



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. Suositelijajärjestelmät . . . . .	3
2.1 Suositustekniikat . . . . .	5
2.1.1 Muistiperustainen yhteisöllinen suodatus . . . . .	6
2.1.2 Mallipohjainen yhteisösuodatus . . . . .	9
3. Apache Spark . . . . .	10
3.1 Scala . . . . .	11
3.1.1 Perustyytit . . . . .	12
3.1.2 Muuttujat . . . . .	12
3.1.3 Funktiot . . . . .	13
3.2 Resilient Distributed Dataset (RDD) . . . . .	14
3.3 Dataset API . . . . .	15
3.4 DataFrame API . . . . .	19
3.5 Matriisin tekijöihinjako . . . . .	19
3.5.1 Alternating Least Squares (ALS) . . . . .	22
4. Toteutus . . . . .	25
4.1 MovieLensRecommendation.scala . . . . .	26
4.2 Apache Pig . . . . .	31
5. Tulokset . . . . .	33
5.1 Sisääntulot . . . . .	33
5.2 Suositukset . . . . .	35
6. Yhteenveto . . . . .	40
6.1 Johtopäätökset . . . . .	40
6.2 Tulevaa työtä . . . . .	40

Bibliography . . . . .	42
------------------------	----

## 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

Suosittelujärjestelmät ovat nykyisin jatkuvasti läsnä jokapäiväisessä elämässämme. Ne auttavat päätöksenteossa verkko-ostoksissa, suoratoistopalveluissa, sosiaalisessa mediassa tai yksinkertaisti 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 käyttäjähajaiseseen suositteluun. Tuotepohjaisessa suosittelussa tarkoituksena on etsiä samankaltaisia tuotteita, sillä käyttäjän ajatellaan olevan mahdollisesti kiinnostunut samankaltaisista tuotteista myös tulevaisuudessa. Käyttäjähajaisessa suosittelussa käyttäjän ajatellaan olevan kiinnostunut tuotteista, joita samankaltaiset käyttäjät ovat ostaneet. Käyttäjähajainen suosittelu yrittää siis etsiä samankaltaisia käyttäjiä, jotta voidaan suositella näiden käyttäjien ostamia tuotteita.

Apache Spark on sovelluskehys, joka mahdollistaa hajautettujen ohjelmien rakentamisen. 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.

Apache Spark on rakennettu Scala ohjelmointikielellä. 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.

Työn päämääränä on tutustua Apache Spark sovelluskehukseen sekä Scala ohjelmointikieleen ja toteuttaa näiden tekniikoiden avulla suosittelujärjestelmä.

Tämä työ on rakentuu seuraavista osista. Luku kaksi kuvailee suosittelujärjestelmiä. Luvussa kolme keskustellaan Apache Sparkista, avoimen lähdekoodin järjestelmästä, joka mahdollistaa hajautettujen ohjelmien rakentamisen. Luku neljä esittää toteutuksen suosittelijajärjestelmälle. Luvussa viisi käydään läpi tulokset. Lopuksi luvussa kuusi esitellään johtopäätökset.

## 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. Suositelijajärjestelmät ovat joukko tekniikoita ja ohjelmistoja jotka tarjoavat suosituksia mahdollisesti hyödyllisistä tuotteista. Tuote tarkoittaa yleistä asiaa jota järjestelmä suosittelee henkilölle. Suosittelemajärjestelmät on yleensä tarkoitettu suosittelemaan tietyn tyyppisiä tuotteita kuten kirjoja tai elokuvia. [14]

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 tuote-arvioita tai ne voidaan tulkita käyttäjän toiminnasta kuten klikkauksista tai sivun katselukerroista. Suosittelemajärjestelmä voisi esimerkiksi tulkita tuotesivulle navigoinnin todisteeksi mieltymyksestä sivun tuotteista. [14]

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. [14]

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. [14]

Viimeaikoina suosittelijajärjestelmät ovat osoittautuneet tehokkaaksi lääkkeeksi kärsivällä 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. [14]

Verrattuna klassisten tietojärjestelmien, kuten tietokantojen ja hakukoneiden, tutkimukseen, suosittelijajärjestelmien tutkimus on verrattain tuoretta. Suosittelijajärjestelmistä 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). [14]

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. [14] Palveluntarjoaja voi esimerkiksi ottaa suosittelujärjestelmän avuksi parantamaan tai monipuolistamaan myyntiä, lisäämään käyttäjien tyytyväisyyttä, lisäämään käyttäjien uskollisuutta tai ymmärtämään paremmin mitä käyttäjä haluaa [14]. Lisäksi käyttäjillä saattaa olla seuraavanlaisia odotuksia suosittelujärjestelmältä. Käyttäjä saattaa haluta suosituksena tuotesarjan, apua selaamiseen

tai mahdollistaa muihin vaikuttamisen. Vaikuttaminen saattaa olla pahantahtoista. [14]

GroupLens, BookLens ja MovieLens olivat uranuurtajia suosittelujärjestelmissä. GroupLens on tutkimuslaboratorio tietojenkäsittelyopin ja tekniikan laitoksella Minnesotan Yliopistossa, joka on erikoistunut suosittelujärjestelmiin, verkkoyhteisöihin, mobiili ja kaikkialla läsnä oleviin teknologioihin, digitaalisiin kirjastoihin ja paikallisen maantieteen tietojärjestelmiin. [2] BookLens on GroupLensin rakentama kirjojen suosittelujärjestelmä [3]. MovieLens on GroupLensin ylläpitämä elokuvien suosittelujärjestelmä [13]. Uranuurtavan tutkimuksen lisäksi nämä sivustot julkaisivat aineistoja, joka ei ollut yleistä. [2]

## 2.1 Suositustekniikat

Suosittelujärjestelmällä täytyy olla jonkunlainen ymmärrys tuotteista, jotta se pysyy suositteluun 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 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. Suositujen elokuvien voidaan siis katsoa olevan kohtuullisen osuvia suosituksia keskivertokäyttäjälle. [14]

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. [14]

Suosittelijajärjestelmät eroavat toisistaan kohdistetun toimialan, käytetyn tiedon ja erityisesti siinä kuinka suositukset tehdään, jolla tarkoitetaan suosittelualgoritmia [14]. Tässä työssä keskitytään vain yhteen suosittelutekniikoiden luokkaan, 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 [14]. Käyttäjä arvostelee tuotteita. Seuraavaksi algoritmi etsii suosituksia perustuen käyttäjiin, jotka ovat ostaneet samanlaisia tuotteita tai perustuen tuotteisiin, jotka ovat eniten samanlaisia käyttäjän ostohistoriaan verrattuna. Yhteisösuodatus voidaan jakaa kahden kategoriaan, jotka ovat tuotepohjainen- ja käyttäjäpohjainen yhteisösuodatus. Yhteisösuodatusta on eniten käytetty ja toteutettu tekniikka suositusjärjestelmissä [7] [14] [4].

Yhteisöllinen suodatus analysoi käyttäjien välisiä suhteita ja tuotteiden välisiä riippuvuuksia tunnistaa uusia käyttäjä-tuote assosiaatioita [10]. Päätelmä siitä, että käyttäjät voisivat pitää samasta laulusta koska molemmat kuuntelevat muita samankaltaisia lauluja on esimerkki yhteisöllisestä suodatuksesta [15].

Koska yhteisöllisessä suodatuksessa suosittelu perustuu pelkästään käyttäjän arvosteluihin tuotteesta, yhteisöllinen suodatus kärsii ongelmista jotka tunnetaan nimillä uusi käyttäjäongelma ja uusi tuoteongelma [7]. 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]

### 2.1.1 Muistiperustainen yhteisöllinen suodatus

Muistiperustaisissa 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 ostohistoriasta [7]. Seuraavaksi mallinnetaan käyttäjän mieltymykset tuotteelle perustuen saman käyttäjän tekemiin arvosteluihin [14]. Alla oleva koodinpätkä esittelee ICBF:n idean jokaiselle uudelle käyttäjälle.

**Program 2.1** *Tuotepohjainen yhteisösuodatus algoritmi [7]*

```
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. Tunnistaa k samankaltaisinta tuotetta jokaiselle tuotteelle

val itemsSorted = sort(similarItems)

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

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

Amerikan suurimman verkkokaupan, Amazon.com:in, on aiemmin tiedetty käyttä-

vät 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. [11]

### 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. [14]

Askeleet jokaiselle uudelle käyttäjälle käyttäjäpohjaisessa yhteisösuodatuksessa ovat:

**Program 2.2** Käyttäjäpohjainen yhteisösuodatus algoritmi [7]

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)
```

### 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. [14]

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

### 3. APACHE SPARK

Apache Spark on avoimen lähdekoodin sovelluskehys, joka yhdistää hajautettujen ohjelmien kirjoittamiseen tarkoitetun järjestelmän sekä elegantin mallin ohjelmien kirjoittamiseen. [15] 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 levyille 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. [15]

Spark ohjelmia voidaan kirjoittaa 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. [15]

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 etteivät koodi tai tulokset pääse katoamaan, vaikkakin REPL tarjoaa samantapaisen muistin komentohistoriasta kuin perinteinen komentorivikin.

SELITYS JAR:ISTA JA EHKÄ JVM:STÄ?

### 3.1 Scala

Scala on hybridiohjelmointikieli, 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, objekti ja piirre (trait). Scala tukee myös kapselointia, perintää, moniperintää ja muita tärkeitä olio-ohjelmoinnin konsepteja. Scala on staattisesti tyyppitetty kieli ja Scala ohjelmat käännetään Scala kääntäjää käyttäen. Scala on JVM (Java Virtual Machine, Java virtuaalikone) perustainen 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. 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. [8]

### 3.1.1 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 true tai false.

Huomionarvoista on se, että Scalassa ei ole ollenkaan primitiivisiä tyyppejä kuten Javassa. Jokainen tyyppi on toteutettu luokkana ja käännöksen aikana kääntäjä tarvittaessa automaattisesti muuntaa Scala tyytit Javan primitiivi tyypeiksi.

### 3.1.2 Muuttujat

Scalassa on kahdentyyppisiä muuttujia: muuttuvia ja muuttumattomia. Muuttuva muuttuja määritellään avainsanan *var* avulla. Muuttuvan muuttuja voidaan antaa uudelleen luomisen jälkeen. Var:ien käyttöä ei suositella, mutta joskus niiden käyttämisellä saadaan aikaan yksinkertaisempaa koodia ja tästä syystä Scala tukee myös muuttuvia muuttujia. Syntaksi *var*:in luomiseksi on

**Program 3.1** *Muuttuvan muuttujan luominen ja uudelleen antaminen*

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

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

**Program 3.2** *Muuttumattoman muuttujan luominen*

```
val y = 10
```

Mikäli muuttumatonta muuttujaa koitetaan antaa uudelleen myöhemmin ohjelmassa, kääntäjä antaa virheen. Huomionarvoista ylläolevassa syntaksissa on se, että Scala kääntäjä ei pakota määrittelemään muuttujan tyyppiä silloin kuin kääntäjä pystyy päättämään sen.

**Program 3.3** *Muuttujan luominen tyyppimäärittelyn avulla*

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

### 3.1.3 Funktiot

Funktio on lohko suoritettavaa koodia joka palauttaa arvon. Se on konseptuaalisesti samankaltainen kuin matematiikassa: funktio ottaa sisääntulon ja palauttaa ulostulon. Scalán funktiot ovat ensimmäisen luokan kansalaisia, jolla tarkoitetaan että funktiota voidaan:

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

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.

***Program 3.4 Funktio***

```
def add(firstInput: Int, secondInput: Int): Int = {  
    val sum = firstInput + secondInput  
    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 3.5 Funktio***

```
def add(firstInput: Int, secondInput: Int): Int = firstInput + secondInput
```

Toinen versio tekee täsmälleen saman asian kuin ensimmäinenkin, mutta se on vain kirjoitettu käyttäen lyhyempää syntaksia. Paluuarvon tyyppi on jätetty antamatta, sillä kääntäjä pystyy päättämään sen koodista. Paluuarvo 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 on arvon palauttavia lausekkeitä, joten funktion rungon viimeisen lausekkeen arvosta tulee funktion paluuarvo. [8]

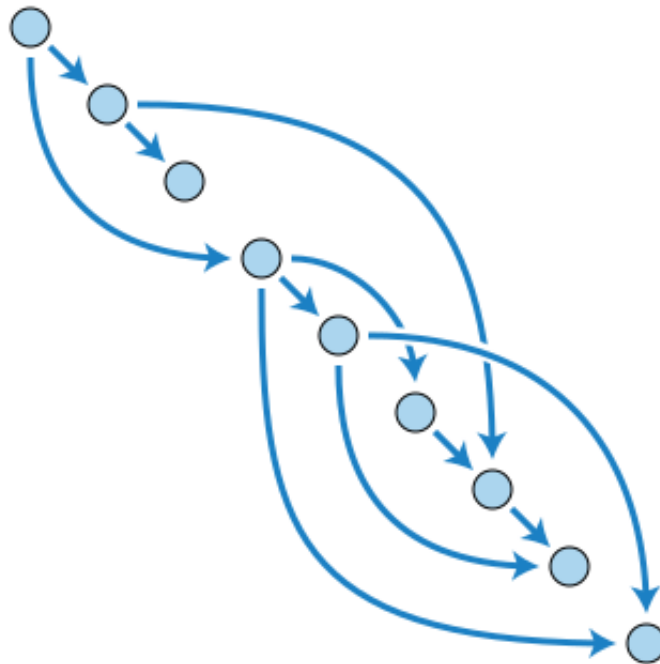
## 3.2 Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset (RDD) on Sparkin tarjoama pääabstraktio. RDD on muuttumaton, osioitu elementtikokoelma, joka voidaan hajauttaa klusterin useiden koneiden välillä. [20]

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:ä, HBase:ssa tai missä tahansa Hadoopin tuemassa tiedostojärjestelmässä. [18]

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 syklistä datavirtaa. Jokaista Spark työtä kohti luodaan DAG klusterissa suoritettavan tehtävän tasoista. Verrattuna MapReduceen, joka luo DAGin kahdesta ennaltamääritetystä tilasta (Map ja Reduce), Sparkin luomat DAGit voivat sisältää minkä tahansa määrän tasoja. Tästä syystä jotkin työt voivat valmistua nopeammin kuin ne valmistuisivat MapReduceessa. Yksinkertaisimmat työt voivat valmistua vain yhden tason jälkeen ja monimutkaisemmat tehtävät valmistuvat yhden monitasoisen ajon jälkeen, ilman että niitä täytyy pilkkoa useampiin töihin. [23]

**Kuva 3.1** *Directed Acyclic Graph [5]*

### 3.3 Dataset API

Dataset (DS) on RDD:n korvaaja Sparkissa. DS on vahvasti tyyppitetty 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. [19]

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 kyselyoptimoiija (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. [19]

Domain-spesifisten olioiden tehokkaaseen tukemiseen tarvitaan enkooderia. Enkoo-

derilla 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 dataprozessointiin. Datan binääriesitys voidaan tarkistaa käyttämällä DS:n tarjoamaa *schema* funktiota. [19]

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 3.6** *Esimerkki JSON tiedosto*

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

Dataset voidaan luoda myös tekemällä muutoksia olemassaoleville Dataset olioille:

**Program 3.7** *Creating a new Dataset through a transformation*

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

**Program 3.8** *Uuden Dataset olion luominen käyttäen read funktiota*

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

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

**Program 3.9** *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 Data-Frame olio, jonka sisäinen mallin (schema) Spark pyrkisi arvaamaan. Tyyppimuunnos tehdään käyttämällä *as* avainsanaa.

**Program 3.10** *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* ketjutamalla rakentaja metodin kutsuja.

[19]

Dataset oliot ovat samankaltaisia kuin RDD:t, sillä nekin tarjoavat vahvan tyyppityksen ja mahdollisuuden käyttää voimakkaita lambda-funktioita [21]. Lambda-funktioita avustaa Spark SQL:n optimoitu suoritusmoottori [21]. 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. [18]

Seuraavassa koodilistauksessa luodaan uusi Dataset lukemalla *json* tiedosto tiedostojärjestelmästä. Seuraavaksi luodaan uusi Dataset muunnoksen kautta. Objektiin kloonaukseksi käytetään case luokan *copy* metodia, koska *people* Dataset oli määriteltä muuttumattomaksi. Lopuksi fyysinen suunnitelma tulostetaan konsoliin käyttämällä *explain* funktiota uudelle Dataset objektile.

**Program 3.11** Dataset olion 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)
```

== 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
```

== 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>

```

### 3.4 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]. [18]

*Program 3.12 DataFrame luominen käyttäen read funktiota*

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

DataFrame objektia luotaessa, Spark arvaa luodun objektin sisäisen mallin.

### 3.5 Matriisin tekijöihinjako

Matriisin tekijöihinjako on toimi, jossa matriisi hajoitetaan matriisien tuloksi. Matriisi voidaan hajottaa tekijöihinsä usealla eri tavalla. Seuraava kappale kuvailee

*Kuva 3.2 DataFrame*

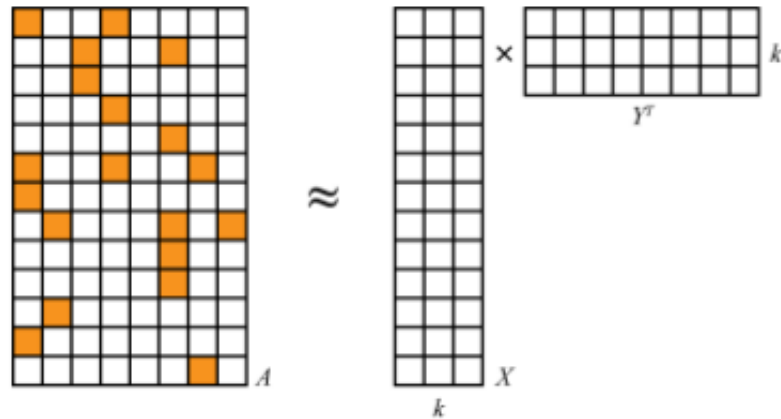
Name	Age	Weight
String	Int	Double
String	Int	Double
String	Int	Double

matriisin tekijöihinjakoa yleisellä tasolla sekä vuorottelevien pienempien 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.

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 mahdollisuuksien joukosta kuvailemalla käyttäjiä ja tuotteita mieltymysten perusteella, joista ei ole mahdollista saada tietoa. [15] 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 arvoa, jonka käyttäjä on antanut tietylle tuotteelle. [15]

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

**Kuva 3.3** Matrix factorization [15]

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 piileviä tekijöitä joita käytetään kuvailemaan tiedossa sijaitsevia vuorovaikutuksia. Hajotelma (factorization) on ainoastaan arvio, sillä  $k$  on pieni. [15]

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. [17] [15]

Usein monissa tosielämän käyttötapauksissa on käytettävissä ainoastaan implisiittistä tietoa 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. Eksplisiittisten arvioioiden sijaan, nämä numerot liittyvät havaintujen käyttäjämieltymysten varmuuteen. Tämän tiedon perusteella malli koettaa etsiä piileviä tekijöitä joiden avulla voidaan ennustaa käyttäjän odotettu arvio tuotteelle. [17]

Näihin algoritmeihin viitataan joskus matriisin täyttö algoritmeina. Tämä johtuu siitä että alkuperäinen matriisi  $A$  saattaa olla harva vaikka matriisitulo  $XY^T$  on

tiheä. Vaikka tulomatriisi sisältää arvon kaikille alkioille, se on kuitenkin vain arvio  $A$ :sta. [15]

### 3.5.1 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 piileviä tekijöitä, joita voidaan käyttää puuttuvien kohtien ennustamiseen. Spark MLlib käyttää vaihtelevien pienimpien neliöiden (Alternating Least Squares, ALS) algoritmia näiden piilevien tekijöiden oppimiseen. [17]

Spark ALS yrittää arvata arvostelumatriisin  $A$  kahden alemman arvon matriisin,  $X$  ja  $Y$ , tulona. [16]

$$A = XY^T \quad (3.1)$$

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. [16] Spark ALS mahdollistaa massiivisen rinnakkaistamisen sillä algoritmia voidaan suorittaa erikseen. Tämä on erinomainen ominaisuus laajamittaiselle (large-scale) laskenta-algoritmille. [15]

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. Vas-

taavasti tuotelohko löytää käyttäjän arviot ja päivittää tuotteita näiden viestien perusteella. [16]

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. [16]

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

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 3.13** *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) on kenties suosituin ennustettujen arvosteluiden tarkkuuden evaluointiin käytetty metriikka. Sitä käytetään yleisesti regressioalgoritmien avulla luotujen mallien evaluointiin. Läheinen metriikka on MSE (Mean Square Error). Regressioalgoritmien yhteydessä virheellä tarkoitetaan havainnon todellisen sekä ennustetun numeroarvon välistä eroa. Kuten nimi viittaa, MSE on virheiden neliöiden keskiarvo. Se voidaan laskea neliöimällä jokaisen havainnon virhe ja laskemalla virheiden neliöiden keskiarvo. RMSE voidaan puolestaan laskea ottamalla neliöjuuri MSE:stä. RMSE sekä MSE edustavat opetusvirhettä. 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. [8]

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

$$RMSE = \sqrt{\frac{1}{|\tau|} \sum_{(u,i) \in \tau} (\hat{r}_{ui} - r_{ui})^2} \quad (3.3)$$

Konvergenssilla tarkoitetaan jonkin ilmiön lähestymistä ajan kuluessa jotain tiettyä arvoa, tässä tapauksessa sitä, että RMSE ei enään pienene tarpeeksi.



## 4. TOTEUTUS

GroupLens Research on kerännyt ja laittanut saataville arvioaineistoja MovieLens sivustolta. Aineistot on kerätty useiden aikajaksojen aikana, riippuen aineiston 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, jotta voitaisiin antaa arvioita elokuville, jotka on oikeasti nähty ja myös laitteiston vuoksi. MovieLens ml-latest-small aineisto koostuu *movies.csv* and *ratings.csv* tiedostoista.

**Taulukko 4.1** *movies.csv*

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 4.2** *ratings.csv*

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 kuvailaan sovelluksen riippuvuudet. Itsenäinen spark sovellus tarkoittaa käyttövalmista *jar* tiedostoa (Java ARchive) joka voidaan jakaa spark klusterille ja se sisältää sekä koodin että 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** *Suosittelujen lataaminen RDD rajapintaa käyttäen*

```
1 val ratings = sc.textFile("ml-latest-small/ratings.csv")
2   .filter(x => !isHeader("userId", x))
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 }

```

**Program 4.4** *Suositusten 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 }

```

**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     val ratings = sc.textFile("ml-latest-small/ratings.csv")
23     .filter(x => !isHeader("userId", x))
24     .map { line =>
25         val fields = line.split(",")
26         val timestamp = fields(3).toLong % 10
27         val userId = fields(0).toInt
28         val movieId = fields(1).toInt
29         val rating = fields(2).toDouble
30
31         (timestamp, Rating(userId, movieId, rating))
32     }
33
34     val movies = sc.textFile("ml-latest-small/movies.csv")
35     .filter(x => !isHeader("movieId", x))
36     .map { line =>
37         val fields = line.split(",")
38         (fields(0).toInt, fields(1))
39     }.collect().toMap
40
41     val numRatings = ratings.count
42     val numUsers = ratings.map(_._2.user).distinct.count
43     val numMovies = ratings.map(_._2.product).distinct.count
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     /* training */
62
63     val ranks = List(8, 12)
64     val lambdas = List(1.0, 10.0)
65     val numIters = List(10, 20)

```

```

66     var bestModel: Option[MatrixFactorizationModel] = None
67     var bestValidationRmse = Double.MaxValue
68     var bestRank = 0
69     var bestLambda = -1.0
70     var bestNumIter = -1
71     for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
72         val model = ALS.train(training, rank, numIter, lambda)
73         val validationRmse =
74             computeRmse(model, validation, numValidation)
75
76         if (validationRmse < bestValidationRmse) {
77             bestModel = Some(model)
78             bestValidationRmse = validationRmse
79             bestRank = rank
80             bestLambda = lambda
81             bestNumIter = numIter
82         }
83     }
84
85     val testRmse = computeRmse(bestModel.get, test, numTest)
86
87     val myRatedMovieIds = personalRatings.map(_.product).toSet
88     val candidates = sc.parallelize(
89         movies.keys.filter(!myRatedMovieIds.contains(_)).toSeq
90     )
91     val recommendations = bestModel.get
92         .predict(candidates.map((0, _)))
93         .collect()
94         .sortBy(-_.rating)
95         .take(10)
96
97     var i = 1
98     println("Movies recommended for you:")
99     recommendations.foreach { r =>
100         println("%2d".format(i) + ": " + movies(r.product))
101         i += 1
102     }
103
104     // clean up
105     sc.stop()
106 }
107
108 def isHeader(id: String, line: String): Boolean = line.contains(id)
109

```

```

110  /** Compute RMSE */
111  def computeRmse(
112      model: MatrixFactorizationModel,
113      data: RDD[Rating],
114      n: Long
115  ): Double = {
116      val predictions: RDD[Rating] =
117          model.predict(data.map(x => (x.user, x.product)))
118      val predictionsAndRatings =
119          predictions.map(x => ((x.user, x.product), x.rating))
120              .join(data.map(x => ((x.user, x.product), x.rating)))
121              .values
122
123      math.sqrt(
124          predictionsAndRatings
125              .map(x => (x._1 - x._2) * (x._1 - x._2))
126              .reduce(_ + _) / n
127      )
128  }
129  }

```

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. Riveillä 22-36 luodaan RDD oliot *ratings* ja *movies* lataamalla kaksi erillistä csv tiedostoa. Tiedostoista suodatetaan ensin pois otsikkorivit käyttäen *isHeader* apufunktiota. 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 muodoste-

taan Tupleja. Riveillä 44-54 valmistellaan opetus, validaatio sekä testidatat. Rivillä 47 opetusdataan lisätään omat henkilökohtaiset arvostelut käyttäen RDD:n union funktiota.

Riveillä 64-83 suoritetaan varsinainen mallin opetus.

Opetus suoritetaan niin, että opetetaan muutama versio mallista, ja lopuksi valitaan opetetuista malleista paras käyttäen RMSE-metriikkaa mittarina.

Varsinainen mallin opetus tehdään käyttäen ALS kirjaston funktiota *train* ja tarkemmin sanottuna *train* funktion ylikuormitettua versiota, joka ottaa sisääntulonaan *ratings*, *rank*, *iterations* sekä *lambda* parametrin. Ratings on RDD Rating olioita, jotka sisältävät käyttäjän id:n, elokuvan id:n ja suosituksen. Rank tarkoittaa piilevien ominaisuuksien sisällytettävää määrää. Iterations tarkoittaa ALS algoritmien iteraatioiden määrää. Lambda tarkoittaa regularisaatio parametria, jolla yritetään ehkäistä mallin ylioppimista.

Riveillä 89-102 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. Rivillä 105 kutsutaan lopuksi *sparkContext* objektin *stop* funktiota, jolla kerrotaan että laskenta on suoritettu loppuun.

## 4.2 Apache Pig

Koska valmiita personoituja suosittelujärjestelmiä ei vaikuttanut olevan saatavilla, tuloksia vertaillaan ei-personoituihin, Apache Pig järjestelmällä saatuihin tuloksiin. Apache Pig on suurten datasettien analysointiin tarkoitettu alusta, joka sisältää korkean tason kielen data-analyysi sovellusten ilmaisemiseen sekä infrastruktuurin näiden ohjelmien käyttämiseen. Pig ohjelmien rakenne mahdollistaa merkittävän rinnakkaistamisen, joka puolestaan mahdollistaa todella suurten datasettien käsittelyn. [6]

Tällä hetkellä Pig:n infrastruktuurikerros koostuu kääntäjästä joka tuottaa Map-Reduce ohjelmien pätkiä, joille suurimittaiset rinnakkaiset toteutukset löytyvät valmiina. Pig:n kielikerros koostuu kielestä nimeltä Pig Latin, jolla on seuraavat ominaisuudet:

Ohjelmoinnin helppous. Rinnakkaisesti suoritettavien data-analyysi ohjelmien kirjoittaminen on helppoa. Monimutkaiset, useista toisistaan riippuvista datamuunnokset enkoodataan eksplisiittisesti data virtaus (flow) sekvensseinä, jolloin niiden kirjoittaminen, ymmärtäminen ja hallinta helpottuu. Optimointimahdollisuudet. Suo-

rituksen automaattinen optimointi, joka antaa käyttäjän keskittyä semantiikkaan tehokkuuden sijaan. Laajennettavuus. Käyttäjät voivat luoda omia funktioitaan erikoiskäyttöiseen prosessointiin. [6]

Toteutus on muunnettu uudemmalle MovieLens datasetin versiolle, mutta muutoin koodi on uudelleen käytetty alkuperäisessä muodossaan. [22] [9]



## **5. TULOKSET**

Tässä kappaleessa käsitellään työn tärkeimpiä tuloksia.

### **5.1 Sisääntulot**

Tässä osassa esitetään suosittelujärjestelmän sisääntulo aineistosta itse annetut arviot.

**Taulukko 5.1** Arvostellut elokuvat

userId	movieId	movieName	rating
0	112897	The Expendables 3 (2014)	4.0
0	116887	Exodus: Gods and Kings (2014)	4.0
0	117529	Jurassic World (2015)	4.0
0	118696	The Hobbit: The Battle of the Five Armies (2014)	4.5
0	128520	The Wedding Ringer (2015)	4.5
0	122882	Mad Max: Fury Road (2015)	4.0
0	122886	Star Wars: Episode VII - The Force Awakens (2015)	4.5
0	131013	Get Hard (2015)	4.0
0	132796	San Andreas (2015)	3.0
0	136305	Sharknado 3: Oh Hell No! (2015)	1.0
0	136598	Vacation (2015)	4.0
0	137595	Magic Mike XXL (2015)	1.0
0	138208	The Walk (2015)	2.0
0	140523	The Visit (2015)	3.5
0	146656	Creed (2015)	4.0
0	148626	The Big Short (2015)	4.5
0	149532	Marco Polo: One Hundred Eyes (2015)	4.5
0	150548	Sherlock: The Abominable Bride (2016)	4.5
0	156609	Neighbors 2: Sorority Rising (2016)	3.5
0	159093	Now You See Me 2 (2016)	4.0
0	160271	Central Intelligence (2016)	4.0

Taulukossa 5.1 on esitetty suosittelujärjestelmän sisääntulona annetut jo nähdyt elokuvat. Sisääntulon rakenne on seuraava: sarakkeessa yksi sijaitsee käyttäjän tunniste, sarakkeessa kaksi sijaitsee elokuvan tunniste, sarakkeeseen kolme on sijoitettu elokuvan nimi ja sarakkeessa neljä sijaitsee elokuvalle annettu arvio asteikolla 0-5. Jonkun työkalun avulla arvosteluiden antaminen olisi ollut paljon mielekkäämpää.

## 5.2 Suositukset

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

**Taulukko 5.2** *Toteutetun järjestelmän suosittelat elokuvat*

number	movieName
1	Death of a superhero (2011)
2	Prisoner of the Mountains (1996)
3	Funeral in Berlin (1966)
4	Caveman (1981)
5	Dream With the Fishes (1997)
6	Erik the Viking (1989)
7	Dead Man's Shoes (2004)
8	Excision (2012)
9	Mifune's Last Song (1999)
10	Maelström (2000)

Taulukossa 5.2 on suosittelujärjestelmältä saaduista suosituksista 10 ensimmäistä.

**Taulukko 5.3** Vertailtavan järjestelmän suosittelat elokuvat

number	movieName
1	Death of a superhero (2011)
2	Prisoner of the Mountains (1996)
3	Funeral in Berlin (1966)
4	Caveman (1981)
5	Dream With the Fishes (1997)
6	Erik the Viking (1989)
7	Dead Man's Shoes (2004)
8	Excision (2012)
9	Mifune's Last Song (1999)
10	Maelström (2000)

Taulukossa 5.2 on suosittelujärjestelmältä saaduista suosituksista 10 ensimmäistä.

Tuloksien yhteiset piirteet on helppo poimia taulukoista 5.2 ja 5.3. Suurimmat erot ovat... Puutteellisia tuloksia on hankala arvioida, sillä molemmat järjestelmät antavat tarvittavat 10 suositusta samalle aineistolle.

## 6. YHTEENVETO

Tässä kappaleessa esitetään yhteenveto.

### 6.1 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.

Olemassaolevien suosittelujärjestelmien tai analytiikkajärjestelmien evaluointi tulisi suorittaa ennen suosittelujärjestelmän valintaa.

Lopulta kaikista vaikein asia oli löytää oikea lähestymistapa tähän kyseiseen tehtävään. Yrittämisen ja lukuisten epäonnistumisien jälkeen oikea teknologioiden joukko sekä varsinainen toimiva esimerkki löydettiin.

Suuremman datasetin käyttäminen, sekä isomman arvostelumäärän tarjoaminen järjestelmälle voisi parantaa tuloksia.

### 6.2 Tulevaa työtä

Mikäli käytettävissä oleva laitteisto sallisi, olisi mahdollista rinnakkaistaa suoritusta sekä samalla kasvattaa käytettävän datasetin kokoa.

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.

Viimeaikoina useat palveluntarjoajat, kuten Telegram ja Microsoft, ovat esitelleet bot sovelluskehityksiä palveluihinsa. Botti on web-palvelu, joka keskustelee käyttäjien



kanssa pikaviesti kanavalla. Käyttäjät voivat aloittaa keskustelun botin kanssa missä tahansa kanavalla, jota bot on määritetty kuuntelemaan. Keskustelut voivat olla vapaamuotoisia tai ne voivat koostua tietyistä, ennaltamääritetyistä valinnoista. [12]

Pitkän aikaa, yrityksillä on ollut jonkinlainen tekstiviestipalvelu, jonka avulla asiakailta on kyselty palautetta esimerkiksi uuden puhelinliittymän tilauksesta. IRC (Internet Relay Chat) -kanavilla botteja on ollut olemassa vieläkin pidempään. Ajatus ei ole uusi, mutta nyt boteille on olemassa suosittuja alustoja. Kuten mikä tahansa ajatus, myös suosittelujärjestelmä voitaisiin toteuttaa botin avulla käytettäväksi. Esimerkiksi Elasticsearch muodostaa RESTful rajapinnan jota vasten botti voisi ajaa kyselyitä.

## 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](https://en.wikipedia.org/wiki/Directed_acyclic_graph#/media/File:Topological_Ordering.svg)
- [6] A. S. Foundation. (2017) Apache pig. [Online]. Available: <https://pig.apache.org/>
- [7] S. K. Gorakala and M. Usielli, *Building a Recommendation Engine with R*, 1st ed. Packt Publishing, 2015.
- [8] M. Guller, *Big Data Analytics with Spark*, 1st ed. Apress, 2015.
- [9] joonne. (2017) Recommendation engine implemented in pig. [Online]. Available: <https://bitbucket.org/joonne/pig-recommendation>
- [10] 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)
- [11] 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>
- [12] Microsoft, “Bots.” [Online]. Available: <https://docs.botframework.com/en-us/>
- [13] MovieLens. [Online]. Available: <https://movielens.org/info/about>

- [14] F. Ricci, L. Rokach, B. Shapira, and P. B. Kanto, *Recommender Systems Handbook*, 1st ed. Springer, 2011.
- [15] S. Ryza, U. Laserson, S. Owen, and J. Wills, *Advanced Analytics with Spark*. O'Reilly Media, Inc., 2015.
- [16] Spark. (2014) ALS. [Online]. Available: <http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS>
- [17] ——. (2014) Collaborative filtering - rdd-based api. [Online]. Available: <http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>
- [18] ——. (2014) Spark programming guide. [Online]. Available: <http://spark.apache.org/docs/latest/programming-guide.html>
- [19] ——. (2016) Dataset. [Online]. Available: <https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Dataset.html>
- [20] ——. (2016) Rdd. [Online]. Available: <https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.rdd.RDD>
- [21] ——. (2016) Spark sql programming guide. [Online]. Available: <http://spark.apache.org/docs/latest/sql-programming-guide.html>
- [22] tcabrol. (2017) Recommendation engine implemented in pig. [Online]. Available: <https://gist.github.com/tcabrol/2319242>
- [23] M. Technologies. Apache spark. [Online]. Available: <https://mapr.com/products/product-overview/apache-spark/>