



TAMPEREEN TEKNILLINEN YLIOPISTO  
TAMPERE UNIVERSITY OF TECHNOLOGY

**JONNE PETTERI PIHLANEN**  
**SUOSITTELIJAJÄRJESTELMÄN RAKENTAMINEN APACHE SPAR-**  
**KILLA**

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

Tampere,

Jonne Pihlanen

# SISÄLLYS

1. Johdanto . . . . .	1
2. Teoria . . . . .	3
2.1 Matriisin tekijöihinjako . . . . .	3
2.1.1 Alternating Least Squares (ALS) . . . . .	5
2.2 Amazon Web Services (AWS) . . . . .	7
2.2.1 Elastic Map Reduce (EMR) . . . . .	7
2.2.2 Simple Storage Service (S3) . . . . .	8
3. Suositelijajärjestelmät . . . . .	10
3.1 Suositustekniikat . . . . .	12
3.1.1 Muistiperustainen yhteisöllinen suodatus . . . . .	13
3.1.2 Mallipohjainen yhteisösuodatus . . . . .	16
4. Apache Spark . . . . .	17
4.1 Scala . . . . .	18
4.1.1 Perustyytit . . . . .	19
4.1.2 Muuttujat . . . . .	19
4.1.3 Funktiot . . . . .	20
4.2 Resilient Distributed Dataset (RDD) . . . . .	21
4.3 Dataset API . . . . .	23
4.4 DataFrame API . . . . .	27
5. Toteutus . . . . .	29
5.1 Opetusdata . . . . .	29
5.2 Projektin rakenne . . . . .	30
5.3 Opetusdatan lataaminen Spark sovellukseen . . . . .	31
5.4 Mallin opettaminen . . . . .	33

5.5	Ennustaminen . . . . .	34
5.6	Apufunktiot . . . . .	35
6.	Tulokset . . . . .	36
6.1	Sisääntulot . . . . .	36
6.2	Suosituksset . . . . .	37
7.	Yhteenveto . . . . .	38
7.1	Johtopäätökset . . . . .	38
7.2	Tulevaa työtä . . . . .	38
	Bibliography . . . . .	39

## LYHENTEET JA MERKINNÄT

Apache Spark	Sovelluskehys hajautettujen ohjelmien rakentamiseen
Scala	Ohjelmointikieli
AWS	Amazon Web Services, pilvipalvelualusta
EMR	Elastic Map Reduce
S3	Simple Storage Service
ALS	Alternating Least Squares
Latentti tekijä	Piilevä tekijä, vaikea tai mahdoton havainnoida
Sparse	Harva
Eksplisiittinen	Tarkasti määritelty
Implisiittinen	
MLlib	Apache Sparkin tarjoama koneoppimis-kirjasto
RMSE	Root Mean Square Error
MSE	Mean Square Error



# 1. JOHDANTO

Suosittelujärjestelmät ovat nykyisin jatkuvasti läsnä jokapäiväisessä elämässämme. Ne auttavat päätöksenteossa verkko-ostoksissa, suoratoistopalveluissa, sosiaalisessa mediassa tai yksinkertaisesti uutisten lukemisessa. Yksinkertaisin ja luonnollisin suosittelun muoto on ihmiseltä ihmiselle suosittelu. Ihmiset voivat kuitenkin tehokkaasti suositella vain niitä asioita, jotka ovat itse henkilökohtaisesti kokeneet. Tällöin suosittelijajärjestelmistä tulee hyödyllisiä, sillä ne voivat mahdollisesti tarjota suosituksia tuhansista tai jopa miljoonista erilaisista tuotteista.

Suosittelu voidaan jakaa kahteen pääkategoriaan: tuotepohjaiseen ja käyttäjäpohjaiseen. [2] Tuotepohjaisessa suosittelussa tarkoituksena on etsiä samankaltaisia tuotteita, sillä käyttäjän ajatellaan olevan mahdollisesti kiinnostunut samankaltaisista tuotteista myös tulevaisuudessa. Käyttäjäpohjaisessa suosittelussa käyttäjän ajatellaan olevan kiinnostunut tuotteista, joita samankaltaiset käyttäjät ovat ostaneet, joten siinä on tarkoituksena etsiä samankaltaisia käyttäjiä, jotta voidaan suositella näiden ostamia tuotteita.

Tämän työn päämääränä on tutustua Apache Spark sovelluskehikseen sekä Scala ohjelmointikieleen ja toteuttaa suosittelujärjestelmä näiden teknologioiden avulla. Toteutus tehdään toimimaan pilvipalvelussa, joten työssä tutustutaan myös kevyesti kahteen pilvipalvelujätti Amazonin AWS:n (Amazon Web Services) tarjoamaan palveluun: EMR (Elastic Map Reduce) sekä S3 (Simple Storage Service). EMR on hallittu klusterialusta, joka yksinkertaistaa big data -sovelluskehysten, kuten Apache Sparkin, käyttämistä AWS:n palveluissa. [3] S3 on tietovarasto, joka on suunniteltu helpottamaan pilvilaskentaa ja se tarjoaa yksinkertaisen rajapinnan tietovaraston hallintaan. [4]

Apache Spark on sovelluskehys, joka mahdollistaa hajautettujen ohjelmien rakentamisen. [10] Hajautetussa ohjelmassa suoritus voidaan jakaa useiden käsittelysolmujen kesken. Jotkin suositteluongelmat voidaan mallintaa hajautettuna ohjelmana, jossa kaksi matriisia, käyttäjät ja tuotteet, prosessoidaan iteratiivisella algoritmilla,

joka mahdollistaa ohjelman suorittamisen rinnakkain. [10]

Apache Spark on rakennettu Scala ohjelmointikielellä. [10] Scala on monikäyttöinen, moniparadigmainen ohjelmointikieli, joka tarjoaa tuen funktionaaliselle ohjelmoinnille sekä vahvan tyyppityksen. Työn käytännön osuus on toteutettu Scalaa käyttäen, joten lyhyt johdanto ohjelmointikieleen tarjotaan lukijalle.

Tämä työ on rakentuu seuraavista osista. Luvuissa kaksi ja kolme esitetään työn kannalta oleellinen teoriaosuus. Luvussa neljä keskustellaan Apache Sparkista, avoimen lähdekoodin järjestelmästä, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Luku viisi esittää toteutuksen suosittelijajärjestelmälle. Luvussa kuusi käydään läpi tulokset. Lopuksi luvussa seitsemän esitellään johtopäätökset.

## 2. TEORIA

Seuraava kappale kuvailee matriisin tekijöihinjakoa yleisellä tasolla sekä vuorottelevien pienempien neliöiden (Alternating Least Squares, ALS) algoritmia. Työssä tarkasteltava Spark-sovelluskehys sisältää toteutuksen ALS-algoritmille, joka on matriisin tekijöihinjako-algoritmi ja se perustuu samalle ajatukselle erään kuuluisan kilpailun voittajan, matriisin tekijöihinjako-mallin kanssa. [18]

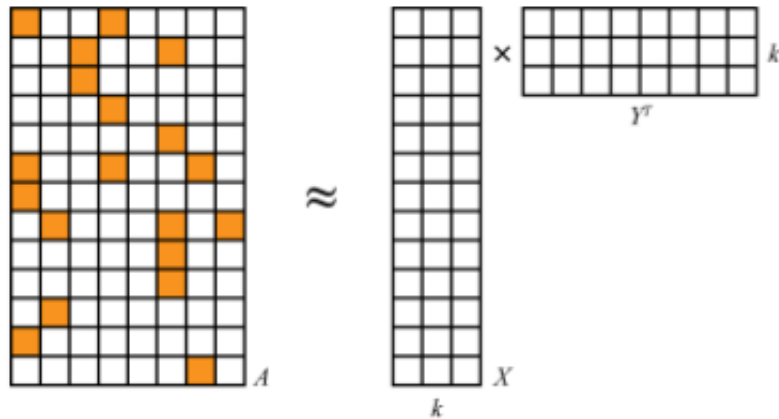
### 2.1 Matriisin tekijöihinjako

Matriisin tekijöihinjaossa matriisi hajoitetaan pienempien matriisien tuloksi. Matriisi voidaan hajoittaa tekijöihinsä usealla eri tavalla. Matriisin tekijöihinjako kuuluu suureen algoritmien luokkaan nimeltä latenttien tekijöiden mallit (Latent-factor models). Latenttien tekijöiden mallit yrittävät selittää usean käyttäjän ja tuotteen välillä havaittuja vuorovaikutuksia käyttämällä suhteellisen pientä määrää havaitsemattomia, latentteja tekijöitä. Voidaan esimerkiksi yrittää selittää miksi ihminen ostaisi tietyn albumin lukemattomien mahdollisuuksien joukosta kuvailemalla käyttäjiä ja tuotteita mieltymysten perusteella, joista ei ole mahdollista saada tietoa. [18] Latenttia tekijää ei ole mahdollista tarkastella sellaisenaan. Esimerkiksi ihmisen terveys on latentti tekijä, 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 alkio esittää arvostelua, jonka käyttäjä on antanut tietylle tuotteelle. [18]

Yleensä  $A$  on harva (sparse), jolla tarkoitetaan että useimmat  $A$ :n alkiot sisältävät arvon nolla. Tämä johtuu siitä, että kaikista mahdollisuuksista usein vain muutama käyttäjä-tuote-kombinaatio on olemassa. [18]

Matriisin tekijöihinjako mallintaa  $A$ :n kahden pienemmän matriisin  $X$  ja  $Y$  tulona,

**Kuva 2.1** Matriisin tekijöihinjako [18]

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 latentteja tekijöitä, joita käytetään kuvailemaan datassa sijaitsevia vuorovaikutuksia. Hajotelma (factorization) on ainoastaan arvio, sillä  $k$  on pieni. [18]

Tavanomainen lähestymistapa matriisin tekijöihinjakoon perustuvassa yhteisöllisessä suodatuksessa on kohdella käyttäjä-tuote matriisin alkioita käyttäjien antamina eksplisiittisinä 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. [21] [18]

Usein monissa tosielämän käyttötapauksissa on käytettävissä ainoastaan implisiittistä tietoa, kuten katselukerrat, klikkaukset, ostokset, 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. Eksplisiittisten arvioiden sijaan, nämä numerot liittyvät havaittujen käyttäjämieltyymysten varmuuteen. Tämän tiedon perusteella malli koettaa etsiä latentteja tekijöitä, joiden avulla voidaan ennustaa käyttäjän arvio tuotteelle. [21]

Näihin algoritmeihin viitataan joskus matriisin täyttö (matrix completion) -algoritmeina. Tämä johtuu siitä, että alkuperäinen matriisi  $A$  saattaa olla harva vaikka matriisitulo  $XY^T$  on tiheä. Vaikka tulomatriisi sisältää arvon kaikille alkioidelle, se on kuitenkin vain arvio  $A$ :sta. [18]

### 2.1.1 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 kuvailaan pienellä määrällä latentteja tekijöitä, joita voidaan käyttää puuttuvien kohtien ennustamiseen. Spark MLlib käyttää vuorottelevien pienimpien neliöiden (Alternating Least Squares, ALS) algoritmia näiden latenttien tekijöiden oppimiseen. [21]

Spark ALS yrittää arvata arvostelumatriisin  $A$  kahden alemman arvon matriisin,  $X$  ja  $Y$ , tulona. [20]

$$A = XY^T \quad (2.1)$$

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. [20] Spark ALS mahdollistaa massiivisen rinnakkaistamisen, sillä algoritmia voidaan suorittaa rinnakkain, toisistaan erillään. Tämä on erinomainen ominaisuus suuren mittakaavan (large-scale) laskenta-algoritmeille. [18]

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 taulukollinen ominaisuusvektoreita jokaisen käyttäjä- ja tuotelohkon välillä. Vastaavasti tuotelohko löytää käyttäjän arviot ja päivittää tuotteita näiden viestien perusteella. [20]

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  $r \leq 0$ . Eksplisiittisen tuotearvion sijaan arvostelut kuvaavat käyttäjän mieltymyksen vahvuuden luottamusarvoa. [20]

$$A_i Y (Y^T Y)^{-1} = X_i \quad (2.2)$$

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 neliömäiseksi, jolloin se voidaan ratkaista optimaalisesti. [1] Ei-konveksilla tarkoitetaan sellaista ongelmaa, jolla saattaa olla olemassa useita paikallisia ratkaisuja ja saattaa kestää kauan tunnistaa, onko ongelmalla ratkaisua lainkaan, tai että löydetty ratkaisu on myös globaali ratkaisu. [26] Alla on [1] mukainen yleinen kuvaus ALS algoritmista:

**Program 2.1** *ALS-algoritmin kuvaus [1]*

1. Alusta matriisi  $V$ .
2. Kiinnitä  $V$ , ratkaise  $U$  minimoimalla RMSE-funktio.
3. Kiinnitä  $U$ , ratkaise  $V$  minimoimalla RMSE-funktio.
4. Toista askeleita 2 ja 3 konvergenssiin asti.

Matriisi  $V$  alustetaan asettamalla ensimmäiseksi riviksi elokuvan keskimääräinen arvio ja pieni satunnaisluku jäljelläoleviin alkioihin. Konvergenssilla tarkoitetaan jonkin ilmiön lähestymistä ajan kuluessa joltain tiettyä arvoa, tässä tapauksessa sitä, että RMSE ei enää pienene tarpeeksi.

RMSE (Root Mean Square Error) on kenties suosituin ennustettujen arvosteluiden tarkkuuden evaluointiin käytetty metriikka. Sitä käytetään yleisesti regressioalgoritmien avulla luotujen mallien evaluointiin. Regressioalgoritmien yhteydessä virheellä tarkoitetaan havainnon todellisen sekä ennustetun numeroarvon välistä eroa. RMSE:n tuntemiseksi tulee tuntea ensin MSE (Mean Square Error). Kuten nimi viittaa, MSE on virheiden neliöiden keskiarvo ja se voidaan laskea neliöimällä jo-

kaisen havainnon virhe ja laskemalla virheiden neliöiden keskiarvo. RMSE voidaan puolestaan laskea ottamalla neliöjuuri MSE:stä. Sekä RMSE että MSE edustavat opetusvirhettä ja ne ilmoittavat kuinka hyvin malli sovittuu opetusdataan. Niiden avulla saadaan selville havaintojen sekä ennustettujen arvojen välinen poikkeavuus. Alhaisemman MSE:n tai RMSE:n omaavan mallin sanotaan sovittuvan paremmin opetusdataan kuin korkeammat virhearvot omaavan mallin. [10]

Suosittelujärjestelmä luo ennustettuja arvosteluita  $\hat{r}_{ui}$  testiaineistolle  $\tau$  käyttäjä-tuote pareja  $(u, i)$  joille todelliset arviot  $r$  tunnetaan. [10] Ennustettujen ja todellisten arvioiden välinen RMSE saadaan laskettua seuraavasti:

$$RMSE = \sqrt{\frac{1}{|\tau|} \sum_{(u,i) \in \tau} (\hat{r}_{ui} - r_{ui})^2} \quad (2.3)$$

## 2.2 Amazon Web Services (AWS)

AWS (Amazon Web Services) on Amazonin tarjoama kokoelma pilvilaskentaan (cloud computing) tarkoitettuja tai sitä avustavia palveluita. Tässä työssä käytetään pääasiassa hyödyksi kahta AWS:n palvelua, EMR:ää (Elastic Map Reduce) sekä S3:a (Simple Storage Service). EMR:n avulla on mahdollista käyttää Big Data sovelluskehysä, kuten Apache Sparkia. S3 on skaalautuva tietovarasto internetin tarpeisiin, jonka tarkoitus on helpottaa ohjelmistokehittäjien elämää. [5] [3] [4]

### 2.2.1 Elastic Map Reduce (EMR)

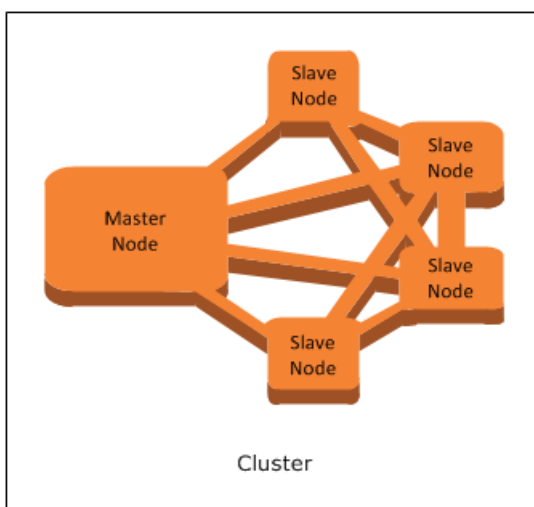
Amazon EMR tarjoaa hallitun klusterialustan (managed cluster platform), joka mahdollistaa suurten datamäärien prosessoinnin. EMR:ssä on mahdollista ajaa Apache Spark:ia ja se on kyvykäs liikuttelevaan suuria datamääriä AWS:n tietovarastoista, kuten S3:sta, ulos ja sisään. Dynaamisesti skaalautuvat Amazon EC2 (Elastic Compute Cloud) instanssit tekevät prosessoinnista helppoa, nopeaa ja kustannustehokasta. [3]

Klusteri on Amazon EMR:n keskeisin komponentti. Klusteri on kokoelma Amazon Elastic Compute Cloud (Amazon EC2) instansseja. Jokaista instanssia klusterissa kutsutaan solmuksi. Jokaisella solmulla on rooli eli tyyppi klusterin sisällä. Amazon

EMR asentaa erilaisia sovelluskomponentteja jokaiseen solmuun, antaen jokaiselle solmulle roolin hajautetussa sovelluksessa kuten Apache Spark. [3]

Amazon EMR:ssä on seuraavanlaisia solmuja: Master, Core ja Task. Master on solmu, joka tarkkailee tehtävien ja klusterin tilaa ajamalla ajaa sovelluskomponentteja, jotka ovat vastuussa datan ja tehtävien jakamisesta Slave-solmuille. Core on Slave-solmu, jossa sijaitsee sovelluskomponentteja, jotka ajavat tehtäviä ja tallentavat dataa HDFS:ään klusterissa. Task on Slave-solmu, jossa sijaitsee sovelluskomponentteja, jotka ajavat vain tehtäviä, Task-solmujen määrittäminen on vapaaehtoista. [3]

**Kuva 2.2** EMR-klusteri [3]



Yllä oleva kuva esittää klusterin, jossa on master-solmu sekä neljä slave-solmua.

### 2.2.2 Simple Storage Service (S3)

Amazon S3 (Simple Storage Service) on tietovarasto Internetin tarpeisiin. Se tarjoaa yksinkertaisen rajapinnan, jonka avulla voidaan tallettaa tai noutaa minkä verran dataa tahansa, milloin vain ja mistä tahansa webissä. Se on suunniteltu tekemään web-mittakaavan (web-scale) laskennasta yksinkertaisempaa kehittäjille. Se antaa kehittäjille pääsyn samoihin hyvin skaalautuvaan, luotettavaan, nopeaan ja edulliseen tietovarasto infrastruktuuriin, jota Amazon käyttää itse globaalin verkkosivustokatraansa ajamiseen. [4]

Tässä työssä S3:a käytetään varastoimaan opetusdataa. Tarvittaessa Apache Sparkin laskennan välivaiheita voidaan tallentaa muistiin jolloin niiden tulokset voitaisiin lukea suoraan ilman että niitä tarvitsee laskea uudelleen. Myös koko opetettu



malli voitaisiin tallentaa S3:een ja esimerkiksi ladata vaikka omalle koneelle, jolloin AWS:ää tulisi myös käytettyä optimaalisesti, sillä EC2 instanssien varaaminen aiheuttaa maksuja koko ajan.

### 3. SUOSITTELIJAJÄRJESTELMÄT

*Suosittelulla* tarkoitetaan tehtävää, jossa tuotteita suositellaan käyttäjille. Yksinkertaisin suosittelu tapahtuu ihmiseltä toiselle, 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 sadoista tai jopa tuhansista erilaisista tuotteista. Suosittelijajärjestelmät ovat joukko tekniikoita ja ohjelmistoja, jotka tarjoavat suosituksia mahdollisesti hyödyllisistä tuotteista. Tuotteella tarkoitetaan tässä yhteydessä yleistä asiaa, jota järjestelmä suosittelee henkilölle. Suosittelujärjestelmät rakennetaan yleensä suositteluun vain tietyn tyyppisiä tuotteita, kuten esimerkiksi kirjoja tai elokuvia. [17]

Suosittelijajärjestelmien tarkoitus on auttaa asiakkaita päätöksenteossa, tuotteiden määrän ollessa 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. Lista, joka sisältää 10 suosituinta tuotetta, on esimerkki räätälöimättömästä suosittelusta. Järjestäminen tehdään ennustamalla kaikista sopivimmat tuotteet käyttäjän mieltymysten tai vaatimusten perusteella. Tämän suorittamiseksi suosittelijajärjestelmän on kerättävä käyttäjältä tämän mieltymykset. Mieltymykset voivat olla suoraan käyttäjältä kysyttyjä tai käyttäjän antamia tuotearvioita tai ne voidaan tulkita käyttäjän toiminnasta kuten klikkauksista, sivun katselukerroista tai ajasta jonka käyttäjä on viipynyt tietyllä tuotesivulla. Suosittelijajärjestelmä voisi esimerkiksi tulkita tuotesivulle päätyksen todisteeksi mieltymyksestä sivun tuotteista. [17]

Suosittelijajärjestelmien kehitys alkoi melko yksinkertaisesta havainnosta: ihmiset tapaavat luottaa toisten ihmisten suosituksiin tehdessään rutiininomaisia päätöksiä. On esimerkiksi yleistä luottaa vertaispalautteeseen valittaessa kirjaa luettavaksi tai luottaa elokuvakriitikoiden kirjoittamiin arvioihin. Ensimmäinen suosittelija-

järjestelmä yritti matkia tätä käytöstä etsimällä yhteisöstä suosituksia aktiiviselle käyttäjälle. Suositukset haettiin käyttämällä algoritmeja. Tämä lähestymistapa on tyypiltään yhteisösuodattamista. Yhteisösuodattamisessa ideana on että, jos käyttäjät pitivät samankaltaisista tuotteista aikaisemmin, he luultavasti pitävät samoja tuotteita ostaneiden henkilöiden suosituksia merkityksellisinä. [17]

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. Vaihtoehdot ovat hyväksi, mutta vaihtoehtojen lisääntyminen on alkanut hyödyn tuottamisen sijaan heikentää kuluttajien hyvinvointia. [17]

Viimeaikoina suosittelijajärjestelmät ovat osoittautuneet tehokkaaksi lääkkeeksi tiedon *ylimääräongelmaa* vastaan. Suositelijajärjestelmät 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* (transactions). 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. [17]

Verrattuna klassisten tietojärjestelmien, kuten tietokantojen ja hakukoneiden, tutkimukseen, suosittelijajärjestelmien tutkimus on verrattain tuoretta. Suositelijajärjestelmistä tuli itsenäisiä tutkimusalueita 90-luvun puolivälissä. Viimeaikoina mielenkiinto suosittelujärjestelmiä kohtaan on kasvanut merkittävästi. Esimerkiksi verkkosivustoissa, kuten Amazon.com, YouTube, Netflix sekä IMDB, suosittelujärjestelmillä on iso rooli. Suosittelujärjestelmien tutkimiseen ja kehittämiseen omistettuja konferensseja on myös olemassa, kuten RecSys ja AI Communications (2008). [17]

Suosittelujärjestelmällä voidaan ajatella olevan kaksi päätarkoitusta: palveluntarjoajan avustaminen ja arvon tuottaminen palvelun käyttäjälle. Suosittelujärjestelmän on siis tasapainoiteltava sekä palveluntarjoajan että käyttäjän tarpeiden välillä. [17] Palveluntarjoaja voi esimerkiksi ottaa suosittelujärjestelmän avuksi parantamaan tai monipuolistamaan myyntiä, lisäämään käyttäjien tyytyväisyyttä, lisäämään käyttäjien uskollisuutta tai ymmärtämään paremmin mitä käyttäjä haluaa [17]. Käyttäjä

puolestaan saattaa haluta suosituksena tuotesarjan, apua selaamiseen tai mahdollistaa muihin vaikuttamisen. [17]

GroupLens, BookLens ja MovieLens olivat uranuurtajia suosittelujärjestelmien tutkimisessa ja kehittämisessä. GroupLens on tutkimuslaboratorio tietojenkäsittelytieteen ja tekniikan laitoksella Minnesotan Yliopistossa, joka on erikoistunut muun muassa suosittelujärjestelmiin ja verkkoyhteisöihin [2]. BookLens on GroupLensin rakentama kirjojen suosittelujärjestelmä [6]. MovieLens on GroupLensin ylläpitämä elokuvien suosittelujärjestelmä [14]. Uranuurtavan tutkimuksen lisäksi nämä sivustot julkaisivat aineistoja, joka ei ollut yleistä tuohon aikaan. [2]

### 3.1 Suositustekniikat

Suosittelujärjestelmällä täytyy olla ymmärrys tuotteista, jotta se pystyy tekemään suosituksia. Tämän mahdollistamiseksi järjestelmän täytyy pystyä ennustamaan tuotteen käytännöllisyys tai ainakin verrata tuotteiden hyödyllisyyttä ja tämän perusteella päättää suositeltavat tuotteet. Ennustamista voidaan luonnostella yksinkertaisella personoimattomalla suosittelualgoritmillä, 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. Suositettujen elokuvien voidaan siis katsoa olevan kohtuullisen osuvia suosituksia keskivertokäyttäjälle. [17]

Tuotteen  $i$  hyödyllisyyttä käyttäjälle  $u$  voidaan mallintaa reaaliarvoisella funktiolla  $R(u, i)$ , kuten yleensä tehdään *yhteisösuodatuksessa* ottamalla huomioon käyttäjien antamat arviot tuotteista. Yhteisösuodatuksessa suosittelijan perustehtävä on ennustaa  $R$ :n arvoa käyttäjä-tuote pareille ja laskea arvio todelliselle funktiolle  $R$ . Laskiessaan tätä ennustetta käyttäjälle  $u$  ja 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. [17]

Suosittelijajärjestelmät eroavat toisistaan kohdistetun toimialan, käytetyn tiedon ja erityisesti siinä kuinka suositukset tehdään, jolla tarkoitetaan suosittelualgoritmia [17]. Tässä työssä keskitytään vain yhteen suosittelutekniikoiden luokkaan, yhteisösuodatukseen, sillä tätä menetelmää käytetään Apache Sparkin MLlib kirjastossa.

Yhteisöllistä suodatusta käyttävät suosittelijajärjestelmät perustuvat käyttäjien yhteistyöhön. Niiden tavoitteena on tunnistaa malleja käyttäjän mielenkiinnoista voidakseen tehdä suunnattuja suosituksia [1]. Tämän lähestymistavan alkuperäisessä toteutuksessa suositellaan aktiiviselle käyttäjälle niitä tuotteita, joita muut samankaltaiset käyttäjät ovat pitäneet aiemmin [17]. Käyttäjä arvostelee tuotteita. Seuraavaksi algoritmi etsii suosituksia perustuen käyttäjiin, jotka ovat ostaneet samankaltaisia tuotteita tai perustuen tuotteisiin, jotka ovat eniten samanlaisia käyttäjän ostohistoriaan verrattuna. Yhteisösuodatus voidaan jakaa kahteen kategoriaan, jotka ovat *tuotepohjainen- ja käyttäjäpohjainen yhteisösuodatus*. Yhteisösuodatus on eniten käytetty ja toteutettu tekniikka suositusjärjestelmissä [9] [17] [7].

Yhteisöllinen suodatus analysoi käyttäjien välisiä suhteita ja tuotteiden välisiä riippuvuuksia tunnistaaakseen uusia käyttäjä-tuote -assosiaatioita [11]. Päätelmä siitä, että käyttäjät voisivat pitää samasta laulusta, koska molemmat kuuntelevat muita samankaltaisia lauluja on esimerkki yhteisöllisestä suodatksesta [18].

Koska yhteisöllisessä suodatuksessa suosittelu perustuu pelkästään käyttäjän arvosteluihin tuotteesta, yhteisöllinen suodatus kärsii ongelmista jotka tunnetaan nimillä *uusi käyttäjäongelma* ja *uusi tuoteongelma* [9]. 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 tyyppillisesti 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 kokomäärästä. [1]

### 3.1.1 Muistiperustainen yhteisöllinen suodatus

*Muistiperustaisissa menetelmissä* 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

Tuotepohjaisessa yhteisösuodatuksessa (Item-based collaborative filtering, IBCF) algoritmi aloittaa etsimällä samankaltaisia tuotteita käyttäjän ostohistoriasta [9]. Seuraavaksi mallinnetaan käyttäjän mieltymykset tuotteelle perustuen saman käyttäjän tekemiin arvosteluihin [17]. Alla oleva koodinpätkä esittelee tuotepohjaisen yhteisösuodatuksen idean jokaiselle uudelle käyttäjälle.

**Program 3.1** *Tuotepohjaisen yhteisösuodatuksen algoritmi [9]*

1. Jokaiselle kahdelle tuotteelle , mittaa kuinka samankaltaisia ne ovat sen suhteen, kuinka samankaltaisia arvioita ne ovat saaneet samankaltaisilta käyttäjiltä . Samankaltaisuutta voidaan arvioida esimerkiksi kosinimitan avulla.

```
case class Item(id: Int, feature1: String, feature2: String)
val items: Seq[Item] = Seq(Item(), Item(), Item())
case class Similarity(item1: Item, item2: Item, similarity : Double)

val similarItems: Seq[Similarity] = items.map { item1 =>
  items.map { item2 =>
    Similarity (item1, item2, cosineSimilarity (item1, item2))
  }
}
```

2. Tunnista k samankaltaisinta tuotetta jokaiselle tuotteelle

```
case class Similarities (item: Item, kMostSimilarItems: Seq[Similarity])

val similarities = findKMostSimilarItems(similarItems)
```

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

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

```
}
}
```

Amerikan suurimman verkkokaupan, Amazon.com:in, on aiemmin tiedetty käyttäneen 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. [12]

### Käyttäjäpohjainen yhteisösuodatus

Tuotepohjaisessa yhteisösuodatuksessa (User-based collaborative filtering, UBCF) algoritmi 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. [17]

Askeleet jokaiselle uudelle käyttäjälle käyttäjäpohjaisessa yhteisösuodatuksessa ovat:

#### ***Program 3.2** Käyttäjäpohjainen yhteisösuodatus algoritmi [9]*

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 kynnsarvon.

```
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)
```

### 3.1.2 Mallipohjainen yhteisösuodatus

Muistipohjaisen 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. [17] ALS-algoritmi on esimerkki mallipohjaisesta yhteisösuodatusalgoritmista.



## 4. APACHE SPARK

Apache Spark on avoimen lähdekoodin sovelluskehys, joka yhdistää hajautettujen ohjelmien kirjoittamiseen tarkoitetun järjestelmän sekä elegantin mallin ohjelmien kirjoittamiseen [18]. Spark tarjoaa korkean tason rajapinnat Java, Scala, Python sekä R ohjelmointikielille.

Jokainen Spark sovelus koostuu driver-ohjelmasta sekä yhdestä tai useammasta executor-ohjelmasta. Driver on ohjelma, joka ajaa käyttäjän pääohjelmaa ja suorittaa erilaisia rinnakkaisia operaatioita klusterissa. Executor on yksi kone klusterissa. [18]

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. [18]

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 levyille kun taas Spark kykenee välittämään tulokset suoraan putkiston (pipeline) seuraavalle vaiheelle. Toiseksi, Apache Spark:in voidaan ajatella kohtelevan 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 putkistossa ja tallentaa sen muistiin. Tämä tarkoittaa sitä, että tulevien askelten 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. [18]

Spark ohjelmia voidaan kirjoittaa Java, Scala, Python tai R-ohjelmointikielellä. Scalan käyttämisellä saavutetaan kuitenkin muutamia etuja, joita muut kielet eivät tarjoa. Tehokkuus saattaa parantua, sillä datan siirtäminen eri kerrosten välillä tai muunnosten suorittaminen datalle voi johtaa heikompaan tehokkuuteen. Spark on kirjoitettu Scala-ohjelmointikielellä, joten viimeisimmät ja parhaimmat ominaisuudet ovat aina käytössä, eikä niiden käännöstä tarvitse odotella. 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. Datatunnti, manipulointi ja koodin lähettäminen klustereihin hoituvat samalla kielellä. [18]

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. Kun REPLissä kehitetyn sovelluksen tai sovelluksen osan voidaan katsoa olevan tarpeeksi valmis, on järkevää tehdä siitä koottu kirjasto (JAR). Näin varmistutaan ettei ohjelmakoodia tai tuloksia pääse katoamaan, vaikkakin REPL tarjoaa samantapaisen muistin komentohistoriasta kuin perinteinen komentorivikin.

JAR eli Java ARchive on suosittuun ZIP tiedostoformaattiin perustuva alustariippumaton tiedostoformaatti, jota käytetään kokoamaan monta tiedostoa yhdeksi tiedostoksi. [15]

JVM (Java Virtual Machine, Java-virtuaalikone) on abstrakti laskentakone (computing machine). Kuten oikea laskentakone, se omaa käskykannan ja muokkaa useita muistialueita ajon aikana. JVM ei tiedä mitään ohjelmointikielistä, kuten Scala tai Java, vaan se operoi ainoastaan class-tiedostoilla, jotka ovat tietynlaisia binääritiedostoja. Class-tiedosto sisältää JVM käskyt sekä symbolitaulun. [16]

## 4.1 Scala

Scala on moniparadigmainen ohjelmointikieli, joka tukee sekä olio- että funktionaalista ohjelmointia. Funktionaalista ohjelmointia varten Scalasta löytyy tuki funktionaalisen ohjelmoinnin konsepteille kuten muuttumattomat tietorakenteet ja funktiot ensimmäisen luokan kansalaisina. Olio-ohjelmointia varten Scalasta löytyy tuki konsepteille kuten luokat, oliot ja piirre (trait). Scala tukee myös kapselointia, perintää, moniperintää ja muita tärkeitä olio-ohjelmoinnin konsepteja. Scala on staattisesti

tyypitetty kieli ja sillä kirjoitetut ohjelmat käännetään Scala-kääntäjää käyttäen. Scala on JVM-perustainen (Java Virtual Machine, Java-virtuaalikone) kieli, joten Scala kääntäjä kääntää sovelluksen Java-tavukoodiksi, joka voidaan ajaa missä tahansa Java-virtuaalikoneessa. Tavukooditasolla Scala ohjelmaa ei voida erottaa Java sovelluksesta. Scalan ollessa JVM-perustainen, Scala on täysin yhteensopiva Javan kanssa ja näin ollen Java-kirjastoja voidaan käyttää suoraan Scala-koodissa. Tästä syystä Scala-sovellukset hyötyvät suuresta Java-koodin määrästä. Vaikka Scala tukee sekä olio- että funktionaalista ohjelmointia, funktionaalista ohjelmointia suositetaan. [10]

### 4.1.1 Perustyytit

Scalan perustyytit numeroiden esittämiseen ovat Byte, Short, Int, Long, Float ja Double. Lisäksi Scalassa on perustyytit Char, String ja Boolean. Char on 16 bittinen etumerkitön Unicode merkki. String on jono Char:eja. Boolean esittää totuusarvoa tosi (true) tai epätosi (false). [10]

Javasta poiketen Scalassa ei ole ollenkaan primitiivisiä tyyppejä vaan jokainen tyyppi on toteutettu luokkana. Käännöksen aikana kääntäjä tarvittaessa automaattisesti muuntaa Scala tyytit Javan primitiivisiksi tyypeiksi. [10]

### 4.1.2 Muuttujat

Scalassa on kahdentyyppisiä muuttujia: muuttuvia ja vakioita. Muuttuva muuttuja määritellään avainsanan *var* avulla. Muuttuvaa muuttujaa ei voida asettaa uudelleen luomisen jälkeen. Var:ien käyttöä ei suositella, mutta joskus niiden käyttämisellä saadaan aikaan yksinkertaisempaa ohjelmakoodia ja tästä syystä Scala tukee myös muuttuvia muuttujia. [10]

Syntaksi *var*:in luomiseksi on

**Program 4.1** *Muuttuvan muuttujan luominen ja uudelleen asettaminen*

```
var x = 10  
x = 20
```

Muuttumatonta muuttujaa, *val*, ei sen sijaan voida antaa uudelleen luomisen jälkeen. Syntaksi *val*:in luomiseksi on

**Program 4.2** Vakion luominen

```
val y = 10
```

Mikäli vakiota yritetään uudelleenmäärittää myöhemmin ohjelmassa, kääntäjä antaa virheen. Huomionarvoista ylläolevassa syntaksissa on se, että Scala kääntäjä ei pakota määrittelemään muuttujan tyyppiä silloin kuin kääntäjä pystyy päättämään (type deduction) sen.

**Program 4.3** Muuttujan luominen tyyppimäärittelyn avulla

```
var x: Int = 10  
val y: Int = 10
```

### 4.1.3 Funktiot

Funktio on lohko suoritettavaa koodia joka palauttaa arvon. Se on konseptuaalisesti samankaltainen kuin matematiikassa: funktio ottaa sisääntulon ja palauttaa ulostulon. [10]

Scalan funktiot ovat ensimmäisen luokan kansalaisia, jolla tarkoitetaan, että funktiota voidaan:

- käyttää kuten muuttujaa
- antaa syötteenä toiselle funktiolle
- määritellä nimettömänä funktioliteraalina
- asettaa muuttujaan
- määritellä toisen funktion sisällä
- palauttaa toisen funktion ulostulona

[10]

Scalassa funktio määritellään avainsanalla *def*. Funktion määrittely aloitetaan funktion nimellä, jota seuraa sulkeissa olevat, pilkulla erotetut, parametrit tyyppimäärittelyineen. Parametrien jälkeen funktiomäärittelyyn tulee kaksoispiste, funktion

ulostulon tyyppi, yhtäsuuruusmerkki sekä funktion runko joko aaltosulkeissa tai ilman. [10]

**Program 4.4** *Funktio*

```
def add(first: Int, second: Int): Int = {  
    val sum = first + second  
    return sum  
}
```

Ylläolevassa esimerkissä funktion nimi on *add* ja se ottaa kaksi *Int* tyyppistä sisääntuloa. Funktio palauttaa *Int* tyyppisen arvon jonka se muodostaa lisäämällä annetut sisääntulot yhteen ja palauttamalla tuloksen.

Scala sallii myös lyhyemmän version samasta funktiosta:

**Program 4.5** *Funktio*

```
def add(first: Int, second: Int): Int = first + second
```

Toinen versio tekee täsmälleen saman asian kuin ensimmäinenkin, mutta se on vain kirjoitettu lyhyemmin. Paluuarvon tyyppi on jätetty antamatta, sillä kääntäjä pysyy päättämään sen koodista. Paluuarvon tyyppi suositellaan kuitenkin annettavan aina. Aaltosulkeet on myöskin jätetty pois, sillä ne ovat pakolliset vain kun funktion runko sisältää useamman kuin yhden käskyn. Lisäksi, *return* avainsana on ohitettu, sillä se on vapaaehtoinen. Scalassa kaikki lausekkeet ovat arvon palauttavia lausekkeitä, joten funktion rungon viimeisen lausekkeen arvosta tulee funktion paluuarvo. [10]

## 4.2 Resilient Distributed Dataset (RDD)

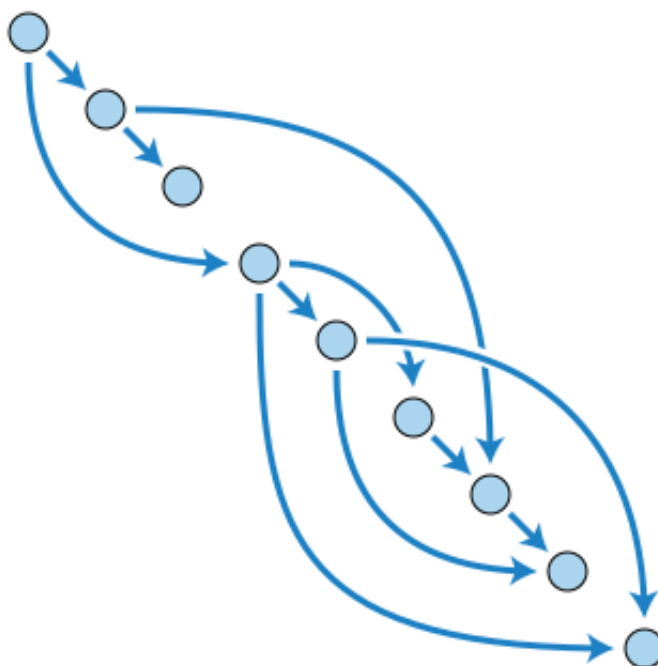
Resilient Distributed Dataset (RDD) on Sparkin tarjoama pääabstraktio. RDD on muuttumaton, partitioitu elementtikokoelma, joka voidaan hajauttaa klusterin useiden koneiden välillä. [24]

RDD:t ovat laiskasti evaluoituvia, jolla tarkoitetaan sitä, että lausekkeen evaluointia viivytetään siihen asti kun sen arvoa tarvitaan. Kun uusi RDD luodaan, mitään laskentaa ei oikeasti vielä tapahdu, vaan Spark tietää missä data sijaitsee tai miten data saadaan laskettua kun tulee aika tehdä sille jotain.

RDD voidaan luoda kahdella tavalla, rinnakkaistamalla (parallelize) tai viittaamalla ulkoiseen aineistoon. Rinnakkaistamisessa olemassaoleva Scala kokoelma voidaan rinnakkaistaa RDD:ksi. Ulkoiseen aineistoon viittaamisella tarkoitetaan viittaamista aineistoon ulkoisessa varastointijärjestelmässä kuten HDFS:sä, HBase:ssa tai missä tahansa Hadoopin tuntemassa tiedostojärjestelmässä. [22]

RDD:t voidaan tallentaa muistiin, jolloin ohjelmistokehittäjä voi uudelleenkäyttää niitä tehokkaasti rinnakkaisissa operaatioissa. RDD:t voivat palautua solmuvirheistä automaattisesti käyttäen Directed Acyclic Graph (DAG) moottoria. DAG tukee asyklistä datavirtaa, jolla tarkoitetaan sitä, että jokainen graafin kaari kulkee topologisessa järjestyksessä aiemmasta myöhempään. 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 MapReduceessa. TODO Yksinkertaisimmat työt voivat valmistua vain yhden tason jälkeen ja monimutkaisemmat tehtävät valmistuvat yhden monitasoisen ajon jälkeen, ilman että niitä täytyy pilkkoa useampiin töihin. [27]

**Kuva 4.1** Directed Acyclic Graph [8]



## 4.3 Dataset API

Dataset (DS) on vahvasti tyyppitetty kokoelma aluespesifisiä (domain specific) objekteja, jotka voidaan muuntaa rinnakkain käyttäen funktionaalisia tai relaatio-operaatioita. DS on RDD:n korvaaja Sparkissa. Dataset:ille olemassa olevat operaatiot on jaettu *muunnoksiin* (transformations) ja *toimiin* (actions). Muunnokset ovat operaatioita, jotka luovat uusia Dataset objekteja, kuten map, filter, select, aggregate. Toimet ovat operaatioita jotka suorittavat laskentaa ja palauttavat tuloksia. Toimia ovat esimerkiksi count, show tai datan kirjoittaminen tiedostojärjestelmään. [23]

Dataset-instanssit ovat laiskasti evaluoituvia, jolla tarkoitetaan sitä, että laskenta aloitetaan vasta kun toimintoa kutsutaan tai instanssin arvoa tarvitaan. 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 optimoitua fyysistä suunnitelmaa, voidaan tutkia käyttämällä DS:n *explain* funktiota. [23]

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. Esimerkiksi luokan *Person* tapauksessa, joka sisältää kentät nimi (merkkijono) ja ikä (kokonaisluku), enkooderia voidaan käyttää käskemään Sparkia luomaan koodia ajon aikana joka sarjallistaa *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. [23]

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

### *Program 4.6 Esimerkki JSON tiedosto*

```
[{
  "name": "Matt",
```

```

    "salary": 5400
  }, {
    "name": "George",
    "salary": 6000
  }]

```

Dataset voidaan luoda myös tekemällä muutoksia olemassaoleville Dataset olioille:

**Program 4.7** Uuden Dataset olion luominen muunnoksella (*transformation*)

```

val people: Dataset<Person> = Dataset(Person())
val names = people.map(_.name)

```

**Program 4.8** Uuden Dataset olion luominen käyttäen *read* funktiota

```

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

```

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

**Program 4.9** *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. *DataFrame* rajapintaa käsitellään seuraavassa aliluvussa. Tyyppimuunnos tehdään käyttämällä *as* avainsanaa.



**Program 4.10** *SparkSession kontekstin luominen*

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

SparkSession on Spark ohjelmoinnin lähtökohta, kun halutaan käyttää Dataset ja DataFrame rajapintoja. Ylläolevassa koodinpätkässä luodaan *SparkSession* ketjutamalla rakentajan kutsuja. [23]

Dataset oliot ovat samankaltaisia kuin RDD:t, sillä nekin tarjoavat vahvan tyyppityksen ja mahdollisuuden käyttää voimakkaita lambda-funktioita [25]. Lambda-funktiolla tarkoitetaan yleisesti anonymiä funktiota, jota ei olla sidottu muuttujaan. Perinteisen sarjallistamisen, kuten Java-sarjallistamisen, sijaan käytetään erikoistunutta enkooderia olioiden sarjallistamiseen. Serialisaatiolla tarkoitetaan olion muuntamista tavuiksi, jolloin olion muistijalanjälki pienenee. Yleisesti sarjallistamista tarvitaan datan prosessointiin tai verkon yli lähettämiseen. Molempia, sekä enkoodereita että sarjallistamista käytetään olioiden muuntamiseen tavuiksi, mutta enkooderit luodaan dynaamisesti koodissa. Enkooderit käyttävät sellaista muotoa, että Spark kykenee suorittamaan monenlaisia operaatioita, kuten suodattamista, järjestämistä ja hajautusta (hashing), ilman että tavuja tarvitsee purkaa takaisin objektiksi. [22]

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

**Program 4.11** *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(true)
```

*Program 4.12 Dataset olion looginen suunnitelma*

== Optimized Logical Plan ==

```
SerializeFromObject [staticinvoke(class org.apache.spark.
  unsafe.types.UTF8String, StringType, fromString,
  assertnotnull(input[0, $line32.$read$$iw$$iw$Person, true
], top level Product input object).name, true) AS name#
67, staticinvoke(class org.apache.spark.sql.types.
  Decimal$, DecimalType(38,0), apply, assertnotnull(input
[0, $line32.$read$$iw$$iw$Person, true], top level
Product input object).salary, true) AS salary#68]
+- MapElements <function1>, class $line32.
  $read$$iw$$iw$Person, [StructField(name,StringType,true),
  StructField(salary,DecimalType(38,0),true)], obj#66:
  $line32.$read$$iw$$iw$Person
+- DeserializeToObject newInstance(class $line32.
  $read$$iw$$iw$Person), obj#65: $line32.
  $read$$iw$$iw$Person
+- Relation [name#55,salary#56L] json
```

*Program 4.13 Dataset olion fyysinen suunnitelma*

== Physical Plan ==

```
*SerializeFromObject [staticinvoke(class org.apache.spark.
  unsafe.types.UTF8String, StringType, fromString,
  assertnotnull(input[0, $line32.$read$$iw$$iw$Person, true
], top level Product input object).name, true) AS name#
67, staticinvoke(class org.apache.spark.sql.types.
  Decimal$, DecimalType(38,0), apply, assertnotnull(input
[0, $line32.$read$$iw$$iw$Person, true], top level
Product input object).salary, true) AS salary#68]
+- *MapElements <function1>, obj#66: $line32.
  $read$$iw$$iw$Person
```

```

+ *DeserializeToObject newInstance(class $line32.
  $read$$iw$$iw$Person), obj#65: $line32.
  $read$$iw$$iw$Person
+ *FileScan json [name#55,salary#56L] Batched: false,
  Format: JSON, Location: InMemoryFileIndex[file:/home/
  joonne/Documents/GitHub/thesis-code/people.json],
  PartitionFilters: [], PushedFilters: [], ReadSchema:
  struct<name:string,salary:bigint>

```

## 4.4 DataFrame API

DataFrame on nimettyihin sarakkeisiin järjestetty Dataset. Se on käsitteellisesti yhtenevä relaatiotietokannan taulun tai R/Python kielten tietokehyksen (data frame) kanssa, mutta DataFrame on optimoitu tehokkaammin. DataFrame voidaan rakentaa useammalla tavalla, kuten esimerkiksi jäsennellyistä tiedostoista, ulkoisista tietokannoista tai olemassaolevista RDD-olioista. DataFrame-rajapinta on saatavilla Scala, Java, Python ja R-ohjelmointikielille. Scala-toteutuksessa DataFrame on riveistä rakentuva Dataset (*Dataset[Row]*). [22]

**Program 4.14** *DataFrame luominen käyttäen read-funktiota*

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

DataFrame-oliota luotaessa Spark arvaa luodun objektin sisäisen mallin.

*Kuva 4.2 DataFrame*

Name	Age	Weight
String	Int	Double
String	Int	Double
String	Int	Double

## 5. TOTEUTUS

Tässä luvussa esitetään työn toteutuksen oleelliset osat. Opetusdata, sen lataaminen ja siistiminen Sparkia varten. Projektin rakenne. Mallin opettaminen. Ennustusten kerääminen mallin avulla.

### 5.1 Opetusdata

GroupLens Research on kerännyt ja laittanut saataville vertailuaineistoja MovieLens -sivustolta. Aineistot on kerätty useiden aikajaksojen aikana, riippuen aineiston koosta. MovieLens 20M aineisto sisältää 20 000 000 (kaksikymmentä miljoonaa) arviota, jotka ovat antaneet 138 000 käyttäjää 27 000 elokuvalle. MovieLens 20M aineisto koostuu *movies.csv* and *ratings.csv* tiedostoista.

**Taulukko 5.1** Näyte *movies.csv* tiedostosta

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

**Taulukko 5.2** Näyte *ratings.csv* tiedostosta

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 ongelmille. Aineiston lataaminen voidaan tehdä dataset-rajapintaa hyödyntäen, mutta varsinainen suositus täytyy tehdä RDD-rajapintaa käyttäen. Dataset-rajapinta tarjoaa useita parannuksia, kuten esimerkiksi yksinkertaisemman tiedon lataamisen.

## 5.2 Projektin rakenne

Ensimmäinen askel itsenäisen Spark-sovelluksen rakentamisessa on tehdä oikeanlainen kansiorakenne ja luoda `< PROJEKTI > .sbt` niminen tiedosto, jossa kuvailaan sovelluksen riippuvuudet. Itsenäisellä Spark-sovelluksella tarkoitetaan käyttövalmista *JAR*-tiedostoa (Java ARchive), joka voidaan jakaa Spark-klusterille ja se sisältää sekä koodin että kaikki riippuvuudet. Tiedostomuoto *.sbt* viittaa SBT (Scala Build Tool) nimiseen ohjelmaan, joka on käännöstyökalu Scala, Java ja C++-kielille [19]. SBT:n avulla lähdekoodi saadaan sekä käännettyä että paketoitua JAR:iksi. Sovelluksia voidaan ottaa käyttöön klusterissa *spark-submit* työkalun avulla, joka mahdollistaa Sparkin kaikkien tuettujen klusterinhoitajien käyttämisen yhteinäisen käyttöliittymän kautta. Tämä ominaisuus osoittautui erittäin hyödylliseksi kun sovellusta ajettiin EMR-klusterissa, sillä *spark-submit* työkalu otti parametrinaan vain käännetyt JAR:in ja alkoi tekemään laskentaa. Tässä työssä "klusteri" tulee sisältämään vain master noodin sekä yhden worker noodin, mutta periaatteessa kyseessä on kuitenkin klusteri, vain erittäin pieni sellainen.

### *Program 5.1 Sovelluksen paketointi sbt työkalulla*

```
sbt package
```

### *Program 5.2 Sovelluksen käyttöönotto klusterissa*

```
spark-submit movieLens-recommendations_2.11-1.0.jar
```

Alla olevassa esimerkissä 4.3 ladataan työssä käytetyt suositukset RDD rajapintaa käyttäen.

**Program 5.3** *Suosituksen lataaminen RDD rajapintaa käyttäen*

```

1  val ratings = sc.textFile("ml-latest-small/ratings.csv")
2    .filter(arr => arr(0) != "userId")
3    .map { line =>
4      val fields = line.split(",")
5      val timestamp = fields(3).toLong % 10
6      val userId = fields(0).toInt
7      val movieId = fields(1).toInt
8      val rating = fields(2).toDouble
9
10     (timestamp, Rating(userId, movieId, rating))
11   }

```

Alla olevassa esimerkissä 4.4 ladataan työssä käytetyt suositukset Dataset-rajapintaa käyttäen.

**Program 5.4** *Suosituksen lataaminen Dataset rajapintaa käyttäen*

```

1  val ratings = spark.read.csv("ml-latest-small/ratings.csv")
2    .filter(arr => arr(0) != "userId")
3    .map { fields =>
4      val userId = fields(0).asInstanceOf[String].toInt
5      val movieId = fields(1).asInstanceOf[String].toInt
6      val rating = fields(2).asInstanceOf[String].toFloat
7      val timestamp = fields(3).asInstanceOf[String].toDouble % 10
8
9      Rating(userId, movieId, rating, timestamp)
10   }

```

Alla olevassa listauksessa on esitetty toteutetun suosittelujärjestelmän implementaatio.

- opetus - parametrien arvoille selitys ja että paperin mukaan - ennustaminen - apufunktiot ja selitys jos tarpeen

## 5.3 Opetusdatan lataaminen Spark sovellukseen

Alla olevassa koodilistauksessa 5.5 on esitetty opetusdatan lataaminen S3:sta.

**Program 5.5** Aineiston lataaminen

```

1 // load personal ratings
2 val personalRatings = sc.textFile("s3n://bucket/personalRatings.txt")
3   .map { line =>
4     val fields = line.split(",")
5     Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble)
6   }.filter(_.rating > 0.0)
7
8 // load ratings
9 val ratings = sc.textFile("s3n://bucket/ratings.csv")
10  .filter(!isHeader("userId"))(_)
11  .map { line =>
12    val fields = line.split(",")
13    val timestamp = fields(3).toLong % 10
14    val userId = fields(0).toInt
15    val movieId = fields(1).toInt
16    val rating = fields(2).toDouble
17
18    (timestamp, Rating(userId, movieId, rating))
19  }
20
21 // load movies
22 val movies = sc.textFile("s3n://bucket/movies.csv")
23  .filter(!isHeader("movieId"))(_)
24  .map { line =>
25    val fields = line.split(",")
26    (fields(0).toInt, fields(1))
27  }.collect().toMap

```

Riveillä 2-6 luodaan RDD *personalRatings* lataamalla henkilökohtaiset suositukset tekstitiedostosta nimeltä *personalRatings.txt*, pilkotaan tiedoston rivit pilkun kohdalta ja luodaan uusia *Rating* -objekteja yhtä monta, kuin tiedostossa on rivejä. Riveillä 22-36 luodaan RDD:t *ratings* ja *movies* lataamalla kaksi erillistä csv-tiedostoa. Tiedostoista suodatetaan ensin pois otsikkorivit ja tämän jälkeen tiedosto käydään läpi rivi kerrallaan ja pätkitään pilkulla erotetut arvot taulukkoon käyttäen Scalan String luokan sisäänrakennettua *split* funktiota. Tämän jälkeen taulukossa olevista arvoista muodostetaan pareja (tuple). Huomionarvoista on se, kuinka tiedostoihin voidaan viitata suoraan S3:n tiedoston nimellä ja Spark osaa hakea tiedostot suoraan S3 bucketista. Opetusdataa ei juuri tarvinnut siistiä, sillä opetukseen käytettiin valmista, hyvin jäsenneltyä datasettiä.



## 5.4 Mallin opettaminen

Alla olevassa koodilistauksessa 5.6 on esitetty mallin opettaminen.

*Program 5.6 Mallin opettaminen*

```

1  val numPartitions = 4
2  val training = ratings.filter(x => x._1 < 6)
3    .values
4    .union(personalRatings)
5    .repartition(numPartitions)
6    .cache()
7  val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
8    .values
9    .repartition(numPartitions)
10   .cache()
11  val test = ratings.filter(x => x._1 >= 8).values.cache()
12
13  val ranks = List(8, 12)
14  val lambdas = List(1.0, 10.0)
15  val numIters = List(10, 20)
16  var bestModel: Option[MatrixFactorizationModel] = None
17  var bestValidationRmse = Double.MaxValue
18  var bestRank = 0
19  var bestLambda = -1.0
20  var bestNumIter = -1
21  for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
22    val model = ALS.train(training, rank, numIter, lambda)
23    val validationRmse =
24      computeRmse(model, validation, numValidation)
25
26    if (validationRmse < bestValidationRmse) {
27      bestModel = Some(model)
28      bestValidationRmse = validationRmse
29      bestRank = rank
30      bestLambda = lambda
31      bestNumIter = numIter
32    }
33  }

```

Riveillä 2-11 valmistellaan opetus-, validaatio- sekä testidatat. Opetusdatan osuus koko aineistosta on 60-, validaatioidatan 20- ja testidatan 20-prosenttia. Rivillä 4 opetusdataan lisätään omat henkilökohtaiset arvostelut käyttäen RDD:n union funktiota, joka yhdistää kaksi erillistä RDD:tä toisiinsa. Riveillä 21-33 suoritetaan varsinainen mallin opetus. Opetus tapahtuu niin, että opetetaan muutama versio mallista

ja valitaan malleista paras käyttäen RMSE-metriikkaa. Koodin tasolla opetus tapahtuu käyttäen MLib / ALS kirjaston funktiota *train*, joka ottaa sisääntulonaan *ratings*, *rank*, *iterations* sekä *lambda* parametrit:

- *ratings* on RDD Rating olioita, jotka sisältävät käyttäjän tunnisteen, elokuvan tunnisteen ja suosituksen
- *rank* on piilevien ominaisuuksien sisällytettävä määrä
- *iterations* on ALS algoritmin iteraatioiden määrä
- *lambda* on regularisaatio-parametri, jolla yritetään ehkäistä mallin ylioppimista

Eräässä tutkimuksessa [13] tutkittiin parhaita parametreja ALS-algoritmillemme ja päädyttiin lambda-arvoon 0.1 ja iteraatioiden määrään 20. Parhautta oltiin tutkittu RMSE metriikan kautta ja kyseisillä parametreilla RMSE saatiin pienimmillään eli mallin voidaan sanoa sovituvan parhaiten opetusdataan. Tutkimuksessa oltiin päädyttiin arvoon XXX, kun taas paras itse opetettu malli päätyi RMSE arvoon 0.8071669083077276, jolloin ne ovat hyvin lähellä toisiaan.

[13]

## 5.5 Ennustaminen

Alla olevassa koodilistauksessa 5.7 on esitetty suositusten ennustaminen.

### *Program 5.7 Suositusten ennustaminen*

```

1
2 val myRatedMovieIds = personalRatings.map(_ . product) . toSet
3 val candidates = sc.parallelize(
4     movies.keys.filter(!myRatedMovieIds.contains(_)) . toSeq
5 )
6 val recommendations = bestModel.get
7     .predict(candidates.map((0, _)))
8     .collect()
9     .sortBy(_ . rating)
10    .take(10)
11
12 var i = 1
13 println("Movies recommended for you:")
14 recommendations.foreach { r =>
15     println("%2d".format(i) + ": " + movies(r.product))

```

```

16     i += 1
17 }

```

Yllä olevassa koodilistauksessa haetaan henkilökohtaiset suositukset käyttämällä mallin *predict* metodia, joka ottaa parametrinaan mahdollisten elokuvien joukon. Mahdollisilla elokuvilla tarkoitetaan elokuvia joita käyttäjä ei ole vielä nähnyt, eli ne eivät sisälly *personalRatings* muuttujan sisältämiin elokuviin.

## 5.6 Apufunktiot

Alla olevassa koodilistauksessa 5.8 esitetään käytetyt apufunktiot.

### *Program 5.8 Apufunktiot*

```

1 def computeRmse(
2     model: MatrixFactorizationModel,
3     data: RDD[Rating],
4     n: Long
5 ): Double = {
6     val predictions: RDD[Rating] =
7         model.predict(data.map(x => (x.user, x.product)))
8     val predictionsAndRatings =
9         predictions.map(x => ((x.user, x.product), x.rating))
10        .join(data.map(x => ((x.user, x.product), x.rating)))
11        .values
12
13    math.sqrt(
14        predictionsAndRatings
15            .map(x => (x._1 - x._2) * (x._1 - x._2))
16            .reduce(_ + _) / n
17    )
18 }

```

Yllä olevassa koodilistauksessa esitetään apufunktio *computeRMSE*, jonka avulla evaluoidaan opetetun mallin virhettä.

## 6. TULOKSET

Tässä kappaleessa käsitellään työn tärkeimpiä tuloksia.

### 6.1 Sisääntulot

Tässä osassa esitetään suosittelevajärjestelmän sisääntulot.

***Taulukko 6.1** Arvostellut elokuvat*

Tunniste	Nimi	Arvostelu
112897	The Expendables 3 (2014)	4.0
116887	Exodus: Gods and Kings (2014)	4.0
117529	Jurassic World (2015)	4.0
118696	The Hobbit: The Battle of the Five Armies (2014)	4.5
128520	The Wedding Ringer (2015)	4.5
122882	Mad Max: Fury Road (2015)	4.0
122886	Star Wars: Episode VII - The Force Awakens (2015)	4.5
131013	Get Hard (2015)	4.0
132796	San Andreas (2015)	3.0
136305	Sharknado 3: Oh Hell No! (2015)	1.0
136598	Vacation (2015)	4.0
137595	Magic Mike XXL (2015)	1.0
138208	The Walk (2015)	2.0
140523	The Visit (2015)	3.5
146656	Creed (2015)	4.0
148626	The Big Short (2015)	4.5
149532	Marco Polo: One Hundred Eyes (2015)	4.5
150548	Sherlock: The Abominable Bride (2016)	4.5
156609	Neighbors 2: Sorority Rising (2016)	3.5
159093	Now You See Me 2 (2016)	4.0
160271	Central Intelligence (2016)	4.0

Taulukossa 5.1 on esitetty suosittelujärjestelmän sisääntulona annetut, aiemmin nähdyt elokuvat. Sisääntulon rakenne on seuraava: sarakkeessa yksi sijaitsee elokuvan tunniste, sarakkeeseen kaksi on sijoitettu elokuvan nimi ja sarakkeessa kolme sijaitsee elokuvalla annettu arvio asteikolla 0-5. Taulukossa nähtävät arvot ovat vain pieni osa kaikesta opetukseen käytetystä aineistosta.

## 6.2 Suositukset

Tässä osassa käsitellään suosittelujärjestelmän tarjoamat suositukset, eli työn varsinaiset tulokset.

**Taulukko 6.2** Toteutetun järjestelmän suosittelat elokuvat

Numero	Nimi	Tyylilajit
1	The War at Home (1979)	Drama
2	Pearl Jam: Imagine in Cornice (2007)	Documentary, Music
3	Octopus (2000)	Adventure, Horror
4	My Brother Tom (2001)	Drama
5	Return to the 36th Chamber (1980)	Action, Comedy
6	Bob Funk (2009)	Comedy
7	Hamoun (1990)	Drama
8	Notebook (2006)	Drama, Musical, Romance
9	Patton Oswalt: My Weakness Is Strong (2009)	Documentary, Comedy
10	Deathstalker II (1987)	Adventure, Comedy, Fantasy

Taulukossa 5.2 on suosittelujärjestelmältä saaduista suosituksista 10 ensimmäistä, mukaan on lisätty myös elokuvien tyylilajit, jotta on helpompi arvioida suosituksien paikkansapitävyyttä.

Tulokset saadaan kysymällä suosittelijajärjestelmältä suosituksia tietyn käyttäjän ID:lle. Järjestelmän tulosjoukosta suodatetaan pois ne elokuvat, jotka on nähty, sillä näille suosituksen pitäisi olla erittäin suuri. Tuloksissa ei ole havaittavissa varsinaisia puutoksia. Tuloksien paikkansapitävyyttä voidaan arvioida esimerkiksi tyylilajien perusteella. Elokuvan hyvyys on hyvinkin henkilökohtainen kokemus, eikä siihen oteta kantaa tässä työssä. Tulokset täyttävät ainakin yhden kriteerin suosittelujärjestelmälle: yllätyksellisyyden, ainakaan vanhojen elokuvien tutkiminen ei koskaan oikein kiinnosta joten siitä plussaa....

## 7. YHTEENVETO

Tässä kappaleessa esitetään yhteenveto.

### 7.1 Johtopäätökset

Suosittelujärjestelmän rakentamiseen on olemassa monia mahdollisia toteutusvaihtoehtoja, kuten SQL ja Elasticsearch. Apache Spark vaikutti mielenkiintoiselta opiskelukohteelta ja tulevaisuuden kannalta hyödylliseltä. Scala ohjelmoinnin oppiminen vaikutti myöskin teknologian valintaan. AWS-palveluiden tuntemus on varsin hyödyllinen taito kaikinensa, ja tässä työssä pieni palanen tuli opittua sieltäkin maailmasta lisää. Olemassaolevien suosittelujärjestelmien tai analytiikkajärjestelmien evaluointi tulisi suorittaa ennen suosittelujärjestelmän valintaa. Suuremman data-setin käyttäminen, sekä isomman arvostelumäärän tarjoaminen järjestelmälle voisi parantaa tuloksia.

### 7.2 Tulevaa työtä

Mllib kirjastoa voitaisiin tutkia uudestaan siinä vaiheessa, kun Dataset rajapintaa voidaan käyttää Mllib:n kanssa. Toteutusta yritettiin myös Dataset rajapintaa hyväksikäyttäen, mutta kaikki toiminnallisuudet eivät olleet vielä käytössä.

Yhteisöllistä suodatusta voisi tietenkin käyttää muuhunkin tarkoitukseen, ja olisi mielenkiintoista tutkia myös jotain muuta ongelmaa ja soveltaa siihen ALS algoritmia.

## 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] Amazon. Amazon emr (elastic map reduce). [Online]. Available: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html>
- [4] ——. (2018) Amazon s3 (simple storage service). [Online]. Available: <https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html>
- [5] ——. (2018) AWS (Amazon Web Services). [Online]. Available: <https://aws.amazon.com/>
- [6] BookLens. [Online]. Available: <https://booklens.umn.edu/>
- [7] 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>
- [8] D. Eppstein. Directed acyclic graph. [Online]. Available: [https://en.wikipedia.org/wiki/Directed\\_acyclic\\_graph#/media/File:Topological\\_Ordering.svg](https://en.wikipedia.org/wiki/Directed_acyclic_graph#/media/File:Topological_Ordering.svg)
- [9] S. K. Gorakala and M. Usulli, *Building a Recommendation Engine with R*, 1st ed. Packt Publishing, 2015.
- [10] M. Guller, *Big Data Analytics with Spark*, 1st ed. Apress, 2015.
- [11] 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](https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf)
- [12] 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>

- [13] G. Miryala, R. Gomes, and K. R. Dayananda. Comparative analysis of movie recommendation system using collaborative filtering in spark engine. [Online]. Available: <http://www.jgrcs.info/index.php/jgrcs/article/view/1015/644>
- [14] MovieLens. [Online]. Available: <https://movielens.org/info/about>
- [15] Oracle. Jar file overview. [Online]. Available: <https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html>
- [16] ——. The java virtual machine specification. [Online]. Available: <https://docs.oracle.com/javase/specs/jvms/se10/html/index.html>
- [17] F. Ricci, L. Rokach, B. Shapira, and P. B. Kanto, *Recommender Systems Handbook*, 1st ed. Springer, 2011.
- [18] S. Ryza, U. Laserson, S. Owen, and J. Wills, *Advanced Analytics with Spark*. O'Reilly Media, Inc., 2015.
- [19] S. SBT. (2015) The interactive build tool. [Online]. Available: <https://www.scala-sbt.org/1.x/docs/index.html>
- [20] Spark. (2014) ALS. [Online]. Available: <http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS>
- [21] ——. (2014) Collaborative filtering - rdd-based api. [Online]. Available: <http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>
- [22] ——. (2014) Spark programming guide. [Online]. Available: <http://spark.apache.org/docs/latest/programming-guide.html>
- [23] ——. (2016) Dataset. [Online]. Available: <https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Dataset.html>
- [24] ——. (2016) Rdd. [Online]. Available: <https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.rdd.RDD>
- [25] ——. (2016) Spark sql programming guide. [Online]. Available: <http://spark.apache.org/docs/latest/sql-programming-guide.html>
- [26] P. A. Stavrou. (2013) What is the difference between convex and non-convex optimization problems? [Online]. Available: [https://www.researchgate.net/post/What\\_is\\_the\\_difference\\_between\\_convex\\_and\\_non-convex\\_optimization\\_problems](https://www.researchgate.net/post/What_is_the_difference_between_convex_and_non-convex_optimization_problems)



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