TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JONNE PETTERI PIHLANEN

SUOSITTELIJAJÄRJESTELMÄN RAKENTAMINEN APACHE SPARKILLA

Diplomityö

Examiner: ????
Examiner and topic approved by the
Faculty Council of the Faculty of
xxxx
on 1st September 2014

ABSTRACT

JONNE PETTERI PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampere University of Technology
Diplomityö, xx pages
September 2016
Master's Degree Program in Signal Processing
Major: Data Engineering
Examiner: ????
Keywords:

The amount of recommendation engines around the Internet is constantly growing.

This paper studies the usage of Apache Spark when building a recommendation engine.

TIIVISTELMÄ

JONNE PETTERI PIHLANEN: Building a Recommendation Engine with Apache Spark
Tampereen teknillinen yliopisto
Diplomityö, xx sivua
syyskuu 2016
Signaalinkäsittelyn koulutusohjelma
Pääaine: Data Engineering
Tarkastajat: ????
Avainsanat:

PREFACE

Thanks to my wife, Noora, for pushing me forward with the thesis when my own interest was completely gone. Without you this would never have been ready.

Tampere,

Jonne Pihlanen

SISÄLLYS

LIST OF ABBREVIATIONS AND SYMBOLS

Spark Fast and general engine for large-scale data processing

Information retrieval (IR) Activity of obtaining relevant information resources from a collection of information resources.

1. JOHDANTO

Suosittelujärjestelmiä on menestyksellisesti käytetty auttamaan asiakkaita päätöksenteossa. Itse asiassa ne ovat jatkuvasti läsnä jokapäiväisessä elämässämme. Mikäli asiakas tekee ostoksia, katsoo elokuvaa Netflixistä, selailee Facebookia tai yksinkertaisesti lukee vain uutisia. Periaatteessa kaikki elämämme osa-alueet sisältävät jonkinlaista suosittelua. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkiläkohtaisesti kokeneet. Tälläin suosittelijajärjestelmistä tulee hyädyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia tuhansista erilaisista tuotteista.

Suosittelu voidaan jakaa kahteen pääkategoriaan: tuotepohjaiseen ja käyttäjäpohjaiseen suositteluun. Tuotepohjaisesa suosittelussa tarkoituksena on etsiä samankaltaisia tuotteita, sillä käyttäjä saattaa haluta mieluummin samankaltaisia tuotteita myäs tulevaisuudessa. Käyttäjäpohjaisessa suosittelussa käyttäjän ajatellaan olevan kiinnostunut tuotteista, joita samankaltaiset käyttäjät ovat ostaneet. Käyttäjäpohjainen suosittelu yrittää siis etsiä samankaltaisia käyttäjiä, jotta voidaan suositella näiden käyttäjien ostamia tuotteita.

Apache Spark on sovelluskehys, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Hajautettu ohjelma tarkoittaa, että ohjelman suoritus on jaettu useiden käsittelysolmujen kesken. Suositteluongelma voidaan mallintaa hajautettuna sovelluksena, jossa kaksi matriisia, käyttäjät ja tuotteet, prosessoidaan iteratiivisella algoritmilla, joka mahdollistaa ohjelman suorittamisen rinnakkain.

Apache Spark on rakennettu Scala ohjelmointikielellä. Scala on monikäyttäinen, moniparadigmainen ohjelmointikieli, joka tarjoaa tuen funktionaaliselle ohjelmoinnille sekä vahvan tyypityksen. Tyässä käytetään Scalaa, joten lyhyt johdanto ohjelmointikieleen tarjotaan.

Tämä työ on rakentuu seuraavista osista. Luku kaksi kuvailee suosittelujärjestelmiä. Luvussa kolme keskutellaan Apache Sparkista, avoimen lähdekoodin järjestel-

mästä, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Luku neljä esittää toteutuksen suosittelijajärjestelmälle. Luvussa viisi käydään läpi tulokset. Lopuksi luvussa kuusi esitellään johtopäätäkset.

2. SUOSITTELIJAJÄRJESTELMÄT

Suosittelulla tarkoitetaan tehtävää, jossa tuotteita suositellaan käyttäjille. Kaikista yksinkertaisin suosittelun keino on vertaisten kesken, täysin ilman tietokoneita. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkilökohtaisesti kokeneet. Tällöin suosittelijajärjestelmistä tulee hyödyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia tuhansista erilaisista tuotteista. Suosittelijajärjestelmät ovat joukko tekniikoita ja ohjelmistoja jotka tarjovat suosituksia mahdollisesti hyödyllisistä tuotteista. Tuote tarkoittaa yleistä asiaa jota järjestelmä suosittelee henkilölle. Suosittelujärjestelmät on yleensä tarkoitettu suosittelemaan tietyn tyyppisiä tuotteita kuten kirjoja tai elokuvia. [10]

Suosittelijajärjestelmien tarkoitus on auttaa asiakkaita päätöksenteossa kun tuotteiden määrä on valtava. Tavallisesti suositukset ovat räätälöityjä, millä tarkoitetaan että suositukset eroavat käyttäjien tai käyttäjäryhmien välillä. Suositukset voivat olla myös räätälöimättömiä ja niiden tuottaminen onkin usein yksinkertaisempaa. (VIITE?) Räätälöimätöntä suosittelua on esimerkiksi yksinkertainen top 10 lista. Järjestäminen tehdään ennustamalla kaikista sopivimmat tuotteet käyttäjän mieltymysten tai vaatimusten perusteella. Tämän suorittamiseksi suosittelijajärjestelmän on kerättävä käyttäjältä tämän mieltymykset. Mieltymykset voivat olla nimenomaisesti ilmaistuja tuotearvioita tai ne voidaan tulkita käyttäjän toiminnasta kuten klikkauksista tai sivun katselukerroista. Suosittelijajärjestelmä voisi esimerkiksi tulkita tuotesivulle navigoinnin todisteeksi mieltymyksestä sivun tuotteista. [10]

Suosittelijajärjestelmien kehitys alkoi melko yksinkertaisesta havainnosta: ihmiset tapaavat luottaa toisten suosituksiin tehdessään rutiininomaisia päätöksiä. On esimerkiksi yleistä luottaa vertaispalautteeseen valitessaan kirjaa luettavaksi tai luottaa elokuvakriitikoiden kirjoittamiin arvioihin. Ensimmäinen suosittelijajärjestelmä yritti matkia tätä käytöstä käyttämällä algoritmeja suosituksien löytämiseen yhteisöstä aktiiviselle käyttäjälle, joka etsi suosituksia. Tämä lähestymistapa on olennaisesti yhteistyösuodattamista ja idea sen takana on että jos käyttäjät pitivät saman-

kaltaisista tuotteista aikaisemmin, he luultavasti pitävät samoja tuotteita ostaneiden henkilöiden suosituksia merkityksellisinä. [10]

Verkkokauppojen kehityksen myötä syntyi tarve suosittelulle vaihtoehtojen rajoittamiseksi. Käyttäjät kokivat aina vain vaikeammaksi löytää oikeat tuotteet sivustojen suurista valikoimista. Tiedon määrän räjähdysmäinen kasvaminen internetissä on ajanut käyttäjät tekemään huonoja päätöksiä. Hyödyn tuottamisen sijaan vaihtoehtojen määrä oli alkanut heikentää kuluttajien hyvinvointia. Vaihtoehdot ovat hyväksi, mutta vaihtoehtojen lisääntyminen ei ole aina parempi. [10]

Viimeaikoina suosittelijajärjestelmät ovat osoittautuneet tehokkaaksi lääkkeeksi käsillä olevaa tiedon ylimääräongelmaa vastaan. Suosittelijajärjestelmät käsittelevät tätä ilmiötä tarjoamalla uusia, aiemmin tuntemattomia tuotteita jotka ovat todennäköisesti merkityksellisiä käyttäjälle tämän nykyisessä tehtävässä. Kun käyttäjä pyytää suosituksia, suosittelujärjestelmä tuottaa suosituksia käyttämällä tietoa ja tuntemusta käyttäjistä, saatavilla olevista tuotteista ja aiemmista tapahtumista suosittelijan tietokannasta. Tutkittuaan tarjotut suositukset, käyttäjä voi hyväksyä tai hylätä ne tarjoten epäsuoraa ja täsmällistä palautetta suosittelijalle. Tätä uutta tietoa voidaan myöhemmin käyttää hyödyksi tuotettaessa uusia suosituksia seuraaviin käyttäjän ja järjestelmän vuorovaikutuksiin. [10]

Verrattuna klassisten tietojärjestelmien, kuten tietokantojen ja hakukoneiden, tutkimukseen, suosittelijajärjestelmien tutkimus on verrattain tuoretta. Suositteljiajärjestelmistä tuli itsenäisiä tutkimusalueita 90-luvun puolivälissä. Viimeaikoina mielenkiinto suosittelujärjestelmiä kohtaan on kasvanut merkittävästi. Esimerkkinä suuren profiilin verkkosivustot kuten Amazon.com, YouTube, Netflix sekä IMDB, joissa suosittelujärjestelmillä on iso rooli. Oma lukunsa ovat myös vain suosittelujärjestelmien tutkimiseen ja kehittämiseen tarkoitetut konferenssit kuten RecSys ja AI Communications (2008). [10]

Suosittelujärjestelmällä voidaan ajatella olevan kaksi päätarkoitusta. Ensimmäinen on avustaa palveluntarjoajaa. Toinen on tuottaa arvoa palvelun käyttäjälle. Suosittelujärjestelmän on siis tasapainoteltava sekä palveluntarjoajan että käyttäjän tarpeiden välillä. [10] Palveluntarjoaja voi esimerkiksi ottaa suosittelujärjestelmän avuksi parantamaan tai monipuolistamaan myyntiä, lisäämään käyttäjien tyytyväisyyttä, lisäämään käyttäjien uskollisuutta tai ymmärtämään paremmin mitä käyttäjä haluaa [10]. Lisäksi käyttäjillä saattaa olla seuraavanlaisia odotuksia suosittelujärjestelmältä. Käyttäjä saattaa haluta suosituksena tuotesarjan, apua selaamiseen

tai mahdollistaa muihin vaikuttamisen. Vaikuttaminen saattaa olla pahantahtoista. [10]

GroupLens, BookLens ja MovieLens olivat uranuurtajia suosittelujärjestelmissä. GroupLens on tutkimuslaboratorio tietojenkäsittelyopin ja tekniikan laitoksella Minnesotan Yliopistossa, Twin Cities:issä, joka on erikoistunut suosittelujärjestelmiin, verkkoyhteisöihin, mobiili ja kaikkialla läsnä oleviin teknologioihin, digitaalisiin kirjastoihin ja paikallisen maantieteen tietojärjestelmiin. [2] BookLens on on GroupLensin rakentama kirjojen suosittelujärjestelmä [3]. MovieLens on GroupLensin ylläpitämä elokuvien suosittelujärjestelmä [9]. Uranuurtavan tutkimuksen lisäksi nämä sivustot julkaisivat aineistoja, joka ei ollut yleistä. [2]

2.1 Suositustekniikat

Suosittelujärjestelmällä täytyy olla jonkunlainen ymmärrys tuotteista, jotta se pystyy suosittelemaan jotain. Tämän mahdollistamiseksi, järjestelmän täytyy pystyä ennustamaan tuotteen käytännöllisyys tai ainakin verrata tuotteiden hyödyllisyyttä ja tämän perusteella päättää suositeltavat tuotteet. Ennustamista voidaan luonnostella yksinkertaisella ei-personoidulla suosittelualgoritmilla joka suosittelee vain suosituimpia elokuvia. Tätä lähestymistapaa voidaan perustella sillä, että tarkemman tiedon puuttuessa käyttäjän mieltymyksistä, elokuva josta muutkin ovat pitäneet on todennäköisesti myös keskivertokäyttäjän mieleen, ainakin enemmän kuin satunaisesti valikoitu elokuva. Suosittujen elokuvien voidaan siis katsoa olevan kohtuullisen osuvia suosituksia keskivertokäyttäjälle. [10]

Tuotteen i hyödyllisyyttä käyttäjälle u voidaan mallintaa reaaliarvoisella funktiolla $R(u, i)$, kuten yleensä tehdään yhteisösuodatuksessa ottamalla huomioon käyttäjien antamat arviot tuotteista. Yhteisösuodatus suosittelijan perustehtävä on ennustaa R:n arvoa käyttäjä-tuote pareille ja laskea arvio todelliselle funktiolle R. Laskiessaan tätä ennustetta käyttäjälle u tuotejoukolle, järjestelmä suosittelee tuotteita joilla on suurin ennustettu hyödyllisyys. Ennustettujen tuotteiden määrä on usein paljon pienempi kuin tuotteiden koko määrä, joten voidaan sanoa että suosittelijajärjestelmä suodattaa käyttäjälle suositeltavat tuotteet. [10]

Suosittelijajärjestelmät eroavat toisistaan kohdistetun toimialan, käytetyn tiedon ja erityisesti siinä kuinka suositukset tehdään, jolla tarkoitetaan suosittelualgoritmia [10]. Tässä työssä keskitytään vain yhteen suosittelutekniikoiden luokkkaan, yhtei-

sölliseen suodatukseen, sillä tätä menetelmää käytetään Apache Sparkin MLlib kirjastossa.

Yhteisöllistä suodatusta käyttävät suosittelijajärjestelmät perustuvat käyttäjien yhteistyöhön. Niiden tavoitteena on tunnistaa kuvioita käyttäjän mielenkiinnoista voidakseen tehdä suunnattuja suosituksia [1]. Tämän lähestymistavan alkuperäisessä toteutuksessa suositellaan aktiiviselle käyttäjälle niitä tuotteita joita muut samankaltaiset mieltymyksen omaavat käyttäjät ovat pitäneet aiemmin [10]. Käyttäjä arvostelee tuotteita. Seuraavaksi algoritmi etsii suosituksia perustuen käyttäjiin, jotka ovat ostaneet samanlaisia tuotteita tai perustuen tuotteisiin, jotka ovat eniten samanlaisia käyttäjän ostoshistoriaan verrattuna. Yhteisösuodatus voidaan jakaa kahteen kategoriaan, jotka ovat tuotepohjainen- ja käyttäjäpohjainen yhteisösuodatus. Yhteisösuodatusta on eniten käytetty ja toteutettu tekniikka suositusjärjestelmissä [6] [10] [4].

Yhteisöllinen suodatus analysoi käyttäjien välisiä suhteita ja tuotteiden välisiä riippuvuuksia tunnistaakseen uusia käyttäjä-tuote assosiaatioita [7]. Päätelmä siitä, että käyttäjät voisivat pitää samasta laulusta koska molemmat kuuntelevat muita samankaltaisia lauluja on esimerkki yhteisöllisestä suodatuksesta [11].

Koska yhteisöllisessä suodatuksessa suosittelu perustuu pelkästään käyttäjän arvosteluihin tuotteesta, yhteisöllinen suodatus kärsii ongelmista jotka tunnetaan nimillä uusi käyttäjäongelma ja uusi tuoteongelma [6]. Ellei käyttäjä ole arvostellut yhtään tuotetta, algoritmi ei kykene tuottamaan myöskään yhtään suositusta. Muita yhteisöllisen suodatuksen haasteita ovat kylmä aloitus sekä niukkuus (sparsity). Kylmällä aloituksella tarkoitetaan sitä, että tarkkojen suositusten tuottamiseen tarvitaan tyypillisesti suuri määrä dataa. Niukkuudella tarkoitetaan sitä, että tuotteiden määrä ylittää usein käyttäjien määrän. Tästä johtuen suhteiden määrä on todella niukka, sillä useat käyttäjät ovat arvostelleet tai ostaneet vain murto-osan tuotteiden koko määrästä. [1]

2.1.1 Muistiperustainen yhteisöllinen suodatus

Muistiperustaississa metodeissa käyttäjä-tuote suosituksia käytetään suoraan uusien tuotteiden ennustamiseksi. Tämä voidaan toteuttaa kahdella tavalla, käyttäjäpohjaisena suositteluna tai tuotepohjaisena suositteluna.

Seuraavat kappaleet kuvaavat käyttäjäpohjaista yhteisösuodatusta ja tuotepohjaista

yhteisösuodatusta.

Tuotepohjainen yhteisösuodatus

Tuotepohjainen yhteisösuodatus (Item-based collaborative filtering, IBCF) aloittaa etsimällä samankaltaisia tuotteita käyttäjän ostoshistoriasta [6]. Seuraavaksi mallinnetaan käyttäjän mieltymykset tuotteelle perustuen saman käyttäjän tekemiin arvosteluihin [10]. Alla oleva koodinpätkä esittelee ICBF:n idean jokaiselle uudelle käyttäjälle.

Program 2.1 *Tuotepohjainen yhteisösuodatus algoritmi [6]*

```
1. For each two items, measure how similar they are in terms of having received
    similar ratings by similar users

1. Jokaiselle kahdelle tuotteelle, mittaa kuinka samankaltaisia ne ovat sen
    suhteen, kuinka samankaltaisia arvioita ne ovat saaneet samankaltaisilta
    käyttäjiltä.

val similarItems = items.foreach { item1 =>
        items.foreach { item2 =>
                val similarity = cosineSimilarity(item1, item2);
        }
}

2. For each item, identify the k−most similar items

2. Tunnistaa k samankaltaisinta tuotetta jokaiselle tuotteelle

val itemsSorted = sort(similarItems)

3. For each user, identify the items that are most similar to the user's
    purchases

3. Jokaiselle käyttäjälle, tunnista tuotteet jotka ovat eniten samankaltaisia
    käyttäjän ostoshistorian kanssa.
```

```
users.foreach { user =>
        user.purchases.foreach { purchase =>
                val mostSimilar = findSimilarItem(purchase)
        }
}
```

Amazon.com:in, Amerikan suurin internetkauppa, on aiemmin tiedetty käyttävät tuote-tuotteeseen yhteisösuodatusta. Tässä toteutuksessa algoritmi rakentaa samankaltaisten tuotteiden taulun etsimällä tuotteita joita käyttäjät tapaavat ostaa yhdessä. Seuraavaksi algoritmi etsii käyttäjän ostoshistoriaa ja arvosteluita vastaavat tuotteet, yhdistää nämä tuotteet ja palauttaa suosituimmat tai eniten korreloivat tuotteet. [8]

Käyttäjäpohjainen yhteisösuodatus

Tuotepohjainen yhteisösuodatus (User-based collaborative filtering, UBCF) aloittaa etsimällä samankaltaisimmat käyttäjät. Seuraava askel on arvostella samankaltaisten käyttäjien ostamat tuotteet. Lopuksi valitaan parhaan arvosanan saaneet tuotteet. Samankaltaisuus saadaan laskettua vertaamalla käyttäjien ostoshistorioiden samankaltaisuutta. [10]

Askeleet jokaiselle uudelle käyttäjälle käyttäjäpohjaisessa yhteisösuodatuksessa ovat:

Program 2.2 *Käyttäjäpohjainen yhteisösuodatus algoritmi [6]*

```
1. Mittaa jokaisen käyttäjän samankaltaisuus uuteen käyttäjään. Kuten IBCF:ssä
    , suosittuja samankaltaisuusarvioita ovat korrelaatio sekä kosinimitta.

case class Similarity(userId1: Int, userId2: Int, score: Int)

val newUser: User = User("Adam", 31, purchases)
val similarities = users.map { user =>
        Similarity(newUser.id, user.id, cosineSimilarity(user, newUser)
}
```

```
2. Tunnista samankaltaisimmat käyttäjät. Vaihtoehtoja on kaksi: Voidaan valita
    joko parhaat k käyttäjää (k−nearest neighbors) tai voidaan valita käyttäjät,
    joiden samankaltaisuus ylittää tietyn kynnysarvon.

val mostSimilarUsers = similarities. filter (_.score > 0.8)

3. Arvostele samankaltaisimpien käyttäjien ostamat tuotteet. Arvostelu saadaan
    joko keskiarvona kaikista tai painotettuna keskiarvona, käyttäen
    samankaltaisuuksia painoina.

val ratedItems = mostSimilarUsers.map { user =>
        user.purchases.map { purchase =>
                val purchases = mostSimilarUsers.map { usr =>
                        usr.purchases. filter (_.id === purchase.id)
                }
        purchases.sum() / purchases.size
        }
}

4. Valitse parhaiten arvostellut tuotteet.

val topRatedItems = ratedItems.take(10)
```

2.1.2 Mallipohjainen yhteisösuodatus

Muistipohjaiseen yhteisösuodatuksen käyttäessä tallennettuja suosituksia suoraan ennustamisen apuna, mallipohjaisissa lähestymistavoissa näitä arvosteluita käytetään ennustavan mallin oppimiseen. Perusajatus on mallintaa käyttäjä-tuote vuorovaikutuksia tekijöillä jotka edustavat käyttäjien ja tuotteiden piileviä ominaisuuksia (latent factors) järjestelmässä. Piileviä ominaisuuksia ovat esimerkiksi käyttäjän mieltymykset ja tuotteiden kategoriat. Tämä malli opetetaan käyttämällä saatavilla olevaa dataa ja myöhemmin käytetään ennustamaan käyttäjien arvioita uusille tuotteille. [10]

Vaihtelevat pienimmät neliöt (Alternating Least Squares, ALS) algoritmi on esimerkki mallipohjaisesta yhteisösuodatusalgoritmista ja se esitetään seuraavassa lu-

vussa.

3. APACHE SPARK

Apache Spark on avoimen lähdekoodin sovelluskehys, joka yhdistää hajautettujen ohjelmien kirjoittamiseen tarkoitetun järjestelmän sekä elegantin mallin ohjelmien kirjoittamiseen. [11] Spark tarjoaa korkean tason rajapinnat Java, Scala, Python sekä R ohjelmointikielille.

Korkealla tasolla, jokainen Spark sovelus koostuu ajaja (driver) ohjelmasta sekä yhdestä tai useammasta täytäntöönpanijasta (executor). Ajaja on ohjelma, joka ajaa käyttäjän pääohjelmaa ja suorittaa erilaisia rinnakkaisia operaatioita klusterissa. Täytäntöönpanija on yksi kone klusterissa.

Spark voidaan esitellä kuvailemalla sen edeltäjää, MapReduce:a, ja sen tarjoamia etuja. MapReduce tarjosi yksinkertaisen mallin ohjelmien kirjoittamiseen ja pystyi suorittamaan kirjoitettua ohjelmaa rinnakkain sadoilla tietokoneilla. MapReduce skaalautuu lähes lineaarisesti datan koon kasvaessa. Suoritusaikaa hallitaan lisäämällä lisää tietokoneita suorittamaan tehtävää.

Apache Spark säilyttää MapReduce:n lineaarisen skaalautuvuuden ja vikasietokyvyn mutta laajentaa sitä kolmella merkittävällä tavalla. Ensiksi, MapReducessa map- ja reduce-tehtävien väliset tulokset täytyy kirjoittaa levylle kun taas Spark kykenee välittämään tulokset suoraan putkiston seuraavalle vaiheelle. Toiseksi, Apache Spark kohtelee kehittäjiä paremmin tarjoamalla rikkaan joukon muunnoksia (transformations) joiden avulla voidaan muutamalla koodirivillä ilmaista monimutkaisia putkistoja. (ESIMERKKI?) Kolmanneksi, Spark esittelee muistissa tapahtuvan prosessoinnin tarjoamalla abstraktion nimeltä Resilient Distributed Dataset (RDD). RDD tarjoaa kehittäjälle mahdollisuuden materialisoida minkä tahansa askeleen liukuhihnassa ja tallentaa sen muistiin. Tämä tarkoittaa sitä, että tulevien askelien ei tarvitse laskea aiempia tuloksia uudelleen ja tällöin on mahdollista jatkaa juuri käyttäjän haluamasta askeleesta. Aiemmin tämänkaltaista ominaisuutta ei ole ollut saatavilla hajautetun laskennan järjestelmissä. [11]

Spark ohjelmia voidaan kirjottaa Java, Scala, Python tai R ohjelmointikielellä. Scalan käyttämisellä saavutetaan kuitenkin muutamia etuja, joita muut kielet eivät tarjoa. Tehokkuus paranee, sillä tehtävät kuten datan siirtäminen eri kerrosten välillä tai muunnosten suorittaminen datalle saattaa johtaa heikompaan tehokkuuteen. Spark on kirjoitettu Scala-ohjelmointikielellä, joten viimeisimmät ja parhaimmat ominaisuudet ovat aina käytössä. Spark ohjelmoinnin filosofia on helpompi ymmärtää kun Sparkia käytetään kielellä, jolla se on rakennettu. Suurin hyöty jonka Scalan käyttäminen tarjoaa, on kuitenkin kehittäjäkokemus joka tulee saman ohjelmointikielen käyttämisestä kaikkeen. Datan tuonti, manipulointi ja koodin lähettäminen klustereihin hoituvat samalla kielellä. [11]

Spark-jakelun mukana toimitetaan luku-evaluointi-tulostus-silmukka, komentorivityökalu, (Read eval print loop, REPL), joka mahdollistaa uusien asioiden nopean testailun konsolissa, eikä sovelluksista tarvitse rakentaa itsenäisiä (self-contained) alusta asti. Usein kun REPLissä kehitetty sovelluksen tai sovelluksen osan voidaan katsoa olevan tarpeeksi valmis, on järkevää tehdä siitä koottu kirjasto (JAR). Näin varmistutaan etteivät koodi tai tulokset pääse katoamaan, vaikkakin REPL tarjoaa samantapaisen muistin komentohistoriasta kuin perinteinen komentorivikin.

SELITYS JAR:ISTA JA EHKÄ JVM:STÄ?

3.1 Resilient Distributed Dataset (RDD)

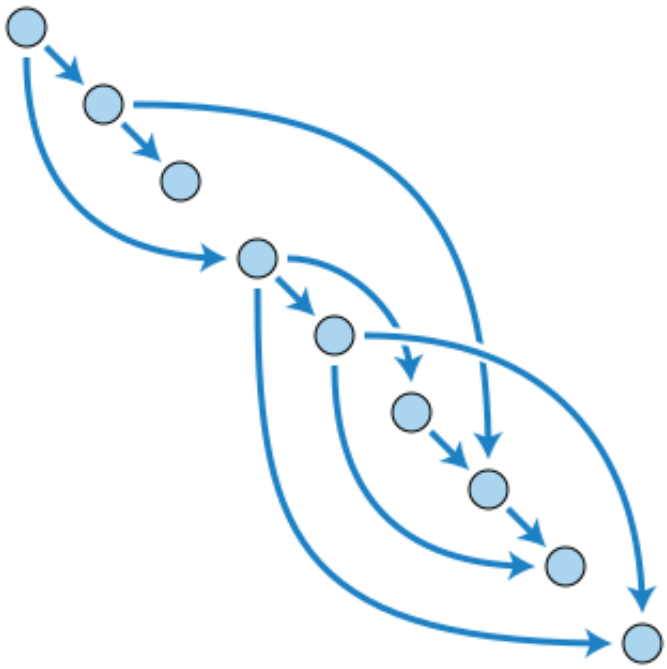
Resilient Distributed Dataset (RDD) on Sparkin tarjoama pääabstraktio. RDD on muuttumaton, osioitu elementtikokoelma joka voidaan hajauttaa klusterin useiden koneiden välillä. [16]

Tärkeä yksityiskohta ymmärtää RDD:stä on että ne ovat laiskasti evaluoituvia. Laiska evaluaatio (lazy evaluation) on evaluointi taktiikka, jossa lausekkeen evaluointia viivytetään siihen asti kun sen arvoa tarvitaan. Kun uusi RDD luodaan, mitään ei oikeasti vielä tapahdu. Spark tietää missä data sijaitsee tai miten data saadaan laskettua kun tulee aika tehdä sille jotain.

RDD voidaan luoda kahdella tavalla. Olemassaoleva Scala kokoelma voidaan rinnakkaistaa (parallelize). Toinen keino on viitata ulkoiseen aineistoon ulkoisessa varastointijärjestelmässä kuten HDFS:sä, HBase:ssa tai missä tahansa Hadoopin tuntemassa tiedostojärjestelmässä. [14]

RDD:t voidaan tallentaa muistiin, jolloin ohjelmistokehittäjä voi uudelleenkäyttää niitä tehokkaasti rinnakkaisissa operaatioissa. RDD:t voivat palautua solmuvirheistä automaattisesti käyttäen Directed Acyclic Graph (DAD) moottoria. DAG tukee syklistä datavirtaa. Jokaista Spark työtä kohti luodaan DAG klusterissa suoritettavan tehtävän tasoista. Verrattuna MapReduceen, joka luo DAGin kahdesta ennaltamäärätystä tilasta (Map ja Reduce), Sparkin luomat DAGit voivat sisältää minkä tahansa määrän tasoja. Tästä syystä jotkin työt voivat valmistua nopeammin kuin ne valmistuisivat MapReducessa. Yksinkertaisimmat työt voivat valmistua vain yhden tason jälkeen ja monimutkaisemmat tehtävät valmistuvat yhden monitasoisen ajon jälkeen, ilman että niitä täytyy pilkkoa useampiin töihin. [18]

Kuva 3.1 *Directed Acyclic Graph [5]*



3.2 Dataset API

Dataset (DS) on RDD:n korvaaja Sparkissa. DS on vahvasti tyypitetty kokoelma aluespesifisiä objekteja jotka voidaan muuntaa rinnakkain käyttäen funktionaalisia tai relaatio-operaatioita. Dataset:ille olemassa olevat operaatiot on jaettu muunnoksiin ja toimiin. Muunnokset ovat operaatioita, jotka luovat uusia Datasettejä, kuten map, filter, select, aggregate. Toimet ovat operaatioita jotka laukaisevat laskentaa ja palauttavat tuloksia. Toimia ovat esimerkiksi count, show tai datan kirjoittaminen tiedostojärjestelmään. [15]

Dataset-instanssit ovat laiskoja luonteeltaan, jolla tarkoitetaan sitä, että laskenta aloitetaan vasta kun toimintoa kutsutaan. Dataset on pohjimmiltaan looginen suunnitelma, jolla kuvataan datan tuottamiseen tarvittava laskenta. Toimea kutsuttaessa, Sparkin kyselyoptimoija (query optimizer) optimoi loogisen suunnitelman ja generoi fyysisen suunnitelman. Fyysinen suunnitelma takaa rinnakkaisesti ja hajautetusti tapahtuvan tehokkaan suorituksen. Loogista suunnitelmaa, kuten myös optimoitu fyysistä suunnitelmaa, voidaan tutkia käyttämällä DS:n *explain* funktiota. [15]

Domain-spesifisten olioiden tehokkaaseen tukemiseen tarvitaan enkooderia. Enkooderilla tarkoitetaan ohjelmaa, joka muuntaa tietoa jonkin algoritmin mukaisesti ja tässä tapauksessa sitä käytetään yhdistämään domain-spesifinen tyyppi T Sparkin sisäiseen tyyppijärjestelmään. Enkooderia voidaan käyttää esimerkiksi luokan *Person*, joka sisältää kentät nimi (merkkijono) ja ikä (kokonaisluku), kertomaan Sparkille generoi koodia ajon aikana joka serialisoi *Person* olion binäärirakenteeksi. Generoidulla binäärirakenteella on usein pienempi muistijalanjälki ja se on myös optimoitu tehokkaaseen dataprosessointiin. Datan binääriesitys voidaan tarkistaa käyttämällä DS:n tarjoamaa *schema* funktiota. [15]

Two ways typically exist to create a Dataset. The most common way is to make use of the read function provided by SparkSession and point Spark to some files on the storage system. Such as the following *json* file.

Dataset voidaan luoda tyypillisesti kahdella eri tavalla. Yleisin tapa on käyttää *SparkSession*:in tarjoamaa *read* funktiota ja osoittaa Spark joihinkin tiedostoihin tiedostojärjestelmässä, kuten seuraavaan *json* tiedostoon.

Program 3.1 *Esimerkki JSON tiedosto*

```
[{
  "name": "Matt",
  "salary": 5400
}, {
  "name": "George",
  "salary": 6000
}]
```

Dataset voidaan luoda myös tekemällä muutoksia olemassaoleville Dataset olioille:

Program 3.2 *Creating a new Dataset through a transformation*

```
val names = people.map(_.name)
```

Program 3.3 *Uuden Dataset olion luominen käyttäen read funktiota*

```
val people = spark.read.json("./people.json").as[Person]
```

jossa *Person* olisi Scala case-luokka, esimerkiksi:

Program 3.4 *case class Person*

```
case class Person(id: BigInt, firstName: String, lastName:
    String)
```

Case-luokat ovat tavallisia Scala-luokkia jotka ovat:

- Oletustarvoisesti muuttumattomia (immutable)
- Hajoitettavia (decomposable) hahmonsovitusta hyväksikäyttäen
- Vertailtavissa viitteiden sijasta rakenteellisen samankaltaisuuden mukaan
- Lyhyitä luoda (instantiate) ja käyttää

Mikäli tyyppimuunnos (casting) jätettäisiin tekemättä, päädyttäisiin luomaan DataFrame olio, jonka sisäinen mallin (schema) Spark pyrkisi arvaamaan. Tyyppimuunnos tehdään käyttämällä *as* avainsanaa.

Program 3.5 *SparkSession kontekstin luominen*

```
val spark = SparkSession
.builder
.appName("MovieLensALS")
.config("spark.executor.memory", "2g")
.getOrCreate()
```

SparkSession on Spark ohjelmoinnin aloituspiste, kun halutaan käyttää Dataset ja Dataframe rajapintoja. Ylläolevassa koodinpätkässä luodaan *SparkSession* ketjuttamalla rakentaja metodin kutsuja.

[15]

Dataset oliot ovat samankaltaisia kuin RDD:t, sillä nekin tarjoavat vahvan tyypityksen ja mahdollisuuden käyttää voimakkaita lambda-funktioita [17]. Lambda-funktioita avustaa Spark SQL:n optimoitu suoritusmoottori [17]. Perinteisen serialisoinnin, kuten Java serialisoinnin, sijaan, käytetään erikoistunutta enkooderia olioiden serialisointiin. Serialisaatiolla tarkoitetaan olion muuntamista tavuiksi, jolloin olion muistijalanjälki pienenee. Yleisesti serialisointia tarvitaan datan prosessointiin tai verkon yli lähettämiseen. Molempia, sekä enkoodereita ja serialisointia käytetään olioiden muuntamiseen tavuiksi, mutta koodi luo enkooderit dynaamisesti. Enkooderit käyttävät sellaista muotoa, että Spark kykenee suorittamaan monenlaisia operaatioita, kuten suodattamista, järjestämistä ja hajautusta (hashing), ilman että tavuja tarvitsee deserialisoida takaisin objektiksi. [14]

Seuraavassa koodilistauksessa luodaan uusi Datase lukemalla *json* tiedosto tiedostojärjestelmästä. Seuraavaksi luodaan uusi Dataset muunnoksen kautta. Objektin kloonaamiseksi käytetään case luokan *copy* metodia, koska *people* Dataset oli määritelty muuttumattomaksi. Lopuksi fyysinen suunnitelma tulostetaan konsoliin käyttämällä *explain* funktiota uudelle Dataset objektille.

Program 3.6 *Dataset olion loogisen ja fyysisen suunnitelman näyttäminen*

```
val people = spark.read.json("./people.json").as[Person]
val peopleWithDoubleSalary = people.map { person =>
        person.copy(salary = person.salary * 2)
}
peopleWithDoubleSalary.explain
```

```
== Physical Plan ==
*SerializeFromObject [staticinvoke(class org.apache.spark.
    unsafe.types.UTF8String, StringType, fromString,
    assertnotnull(input[0, $line62.$read$$iw$$iw$Person, true
    ], top level Product input object).name, true) AS name#
    212, staticinvoke(class org.apache.spark.sql.types.
    Decimal$, DecimalType(38,0), apply, assertnotnull(input
    [0, $line62.$read$$iw$$iw$Person, true], top level
    Product input object).salary, true) AS salary#213]
+- *MapElements <function1>, obj#211: $line62.
```

```
   $read$$iw$$iw$Person
+- *DeserializeToObject newInstance(class $line62.
   $read$$iw$$iw$Person), obj#210: $line62.
   $read$$iw$$iw$Person
+- *FileScan json [name#200,salary#201L] Batched: false,
   Format: JSON, Location: InMemoryFileIndex[file:/home/
   joonne/Documents/GitHub/thesis-code/people.json],
   PartitionFilters: [], PushedFilters: [], ReadSchema:
   struct<name:string,salary:bigint>
```

3.3 DataFrame API

DataFrame on pohjimmiltaan nimettyihin sarakkeisiin järjestetty Dataset. Se on käsitteellisesti yhtenevä relaatiotietokannan taulun tai R/Python kielten tietokehyksen (data frame) kanssa, mutta DataFrame omaa rikkaammat optimoinnit konepellin alla. DataFrame voidaan rakentaa useammalla tavalla, kuten esimerkiksi jäsennellyistä tiedostoista, Hive tauluista, ulkoisista tietokannoista tai olemassaolevista RDD olioista. DataFrame rajapinta on saatavilla Scala, Java, Python ja R -ohjelmointikielille. Scala rajapinnassa DataFrame on riveistä rakentuva Dataset, se on siis yksinkertaisesti tyyppialias Dataset[Row]. [14]

Program 3.7 *DataFrame luominen käyttäen read funktiota*

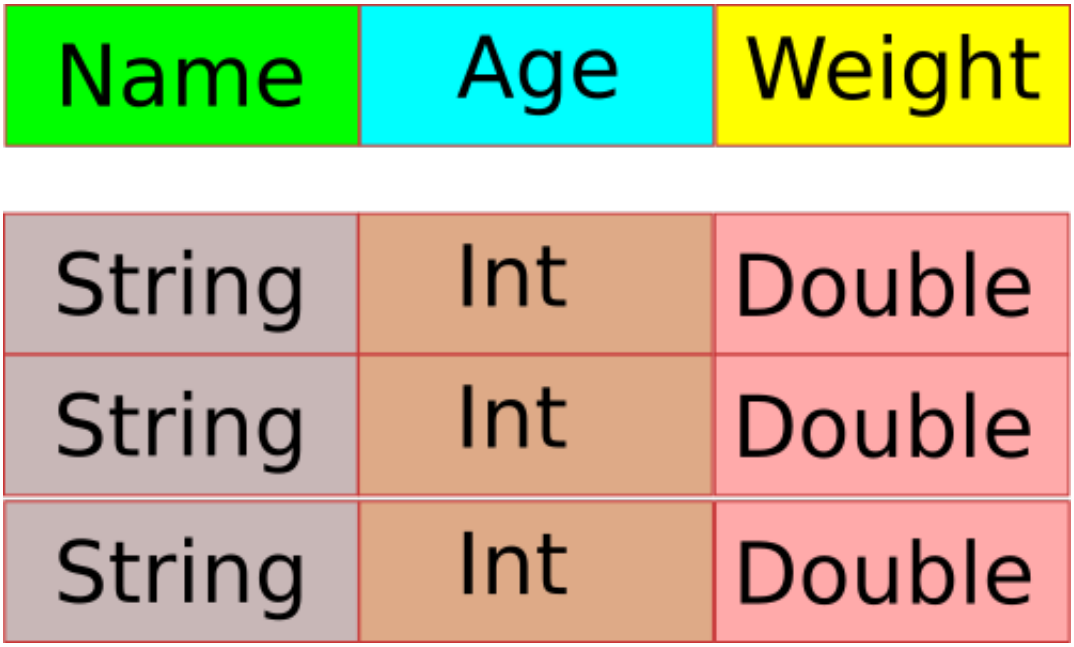
```
val people = spark.read.json("./people.json")
```

DataFrame objektia luotaessa, Spark arvaa luodun objektin sisäisen mallin.

3.4 Matriisin tekijöihinjako

Matriisin tekijöihinjako on toimi, jossa matriisi hajoitetaan matriisien tuloksi. Matriisi voidaan hajoittaa tekijöihinsä usealla eri tavalla. Seuraava kappale kuvailee matriisin tekijöihinjakoa yleisellä tasolla sekä vuorottelevien pienenmpien neliöiden (Alternating Least Squares, ALS) algoritmia. ALS on Sparkin toteuttama matriisin tekijöihinjako algoritmi ja se perustuu samalle ajatukselle Netflix prize kilpailun voittajan, matriisin tekijöihinjako mallin kanssa.

Kuva 3.2 *DataFrame*



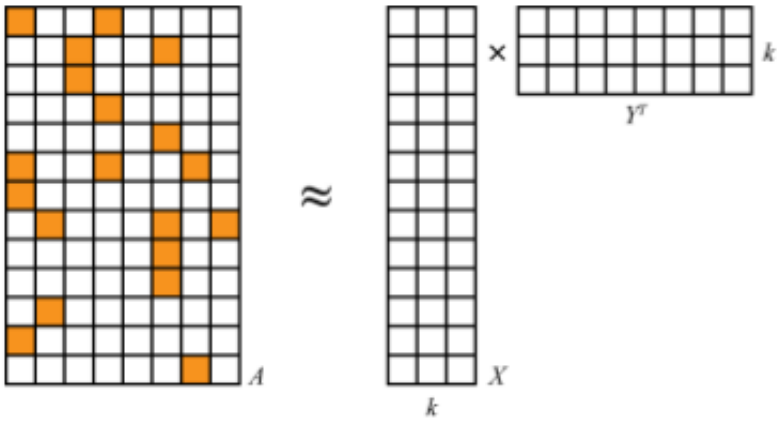
Matriisin tekijöihinjako kuuluu suureen algoritmien luokkaan nimeltä piilevien tekijöiden mallit (Latent-factor models). Piilevien tekijöiden mallit yrittävät selittää usean käyttäjän ja tuotteen välillä havaittuja vuorovaikutuksia käyttämällä suhteellisen pientä määrää havaitsemattomia, piileviä syitä. Voidaan esimerkiksi yrittää selittää miksi ihminen ostaisi tietyn albumin lukemattomien mahdolisuuksien joukosta kuvailemalla käyttäjiä ja tuotteita mieltymysten perusteella, joista ei ole mahdollista saada tietoa. [11] Piilevää tekijää ei ole mahdollista tarkastella sellaisenaan. Ihmisen terveys on esimerkki piilevästä tekijästä, sillä sitä ei ole mahdollista mitata kuten esimerkiksi verenpainetta.

Matriisin tekijöihinjako algoritmit käsittelevät käyttäjä- ja tuotetietoja suurena matriisina A. Jokainen rivissä i sekä sarakkeessa j sijaitseva kohta esittää arvostelua jonka käyttäjä on antanut tietylle tuotteelle. [11]

Yleensä A on harva (sparse), jolla tarkoitetaan että useimmat A:n alkiot sisältävät 0. Tämä johtuu siitä, että usein vain muutama käyttäjä-tuote kombinaatio on olemassa kaikista mahdollisuuksista.

Matriisin tekijöihinjako mallintaa A:n kahden pienemmän matriisin X ja Y tulona, jotka ovat varsin pieniä. Koska A:ssa on monta riviä ja saraketta, X ja Y sisältävät paljon rivejä mutta vain muutaman (k) sarakkeen. Nämä k saraketta vastaavat

Kuva 3.3 *Matrix factorization [11]*



piileviä tekijöitä joita käytetään kuvailemaan tiedossa sijaitsevia vuorovaikutuksia. Hajotelma (factorization) on ainoastaan arvio, sillä k on pieni. [11]

Tavanomainen lähestymistapa matriisin tekijöihinjakoon perustuvassa yhteisöllisessä suodatuksessa on kohdella käyttäjä-tuote matriisin alkioita käyttäjien antamina täsmällisinä arvosteluina. Eksplisiittistä tietoa on esimerkiksi käyttäjän antama arvio tuotteelle. Spark ALS kykenee käsittelemään sekä implisiittistä että eksplisiittistä tietoa. Implisiittistä tietoa on esimerkiksi sivujen katselukerrat tai tieto siitä, onko käyttäjä kuunnellut tiettyä artistia. [13] [11]

Usein monissa tosielämän käyttötapauksissa on käytettävissä ainoastaan implisiittistä tieto kuten katselukerrat, klikkaukset, ostos, tykkäykset tai jakamiset. Spark MLlib kohtelee tietoa numeroina jotka esittävät havaintojen vahvuutta kuten klikkausten määrä tai kumulatiivinen aika joka käytetään elokuvan katseluun, sen sijaan että mallinnettaisiin arviomatriisia suoraan. Ekplisiittisten arvioioiden sijaan, nämä numerot liittyvät havaittujen käyttäjämieltymysten varmuuteen. Tämän tiedon perusteella malli koettaa etsiä piileviä tekijöitä joiden avulla voidaan ennustaa käyttäjän odotettu arvio tuotteelle. [13]

Näihin algoritmeihin viitataan joskus matriisin täyttö algoritmeina. Tämä johtuu siitä että alkuperäinen matriisi A saattaa olla harva vaikka matriisitulo XY^T on tiheä. Vaikka tulosmatriisi sisältää arvon kaikille alkioille, se on kuitenkin vain arvio A:sta. [11]

3.4.1 Alternating Least Squares (ALS)

Yhteisöllistä suodatusta käytetään usein suosittelijajärjestelmissä. Nämä tekniikat pyrkivät täyttämään käyttäjä-tuote assosiaatiomatriisin puuttuvat kohdat. Spark MLlib tukee mallipohjaista yhteisösuodatusta, jossa käyttäjiä ja tuotteita kuvaillaan pienellä määrällä piileviä tekijöitä, joita voidaan käyttää puuttuvien kohtien ennustamiseen. Spark MLlib käyttää vaihtelevien pienimpien neliöiden (Alternating Least Squares, ALS) algoritmia näiden piilevien tekijöiden oppimiseen. [13]

Spark ALS yrittää arvioida arvostelumatriisin A kahden alemman arvon matriisin, X ja Y, tulona. [12]

$$A = XY^T \tag{3.1}$$

Tyypillisesti näihin arvioihin viitataan tekijämatriiseina. Perinteinen lähestymistapa on iteratiivinen. Jokaisen iteraation aikana, toista tekijämatriisia pidetään vakiona ja toinen ratkaistaan käyttäen pienimpien summien algoritmia. Juuri ratkaistua tekijämatriisia pidetään vuorostaan vakiona kun ratkaistaan toista tekijämatriisia. [12] Spark ALS mahdollistaa massiivisen rinnakkaistamisen sillä algoritmia voidaan suorittaa erikseen. Tämä on erinomainen ominaisuus laajamittaiselle (large-scale) laskenta-algoritmille. [11]

Spark ALS on lohkotettu versio ALS tekijöihinjako algoritmista. Ajatuksena on ryhmittää kaksi tekijäryhmää, *käyttäjät* ja *tuotteet*, lohkoihin. Ryhmittämistä seuraa kommunikaation vähentäminen lähettämällä jokaiseen tuotelohkoon vain yksi kopio jokaisesta käyttäjävektorista iteraation aikana. Vain ne käyttäjä vektorit lähetetään, joita tarvitaan tuotelohkoissa. Vähennetty kommunikaatio saavutetaan valmiiksi laskemalla joitain tietoja suositusmatriisista jotta voidaan päätellä jokaisen käyttäjän ulostulot ja jokaisen tuotteen sisääntulot. Ulostulolla tarkoitetaan niitä tuotelohkoja, joihin käyttäjä tulee myötävaikuttamaan. Sisääntulolla tarkoitetaan niitä ominaisuusvektoreita jotka jokainen tuote ottaa vastaan niiltä käyttäjälohkoilta joista ne ovat riippuvaisia. Tämä mahdollistaa sen, että voidaan lähettää vain taulukollisen ominaisuusvektoreita jokaisen käyttäjä- ja tuotelohkon välillä. Vastaavasti tuotelohko löytää käyttäjän arviot ja päivittää tuotteita näiden viestien perusteella. [12]

Sen sijaan että etsittäisiin, alemman tason arviot suositusmatriisille A, etsitäänkin

arviot mieltymysmatriisi P:lle, jossa P:n alkiot saavat arvon 1 kun $r > 0$ ja arvon 0 kun $r <= 0$. Eksplisiittisen tuotearvion sijaan arvostelut kuvaavat käyttäjän mieltymyksen vahvuuden luottamusarvoa. [12]

$$A_i Y (Y^T Y)^{-1} = X_i \tag{3.2}$$

ALS operoi kiinnittämällä yhden tuntemattomista u_i ja v_j ja vaihtelemalla tätä kiinnittämistä. Kun toinen on kiinnitetty, toinen voidaan laskea ratkaisemalla pienimpien neliöiden ongelma. Tämä lähestymistapa on hyödyllinen, koska se muuttaa aiemman, ei-konveksin, ongelman kvadraattiseksi, eli neliömäiseksi, jolloin se voidaan ratkaista optimaalisesti. [1] Alla on [1] mukainen yleinen kuvaus ALS algoritmista:

Program 3.8 *Vaihtelevien pienimpien neliöiden algoritmi (ALS) [1]*

1. Alusta matriisi V asettamalla ensimmäiseksi riviksi elokuvan keskimääräinen arvio ja pieni satunnaisluku jäljelläoleviin alkioihin.

2. Kiinnitä V, ratkaise U minimoimalla RMSE funktio.

3. Kiinnitä U, ratkaise V minimoimalla RMSE funktio.

4. Toista askeleita 2 ja 3 konvergenssiin asti.

RMSE (Root Mean Square Error) funktion minimoinnilla tarkoitetaan toimea, jossa

4. TOTEUTUS

GroupLens Research on kerännyt ja laittanut saataville arvioaineistoja MovieLens sivustolta. Aineistot on kerätty useiden aikajaksojen aikana, riippuen aineston koosta. MovieLens ml-latest-small aineisto sisältää 100 000 arviota, jotka ovat antaneet 700 käyttäjää 9000 elokuvalle. Näiden aineistojen haittapuolena on, että ne muuttuvat ajan myötä, eivätkä näin ollen ole sopivia tutkimustulosten raportointiin. Nykyinen, lokakuussa 2016 julkistettu versio on saatavilla projektin versionhallinnassa. Tämä aineisto valittiin, jottai voitaisiin antaa arvioita elokuville, jotka on oikeasti nähty ja myös laitteiston vuoksi. MovieLens ml-latest-small aineisto koostuu *movies.csv* and *ratings.csv* tiedostoista.

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

userId	movieId	rating	timestamp
1	31	2.5	1260759144
1	1029	3.0	1260759179
1	1061	3.0	1260759182
1	1129	2.0	1260759185
1	1172	4.0	1260759205
1	1263	2.0	1260759151
1	1287	2.0	1260759187
1	1293	2.0	1260759148
1	1339	3.5	1260759125

Toteutuksessa käytettiin RDD-pohjaista rajapintaa, sillä dataset-pohjainen rajapinta ei ole vielä täysin toiminnallinen yhteisöllisen suodatuksen tehtävissä. Aineiston lataaminen voidaan tehdä dataset rajapintaa hyödyntäen, mutta varsinaisen suositus täytyy tehdä RDD rajapintaa käyttäen. Dataset rajapinta tarjoaa useita parannuksia, kuten esimerkiksi yksinkertaisemman tiedon lataamisen.

4.1 MovieLensRecommendation.scala

Ensimmäinen askel itsenäisen spark sovelluksen rakentamisessa on tehdä oikeanlainen kansiorakenne ja tehdä $< PROJEKTI >$ *.sbt* niminen tiedosto, jossa kuvaillaan sovelluksen riippuvuudet. Itsenäinen spark sovellus tarkoittaa käyttövalmista *jar* tiedostoa (Java ARchive) joka voidaan jakaa spark klusterille ja se sisältää koodin ja kaikki riippuvuudet.

Sovelluksia voidaan ottaa käyttöön klusterissa spark-submit työkalun avulla. Spark-submit mahdollistaa Sparkin kaikkien tuettujen klusterinhoitajien käyttämisen yhteinäisen käyttöliittymän kautta, joten käyttäjän ei tarvitse määrittää sovellusta toimimaan erikseen kaikkien kanssa.

Program 4.1 *Kokoonpano jar-tiedoston tekeminen sbt työkalulla*

```
sbt package
```

Program 4.2 *Sovelluksen käyttöönotto klusterissa*

```
spark-submit --class "MovieLensALS" --master local[4] movielens-recom
```

Program 4.3 *Suositusten lataaminen RDD rajapintaa käyttäen*

```
val ratings = sc.textFile("ml-latest-small/ratings.csv")
  .filter(x => !isHeader("userId", x))
  .map { line =>
    val fields = line.split(",")
    (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt,
  }
```

Program 4.4 *Suositusten lataaminen dataset rajapintaa käyttäen*

```
val ratings = spark.read.csv("ml-latest-small/ratings.csv")
  .filter(arr => arr(0) != "userId")
  .map { fields =>
        Rating(fields(0).asInstanceOf[String].toInt, fields(1).asInst
  }
```

Program 4.5 *MovieLensALS.scala*

```
1  import org.apache.spark.mllib.recommendation._
2
3  object MovieLensALS {
4   def main(args: Array[String]) {
5
6   val conf = new SparkConf()
7    .setAppName("MovieLensALS")
8    .set("spark.executor.memory", "4g")
9   val sc = new SparkContext(conf)
10
11  /* load personal ratings */
12  val personalRatings = Source.fromFile("personalRatings.txt")
13   .getLines()
14   .map { line =>
15     val fields = line.split(",")
16     Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble)
17   }.toSeq
18
19  val personalRatingsRDD = sc.parallelize(personalRatings, 1)
20
21  /* load ratings and movie titles */
22
23  val ratings = sc.textFile("ml-latest-small/ratings.csv")
24   .filter(x => !isHeader("userId", x))
25   .map { line =>
26     val fields = line.split(",")
```

```
27      /* format: (timestamp % 10, Rating(userId, movieId, rating)) */
28      (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt,
            fields(2).toDouble))
29    }
30
31    val movies = sc.textFile("ml-latest-small/movies.csv")
32     .filter(x => !isHeader("movieId", x))
33     .map { line =>
34       val fields = line.split(",")
35       // format: (movieId, movieName)
36       (fields(0).toInt, fields(1))
37     }.collect().toMap
38
39    val numRatings = ratings.count
40    val numUsers = ratings.map(_._2.user).distinct.count
41    val numMovies = ratings.map(_._2.product).distinct.count
42
43    println(s"Got $numRatings ratings from $numUsers users on $numMovies
          movies.")
44
45    val numPartitions = 4
46    val training = ratings.filter(x => x._1 < 6)
47    .values
48    .union(personalRatingsRDD)
49    .repartition(numPartitions)
50    .cache()
51    val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
52    .values
53    .repartition(numPartitions)
54    .cache()
55    val test = ratings.filter(x => x._1 >= 8).values.cache()
56
57    val numTraining = training.count()
58    val numValidation = validation.count()
59    val numTest = test.count()
60
61    println(s"Training: $numTraining, validation: $numValidation, test:
          $numTest")
62
63    // TRAINING
64
65    val ranks = List(8, 12)
66    val lambdas = List(1.0, 10.0)
67    val numIters = List(10, 20)
```

```
68  var bestModel: Option[MatrixFactorizationModel] = None
69  var bestValidationRmse = Double.MaxValue
70  var bestRank = 0
71  var bestLambda = -1.0
72  var bestNumIter = -1
73  for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
74  val model = ALS.train(training, rank, numIter, lambda)
75  val validationRmse = computeRmse(model, validation, numValidation)
76  println(s"RMSE (validation) = $validationRmse for the model trained
        with rank = $rank, lambda = $lambda, and numIter = $numIter.")
77  if (validationRmse < bestValidationRmse) {
78  bestModel = Some(model)
79  bestValidationRmse = validationRmse
80  bestRank = rank
81  bestLambda = lambda
82  bestNumIter = numIter
83  }
84  }
85
86  val testRmse = computeRmse(bestModel.get, test, numTest)
87
88  println(s"The best model was trained with rank = $bestRank and lambda
       = $bestLambda and numIter = $bestNumIter and its RMSE on the test
        set is $testRmse .")
89
90  val myRatedMovieIds = personalRatings.map(_.product).toSet
91  val candidates = sc.parallelize(movies.keys.filter(!myRatedMovieIds.
        contains(_)).toSeq)
92  val recommendations = bestModel.get
93  .predict(candidates.map((0, _)))
94  .collect()
95  .sortBy(- _.rating)
96  .take(10)
97
98  var i = 1
99  println("Movies recommended for you:")
100 recommendations.foreach { r =>
101 println("%2d".format(i) + ": " + movies(r.product))
102 i += 1
103 }
104
105 // clean up
106 sc.stop()
107 }
```

```
108
109   def isHeader(headerId: String, line: String): Boolean = line.contains(
          headerId)
110
111   /** Compute RMSE (Root Mean Squared Error). */
112   def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n:
          Long): Double = {
113   val predictions: RDD[Rating] = model.predict(data.map(x => (x.user, x.
          product)))
114   val predictionsAndRatings = predictions.map(x => ((x.user, x.product),
          x.rating))
115   .join(data.map(x => ((x.user, x.product), x.rating)))
116   .values
117   math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)
          ).reduce(_ + _) / n)
118   }
119 }
```

Rivillä 1 tuodaan saataville kaikki recommendation paketin sisältämät kentät tai metodit käyttäen *import* avainsanaa. Rivillä 3 määritellään MovieLensALS niminen objekti. Objekti on nimetty instanssi joka sisältää jäseniä kuten kenttiä (field) sekä metodeita (method). Rivillä 4 on määritelty *main* funktio tarkoittaa sitä, että määritelty objekti *MovieLensALS* on ohjelman aloituspiste (entry point) sillä *main* funktio sisältää tietynlaisen allekirjoituksen eli tietynlaiset parametrit. Riveillä 6-9 luodaan *SparkConf* objekti, jonka avulla luodaan ohjelman käyttöön uusi *SparkContext* objekti. *SparkContext* objektin avulla päästään käsiksi Sparkin sisäisiin toiminnallisuuksiin. Riveillä 11-17 ladataan henkilökohtaiset suositukset tekstitiedostosta nimeltä *personalRatings.txt*, pilkotaan tiedoston rivit pilkun kohdalta ja luodaan uusia *Rating* objekteja yhtä monta, kuin tiedostossa on rivejä. Rivillä 19 ladatut suositukset muutetaan vielä RDD (Resilient Distributed Dataset) muotoiseksi käyttäen *sc.parallelize* funktiota. Funktiolle annettava toinen parametri tarkoittaa hajautuksen määrää, eli kuinka monelle solmulle klusterissa tiedosto halutaan hajauttaa.

BIBLIOGRAPHY

[1] C. Aberger, "Recommender: An analysis of collaborative filtering techniques," 2014. [Online]. Available: http://cs229.stanford.edu/proj2014/Christopher%20Aberger,%20Recommender.pdf

[2] C. C. Aggarwal, *Recommender Systems*. Springer International Publishing, 2016.

[3] BookLens. [Online]. Available: https://booklens.umn.edu/

[4] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370. [Online]. Available: http://dx.doi.org/10.1023/A:1021240730564

[5] D. Eppstein. Directed acyclic graph. [Online]. Available: https://en.wikipedia.org/wiki/Directed_acyclic_graph#/media/File:Topological_Ordering.svg

[6] S. K. Gorakala and M. Usuelli, *Building a Recommendation Engine with R*, 1st ed. Packt Publishing, 2015.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," 2009. [Online]. Available: https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[8] G. Linden, B. Smith, and J. York, "Amazon.com recommendations," *IEEE INTERNET COMPUTING*, pp. 76–79, 2003. [Online]. Available: http://www.cin.ufpe.br/~idal/rs/Amazon-Recommendations.pdf

[9] MovieLens. [Online]. Available: https://movielens.org/info/about

[10] F. Ricci, L. Rokach, B. Shapira, and P. B. Kanto, *Recommender Systems Handbook*, 1st ed. Springer, 2011.

[11] S. Ryza, U. Laserson, S. Owen, and J. Wills, *Advanced Analytics with Spark*. O'Reilly Media, Inc., 2015.

[12] Spark. (2014) ALS. [Online]. Available: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS

[13] ——. (2014) Collaborative filtering - rdd-based api. [Online]. Available: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

[14] ——. (2014) Spark programming guide. [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html

[15] ——. (2016) Dataset. [Online]. Available: https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Dataset.html

[16] ——. (2016) Rdd. [Online]. Available: https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.rdd.RDD

[17] ——. (2016) Spark sql programming guide. [Online]. Available: http://spark.apache.org/docs/latest/sql-programming-guide.html

[18] M. Technologies. Apache spark. [Online]. Available: https://mapr.com/products/product-overview/apache-spark/