljubavi, razumijevanju i podršci, a posebno mojim roditeljima koji su uvijek tu za mene bez ikakve zadrške. Bez vas ovo ne bi bilo moguće. HVALA!

SADRŽAJ

1. Uvod

U posljednjih nekoliko godina količina informacija kojima smo okruženi eksponencijalno raste. Istraživanja pokazuju da je 90% digitalnog sadržaja koji trenutno postoji nastalo u protekle dvije godine. S povećanjem količine podataka, pojavljuje se problem snalaženja među njima i izvlačenja korisnih informacija.

Iako su mnoge tehnike obrade i analize podataka poznate već desetljećima, tradicionalne metode i alati više nisu dostatni za upravljanje tako velikom količinom podataka (engl. *big data*). Glavni problemi s kojima se danas suočavamo odnose se na takozvana '3V' velikih podataka. Pojam '3V' obuhvaća tri osnovne karakteristike velikih podatka. Brzina (engl. *velocity*) pristizanja podataka zahtjeva obradu u stvarnom vremenu, volumen (engl. *volume*) odnosno količina podataka koja prelazi dosadašnje spremišne kapacitete i naposljetku varijabilnost (engl. *variety*) podataka odnosno raznovrsnost strukturiranih i nestrukturiranih formata u kojima dolaze.

Iz navedenih razloga analiza, prikupljanje i obrada podataka postaje veoma složena i zamorna za ručnu obradu pa se javlja potreba za novim rješenjima. Dolazi do naglog razvoja znanosti o podacima (engl. *data science*) s ciljem izvlačenja znanja iz velikih podataka. Pritom se služi teorijama i tehnikama brojnih područja kao što su matematika, statistika, teorija informacije, strojno učenje, dubinsko pretraživanje podataka, baze podataka, vizualizacija i drugi.

U sljedećim poglavljima bit će opisan način rada radnog okvira Apache Spark, jednog od rješenja za problem obrade velikih podatak s detaljnim osvrtom na knjižicu MLlib. Konačno, dan je i opis praktičnog dijela kojim su obuhvaćeni osnovni koncepti rada s MLlibom.

2. Uvod u strojno učenje

Strojno učenje grana je umjetne inteligencije čijom pojavom programiranje doživljava svoj procvat. Kod tradicionalnog načina programiranja računalo radi isključivo ono što mu nalažu linije koda, a svako ažuriranje zahtijeva rad inženjera odnosno programera koji bi mu promijenio izvorni kod. S druge strane, područje strojnog učenja pronalazi metode programiranja koje bi računalu omogućile samostalno optimiranje svog rada na temelju prijašnjih iskustava ili podataka koji mu pristižu [3].

Učenje je najlakše objasniti na jednostavnom klasifikacijskom problemu. Primjerice, da računalu zadamo da raspoznaje nalazi li se na pojedinoj slici automobil ili bicikl, njemu to predstavlja izuzetno težak zadatak. Iako je to ljudima intuitivno jasno na prvi pogled, računalo ne razmišlja kao mi, već mu je potrebno nizom matematičkih i logičkih operacija simulirati "razmišljanje".

Prvi korak u procesu izgradnje modela bio bi prikupiti podatke, odnosno u konkretnom primjeru, slike automobila i bicikala. Prikupljeni se podaci zatim razdvajaju u dva dijela - podatke za učenje i podatke za testiranje, najčešće u omjeru 70:30 ili 80:20. Dio podataka za učenje koristimo kako bi izgradili model, dok dio za testiranje služi za vrednovanje performansi našeg modela. Bitno je da su podaci kojima učimo model različiti od testnih kako bi vrednovanje bilo nepristrano.

Postoji veliki broj algoritama za učenje, od kojih su neki bolje prilagođeni za pojedine zadatke od drugih. Ovisno o odabranom algoritmu učenja, model traži uzorke u prikupljenim podacima i njihovim karakteristikama. Upravo na temelju uočenih uzoraka model uči i ispravlja svoje pogreške te inkrementalno poboljšava svoju sposobnost predviđanja. Unatoč tome što postoji puno karakteristika temeljem kojih možemo razlikovati automobile od bicikala, najjednostavniji je način koristiti broj kotača. Međutim, što je s vozilima koja imaju tri kotača (slika 2.1)? Potrebno je izabrati model koji uspijeva pogoditi ispravnu kategoriju većinu puta.

Slika 2.1: Primjer automobila koji se neće ispravno klasificirati

Zato je vrlo bitan korak vrednovanje, dio gdje pustimo naš model da napravi svoja predviđanja nad testnim skupom podataka. U ovom trenutku, model više ne uči, ne poboljšava se, već koristi stečeno znanje da izgradi vlastite pretpostavke o kategoriji kojoj podatak pripada. Usporedbom predviđanja i stvarnih kategorija kojima pojedini podatak pripada, možemo izračunati metriku pouzdanosti našeg modela. Istražujući dobivenu metriku, možemo naći načine za poboljšanje modela kalibriranjem parametara, mijenjanjem skupa podataka dodavanjem ili uklanjanjem nekih opisnih značajki ili čak mijenjanjem izabrane metode [2].

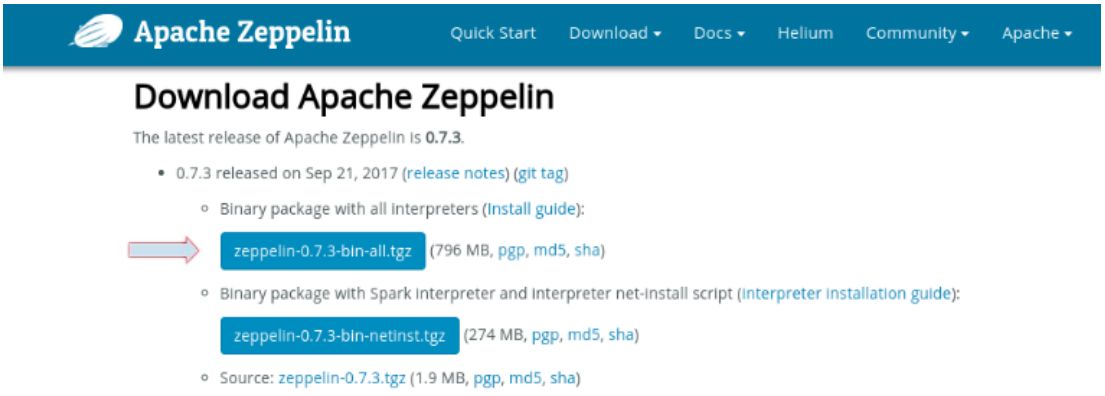
3. Apache Spark

Jedno od rješenja za problem obrade velikih podataka nudi alat Apache Spark. Apache Spark je računalna platforma otvorenog koda koja omogućuje brzu i skalabilnu obradu velikih skupova podataka te izvođenje programskog koda na grozdu računala uz minimalne izmjene [15]. Svojim korisnicima pruža mogućnost programiranja u Scali, Javi, Pythonu ili R-u putem naprednih aplikacijskih programskih sučelja (API-ja).

3.1. Instalacija Apache Sparka uz Apache Zeppelin

Budući da Apache Spark nema ugrađen alat za vizualizaciju, za te potrebe moguće je koristiti Apache Zeppelin. Apache Zeppelin je višenamjenska virtualna bilježnica koja podržava interpretere za brojne programske jezike i programe za obradu podataka kao što su Java, R, Scala, Python, PostgreSQL, Apache Spark i mnogi drugi.

Integracija Sparka i Zeppelina uvelike olakšava postupak instalacije potrebne programske podrške. Apache Zeppelin već ima podržan Spark interpreter te, osim instalacije Zeppelina, nikakve dodatne konfiguracije nisu potrebne.



Slika 3.1: Preuzimanje programskog paketa Apache Zeppelin

Instalacija Apache Zeppelina na operacijskom sustavu Fedora 25 provodi se sljedećim koracima:

1. S poveznice https://zeppelin.apache.org/download.html dohvatite tar.gz paket s podržanim Spark interpreterom (slika 3.1.)

2. Preuzetu arhivu raspakirajte te premjestite u direktorij /opt.

```
$ tar -zxvf zeppelin-0.7.3-bin-all.tgz
$ sudo mv zeppelin-0.7.3-bin-all /opt/
```

Apache Zeppelin spreman je za korištenje. Za njegovo pokretanje potrebno je pokrenuti skriptu zeppelin-daemon.sh koja se nalazi unutar bin direktorija.

```
$ ./opt/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh start
```

Ovom naredbom Apache Zeppelin postaje dostupan na portu 8080, te mu se može pristupiti putem web preglednika na lokaciji http://localhost:8080/.

Za zaustavljanje koristi se ista skripta s parametrom stop.

```
$ ./opt/zeppelin-0.7.3-bin-all/bin/zeppelin-daemon.sh stop
```

3.2. Osnovne karakteristike

U izvornoj dokumentaciji Apache Spark opisuje se riječima svjetlosno brz te alat opće namjene. Ovim riječima istaknute su glavne prednosti koje Spark izdvajaju pored ostalih radnih okvira namjenjenih za obradu podataka [15].

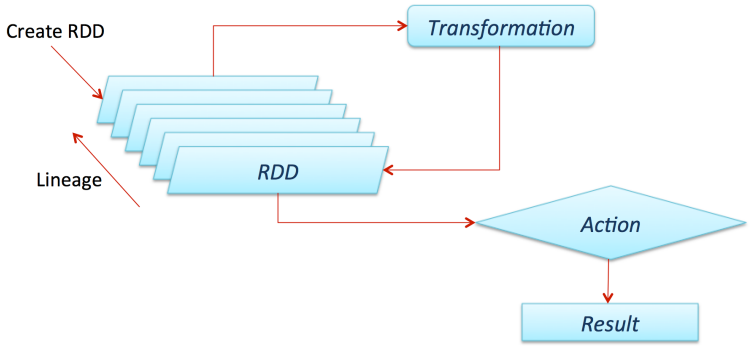
Apache Spark uspješno rješava neke probleme uskog grla prilikom zapisivanja i čitanja sa diska. Osnovna karakteristika koja mu to osigurava je sposobnost obrade unutar radne memorije (RAM). Izvođenje u radnoj memoriji osigurava izuzetnu brzinu i poboljšanje performansi. Umjesto da se podaci drže na nekom sporom disku, oni se drže u radnoj memoriji i obrađuju se paralelno.

Opća namjena podrazumijeva da se unutar jedinstvenog radnog okvira može raditi veliki broj različitih obrada i analize podataka. U prošlosti, bilo je potrebno ukomponirati razne alate kako bi se omogućila iterativnost, raspodijeljena obrada podataka, obrada tokova podataka i slično. Međutim, Apache Spark ujedinjuje nekoliko modula

koje je moguće kombinirati te izbjeći snalaženje i komponiranje nepovezanih alata. Apache Spark moduli omogućavaju provođenje SQL upita, izgradnja cjevovoda podataka, pružaju razne statističke metode, ali i metode za duboku analizu podataka. Nadalje, moguće je raditi s grafovima, tokovima podataka i mnogim drugim.

Primjer iz knjige Learning Spark [19] odlično ilustrira međusobnu povezanost modula. "Na primjer, u Sparku možete napisati jednu aplikaciju koja koristi strojno učenje za klasifikaciju podataka u stvarnom vremenu, dok podaci pristižu kao tok podataka. Istovremeno analitičari mogu raditi SQL upite nad rezultatima, također u stvarnom vremenu. Nadalje, inženjeri i znanstvenici istovremeno tim rezultatima mogu pristupiti preko Python ljuske za *ad hoc* analizu. Za sve to vrijeme, potporni tim mora održavati samo jedan jedini sustav."

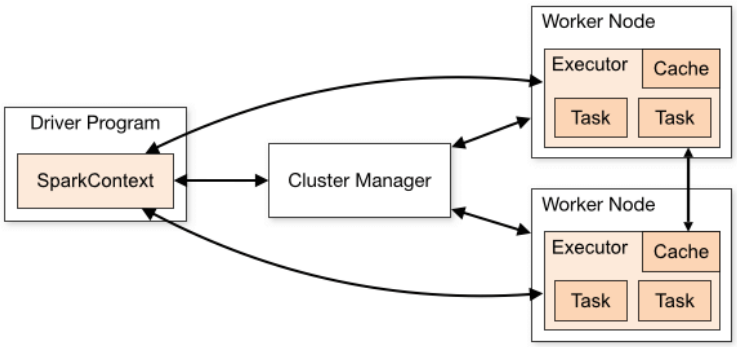
Još jedna od optimizacija je lijeno izvođenje (engl. *lazy evaluation*), sposobnost da se podaci ne obrađuju nakon svake operacije. Razlikuju se operacije transformacija od operacija akcije. Transformacije zapravo ne rade ništa već ih Spark sprema u svoje lokalno spremište te iz njih gradi direktni aciklični graf (DAG). Tek akcije potiču izvođenje pojedinih transformacija. Ovo omogućava optimiranje niza operacija, pa čak i izbacivanje nekih te učinkovitije korištenje resursa.



Slika 3.2: Operacije nad RDD tipom podataka - lijeno izvođenje

Ispod haube, rad se temelji na gospodar - rob arhitekturi (engl. *master-slave architecture*). Odnosno, postoji jedan centralni Java proces, nazvan pogonski program (engl. *driver program*), koji koordinira velik broj radnih čvorova (engl. *worker node*). Svaki je od radnih čvorova zaseban Java proces. Jedna Spark aplikacija je kombinacija mnoštva pogonskih programa i njihovih radnih čvorova raspoređenih na grozdu računala (slika 3.2). Svim procesima upravlja glavni upravitelj grozda (engl. *cluster manager*) zadužen za alociranje resursa. Iako podržava vanjske upravitelje grozda kao

što su Hadoop Yarn, Apache Mesos i drugi, Spark dolazi i sa samostalnim upraviteljem Spark Standalone.



Slika 3.3: Prikaz Sparkove master - slave arhitekture

3.3. Komponente Apache Spark arhitekture

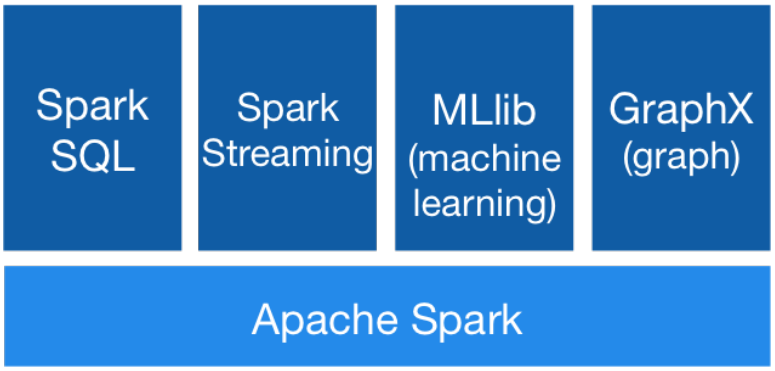
Pod imenom Apache Spark podrazumijevamo više od jednog modula od kojih je svaki specijaliziran za zaseban aspekt znanosti o podacima. Kao što je već spomenuto, ti moduli mogu se međusobno kombinirati te ostvariti složene sustave. Budući da su moduli usko povezani, kombiniranje je prilično jednostavno, a dodatna je prednost ta da, kada se u arhitekturu uvede neki novi modul, on se u postojeći projekt može uključiti poput ugrađenih biblioteka (engl. *built-in libraries*) [19].

Spark Core temelj je Apache Spark stoga, a nad njim nalaze se još četiri ugrađena modula koje zajedno čine kompletni ekosustav. To su: Spark SQL, Spark Streaming, MLlib, GraphX. Ovakva slojevita arhitektura osigurava da se svako ažuriranje provedeno na Spark Coreu, bez odgode vidi i u ostalim slojevima.

Sljedeća potpoglavlja daju detaljniji opis pojedinih modula na temelju službene dokumentacije svakog od njih.

3.3.1. Spark Core

Apache Spark Core jezgra je cjelokupne arhitekture. On podržava osnovne operacije za raspoređivanje zadataka, upravljanje memorijom, toleranciju na pogreške, osiguravanje I/O operacija i slično, a bez kojih bi ostali moduli izgrađeni nad Spark Coreom bili beskorisni. Upravo Spark Core je taj koji ima sposobnost unutar memorijske



Slika 3.4: Apache Spark ekosustav

obrade čime osigurava brzinu sustava. Korištenje gore navedenih operacija omogućeno je korisnicima, odnosno programerima, kroz aplikacijsko programsko sučelje (API) s podrškom za: Javu, Python, Scalu i R. Navedeni API-ji od korisnika skrivaju implementacijske detalje i pružaju mogućnost programiranja na visokoj razini apstrakcije čineći uporabu Sparka izuzetno jednostavnom za korištenje u praksi.
Osnova rada temelji se na tipu podataka koji se naziva *Resilient Distributed Dataset*, kraće RDD. RDD je apstrakcija podataka specifična za Spark, a predstavlja kolekciju objekata nad kojom se operacije mogu izvršavati paralelno. Budući da su oni nepromjenjivi, svaka operacija rezultira stvaranjem novog RDD-a.

3.3.2. Spark SQL

Spark SQL je knjižica za rad sa strukturiranim podacima. S njenom pojavom uveden je i drugi tip podataka pod nazivom DataFrame koji je produžetak nad RDD API-jem, a nad kojim se mogu izvoditi SQL upiti. Prednosti ovog modula su deklarativni upiti i optimirana pohrana. Podaci se mogu crpiti iz raznih izvora kao što su na primjer JSON, Parquet, Hive tablice. Spark SQL vrlo je lako integrirati i koristiti s ostalim dijelovima ekosustava Spark. Na taj način dozvoljeno je miješanje SQL upita nad DataFrameom s operacijama transformacija i akcija nad RDD-jem. Primjer gdje se to pokazalo izuzetno korisno je integracija s MLlibom, knjižicom za strojno učenje.

3.3.3. Spark Streaming

Spark Streaming knjižica je pomoću koje Spark osigurava kompletno novu dimenziju obrade toka podataka u stvarnom vremenu. Kontinuirani tok podataka stvara se takozvanim mikro gomilama/serijama (engl. *batch*). Naime, podaci koji dolaze iz nekog izvora razdvajaju se u batcheve koji se tada obrađuju u Spark Coreu. Za ovaj tip

obrade, Spark pruža apstrakciju podataka nazvanu diskretizirani tok (engl. *discretized stream*) ili DStream, koja je realizirana kao niz RDD-jeva. Nad DStreamom moguće je provoditi operacije kao i nad samim RDD-jevima i generirati rezultate u novom toku podataka. Ugrađena je podrška za HDFS, Kafka, Twitter i druge izvore podataka.

3.3.4. MLlib

MLlib knjižica je koja predstavlja skup naprednih algoritama namijenjenih strojnom učenju. Kategorije, od kojih neke sadrže desetke različitih algoritama, su klasifikacija, regresija, grupiranje podataka te kolaborativno filtriranje. Sparkova sposobnost da čuva podatke u radnoj memoriji čini ga pogodnim za izvršavanje takvih algoritama koji uzastopno izvode istu funkciju nad istim skupom podataka. Osim algoritama za strojno učenje knjižica ima i bogat skup funkcija za ekstrakciju svojstava (engl. *featurization*), pretvorbu i slično. Također je moguće graditi cjevovode, te provoditi optimizacije nad pojedinim dijelovima ili čak na cijelim cjevovodima. Pruža i alate za vrednovanje te hiperparametrizacijsko ugađanje.

3.3.5. GraphX

Modul GraphX namijenjen je radu s grafovima. Biblioteka sadrži operatore za manipulaciju grafovima te algoritme i funkcije za izgradnju grafova. Graf u GraphXu apstrahiran je tipom podataka Graph, koji je direktni usmjereni multigraf s proizvoljnim svojstvima na svakom bridu i čvoru.

3.4. Tipovi podataka RDD i DataFrame

3.4.1. RDD

Kao što je već spomenuto u poglavlju 3.3.1. Apache Spark uveo je novi tip podatka nazvan *Resilient Distributed Dataset*. Doslovni prijevod naziva RDD bio bi otporni raspodijeljeni skup podataka. Ova memorijska apstrakcija podataka osnovna je podatkovna struktura u Sparku, otporna na pogreške i može se pohraniti i manipulirati s njom na grozdu računala.

Otpornost na pogreške omogućena je već spomenutim DAG-om koji prati povijest svih transformacija pa se, u slučaju pogreške, može rekonstruirati traženi RDD. Povijest se

čuva sve dok se RDD ne prestane koristiti, tek onda skupljač smeća (engl. *garbage collector*) čisti privremeno spremište podataka. Za aplikacije koje se dugo izvode moguće je prijevremeno brisanje povijesti zato što bi u protivnom koristile jako puno diskovnog prostora.

Stvaranje RDD-a moguće je na dva načina: paraleliziranjem postojeće kolekcije ili referenciranjem na podatke u eksternoj memoriji kao što je HDFS, HBase i ostali [8].

3.4.2. DataFrame

U želji da se dobre strane RDD-a povežu s dobrim stranama relacijskog modeliranja podataka, nastao je tip podataka nazvan DataFrame. On predstavlja raspodijeljenu kolekciju podataka organiziranih u imenovane stupce. Strukturom je istovjetan tablici u relacijskoj bazi, no u usporedbi s njom pokazuje bolje performanse [14].

Moguć je prelazak iz DataFramea u RDD i obrnuto. Ovo je ponekad izuzetno korisno jer su poneke pretvorbe moguće samo nad RDD tipom podataka, a s druge strane DataFrame je intuitivniji za korištenje.
Postoje dva načina pretvorbe:

1. Prvi način koristi automatsko prepoznavanje sheme iz strukture podataka. Prednost ovog načina je sažet kod, ali prikladan je samo onda kada unaprijed znamo kako shema treba izgledati.

2. Drugi način je ručno definiranje sheme. Ovaj način detaljnije je prikazan u poglavlju 6.2.

4. Osnovni koncepti modula za strojno učenje - MLlib

U poglavlju 3.3.4. dan je kratki opis modula MLlib, no bitno je naglasiti da MLlib u sebi sadrži dva različita API-ja zbog čega često dolazi do zabune. Originalni API podržava RDD tip podataka, a drugi je dodan naknadno i služi za podršku DataFrame tipa podataka. Iako se API za DataFrame u neslužbenim izvorima naziva Spark ML, to nije službeni naziv te je on dio knjižice MLlib, a ne zaseban modul.

S druge strane, API za RDD od verzije Sparka 2.0 prešao je u održavanje, što znači da je još uvijek moguće korištenje te će se i dalje vršiti otklanjanje pogreški (engl. *bug fixes*), ali se API-ju neće dodavati nove funkcionalnosti. Od verzije 3.0 u planu je kompletno uklanjanje ovog API-ja iz MLlib modula [11].

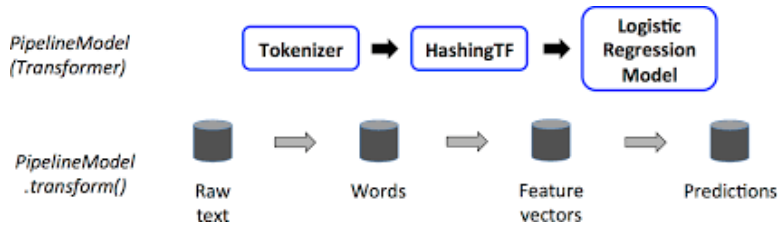
Razloga za gore navedeno je nekoliko, ali najvažniji su:

- DataFrame kao tip podataka programerima je mnogo ugodniji i intuitivniji za rad. Također, ima puno prednosti kao što su već spomenuti relacijski upiti i optimizacije.
- API za DataFrameove omogućava uniformno pisanje koda neovisno o programskom jeziku.
- DataFrameovi su prikladniji za korištenje cjevovoda (engl. *pipeline*).

4.1. Cjevovodi u MLlibu

Ono što MLlib čini posebno zanimljivim i drugačijim je to što nam pruža alate za izgradnju cjevovoda tako olakšavajući kombinaciju više algoritama u jednoj aplikaciji, ali i pojednostavljenje složenih algoritama razdvajajući ih na manje cjeline.
Naime, cjevovod se sastoji od dvije ili više međusobno odvojenih logičkih cjelina kroz

koje podaci prolaze točno određenim redoslijedom. U MLlibu jedan cjevovod sastoji se od nekoliko transformatora i estimatora koji zajedno specificiraju tijek rada (engl. *workflow*). Primjer iz službene dokumentacije dan je na slici 4.1.



Slika 4.1: Primjer izgradnje cjevovoda za jednostavnu obradu tekstualnog dokumenta

Transformator je algoritam kojim je moguće preoblikovati jedan DataFrame u drugi koristeći pretvorbe kojima se preslikavaju stupci čime se dobiju vektori značajki (engl. *feature vectors*), dok je estimator algoritam koji, upotrijebljen nad DataFrameom, rezultira novim transformatorom.

Pojašnjenje: podaci kroz cjevovod prolaze striktno definiranim redoslijedom, a na tom putu "doživljavaju" razne pretvorbe dok prolaze kroz svaku od faza (kroz svaki transformator odnosno estimator). Kroz transformatorske cjeline najčešće se odvija priprema podataka za provođenje estimatora iz kojeg se potom gradi model. Bitno je da se nakon svakog estimatora (ako ih ima više od jednog u cjevovodu) nad modelom pozove funkcija *transform()* kako bi se dobio DataFrame koji se može dalje obrađivati sljedećim transformatorom odnosno estimatorom.

Gornji opis odgovara linearnim cjevovodima, gdje se rezultat jednog koraka koristi u sljedećem koraku. Međutim, postoje i nelinearni cjevovodi, ali tijekom njihove izgradnje potrebno je biti jako oprezan i složiti ga tako da bude isključivo direktni aciklički graf (DAG).

4.2. Implementirane fukcije

Budući da Spark ima bogat skup funkcija namijenjenih za strojno učenje i dubinsku analizu podataka, u ovom poglavlju dan je pregled samo onih koji se odnose na probleme klasifikacije i regresije.

Oba tipa učenja pripadaju nadziranom učenju, ali razlika je u obliku ciljne varijable. Klasifikacija je proces svrstavanja podataka u predefinirane kategorije, dok s druge

strane, regresija nastoji predvidjeti numeričku vrijednost kontinuirane izlazne varijable [7].

Većina popularno korištenih metoda za klasifikacijske i regresijske probleme dostupne su u MLlibu. Samo neke od njih su sljedeće:

- **Logistička regresija** (engl. *logistic regression*) - Statistička metoda koja svoju odluku temelji na linearnoj kombinaciji vjerojatnosti događaja. U MLlibu dostupne su metode za binarnu i multinomnu regresiju.
- **Stablo odluke** (engl. *decision tree classifier*) - Odluku donosi na principu raspoznavanja uzoraka. Iz više ulaznih značajki gradi se struktura stabla koje predstavlja niz koraka odnosno odluka koje u konačnici rezultiraju predikcijom.
- **Slučajna šuma** (engl. *random forest classifier*) - Osnova rada ista je kao i kod stabla odluke. Umjesto jednog stabla, gradi više neovisnih te analizom njihovih odluka odabire prikladan rezultat.
- **Linearna regresija** (engl. *linear regression*) - Na temelju ulaznih vrijednosti određuju se koeficijenti pravca najčešće metodom najmanjih kvadrata.
- **Stroj potpornih vektora** (engl. *linear support vector machine*) - Model reprezentira uzorke u prostoru te među kategorijama stvara hiperravninu nastojeći napraviti maksimalan razmak između kategorija.
- **Naivan Bayesov klasifikator** (engl. *naive Bayes*) - Jednostavan klasifikator kojemu je osnova rada Bayesov teorem koji "naivno" tvrdi da između bilo kojeg para značajki postoji nezavisnost. MLlib trenutno podržava *multinomial naive Bayes* te *Bernoulli naive Bayes*.
- **Višeslojni perceptron** (engl. *multilayer perceptron classifier*) - Jednostavna neuronska mreža koja se sastoji od više povezanih slojeva čvorova. Za sada je u MLlibu to jedini algoritam povezan s neuronskim mrežama.

5. Srodni radovi

Zahvaljujući popularnosti Apache Sparka, moguće je pronaći mnoštvo primjera korištenja njegovih modula u praksi. Osim toga, postoje i službeni primjeri za pojedine metode koji demonstriraju načine korištenja, a mogu se pronaći na Sparkovoj službenoj GitHub stranici [4].

Jedan od primjera korištenja MLliba dan je u članku "*Spam classification using Spark's DataFrames, ML and Zeppelin*" [6]. Navedeni članak demonstrira izgradnju modela koji bi predviđao spada li neki email u neželjenu poštu (engl. *spam*) ili ne. Članak izvrsno prikazuje proces obrade prirodnog jezika i ukazuje na važnost pripreme podataka i izvlačenja relevantnih značajki. Posebno je istaknuto i vrednovanje rezultata korištenjem ugrađenih metoda, ali ukazuje na njihove nedostatke.

Još jedan primjer je analiza zločina u San Franciscu [18]. U primjeru dana je detaljna statistička analiza i fokus je stavljen na upoznavanje skupa podataka i pojedinih značajki te uočavanje nepravilnosti među njima. Cilj rada bio je saznati je li moguće predvidjeti naselje u kojem se zločin odvio na temelju nekih značajki kao što je dan u tjednu ili opis zločina. Ovaj rad koristi dvije metode za izgradnju modela: logističku regresiju i algoritam slučajne šume, gdje se slučajna šuma pokazala boljim izborom s 80% preciznošću.

Postoje i mnogi drugi srodni projekti, međutim oni često koriste starije verzije Apache Sparka koja nije sadržavala mnoge, sada implementirane i ustaljene metode.

6. Praktični dio

6.1. Opis skupa podataka - puzlatka

Puzlatka (engl. *abalone*, lat. *Haliotis*) je naziv za porodicu morskih puževa čija veličina varira od 20mm do 30cm. Ono što ih čini toliko privlačnima za ljude je, osim ukusnog mesa koje se može pripremati na razne načine, i šarena unutrašnjost same školjke od koje se izrađuje nakit [1].

Slika 6.1: Školjka *Haliotis* (engl. *abalone*), u hrvatskom jeziku poznata je i pod nazivima: puzlatka, ušenak, divojačko uho, srebrnica ili Petrovo uho

Najprecizniji način za određivanje dobi puzlatke je čiščenjem same školjke te mikroskopsko brojanje prstenova. Ovo je jako spor i naporan proces, a u ovom radu taj proces pokušat će se olakšati izradom modela koji bi predviđao dob na temelju fizičkih karak-

teristika same školjke [17].

Skup podataka nastao je tijekom biološkog istraživanja *"The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait"* 1994. godine, a sadrži zapise za 4177 školjki, ima 9 vrijednosti te nema nedostajućih vrijednosti. Svaka je školjka opisana s osam atributa navedenih u tablici 6.1. Posljednja vrijednost, broj prstenova, je vrijednost koju se pokušava predvidjeti. Opisani skup podataka javno je dostupan putem UCI repozitorija [9].

Tablica 6.1: Prikaz atributa u skupu podataka

Ime	Tip podataka	Mjerna jedinica	Opis
Spol	*string*	–	M, F i I (muško, žensko, mlado)
Duljina	*double*	mm	Najdulja stranica školjke
Promjer	*double*	mm	Okomito na duljinu
Visina	*double*	mm	Visina neočišćene školjke (s mesom)
Težina	*double*	g	Ukupna težina školjke
Meso	*double*	g	Težina samog mesa
Utroba	*double*	g	Težina utrobe nakon iskrvarenja
Školjka	*double*	g	Težina školjke nakon sušenja
Prstenovi	*integer*	–	Broj mikroskopski izbrojenih prstenova

6.2. Učitavanje podataka u Apache Spark

Budući da je skup podataka *Abalone* već pročišćen i ima jasno definiranu strukturu, bez nedostajućih vrijednosti, moguće ga je bez poteškoća učitati u MLlibu prikladan oblik - DataFrame.

```
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
dataFrame = sqlContext.read
        .format('com.databricks.spark.csv')
        .options(header='False', mode='DROPMALFORMED')
        .schema(schema)
        .load('abalone.data.csv')
```

Prvo je potrebno učitati i stvoriti primjerak razreda SQLContext koji omogućava učitavanje podataka u DataFrame. Tek nakon toga, koristeći varijablu sqlContext čitamo iz datoteke 'abalone.data.csv' zapisane u csv (engl. *comma-separated values*) formatu. Način čitanja csv formata dostupan je putem csv biblioteke '*com.databricks.spark.csv*'.

Zahvaljujući dobroj uređenosti izabranog skupa, iskorišteni su samo neki od brojnih argumenata, a njihovo je značenje objašnjeno ispod.

Opcija *header* postavlja se na '*True*' kada želimo zapamtiti sve vrijednosti atributa koje se nalaze u prvom redu zapisa kao nazive pojedinih tablica u novostvorenom DataFrameu. Unatoč tomu što je zadana vrijednost opcije *header*='*False*', ovdje je ona navedena kako bi se naglasila važnost razumijevanja načina na koji su podaci zapisani.

Opcija *mode* definira način parsiranja podataka. Postavljanjem u način '*DROPMALFOMRED*' osiguravamo da se svi nevaljani redci izuzmu iz obrade. Budući da u danom skupu podataka nemamo zaglavlje kojim bi imenovali imena stupaca tablice, potrebno je shemu ručno definirati što se može na sljedeći način.

```
schema = StructType([
        StructField('Sex', StringType(), True),
        StructField('Length', DoubleType(), True),
        StructField('Diameter', DoubleType(), True),
        StructField('Height', DoubleType(), True),
        StructField('Whole_weight', DoubleType(), True),
        StructField('Shucked_weight', DoubleType(), True),
        StructField('Viscera_weight', DoubleType(), True),
        StructField('Shell_weight', DoubleType(), True),
        StructField('Rings', IntegerType(), True)])
```

Definirana shema može se predati kao argument funkcije *schema()* kako bi se iskoristila za izgradnju relacijske strukture.

Na ovaj način, vrlo jednostavno je izgrađen DataFrame s devet imenovanih stupaca i 4177 redaka nad kojim se mogu provoditi deklarativni upiti.

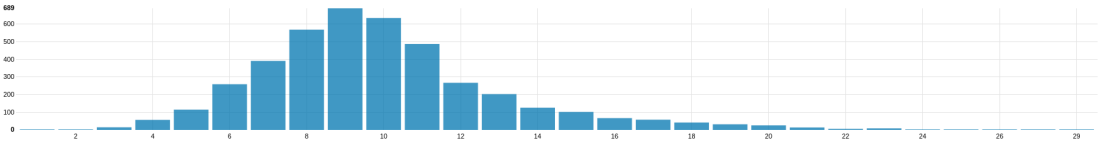
6.3. Istraživanje skupa podataka

Nad izabranim skupom podataka želimo odrediti kategoriju školjke temeljem njezinih fizičkih karakteristika. Kako bi bolje razumjeli tematiku problema, potrebno je istražiti koliko kategorija uopće postoji, odnosno koliko različitih vrijednosti ima u stupcu 'Rings'.

```
from pyspark.sql.functions import col

categories = dataFrame.groupBy("Rings").count() \
                .orderBy(col("Rings").desc())
```

Ovakvim grupiranjem dobije se sljedeća raspodjela po kategorijama:

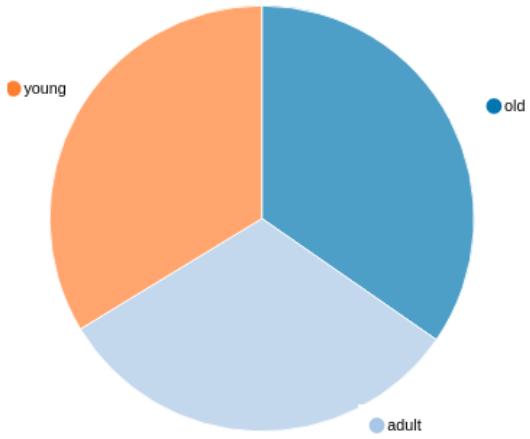


Slika 6.2: Raspodjela uzoraka po broju prstenova

Kao što je vidljivo sa slike 6.2, varijable stupca 'Rings' mogu poprimiti 29 različitih vrijednosti. To znači da ima 29 mogućih kategorija u koje je moguće smjestiti pojedini uzorak. S obzirom na to da bi velik broj kategorija imao jak učinak na performanse i memoriju, u svrhu pojednostavljenja problema, grupirali smo uzorke u tri manje skupine na način prikazan tablicom 6.2, a koji rezultira jednolikom raspodjelom po kategorijama (slika 6.3). Kriterij za raspodjelu po kategorijama temelji se na individualnoj procjeni.

Tablica 6.2: Nova kategorizacija temeljena na individualnoj procjeni

Broj prstenova	Kategorija
0-8	Mlade
9-10	Odrasle
11-29	Stare

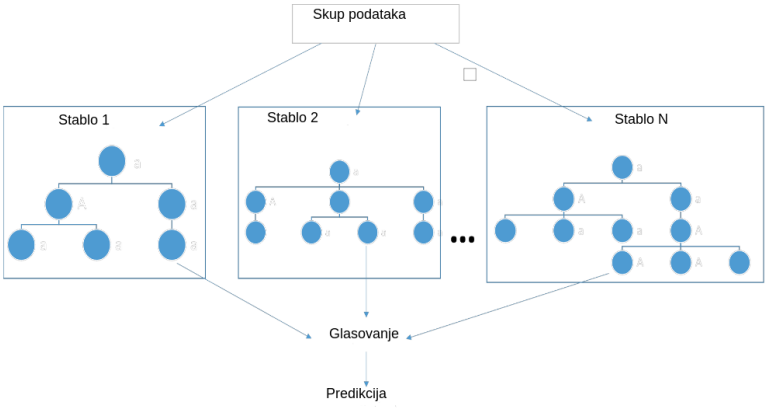


Slika 6.3: Prikaz raspodjele uzoraka prema novim kategorijama

6.4. Algoritam slučajne šume

Slučajna šuma (engl. *random forest*) jedan je od najčešće korištenih algoritama za strojno učenje. Popularnost zahvaljuje činjenici da daje dobre rezultate u većini slučajeva. Također je i jednostavan za korištenje te je njime moguće rješavati klasifikacijske i regresijske probleme.

Princip rada (slika 6.4) isti je kao kod stabla odluke. Međutim, slučajna šuma gradi više od jednog stabla te, nakon što sva stabla donesu svoju odluku, analizira sve rezultate i odabire onaj za koji se odlučilo najviše stabala. Stabla odluke grade se iz nasumično odabranih značajki pojedinog skupa podataka [16].



Slika 6.4: Princip rada agloritma slučajne šume

Postoje dva izvora slučajnosti kod slučajne šume [5]:

1. Bootstrapping - podaci za svako stablo u šumi se nasumično odabiru u n primjeraka, svaki put s ponavljanjem.

2. Metoda slučajnog potprostora - u svakom čvoru, svakog stabla, radi se nasumični izbor između *sqrt(M)* broja značajki na kojima se razmatra podjela vrijednosti. M je ukupni broj značajki skupa podataka.

6.5. Izgradnja cjevovoda

Nakon analize skupa podataka te odabira predikcijskog algoritma, moguće je izgraditi cjevovod. Zahvaljujući činjenici da su sva relevantna svojstva u skupu podataka tipa *double*, nema potrebe za složenim pretvorbama kao što bi to bio slučaj da se radi o neuređenom skupu podataka.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier


labelIndexer = StringIndexer(inputCol="Category", \
            outputCol="indexedLabel") \
            .fit(dataFrame)
vecAssembler = VectorAssembler(inputCols = \
              ["Length",
              "Diameter",
              "Height",
              "Whole weight",
              "Shucked weight",
              "Viscera weight",
              "Shell weight"],
              outputCol="features")
rf = RandomForestClassifier(labelCol="indexedLabel")
pipeline = Pipeline(stages=[labelIndexer, vecAssembler, rf])
```

Prije svega, potrebno je izvršiti potrebne pretvorbe nad novostvorenim stupcem "Category" jer MLlib algoritmi razumiju isključivo numeričke vrijednosti, a taj stupac sadrži vrijednosti tipa *string*. U tu svrhu pozivamo nad tim stupcem funkciju *StringIndexer()*. Ovaj transformator kodira stupac s labelama u numeričke vrijednosti. Moguće vrijednosti su između 0 i broja ukupnih kategorija [12].

S druge strane, nad *double* svojstvima je iz gore navedenih razloga dovoljno pozvati VectorAssembler, transformator čija je zadaća iskombinirati listu zadanih stupaca u jedan vektor. Kao argumente prihvaća sve tipove numeričkih vrijednosti, *boolean* tip podataka i vektore. Transformator rezultira konkatenacijom svih predanih stupaca u jedinstven vektor u zadanom redoslijedu [12].

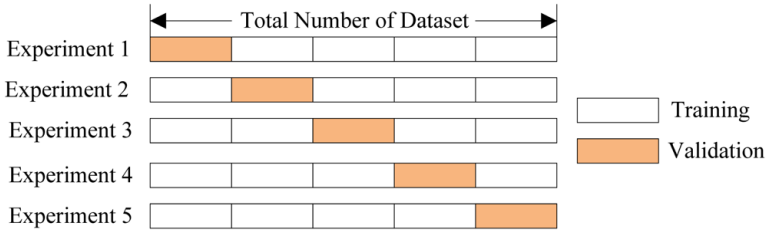
Sljedeći korak je stvoriti primjerak razreda RandomForestClassifier. U sljedećem poglavlju bit će detaljno objašnjeni razlozi zbog kojih nema specificiranih parametara.

Posljednje, pozivamo funkciju *Pipeline()* koja gradi cjevovod s tri faze.

6.6. Vrednovanje rezultata

U ovom dijelu osvrnut ćemo se na mogućnost ugađanja parametara i vrednovanja rezultata.
Jedna od stvari koja može jako utjecati na rezultate je izbor samog modela, ali čak i kada je model odabran, ostaje još mnogo parametara koji trebaju ugađanje te u ovisnosti o kojima rezultat jako varira. MLlib pruža mogućnost ugađanja pojedinih faza ili čak kompletnog cjevovoda.

U konkretnom primjeru izabrana je unakrsna provjera (engl. *cross validation*). U MLlibu, kod za unakrsnu provjeru temeljem parametra k dijeli skup podataka u određeni broj parova (učenje, testiranje). Primjerice, ako se postavi da je k=10, podijelit će se dani skup podataka u 10 slučajnim odabirom stvorenih parova u omjeru 2/3 i 1/3. Na ovaj način, pronalaze se optimalni parametri za odabrani model.



Slika 6.5: Primjer podjele skupa podataka za unakrsnu provjeru

```
from pyspark.ml.evaluation import \
  MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.tuning import ParamGridBuilder

paramGrid = ParamGridBuilder().build()

crossval = CrossValidator(
  estimator=pipeline,
  estimatorParamMaps=paramGrid,
  evaluator=MulticlassClassificationEvaluator(
            predictionCol="prediction",
            labelCol="indexedLabel"),
            numFolds=10)

model = crossval.fit(trainingData)
prediction = model.transform(testData)
```

Pozivom metode *fit()* generira se model, dok metoda *transform()* stvara novi DataFrame s dodatnim stupcem predikcija. Testni podaci koji su predani metodi *transform()* moraju biti podaci koje model još nikada nije vidio. Temeljem stupca '*prediction*' moguće je odrediti preciznost, odnosno pouzdanost našeg modela.
MLlibov ugrađeni *MultiClassificationEvaluator* pozivom metode *evaluate()*, vraća upravo preciznost tipa *double*.

```
evaluator = MulticlassClassificationEvaluator(
            predictionCol="prediction",
            labelCol="indexedLabel")
evaluator.evaluate(prediction)
```

Preciznost modela ispala je 64,3%. U usporedbi s nekim sličnim radovima kao što su "*Predicting the age of abalone*" [17] i "*Data Mining For Abalone*"[10] kojima preciznost varira (budući da su koristili i druge metode osim algoritma slučajne šume) između 60% i 70% to je zaista dobar rezultat uzimajući u obzir tematiku problema.

7. Zaključak

Zahvaljujući slojevitoj arhitekturi Apache Spark svojim korisnicima omogućava visoku razinu apstrakcije za vrijeme programiranja. Pogodan je za korištenje početnicima, ali i naprednijim korisnicima, uz module bogate mnoštvom metoda koje skrivaju složenost pojedinih algoritama te API-je za razne programske jezike. Apstrakcija rezultira i smanjenim brojem linija koda te bržim i jednostavnijim razvojem programske potpore.

Jednostavnost, skalabilnost i lakoća integracije s ostalim modulima vidljive su i u knjižici MLlib. Glavna prednost jest to što omogućava korisniku da se koncentrira na sam problem, bez potrebe za namještanjem konfiguracije ili razmišljanja o infrastrukturi i pozadinskoj implementaciji.

Osnova na kojoj počiva MLlib su takozvani cjevovodi. *ML Pipelines* je skup API-ja izgrađenih nad tipom podataka DataFrame koji omogućavaju razdvajanje cjelokupnog procesa (ili složenog algoritma) na manje logičke cjeline [13].

Unatoč brojnim prednostima, MLlib je još uvijek u razvoju te za sad implementira samo popularne algoritme i manjka neke funkcionalnosti. No, očekuje se da će, s porastom popularnosti Apache Sparka, rasti i popularnost njegovog korištenja za strojno učenje te da će potaknuti brži i nagliji razvoj MLliba.

LITERATURA

[1] Abalone. URL https://en.wikipedia.org/wiki/Abalone.

[2] The 7 steps of machine learning. URL https://www.youtube.com/watch?v=nKW8Ndu7Mjw.

[3] Što je to "machine learning" ili strojno učenje? URL http://pcchip.hr/helpdesk/sto-je-to-machine-learning-ili-strojno-ucenje.

[4] Apache github. URL https://github.com/apache/spark/tree/master/examples/src/main/python.

[5] Random forests. URL http://lis.irb.hr/KDSA2008/presentations/bosnjak.pdf.

[6] Spam classification using spark's dataframes, ml and zeppelin. URL https://blog.codecentric.de/en/2016/06/spam-classification-using-sparks-dataframes-ml-zeppelin.

[7] Strojno učenje struktura metoda/algoritama strojnog učenja. URL https://web.math.pmf.unizg.hr/nastava/su/index.php/download_file/-/view/158/.

[8] Apache spark - rdd. URL https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm.

[9] Abalone data set, 1995. URL https://archive.ics.uci.edu/ml/datasets/abalone.

[10] Ahmetjan Asim ans Yan Li, Yinghong Xie, i Yongfang Zhu. Data mining for abalone, 2013. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.2321&rep=rep1&type=pdf.

[11] *Machine Learning Library (MLlib) Guide*. Apache Spark, . URL https://spark.apache.org/docs/latest/ml-guide.html.

[12] *Extracting, transforming and selecting features*. Apache Spark, . URL https://spark.apache.org/docs/latest/ml-features.html.

[13] *ML Pipelines*. Apache Spark, . URL https://spark.apache.org/docs/latest/ml-pipeline.html.

[14] *SQL programming guide*. Apache Spark, . URL https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes.

[15] *Spark Overview*. Apache Spark, 2002. URL https://spark.apache.org/docs/latest/.

[16] Niklas Donges. The random forest algorithm. 2018. URL https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.

[17] Jozef Janovsky. Predicting the age of abalone, 2013. URL https://www.slideshare.net/hyperak/predicting-the-age-of-abalone.

[18] Ofer Mendelevitch. Data science demo: predicting crime resolution in san francisco. URL https://github.com/ofermend/Zeppelin-Notebooks.

[19] Matei Zaharia, Patrick Wendell, Andy Konwinski, i Holden Karau. *Learning Spark*. O'Reilly Media, Inc., 2015.

Dubinska analiza podataka u radnom okviru Apache Spark pomoću knjižnice MLlib

Sažetak

Količina informacija kojima smo okruženi eksponencijalno raste i s njom javlja se potreba za novim rješenjima na području znanosti o podacima. Kao jedno od rješenja, pojavio se Apache Spark, sa svojom izuzetno brzom jezgrom namijenjenoj za obradu velikih skupova podataka. Spark ujedinjuje nekoliko ugrađenih biblioteka koje je moguće međusobno kombinirati unutar jedne aplikacije. Ovaj rad fokusira se na korištenje MLliba, knjižice za strojno učenje, u procesu izvlačenja korisnih informacija iz velike količine podataka. Taj proces objašnjen je detaljno, počevši s uputama za instalaciju Apache Sparka, kratkim osvrtom na osnovne koncepte i naposljetku praktičnim primjerom izgradnje cjevovoda za strojno učenje te vrednovanjem rezultata koristeći MLlib.

Ključne riječi: znanost o podacima, Spark ML, strojno učenje, analiza podataka, obrada podataka, slučajna šuma, DataFrame, RDD, veliki podaci, cjevovod, vrednovanje modela

Data mining in Apache Spark framework using MLlib library

Abstract

The amount of digital data that exists is growing at a rapid rate and with it the need for new solutions in the field of data science. As one of the solutions Apache Spark emerged with its lightening fast unified analytics engine for large-scale data processing. It provides several built-in libraries that you can easily combine in the same application. This thesis focuses on the use of MLlib, a machine learning library, in the process of extracting useful knowledge from volumes of data. The process is explained in details, starting with Apache Spark installation guide, an overview of the main concepts and culminating with a practical example of creating a machine learning pipeline and model evaluation via MLlib library.

Keywords: Data Science, Spark ML, machine learning, data analysis, data processing, Random Forest algorithm, DataFrame, RDD, Big Data, pipeline, model evaluation