TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Jaak Tamre 130411IABMM

TELIA TV SOOVITUSMOOTORI VALIK JA KASUTUSELEVÕTT

Magistritöö

Juhendaja: Jaak Tepandi
PhD

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Jaak Tamre

05.05.2017

Annotatsioon

Käesolev lõputöö tutvustab hetkeolukorda Telia TV rakenduse probleemidest, mis puudutavad kasutaja sisu leidmist ja üritab leida parimat soovitusmootorit selle lahendamiseks. Lõputöö eesmärgid on valida parim soovitusmootor, analüüsida selle masinõppimise algoritmi ning võtta see kasutusele Telia TV rakenduste jaoks.

Otsuse tegemiseks kasutatakse täppismeetodit, mille loojaks on Thomas L. Saaty, ning luuakse otsustusmudel. Otsustusmudeli põhikriteeriumiteks sai valitud hind, keerukus, kvaliteet, tehniline võimekus ja haldus. Valiku alternatiividekson IBM Watson, Weka, Scikit-learn, PredictionIO ning R-Project. Lõpptulemusena osutus valituks PredictionIO lahendus.

PredictionIO jaoks sai võetud kasutusele soovitusmootori mall The Universal Recommender, mis leiab ristkorrelatsioonide (Correlated Cross-Occurrence) põhjal indikaatorid soovituste tegemiseks. Kasutajale soovitatakse eelkõige neid asju, mida teised kasutajad on vaadanud lisaks tema vaadatud asjadele, seejuures pidades anomaalseid käitumismustreid olulisemaks.

PredictionIO'ga suhtlemiseks tuli luua Telia TV API poolel uued komponendid ja skriptid andmete importimiseks ning soovituste väljastamiseks kasutajaliidesele. Kasutaja jaoks sai lõputöö skoobis lisatud videolaenutuse menüüsse uus soovituste kategooria, et valideerida selle vajalikkust. Pärast funktsionaalsuse klientidele avalikuks tegemist on soovituste alt käivitatud ligikaudu 20% kõikidest vaatamistest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 5 peatükki, 27 joonist, 2 tabelit.

Abstract

Telia TV Recommendation Engine Choice and Implementation

The master's thesis at hand introduces a common problem for most Telia TV users who want to watch on-demand movies and series. Users have a hard time finding relevant or specific content in the current user interface. The main goal of this paper is to research different machine learning platforms, choose the most suitable one and integrate it into the TV service.

Thomas L. Saaty's Analytical Hierarchy Process is used in order to make a precise decision. Price, complexity, quality, maintenance and technical capability are the main criteria used in the decisive model. It compares 5 different alternatives: IBM Watson, Scikit-learn, R-Project, Weka and PredictionIO. After completing all pairwise comparisons for each criteria and alternative, IBM Watson and PredictionIO scored equal highest results. Finally PredictionIO was chosen as an open-source solution compared to Watson which has additional fees.

PredictionIO's algorithm had to be analyzed before the integration could start. The most suitable movie recommendation engine template for PredictionIO seems to be The Universal Recommender, which uses correlated cross-occurrences to find indicators for each item. Using recent event history, some items may be more relevant to a user based on those indicators. ElasticSearch is used in the backend as an indexed database for finding recommendations and boosting the relevance of items with the given indicators.

To integrate PredictionIO for Telia TV it was necessary to install all the components on a new server instance. An existing SDK for PHP was installed and extended in the TV API. EventClient interacts with the event server to import users, items and events between them into the history matrix. These events are converted to indicators upon training the engine. After the training is complete the engine can be deployed and is available for returning predictions in response to an EngineClient's query.

In order to provide recommendations to the user interface, a new endpoint was added which uses the EngineClient. Those recommendations are fetched by the UI on bootup and displayed in a new VOD category for personal recommendations. Whenever a user completes watching a movie, the event will be sent to the event server and UI will fetch updated recommendations which exclude seen items. This also results in different relevances due to new indicators available immediately. The engine must be trained whenever new movies are added to the database and periodically to follow user's trends.

About 20% of the movies have been chosen from the new recommendation category after it was released to the end user. The project was considered successful by the stakeholders and an approval was given to continue with the developments of the recommendation engine.

The thesis is in Estonian and contains 45 pages of text, 5 chapters, 27 figures, 2 tables.

Lühendite ja mõistete sõnastik

AHP	Analytical Hierarchy Process, analüütiline hierarhiate meetod
API	Application Programming Interface, rakenduse programmiline liides
SDK	Software Development Kit, tarkvara arendamise komplekt
TPOC	Technical Proof of Concept, tehnilise konseptsiooni tõestamine
VOD	Video On-Demand, videolaenutus
UI	User Interface, kasutajaliides
UR	Universal Recommender, PredictionIO soovitusmootori mall

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Televisioon leidis laiemat kasutust juba pärast Teist Maailmasõda, kui rahvale hakati edastama uudiseid ja telesaateid otse-eetris. Alates 1975. aastast tekkis võimalus sisu salvestamiseks magnetlintidele (Betamax, VHS), et neid hiljem taasesitada. Varasemalt sai filme vaadata vaid kinosaalides, kuid seejärel hakkas filmitööstus arenema. Järgmine suurem hüpe toimus interneti kiiruste kasvuga ning üleminekuga digitaalsele televisioonile, mis võimaldas parema kvaliteediga pilti edastada. Samuti on palju sisu saadaval otse interneti vahendusel erinevate *streaming* teenusepakkujate poolt nagu Netflix, YouTube, Hulu jne (Television, 2017).

Tänapäeval on kinos käimine endiselt väga populaarne, kuid tänu lai-ekraan televiisoritele on võimalik hea kino-efekt saavutada ka kodus. Inimesed saavad sõltuvalt seadmete võimekusest kasutada erinevaid teenuseid – SmartTV omanikud sealhulgas ka Netflix, YouTube ja muid rakendusi. Suurem osa Eesti elanikkonnast kasutab enamus ajast ikkagi oma teenusepakkuja (Telia, Starman, STV) lahendust ning peamiselt Live TV (kanali otse-edastus) teenust. Hiljutised raportid (Ofcom, 2016) on aga näidanud selle trendi muutumist ning Live TV osakaal väheneb pidevalt kõikides vanusegruppides kuni 64-aastasteni. See on peamiselt tingitud VOD (videolaenutus) osakaalu suurenemisest – kõige enam 16-24-aastaste vanusegrupis.

Käesolev magistritöö keskendub TeliaTV VOD teenuse parendamisele ning kasutajasõbralikumaks tegemisele. Kasutajad on praeguse teenuse kohta andnud kriitikat, et sisu on raske üles leida ja puudub otsing. Tihti ei oska kasutaja midagi konkreetset otsida, mistõttu tuleks kasutajale pakkuda sisu lähtuvalt tema personaalsetest eelistustest. Paljud suured ettevõtted (Netflix, Google, Spotify) kasutavad sisu personaalsemaks muutmiseks iseõppivaid intelligentseid süsteeme ning käesoleva töö eesmärk on sarnane lahendus tööle panna ka TeliaTV klientide jaoks.

2 Analüüs, iseõppiva süsteemi valik

Probleemi lahendamiseks on vajalik uurida olemasolevaid võimalusi ning valida nendest sobivaim. Selleks kasutab autor USA matemaatiku Thomas L. Saaty poolt välja töötatud analüütilise hierarhiate meetodit (AHP – Analytical Hierarchy Process). Antud meetod on mõeldud subjektiivsete hinnangute põhjal tegutsevatele süsteemidele otsuste tegemiseks. Selle meetodiga tutvus autor aine „Täppismeetodid otsuste tegemiseks" raames.

2.1 Otsustusmudeli koostamine

Vastavalt Saaty meetodile tuleb mudeli koostamist alustada planeerimisest ehk faktorite välja mõtlemisest ja nende hierarhiasse paigutamisest. Kriteeriumite valikul sai lähtutud olulistest aspektidest, mis puudutavad arenduse ja halduse meeskondi. Projektijuhtimise kolmnurga[1] järgi on oluline leida tasakaal ajakulu, hinna ja kvaliteedi vahel. Ajakulu määrab antud puhul süsteemi keerukus. Haldusmeeskonna jaoks on olulised erinevad aspektid, mis võimaldavad serverit monitoorida ja hallata – neid vaadeldakse eraldi kriteeriumina. Lisaks on oluline see, et süsteem oleks lihtsasti integreeritav teiste olemasolevate süsteemidega ehk tehniline võimekus. Sellest tulenevalt sai iseõppiva süsteemi jaoks valitud 5 kriteeriumit.

Valikukriteerium	**Kirjeldus**
Hind	Kas tegemist on avatud koodiga tarkvaraga või tasulise teenusega ning kui suurt investeeringut see ettevõttelt nõuab. Kriteerium on oluline seetõttu, et kalli hinna tõttu võidakse projektile rahastust mitte anda ning sellega tuleb arvestada.
Tehniline võimekus	Kas tarkvara saab hallata läbi terminali ja jookseb Linux masinas. Kas tarkvaral on API liidestus, millega saab suhelda väline server.

[1] https://en.wikipedia.org/wiki/Project_management_triangle

	Ilma minimaalsete nõueteta ei saa uut süsteemi olemasolevaga liidestada, seetõttu on selline kriteerium valitud.
Kvaliteet	Dokumentatsiooni ja koodi kvaliteet, funktsionaalsus. Pikas perspektiivis võib halb kvaliteet ja dokumentatsioon mõjuda süsteemi arendamist takistavalt või tähendada lausa projekti sulgemist.
Keerukus	Kas süsteemi tundma õppimine ja seadistamine on aeganõudev või lihtne protsess. See on oluline, et lühendada projekti kestvust ning lahendus lõppkasutajani tuua.
Haldus	Kas süsteem on paigaldatav enda serverisse ning hallatav enda administraatorite poolt või on tegemist pilveteenusega. Kas serveri install ja monitoorimine on lihtne ning võimaldab klastrit ehitada. Antud juhul on oluline teada, kas enda haldusüksus omab täielikku kontrolli süsteemi üle, kuna firma poliitika nõuab seda, ning kuivõrd keeruline on seda hallata.

Seejärel tuleb paika panna võimalikud alternatiivid. Valikus on variandid millel on olemas hea dokumentatsioon, pikaajaline toetus ning laialdane kasutus.

Alternatiiv	**Kirjeldus**
Weka	Masinõppe ja andmekaevandamise jaoks mõeldud vabavaraline tarkvara, kirjutatud Javas. http://www.cs.waikato.ac.nz/ml/weka/
Scikit-learn	Vabavaraline andmekaevandamise ja –analüüsi jaoks mõeldud tööriistade tarkvara, kirjutatud Pythonis. http://scikit-learn.org/
R	Laialtkasutatud statistilise programmeerimise ja visualiseerimise platvorm. https://www.r-project.org/
PredictionIO	Apache poolt arendatud vabavaraline masinõppe server-lahendus. http://predictionio.incubator.apache.org/
IBM Watson	Intelligentne andmeanalüüsi ja visualiseerimise teenus. https://www.ibm.com/us-en/marketplace/watson-analytics

AHP Mudeli koostamiseks ja alternatiivide võrldlemiseks on olemas mitmeid erinevaid tööriistu, kuid paljud neist on tasulised või kasutavad vanu Java lahendusi, mis on tänapäeva brauserites blokeeritud või keeruline tööle saada. Üks tasuta variant, mida

lihtsalt saab kasutada, on BPMSG AHP Online System - http://bpmsg.com/academic/. Selleks tuleb registreerida kasutajaks, mis ei too kaasa mingeid kohustusi, aga võimaldab mudelit salvestada. Mudeli koostamiseks tuleb alustuseks luua uus hierarhia, sisestada sinna valikukriteeriumid ja alam-kriteeriumid ning neid omavahel võrrelda.

2.2 Alamkriteeriumid ja hindamisskaala

Igal kriteeriumil võib olla 2 või enam alam-kriteeriumit, mille prioriteedid tuleb leida omavahelise võrlduse teel kasutades selleks AHP meetodit. Selleks tuleb igat kriteeriumit võrrelda ning hinnata, kui palju on üks kriteerium teisest olulisem. Hinne 1 tähendab, et kriteeriumid on sama olulised, hinde 9 puhul on üks ekstreemselt tähtsam.

2.2.1 Hind

Hind on teatud määral oluline, kuna tasuta lahenduse kasutamine võimaldab projekti ärijuhtide ees kergemini kaitsta. Samas ei saa hinda panna olulisemaks teistest kriteeriumitest. Hinna määrab lahenduse maksumus ning sellel puuduvad alam-kriteeriumid.

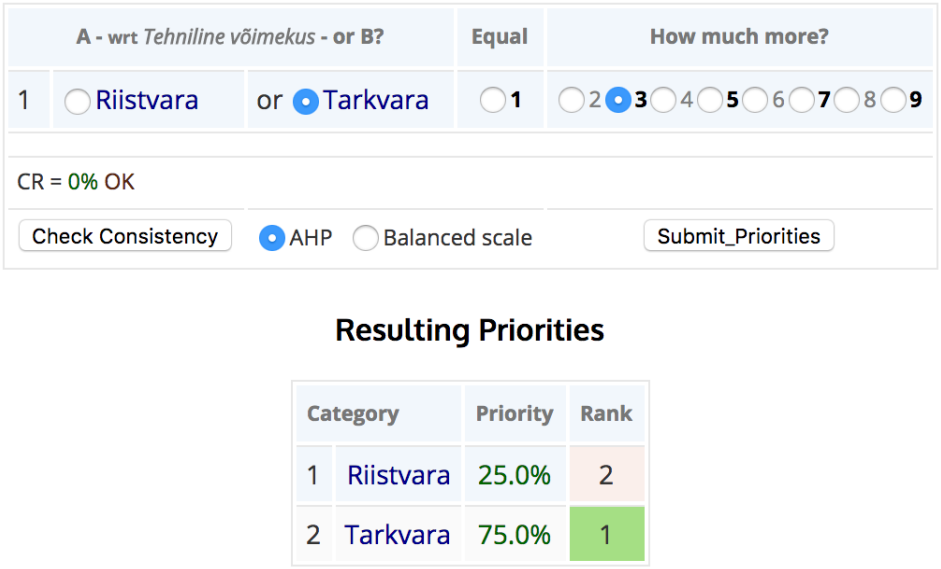
2.2.2 Tehniline võimekus

Tehniline võimekus määrab ära, kas seda lahendust on võimalik Telia TV arhitektuuri osana kasutada. Selleks on määratud kaks alam-kriteeriumit:

1. Riistvara – määrab ära, kas serverit on võimalik jooksutada virtuaalse Linuxi masina peal ning seda on võimalik kirjeldada SaltStack konfiguratsioonis, mida kasutab haldusmeeskond.

2. Tarkvara – kas antud lahendusel on olemas HTTP või muu protokoll, mille kaudu saab andmeid pärida ja sisestada. See on vajalik selleks, et olemasolevad süsteemid saaksid iseõppiva süsteemiga suhelda. Andmevahetuseks on kõige eelistatum formaat JSON, seejärel SOAP ja XML-RPC.

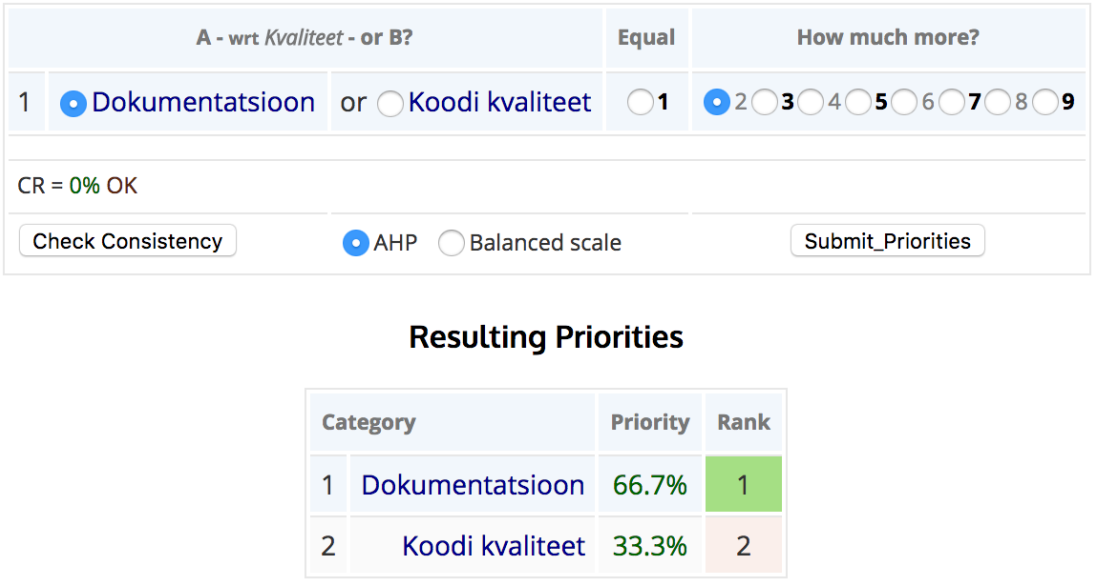
Otsustusmudelis on tarkvara 3 korda tähtsam kui riistvara (Joonis 1), sest isegi kui lahendus ei vasta kõikidele riistvaralistele nõuetele, on võimalik teha erandeid ja süsteem ikkagi tööle saada. Tarkvaraliste puuduste kõrvaldamine ei ole aga mõistlik ning tuleks kohe valida sobiv variant.



Joonis 1 – Tehnilise võimekuse alam-kriteeriumid

2.2.3 Kvaliteet

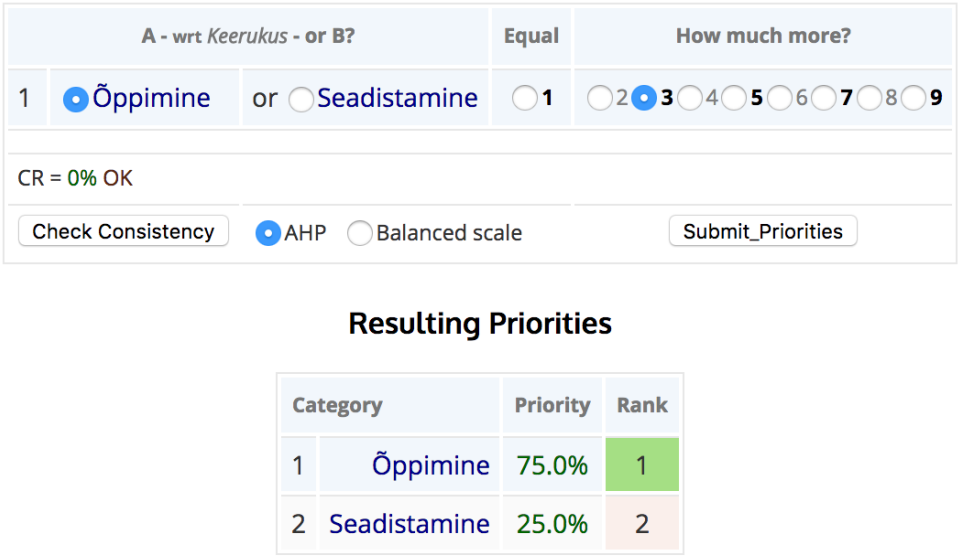
Olulised kvaliteedi alam-kriteeriumid on hea dokumentatsioon ning koodi kvaliteet. Kuigi koodi kvaliteeti on raske hinnata ilma põhjalikku uurimist tegemata, tuleb arvestada üldiste arvustustega. Juhul, kui tegemist on kinnise koodiga, tuleb arvestada firma üldist kvaliteeti. Hea dokumentatsioon korvab tihtipeale halva koodikvaliteedi, seetõttu on valitud dokumentatsioon kaks korda olulisemaks kui koodi kvaliteet (Joonis 2).



Joonis 2 – Kvaliteedi alam-kriteeriumid

2.2.4 Keerukus

Keerukuse alam-kriteeriumiteks on õppimine ja seadistamine. Õppimine on hinnanguline süsteemi selgeks tegemise kiirus. Seadistamise puhul tuleks arvestada esialgse töötava lahenduse tööle panemise aega, kuid ka seda kui palju on süsteemis üldse võimalik konfigureerida vastavalt vajadusele. Joonis 3 näitab, et õppimine on valitud mudeli jaoks 3 korda olulisem kui seadistamine, kuna seadistamine on vaid väike osa süsteemist.



Joonis 3 – Keerukuse alam-kriteeriumid

2.2.5 Haldus

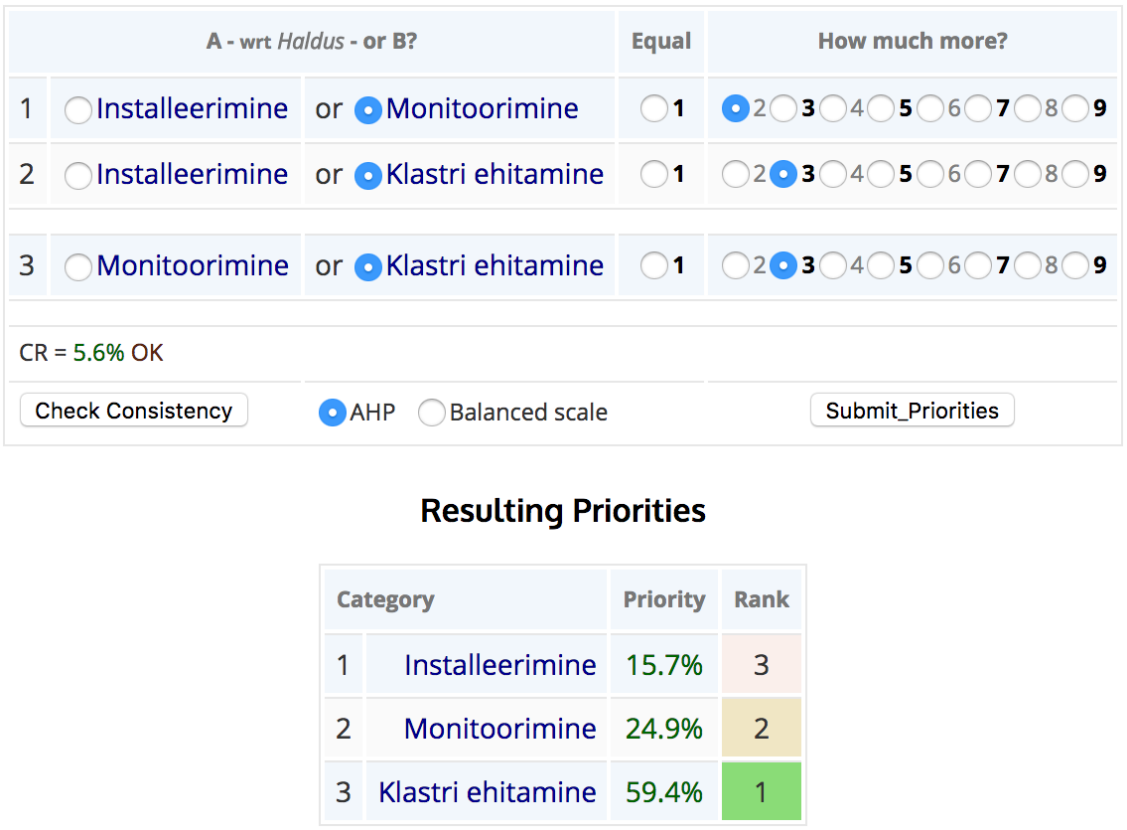
Haldus on ainuke kriteerium, millel on määratud 3 alam-kriteeriumit ning sel juhul tuleb ka arvestada kooskõla indeksit (joonisel tähistatud CR – consistency rating). Kui see tuleb liiga suur, siis antud süsteem soovitab, milliseid võrdlusi tuleks parandada, et seda vähendada.

1. Installeerimine – määrab ära, kui lihtne on kogu lahenduse paigaldamine serverisse ning tööle saamine.

2. Monitoorimine – kas süsteem kirjutab sobival kujul logisid, et neid saaks lisada monitooringusse ja jälgida süsteemi staatust.

3. Klastri ehitamine – kas süsteemi on lihtne klasterdada selliselt, et server koosneks mitmest masinast, mis omavahel andmeid sünkroniseerivad. Sinna ette saaks panna *load-balancer* tüüpi serveri koormuse jaotamiseks, mis tõstaks märgatavalt süsteemi töökindlust ja välistaks pudelikaela tekkimist.

Pikas perspektiivis on klastri ehitamise võimekus kindlasti kõige olulisem, kuid esimeses etapis ei ole see vajalik. Installeerimine on ühekordne tegevus eeldusel, et serveri konfiguratsiooni on võimalik kirjeldada SaltStack'i abil. Monitoorimine on keskmise tähtsusega. Võrdlused on leitavad allolevalt jooniselt (Joonis 4).



Joonis 4 – Halduse alam-kriteeriumid

2.3 Põhikriteeriumite võrdlus

Alljärgneval joonisel (Joonis 5) on välja toodud kõikide põhikriteeriumite omavahelised võrdlused ning nende prioriteetide tabel. Kõige olulisemaks osutus tehniline võimekus, kuna see on kriitiline projekti elluviimiseks. Teisel kohal on haldus, mis võimaldab süsteemi ka pikas perspektiivis kasutada ning kolmandale kohale jäi keerukus, mis määrab esimese iteratsiooni ajalise töömahu. Kvaliteet ja hind on omavahel enam-vähem võrdselt olulised.

	A - wrt *Iseõppiva süsteemi valik* - or B?		Equal	How much more?
1	Hind	or Tehniline võimekus	1	2 3 4 5 6 7 8 9
2	Hind	or Kvaliteet	1	2 3 4 5 6 7 8 9
3	Hind	or Keerukus	1	2 3 4 5 6 7 8 9
4	Hind	or Haldus	1	2 3 4 5 6 7 8 9
5	Tehniline võimekus	or Kvaliteet	1	2 3 4 5 6 7 8 9
6	Tehniline võimekus	or Keerukus	1	2 3 4 5 6 7 8 9
7	Tehniline võimekus	or Haldus	1	2 3 4 5 6 7 8 9
8	Kvaliteet	or Keerukus	1	2 3 4 5 6 7 8 9
9	Kvaliteet	or Haldus	1	2 3 4 5 6 7 8 9
10	Keerukus	or Haldus	1	2 3 4 5 6 7 8 9

CR = 1.5% OK

Check Consistency | AHP | Balanced scale | Submit_Priorities

Resulting Priorities

	Category	Priority	Rank
1	Hind	8.0%	5
2	Tehniline võimekus	47.2%	1
3	Kvaliteet	8.7%	4
4	Keerukus	14.7%	3
5	Haldus	21.3%	2

Joonis 5 – Otsustusmudeli põhikriteeriumite võrldus

2.4 Alternatiivide analüüs

Selleks, et koostatud mudeli põhjal valida parim alternatiiv, tuleb sisestada mudelisse kõikide kriteeriumite kohta võrdlused iga alternatiivi jaoks. Siin saab samuti kasutada AHP meetodit. Alternatiivi valikuks tuleb vajutada mudeli juures nuppu „Use consol. priorities" ning lisada alternatiivid süsteemi. Sellepeale luuakse uus projekt ning saab alternatiive võrdlema hakata.

Kõigepealt tuleb uurida iga alternatiivi dokumentatsiooni, et saaks välja selgitada tugevused ja nõrkused valitud kriteeriumites.

2.4.1 Weka

Uurides lähemalt Weka dokumentatsiooni, võib leida viiteid erinevatele allikatele: wiki leht, API dokumentatsioon, kasutajajuhendid ja õpetused algajatele (Weka Documentation). Installeerimisjuhendis on olemas Linuxi tugi, kuigi selle installeerimine käib zip faili allalaadimisega ja selle lahti pakkimisega. Serveri käivitamisel saab öelda, kas tegemist on *master* või *slave* masinaga ning samuti on juhendis eraldi punkt „Clustering", mis viitavad sellele, et klastrisse ehitamine on võimalik.

Tarkvara nõuete osas paistab Wekal olevat puudujääke. Puuduvad selged viited selle kohta, et serveriga saaks suhelda üle HTTP protokolli andmete küsimiseks või salvestamiseks. Samas näitekood on kenasti struktureeritud ja Weka omab enda testimise raamistikku ning soovitab koodi testidega katta.

Esmapilgul tundub soovitusmootori ehitamine Weka platvormil üpriski keerukas, vajab sügavamat uurimist ning eeldab tugevat Java koodi kirjutamise ja lugemise oskust. Samas paistab, et erinevat funktsionaalsust saab juurde lisada lisapakkide installeerimise teel.

2.4.2 Scikit-learn

Scikit-learn paigaldusjuhend (Documentation of scikit-learn 0.18, 2016) toob välja tehnilised nõuded ning need on aksepteeritavad Linuxi serverisse installimiseks ja SaltStack'is kirjeldamiseks. Samuti on juhendis kirjeldus testide jooksutamiseks ja koodistandardid, mis mõjuvad soodsalt koodi kvaliteedi kriteeriumile.

Master-slave replikatsiooni ega klastrisse ehitamise kohta internetist esmapilgul informatsiooni ei leia, samas monitoorimine ja logimine on Telia TV süsteemides Pythoni rakenduste puhul laialtlevinud.

Soovitusmootori ehitamiseks on leitav juhend, mis sisaldab ka koodinäiteid[1]. See vähendab küll õppimise keerukust, kuid siiski paistab, et üpris palju peab tundma Pythonit, et see valmis teha vastavalt oma vajadustele. Lisaks võib Pythoni lahendus jääda aeglasemaks võrreldes teiste keeltega.

2.4.3 R Project

R dokumentatsioon (The R Manuals) on mahukas ning sisaldab endas informatsiooni arendamise, haldamise ja andmete importimise kohta. R kasutab erilist programmeerimiskeelt S, millest autor pole varem kuulnudki. Pythoniga võrreldes on selle õppimiskõver ilmselt pikem.

R jookseb Linuxi peal ning on paigaldatav käsurealt. Juhendis on mainitud küll, et süsteemiga on võimalik üle HTTP protokolli suhelda, kuid lahendus on liiga lihtsakoeline ning vajab kas lisapaketti või süsteemi täiendamist. Üks R piiranguid on see, et ta hoiab andmeid mälus (RAM), mis võib tekitada probleeme suuremate andmemahtude puhul (*Big Data*) ning seab riistvarale suuremad nõudmised.

Koodistandardid ja testide kirjutamise juhendid leiab dokumentatsioonist „R Internals" juhendist, mis annab lootust heale koodikvaliteedile.

Kuna R on üpris laialt kasutatav, siis internetist leiab lihtsama soovitusmootori ehitamise juhendi üpris kiiresti. R jaoks on kirjutatud üle 5000 laienduse ning uue laienduse kirjutamise juhend on samuti dokumentatsioonis olemas.

2.4.4 PredictionIO

PredictionIO dokumentatsioon (PredictionIO Docs) on võrreldes eelmistega moodsam ja parema üles-ehitusega. Samuti paistab lahendus olevat üpris paindlik, kuna võimaldab erinevate andmebaaside kasutamist (PostgreSQL, MySQL või ElasticSearch) ja kohe tehakse juttu ka klastrist (mitme serveri hajussüsteemist).

Suurim eelis teiste ees on kindlasti mallide ehk *template* loogika. „Template gallery" sektsioonist leiab erinevaid malle (soovitusmootorid, klassifitseerijad jne), mida PredictionIO arendajad pidevalt täiendavad. Tuleb leida sobiv mall, see paigaldada ja

[1] http://www.mickaellegal.com/blog/2014/1/30/how-to-build-a-recommender

seadistada ning koodi polegi vaja kirjutada. Enamus malle kasutavad Scala keelt, mis sarnaneb natuke JavaScriptile ja Javale, mis teeb selle mõistmise autorile kergemaks kui näiteks Pythoni puhul. Malli koodiga on ka näited kaasas, ning sisaldavad osaliselt ka funktsionaalseid teste. Testide katvus on mallidel erinev ning kohati puudulik,

PredictionIO on üles ehitatud olemasolevatest Apache komponentidest, andmebaasist ja neid ühendavast kihist. Positiivne on see, et kohe esimene suurem peatükk „Integrating with Your App" selgitab, et serveriga saab suhelda üle HTTP protokolli ning REST standardit kasutades. See on TeliaTV süsteemi jaoks kindlasti aspekt, mis lihtsustab kasutuselevõtmist.

Süsteemi paigaldamise juhend on üpris lihtne ning monitoorimise jaoks soovitab kasutada Linuxil *monit* tööriista, et jälgida süsteemi töökorda ja staatust. Kõiki asju saab kergesti installeerida käsurealt, mis võimaldab serveri kirjeldamise SaltStack'is.

2.4.5 IBM Watson

Watson on olemuselt sarnane PredictionIO süsteemile ning pakub valmislahendust. Lisaks Watsonile on võimalik kasutada ka IBM Graph'i, mis pakub samuti soovitusmootori jaoks sobilikku liidestust ja platvormi.

Watsonil on olemas REST API liidestus (Getting started with Machine Learning), mis sobib TeliaTV süsteemi jaoks ideaalselt. Kogu serveriparki, klasterdamist ja koormusjaotust haldab ja hooldab IBM, mis tähendab vähem vastutust enda haldusmeeskonnale, kuid samas tekitab välise sõltuvuse. Süsteemi õppimine ja seadistamine on oluliselt lihtsam kui paljudel alternatiivsetel süsteemidel.

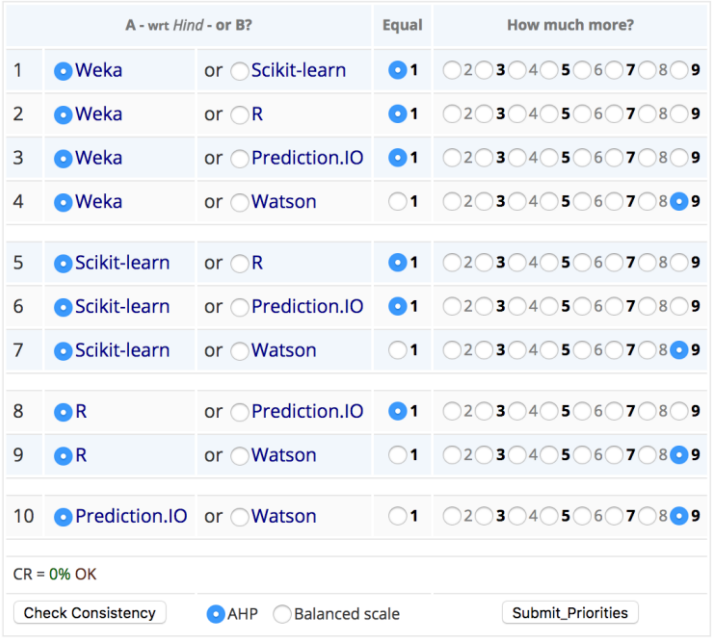
Miinuseks on hinnastuspoliitika. Watson maksaks ettevõttele 9200 USA dollarit kuus iga serveri kohta klastris. IBM Graph Standardpakett sisaldab kuni 25000 API päringut kuus ja 500MB kettaruumi tasuta. Iga 1000 päringu eest tuleb maksta 0.2 USA dollarit ja iga ületatud GB eest 15 dollarit. Päringute arvu saab vähendada, kui koguda andmeid perioodiliselt kokku ning saata suurem kogus andmeid korraga (API Reference, Bulk input APIs). Vajaminevat kettaruumi on selles etapis üpris keeruline ennustada.

2.5 Alternatiivide võrldus kriteeriumite kaupa

Analüüsi tulemusena selgus, et Watson ja PredictionIO on teistega võrreldes valmislahendused ning nende tööle saamine võtab märgatavalt vähem aega ja haldamine on lihtsam. Watsoni ja PredictionIO peamised erinevused on hinnastuspoliitika ja see, kas platvorm asub pilveserveris või enda juures. Lähtudes kogutud informatsioonist tuleb kõiki alternatiive omavahel igas kriteeriumis võrrelda ning kanda need otsustusmudelisse. Selle tulemusena tekibki järjestus, mille alusel saab valida Telia TV jaoks sobivaima iseõppiva süsteemi.

2.5.1 Hind

Kuna Watson on ainuke tasuline lahendus, siis kõik teised on selles osas eelistatud ja omavahel võrdsed. Watsoni kasutamisega kaasneb litsentsikulu iga kasutaja kohta, mis Telia TV kasutajate arvu puhul tähendab märgatavat iga-aastast kulu. Seetõttu on kõik teised alternatiivid Watsoni suhtes ekstreemselt eelistatud - hinnang 9 (Joonis 6).

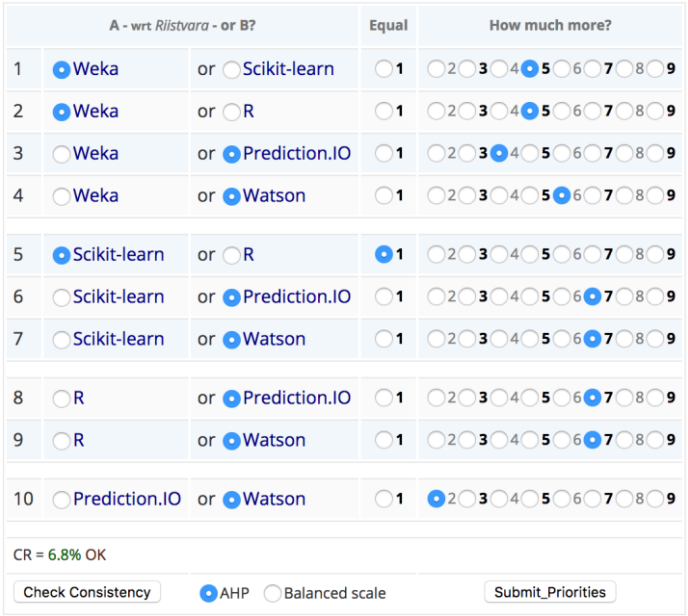


Joonis 6 – Alternatiivide võrdlus hinna suhtes

2.5.2 Tehniline võimekus – riistvara

Osadel alternatiividel selgusid teatud puudused ideaalsest süsteemist: Wekal installeerimises, Scikit-learnil klastrisse ehitamine, R puhul suurte andmemahtudega

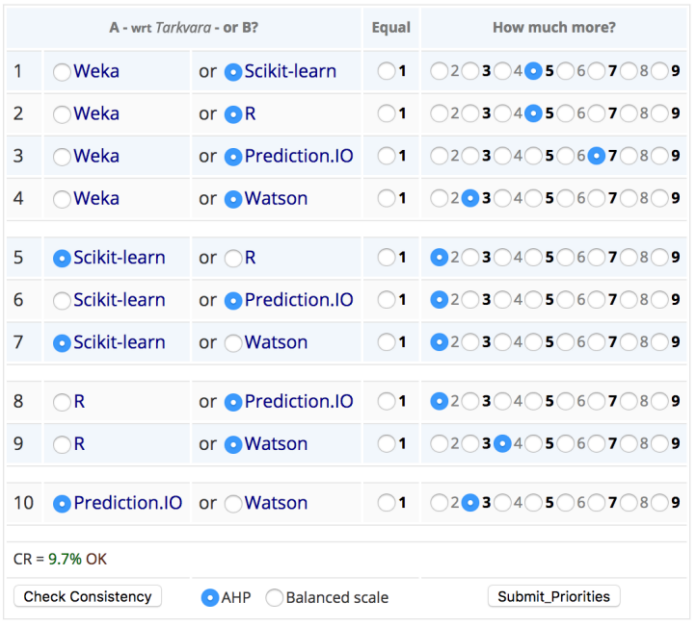
võimalikud probleemid. Watson ja PredictionIO paistavad sobima kõige paremini. Omavaheline võrdluste tulemus on näha joonisel 7.



Joonis 7 – Alternatiivide võrdlus riistvara nõuetele vastavuse suhtes

2.5.3 Tehniline võimekus – tarkvara

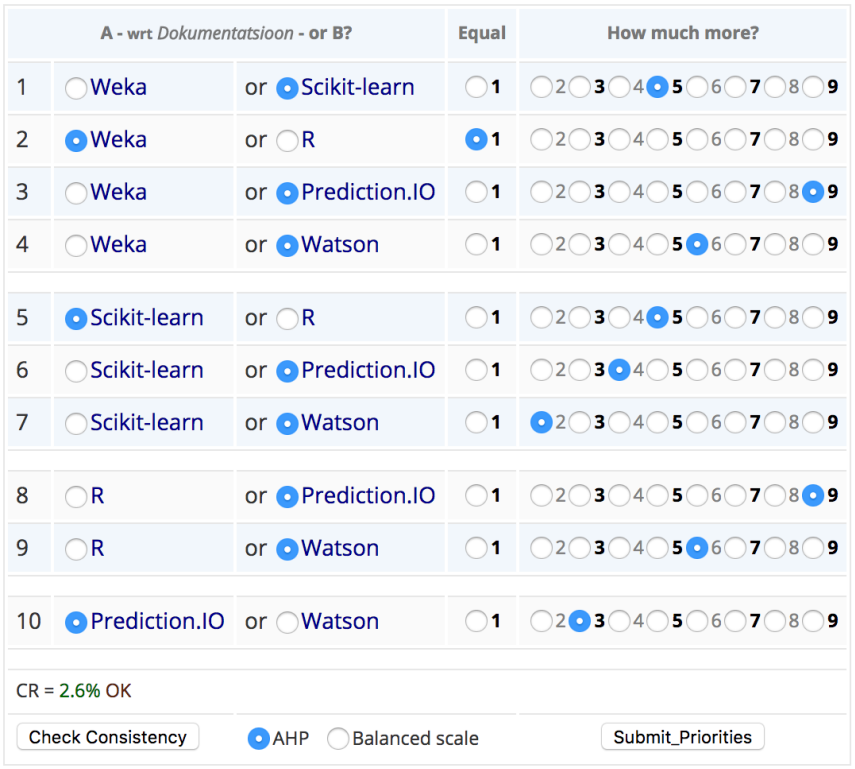
Tarkvaralisi puudusi leidus kõikidel: Wekal väliste süsteemidega liidestumise osas, Scikit-learnil kiirus, R puhul võõras keel S, PredictionIO puhul kohati mallide testide puudulikkus, Watsonil suletud kood. Otsused, millised puudujäägid mõjuvad raskemalt, on kujutatud alloleval joonisel (Joonis 8). Esialgne CR tuli 30%, mille tõttu tuli teha väikseid korrektuure kuni see langes alla 10%.



Joonis 8 – Alternatiivide võrldus tarkvara nõuetele vastavuse osas

2.5.4 Kvaliteet – dokumentatsioon

Kõige selgem ja loetavam dokumentatsioon on PredictionIO oma. Watsoni puhul oli probleeme erinevate süsteemide paljususega, kuid õige koha leidmisel oli juhend arusaadav ja lihtne. Weka ja R dokumentatsioonid on laialipaisatud ja mahukad. Scikit-learn dokumentatsioon on üle keskmise hea struktuuriga ja detailsusega. Tulemused on kantud joonisele (Joonis 9).



Joonis 9 – Alternatiivide võrdlus dokumentasioonide osas

2.5.5 Kvaliteet - koodi kvaliteet

Koodi kvaliteedi paremaks hindamiseks tuleb otsida üles iga projekti koodi repositoorium ning jälgida kui aktiivselt seda kasutatakse.

Weka kood on avalikult üleval[1], kuid vigasid raporteeritakse e-maili teel, avalikult neid ei leia. Scikit-learn repositoorium[2] on väga aktiivne ja sinna tehakse pidevalt täiendusi. R repositooriumis[3] paistab üpris aktiivne tegevus, kuigi viimane versioon on välja lastud üle aasta tagasi. PredictionIO repositoorium[4] on samuti aktiivne ja parandusi tehakse pidevalt. Watsonil on suletud kood ning siin tuleb usaldada IBM brändi ja

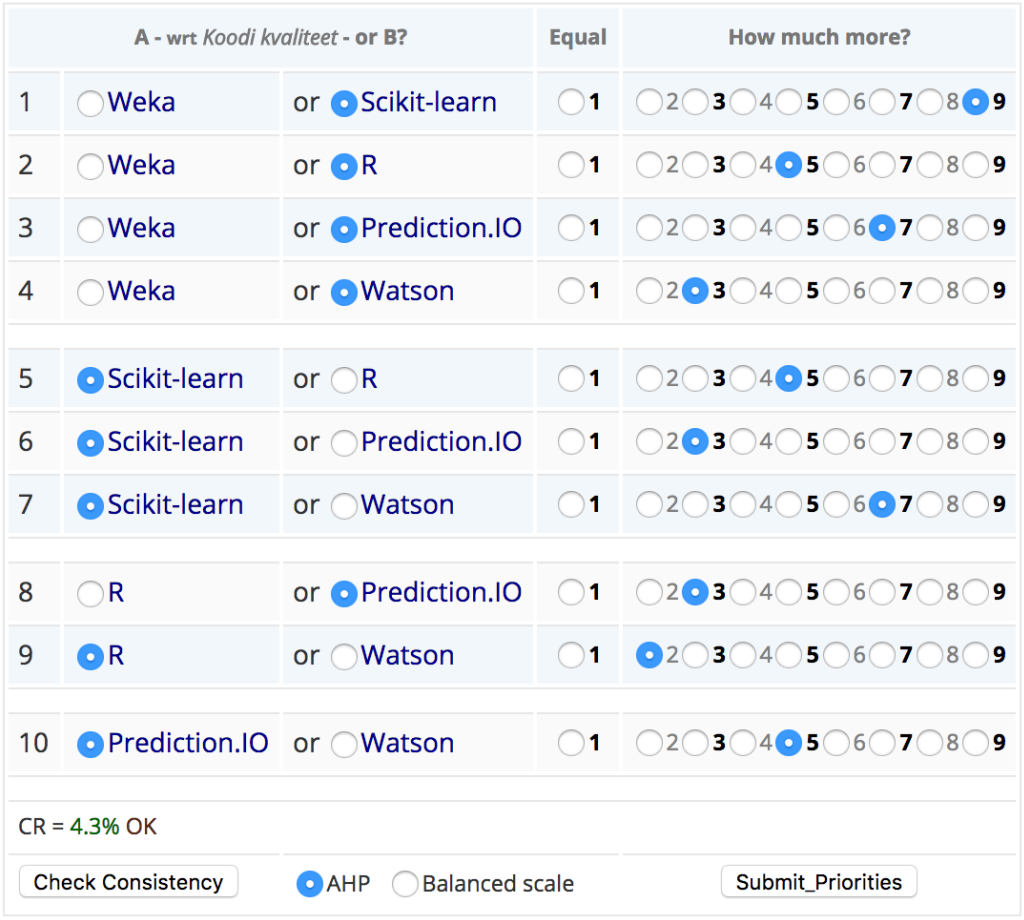
[1] https://svn.cms.waikato.ac.nz/svn/weka/

[2] https://github.com/scikit-learn/scikit-learn

[3] https://github.com/mlr-org/mlr

[4] https://github.com/apache/incubator-predictionio

kvaliteeti. Küsimusi ja vigu näeb selleks eraldi loodud veebilehelt[1]. Võrdlustulemused on kantud joonisele 10.

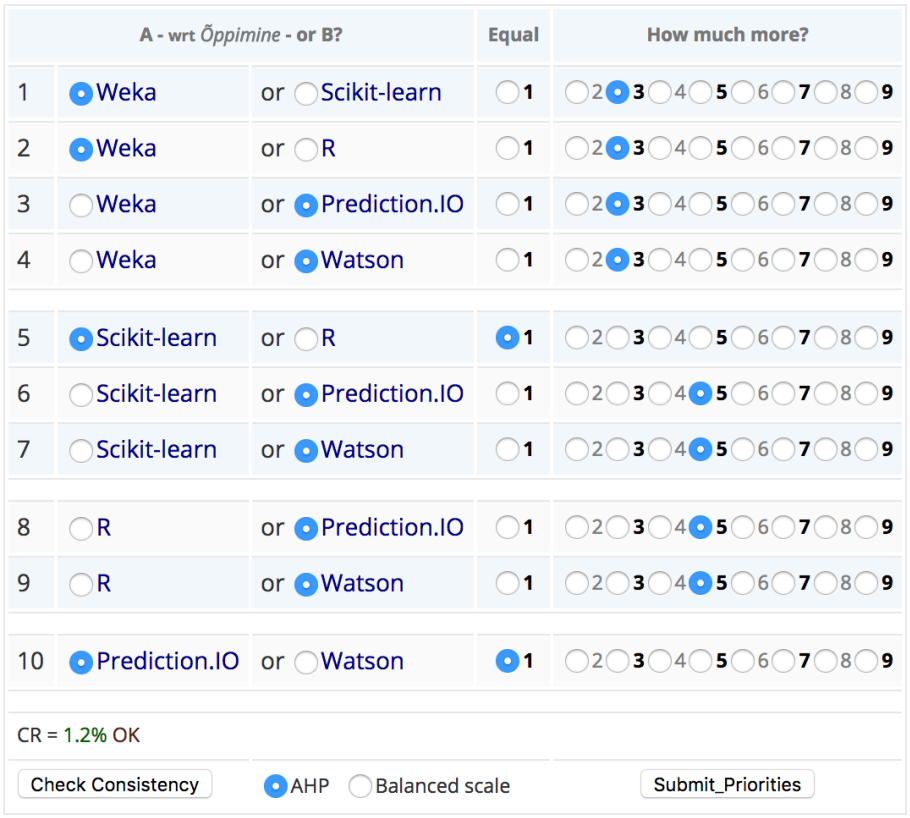


Joonis 10 – Alternatiivide võrldus koodi kvaliteedi suhtes

2.5.6 Keerukus – õppimine

Õppimise keerukus sõltub paljuski arendaja oskustest ja kogemustest erinevate programmeerimiskeeltega. Seetõttu peaks mudeli kasutaja lähtuma siin eelkõige personaalsetest eelistustest. Kuna töö autor kasutab igapäevaselt PHP keelt ja kohati Javascripti, siis eelistatud on PredictionIO ja Watson, kus on vaja vähem koodi kirjutada ning kood on mõnevõrra tuttav. Mõlemal on olemas ka PHP moodul, mis oskab vastava serveriga suhelda. Teiste puhul on natuke eelistatud Java keelt kasutav Weka. Sellest lähtuvalt on koostatud alljärgnevad paaritiste võrldused (Joonis 11).

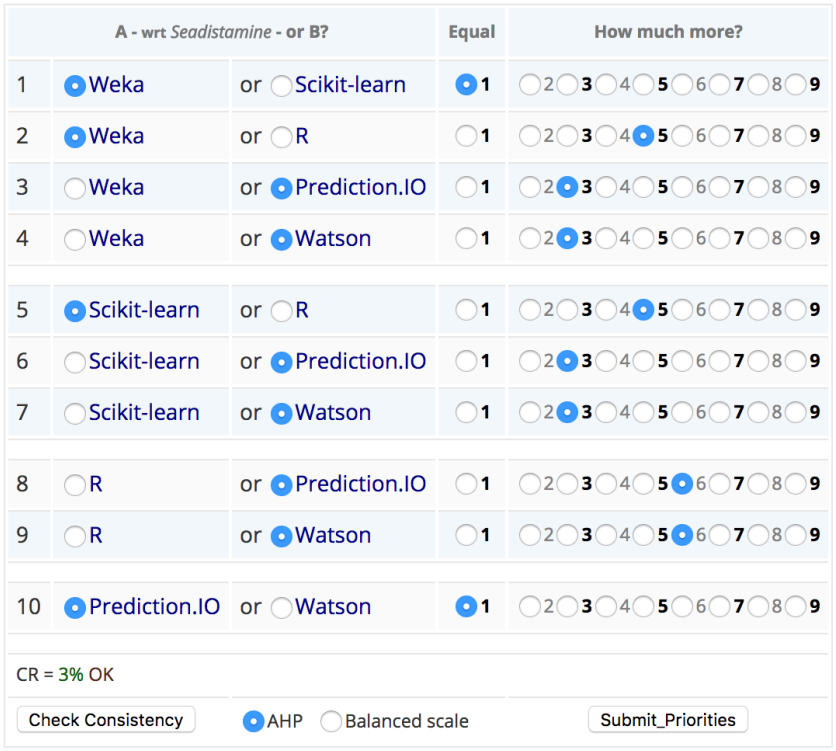
[1] developer.ibm.com/answers



Joonis 11 – Alternatiivide võrdlus õppimise keerukuse suhtes

2.5.7 Keerukus – seadistamine

Weka võimaldab valida erinevate algoritmide vahel ning üpris palju peenhäälestust analüüsimise ja treenimise juures. Scikit-learn omab mitmeid seadistusi ja mõistlikke vaikeväärtusi. R projektil tundub üldkonfiguratsioon olema suhteliselt piiratud, täpsemate seadistuste jaoks tuleb lähemalt uurida. PredictionIO võimaldab valida erinevate komponentide vahel, mis võivad paikneda erinevates serverites. Samuti saab valida erinevate algoritmide vahel ning seadistada põhjalikult soovitusmootori treenimise parameetreid. Samas on vaikeväärtused head ning tüüpinstallatsiooni jaoks pole vaja neid muuta. Ka Watsoni puhul on süsteem ülimalt seadistatav ning seadistused dokumenteeritud. Sellest lähtuvalt on viimased kaks eelistatud ülejäänud alternatiivide suhtes (Joonis 12).



Joonis 12 – Alternatiivide võrdlus seadistamise keerukuse kohta

2.5.8 Haldus – installeerimine

- Weka installeerimine on üpris lihtne ning vajab ainult Javat. Kõik on installeeritav zip faili all tõmbamise ja lahti pakkimise teel[1].

- Scikit-learn paigaldamine nõuab rohkem tegevusi, kuid kõike saab teha käsurealt ning detailne juhend on selleks olemas[2].

- R puhul on vaja rohkem tegevusi, kuid juhendit järgides peaks sellega toime tulema[3].

- Prediction.IO juhendis on välja toodud mitu sõltuvust (Java, Hadoop, Spark, andmebaas), kuid nende paigaldamiseks eraldi juhendit kodulehel pole. Küll aga

[1] http://machinelearningmastery.com/download-install-weka-machine-learning-workbench/

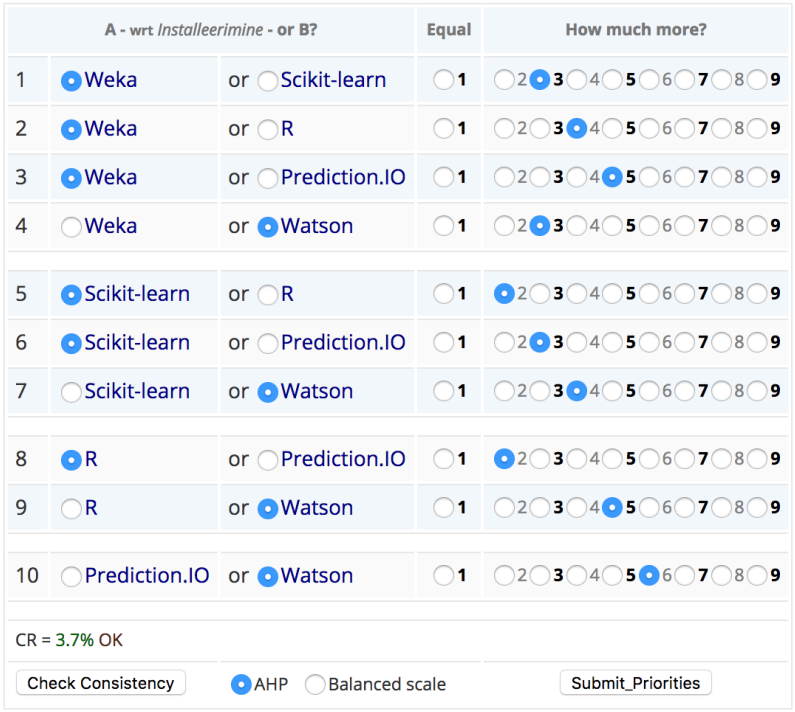
[2] http://scikit-learn.org/stable/developers/advanced_installation.html

[3] https://cran.r-project.org/bin/linux/debian/

on seal kirjeldatud kõik muud tegevused süsteemi paigaldamiseks[1]. Lisaks on olemas Vagrant ja Docker projektid virtuaalserveri loomiseks, kust saab võtta näiteid ka süsteemi SaltStack'is kirjeldamiseks.

- IBM Watsoni puhul ei tule ise midagi installeerida, vaid tuleb sisse logida bluemix kontoga (console.ng.bluemix.net) ning luua sealt uus sobivat tüüpi rakendus. Paigaldamine on lihtne ja juhendid on iga rakenduse juures saadaval.

Lähtuvalt kogutud informatsioonist saab alternatiivide võrdlused sisestada mudelisse (Joonis 13).



Joonis 13 – Alternatiivide võrldus installeerimise kohta

2.5.9 Haldus – monitoorimine

- Weka juhendis logimise ja monitoorimise kohta palju infot esmapilgul ei paista. Internetist leiab juhendi, kus mainitakse, et veebiliidesest pordilt 8085 saab vastavat infot näha[1]. Kasutusele võtmisel tuleb lähemalt uurida.

[1] http://predictionio.incubator.apache.org/install

- Scikit-learn on Pythoni rakendus ning selle monitoorimiseks sobiks näiteks New Relic[2]. TeliaTV puhul on Pythoni monitoorimise vahendid juba olemas.

- R projekti dokumetatsioonist süsteemi monitoorimise kohta midagi ei leia. Eraldi paigaldatavate pakkide nimekirjast[3] võib midagi leida, kuid vajab lähemat uurimist.

- PredictionIO puhul leidub juhendis eraldi sektsioon[4] monitoorimise kohta, kus soovitatakse kasutada Linuxil *monit* tööriista, et jälgida süsteemi töökorda ja staatust. See omab graafilist liidest ja võimaldab ka teistel süsteemidel andmeid töödelda.

- Watsoni monitoorimise jaoks on Bluemix platvormil olemas erinevad tööriistad, mille kasutamine on lihtne - näiteks Availability Monitoring.[5]

Watson on antud juhul kindlasti kõige eelistatum ning Weka ja R nõuavad täpsemat uurimist, PredictionIO ning Scikit-learn jäävad nende vahele (Joonis 14).

---

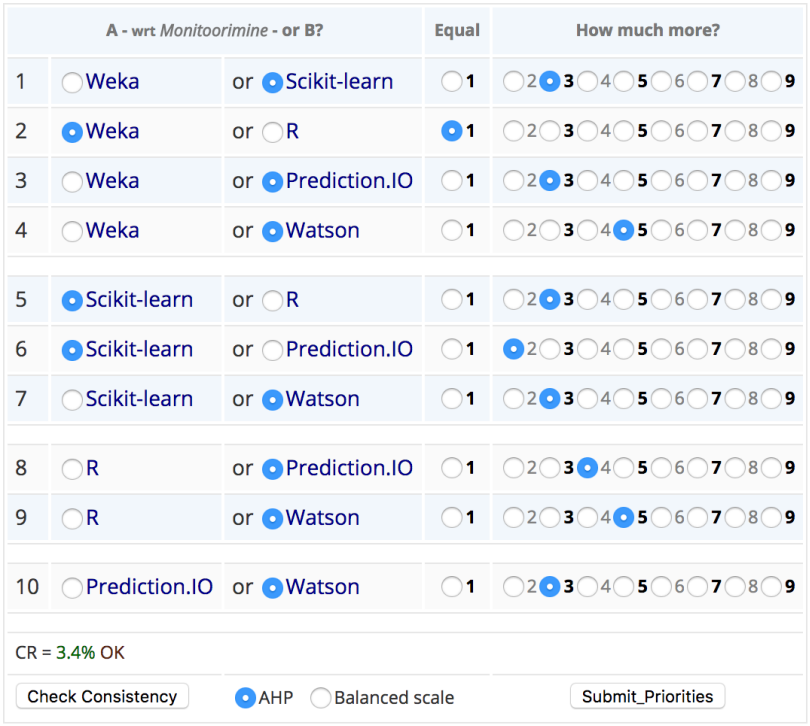
[1] http://wiki.pentaho.com/display/DATAMINING/Weka+Server

[2] https://newrelic.com/python

[3] https://cran.r-project.org/web/packages/available_packages_by_name.html

[4] http://predictionio.incubator.apache.org/deploy/monitoring/

[5] https://console.ng.bluemix.net/catalog/services/availability-monitoring/



Joonis 14 – Alternatiivide võrdlus monitoorimise kohta

2.5.10 Haldus – klastri ehitamine

- Weka puhul saab ehitada erinevaid *master-slave* lahendusi. *Slave* serveri käivitamisel tuleb kaasa anda parameetritena *-master masterHost:masterPort*, kus on *master* serveri andmed.

- Scikit-learni jaoks ei leidu esmapilgul juhendist ega internetist klastri ehitamise kohta midagi.

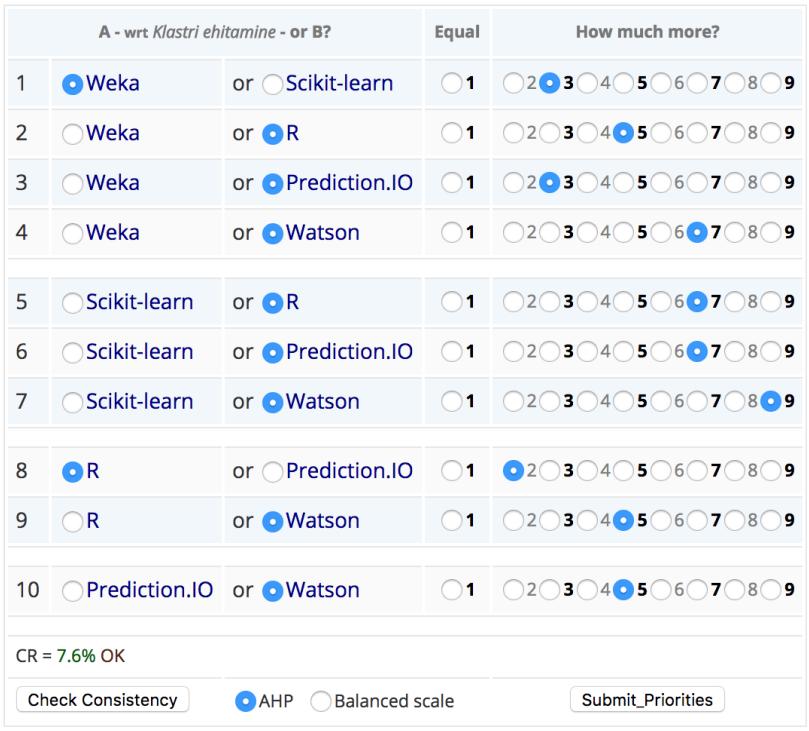
- R projekti jaoks on loodud täiendav tarkvara snowFT[1] (Fault-Tolerant Simple Network of Workstations), mis toetab paralleelsete serverite kasutamist. See tegeleb koormuse jaotamise ja vigade käitlemisega ning on kerge kasutada.

- PredictionIO koosneb mitmest komponendist (Hadoop, Spark, Elasticsearch, HBase), mida saab kõiki omakorda klastrisse ehitada. Selle kohta on ka olemas juhend[1], kuid see vajab põhjalikku uurimist ja rohkem tööd.

[1] https://cran.r-project.org/web/packages/snowFT/snowFT.pdf

- IBM pakub pilveteenust ning seejuures on tagatud ka kõiksugused koormusjaotused ja vea käsitlused. Teenuse eest tuleb muidugi tasuda iga püsti pandud instantsi eest.

Watson on taaskord siin esirinnas, kuna pakub täisteenust, mis sisaldab klastri struktuuri ning on juba sellise eeldusega üles ehitatud. R projektil on olemas valmis pakk, mida saab kasutada ning PredictionIO puhul on olemas paindlik ja tehniline juhend klastri jaoks. Kõik võrdlused on kantud mudelisse (Joonis 15).



Joonis 15 – Alternatiivide võrdlus klastri ehitamise kohta

2.6 Tulemuste kokkuvõte

Pärast kõikide võrdluste mudelisse kandmist (Joonis 16) annab mudel meile paremusjärjestuse valitud alternatiivide jaoks (Joonis 17). Nagu näha said Watson ja PredictionIO võrdse tulemuse 30,9% ning otsuse tegemiseks tuleb kaaluda kriteeriumeid, mis annavad kõige suurema efekti. Kõige suurema kaaluga oli tarkvaraline tehniline võimekus, mis PredictionIO puhul oli parem kui Watsonil. Teine oluline faktor ettevõtte jaoks on kindlasti hind – tasuta lahendust on märksa lihtsam

[1] https://github.com/actionml/docs.actionml.com/blob/master/small_ha_cluster.md

projektiks vormistada, kuna see sisaldab vaid sisemisi arendus- ja haldustöid. Seetõttu saab esmaseks valikuks PredictionIO, mida järgnevas peatükis autor detailsemalt uurima hakkab.

Kindlasti tuleks lugejal arvesse võtta, et kõik võrdlused on tehtud subjektiivsetel hinnangutel ning lähtudes konkreetsetest vajadustest. Mudelit saab alati täiustada valides uusi alternatiive või muutes võrdluste tulemusi.

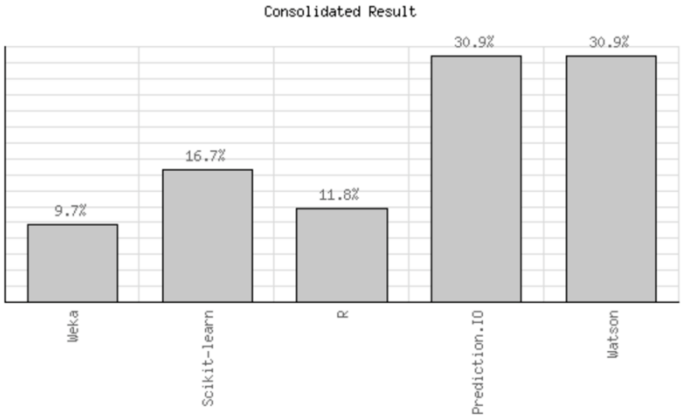
Tänu teostatud analüüsile on võimalik valitud lahendust kergemini kasutusele võtta ning ületamatute takistuste korral valida hoopis IBM Watson või kolmandale kohale jäänud Scikit-learn.

	Criterion	Node	Glb Priorities	Compare	Weka	Scikit-learn	R	Prediction.IO	Watson
1.	Hind	Iseõppiva süsteemi valik	8%	AHP	0.243	0.243	0.243	0.243	0.027
2.	Riistvara	Tehniline võimekus	11.8%	AHP	0.134	0.042	0.042	0.314	0.468
3.	Tarkvara	Tehniline võimekus	35.4%	AHP	0.044	0.244	0.131	0.377	0.204
4.	Dokumentatsioon	Kvaliteet	5.8%	AHP	0.042	0.161	0.042	0.514	0.241
5.	Koodi kvaliteet	Kvaliteet	2.9%	AHP	0.034	0.516	0.118	0.263	0.068
6.	Õppimine	Keerukus	11%	AHP	0.153	0.064	0.064	0.359	0.359
7.	Seadistamine	Keerukus	3.7%	AHP	0.138	0.138	0.041	0.342	0.342
8.	Installeerimine	Haldus	3.3%	AHP	0.266	0.125	0.079	0.051	0.48
9.	Monitoorimine	Haldus	5.3%	AHP	0.071	0.223	0.068	0.18	0.458
10.	Klastri ehitamine	Haldus	12.7%	AHP	0.06	0.03	0.208	0.14	0.561
				Total weight of alternatives:	**0.097**	**0.167**	**0.118**	**0.309**	**0.309**

Done! Go back to hierarchy.

Joonis 16 – Alternatiivide võrdluste kaalud

Result for Alternatives



Ranking for Iseõppiva süsteemi valik:

	Category	Priority	Rank
1	Weka	9.7%	5
2	Scikit-learn	16.7%	3
3	R	11.8%	4
4	Prediction.IO	30.9%	2
5	Watson	30.9%	1

Joonis 17 – Alternatiivide võrdluste lõpptulemus

3 Soovitusmootori detailanalüüs

Parima lahenduse leidmiseks sai koostatud otsustumudel, mis aitab teoreetiliselt teha õiget valikut. Enne kui tehnilise lahenduse kallal tööle saab asuda, on mõistlik uurida lähemalt PredictionIO arhitektuuri ja kasutatavat algoritmi.

3.1 Komponentide arhitektuur

PredictionIO on vabavaraline masinõppe serverlahendus, mis kasutab uusimaid vabavaralisi lahendusi, et luua ennustusmootoreid erinevate masinõppe ülesannete lahendamiseks (PredictionIO Docs). Masinõppe ja andmetöötluse jaoks on kasutusel Spark MLLib ning andmete salvestamiseks MySql, PostgreSQL või ElasticSearch. PredictionIO ise koosneb sündmuste serverist (*event server*) ja mootoritest (*engine*). Rakendused saavad saata andmeid sündmuste serverisse. Mootor saab andmetele ligi nii treenimise käigus kui ka soovitusi välja andes vajadusel neid filtreerides. Kui rakendused küsivad PredictionIO käest ennustust, siis kogutakse iga mootori tulemused kokku ning tagastatakse lõpptulemus rakendusele tagasi (PredictionIO Docs).

Mootorite jaoks on loodud mallid (*Engine Template*), mis täidavad erinevat tüüpi masinõppe ülesandeid. Kõik mootori mallid peavad järgima DASE (Data, Algorithm, Serving, Evaluator) arhitektuuri[1]:

- **D**ata – koosneb kahest komponentist: *Data Source* ja *Data Preparator*. *Data Source* loeb andmeid *Event Store*'st ja tagastab treeningandmed. *Data Preparator* võtab treeningandmed ning teostab erinevat tüüpi andmetöötlust.

- **A**lgorithm - sisaldab kahte meetodit: *train* ja *predict. Train* vastutab ennustusmudeli treenimise eest, mille PredictionIO salvestab pärastiseks kasutamiseks. *Predict* kasutab ennustusmudelit soovituste tegemiseks lähtuvalt päringu andmetest.

- **S**erving – töötleb ennustuse tulemusi ja liidab vajadusel erinevate ennustuste tulemused ühte, kui kasutusel on mitu algoritmi. Siin on võimalik ka andmeid

[1] http://predictionio.incubator.apache.org/templates/recommendation/dase/

filtreerida (näiteks objektide *black-list*). Lõpptulemus muudetakse automaatselt PredictionIO poolt JSON formaati ning tagastatakse rakendusele.

- **E**valuator – moodul, mis võimaldab hinnata soovitusmootori tööd ja tagastab optimaalsed algoritmi parameetrid. Kuna PredictionIO mootor võimaldab valida erinevaid algoritme ning seadistada neid erinevate parameetritega, siis see võimaldab mootori täppisseadistamist.

3.2 Mootori malli analüüs

TeliaTV jaoks on vaja valida sobilik mootori mall, millega saab tutvuda PredictionIO kodulehel[1]. Soovitusmootori jaoks sai valitud soovituste kategooriast levinuim mall *The Universal Recommender* (UR). Osa malle võimaldab kasutada ka muid andmebaase (näiteks SQL), kuid valitud nõuab just ElasticSearch'i, nagu malli dokumetatsioonist[2] saab lugeda.

UR loojad on oma algoritmi ehitamisel lähtunud Ted Dunning'u ja Ellen Friedman'i raamatu „Practical Machine Learning: Innovations in Recommendation" põhjal. Töö autor tutvus samuti selle raamatuga ning uuris UR koodi Githubis.

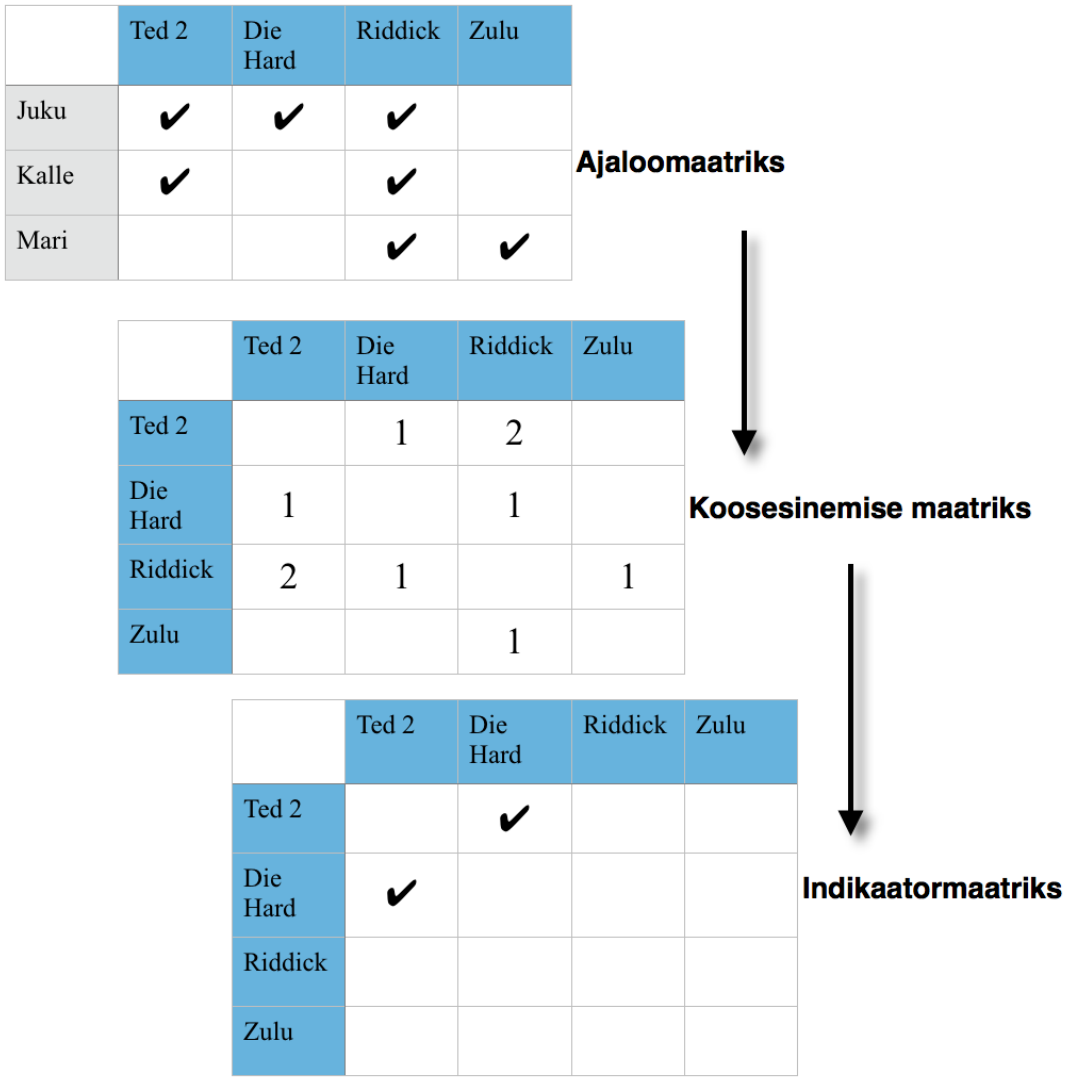
Iseõppivaid süsteeme on viimasel ajal loodud palju, kuid nendelt süsteemidelt on tihtipeale keeruline küsida, millele nende tehtud otsus tugineb. Juhul kui süsteem õpib iseseisvalt suletud süsteemis, võib juhtuda, et uues taustsüsteemis ei suuda see teha õigeid otsuseid. Arendajatel on sellisel juhul väga keeruline välja uurida valede otsuste põhjus. Ka Dunning selgitab oma raamatus, et „keep it simple" on uus mantra *big data* maailmas. Süsteem peab olema võimalikult lihtne, kuid mitte lihtsam. Kui lihtsustamine peab olema tehtud õigesti, siis on süsteem paremini hallatav, jälgitav ja arendatav. (Ted Dunning, 2014).

[1] http://predictionio.incubator.apache.org/gallery/template-gallery/s

[2] https://github.com/PredictionIO/template-scala-parallel-universal-recommendation/blob/master/README.md

3.3 Masinõppe meetodi kirjeldus

Dunning kirjeldab oma raamatu 4. peatükis lihtsa soovitusmootori disainimist. Seda metoodikat kasutab ka Universal Recommender. Esimeses etapis kogutakse andmeid sündmuste kohta kasutajate ja asjade vahel ning kantakse need maatriksisse (History Matrix). Algoritmi treenimise käigus võetakse kõik sündmused ning luuakse koosesinemise maatriks (*co-occurrence matrix*), kus on nii tulpades kui ridades esemed. Maatriksis on välja toodud kui mitu kasutajat omavad sündmust mõlema eseme jaoks. Koosesinemise maatriksist otsitakse välja anomaalsed käitumismustrid ja koostatakse indikaatormaatriks (indicator matrix).



Joonis 18 – Soovitusmootori mudeli koostamine indikaatormaatriksi abil

Ajaloomaatriksisse satuvad andmed sündmuste serveri kaudu ning neid hoiustakse *Event Store*'s (sündmuste serveri andmebaas). Need andmed antakse treenimise ajaks algoritmile kaasa. Antud süsteemis on asjadeks filmid ja sarjad nagu näidisena joonisel näha (Joonis 18). Koosesinemise maatriksi iga kast näitab, mitu kasutajat vaatas mõlemat filmi. See ei ole aga veel piisav alus soovituste tegemiseks. Dunning selgitab, et kõik seosed ei ole alati relevantsed, kuna populaarsed asjad ei ole soovitusteks head infikaatorid vaid üldine käitumismuster. Näiteks kui kui enamus kasutajatele meeldib film „Riddick" (Joonis 18), ei tähenda see, et igale kasutajale peaks kohe soovitama seda vaadata. Kasutaja ajaloost otsitakse huvitavat ja anomaalset käitumismustrit, mis võimaldab teha häid soovitusi. Sellistest koosesinemistest luuakse indikaatormaatriks (*indicator matrix*) (Ted Dunning, 2014). Joonise 19 põhjal tuleks „Ted 2" vaadanud kasutajale Kalle soovitada „Die Hard" vaatamist.

Univeral Recommender paneb iga eseme külge indikaatoritena viited teistele seotud esemetele ning salvestab need ElasticSearch indeksisse. Sellega on treeningu protsess lõppenud.

Soovituste küsimisel koostab Universal Recommender lähtuvalt päringust ja kasutaja ajaloost ElasticSearch päringu. ElasticSearch sorteerib indekseeritud dokumente (antud juhul filme ja nende metaandmeid) relevantsuse ehk kaalude järgi lähtuvalt päringust. Päringu koostamisel otsitakse kasutaja viimased sündmused ning nendega seotud esemed (vaadatud filmid) ja lisatakse need ElasticSearch päringusse, et tõsta selliste esemete kaalusid, millel on indikaatoriteks juba vaadatud filmid. Need samad filmid lisatakse ka filtrina, et juba nähtud filme ei tagastataks. Soovituste leidmisel kasutatakse alljärgnevat valemit[1]:

$r = [P^tP]h_p$

r – soovitused
h_p – konkreetse kasutaja ajalugu mõne põhisündmuse kohta (näiteks *view*)
P – kõikide kasutajate ajalugu põhisündmuste kohta, ridades kasutajad, veergudes objektid

[1] http://www.actionml.com/blog/cco

[PtP] – võrldused veerg-veeru kaupa kasutades tõepärafunktsiooni (log-likelihood) põhist korrelatsioonide leidmist

Juhul kui kasutusel on mitu erinevat sündmust, siis tuleb kasutada ristkorrelatsioonide (Correlated Cross-Occurrence) leidmist:

$r = [P^tP]h_p + [P^tV]h_v + [P^tC]h_c + ...$

Teoreetiliselt saab kõike, mida kasutaja kohta teame, kasutada soovituste parandamiseks – ostud, vaatamised, kategooria eelistused, asukoha eelistused, seadme eelistused, sugu jne.

3.4 Sündmuste liigid

Ajaloomaatriksi salvestamisel kasutab UR väga tugevat põhisündmust, mis selgelt viitab kasutaja eelistustele ning teisejärgulisi sündmusi, mis samuti iseloomustavad kasutaja huvi objektide vastu. TeliaTV puhul on põhisündmuseks filmi või sarja vaatamine ning sündmuse nimeks *„view”*. Mida rohkem on kasutajal põhisündmusi, seda täpsemad on soovitused, kuid teisejärgulised sündmused aitavad samuti kasutajal leida soovitusi, et jõuda põhisündmuseni.

UR ei soovita vaikimisi kasutajale asju, mida ta on juba vaadanud (põhisündmus view eksisteerib kasutaja ja objekti vahel). Probleemne koht on aga sarjade puhul, kus kasutaja ei vaata mitte sarja vaid konkreetset sarja episoodi. Ärinõuete kohaselt peab sari olema alati soovitustes näha ning sarja episoode ei tohi tulla üksikutena tulemustes välja. Selle lahendamiseks tuleb sarja episoodi vaatamisel salvestada põhisündmuse asemel teine sündmus sarja jaoks – *view_episode*. Kui kasutaja vaatab sama sarja erinevaid episoode, siis tekib mitmekordselt sama sündmus, mis kirjutab eelmise üle. Kuna *view_episode* ei ole põhisündmus, siis tulevad sarjad soovitustest ikkagi välja.

3.5 Soovituste küsimine

Universal Recommender võimaldab soovitusi küsida mitmel eri viisil:

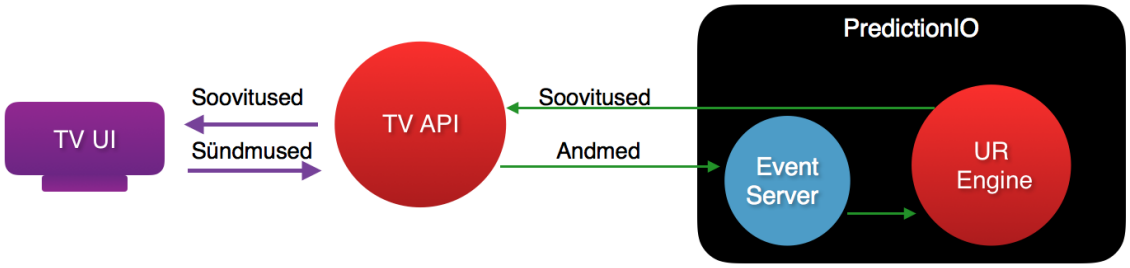
- Populaarsed filmid – päringule ei anta kaasa kasutaja ID (user ID) ega filmi ID (item ID). Selline päring kasutab eraldi meetodit populaarsete filmide leidmiseks, mida saab kasutada anonüümsele kasutajale soovituste tegemiseks.

- Kasutaja soovitused – päringule on kaasa antud kasutaja ID, aga mitte filmi ID. Päring tagastab esiletõstetuna filmid, millel on indikaatoritena küljes kasutaja viimati vaadatud filmid. Seda saab kasutada kasutajale üldsoovituste tegemiseks ning filtrite abil ka kategooriapõhiselt.

- Sarnased filmid – päringule on antud kaasa filmi ID, aga mitte kasutaja ID. Päring tõstab esile filme, millel on valitud filmile sarnased metaandmed ning võimaldab kuvada anonüümsele kasutajale valitud filmiga sarnaseid filme.

- Filmisoovitused – päringule antakse kaasa nii kasutaja ID kui filmi ID. Sisuliselt kombineeritakse kokku kasutaja soovituste ja sarnaste filmide päringud. Võimaldab kasutajale soovitada sarnaseid filme detailvaates või pärast filmi vaatamist arvestades kasutaja käitumisajalugu.

Kokkuvõttes tundub Universal Recommender olevat sobilik algoritm, mida Telia TV jaoks kasutusele võtta, kuna selle abil on võimalik realiseerida esialgu vajaminevad kasutuslood, see toetub konkreetsele allikale ning võimaldab üpris palju seadistamist. Alloleval joonisel on välja toodud ka komponentskeem, kus on näha UI, API ja PredictionIO komponentidevaheline suhtlus (Joonis 19). Sellega on mootori malli analüüs piisavalt detailne, et jätkata järgmine etapiga, milleks on PredictionIO serveri paigaldus ja Universal Recommenderi malli kasutuselevõtt.



Joonis 19 – Telia TV ja PredictionIO üldistatud komponentskeem

4 Tehniline lahendus

Parima lahenduse leidmiseks sai koostatud otsustumudel, mis aitab teoreetiliselt teha õiget valikut. Lisaks sai uuritud lähemalt PredictionIO struktuuri ja Universal Recommender algoritmi malli. Selleks, et veenduda valitud lahenduse tehnilises sobivuses, tuleb tõestada, et Telia TV jaoks on võimalik süsteem minimaalse funktsionaalsusega tööle saada. Sellise eesmärgiga projekti tüüp on TPOC (Technical Proof of Concept) ning see on antud magistritöö peamine ülesanne.

4.1 Tegevuskava planeerimine

Autor sai heakskiidu selline projekt ellu viia. Projekti tunnuseks sai RECOMTPOC. Esimese asjana tuli välja mõelda kasutuslood, need ära kirjeldada ning hinnata tööde maht. Kokku sai loodud 10 kannet (Joonis 20).

T	Key ↑	Summary	Assignee	Reporter	Original Estimate
+	RECOMTPOC-1	Töötundide kapitaliseerimine	Jaak Tamre	Jaak Tamre	
+	RECOMTPOC-2	Mina kasutajana tahan saada isiklikke soovitusi	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-3	Mina kasutajana tahan näha sarnaseid filme	Jaak Tamre	Jaak Tamre	8h
+	RECOMTPOC-4	Mina kasutajana tahan näha soovitusi pärast filmi vaatamist	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-5	Sisestada soovitussüsteemi kõik filmid	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-6	Tagada kasutaja kirje soovitussüsteemis	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-7	Tagada aktiivsete kasutajate kirjed soovitussüsteemis	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-8	Tekitada viimaste tellimuste kohta kirjed soovitussüsteemi	Jaak Tamre	Jaak Tamre	4h
+	RECOMTPOC-9	Filmi vaatamise lõpetamisel salvestada sündmus soovitussüsteemi	Jaak Tamre	Jaak Tamre	2h
+	RECOMTPOC-10	Implementeerida otsingusõna soovitaja	Jaak Tamre	Jaak Tamre	16h

Joonis 20 – RECOMTPOC projekti kasutuslood

RECOMTPOC-1 kanne sai loodud projektijuhtimise, analüüsi ja serveri esmase installeerimisega seotud tööde kapitaliseerimiseks. Ülejäänud kanded sisaldavad süsteemi seadistamist, implementeerimist ning andmete importimiseks vajalike skriptide kirjutamist. Kõige keerulisemaks sai hinnatud otsingusõna soovitaja loomine. Projekti juhtimisel lähtuti agiilsete arendusmetoodikate põhimõtetest ning töid planeeriti sprintide kaupa paralleelselt teiste projektidga. Selleks, et neid arendusi tegema saaks

hakata, tuleb eelnevalt PredictionIO installeerida ja sündmuste server tööle saada. See võimaldab TV baasist importida ajaloomaatriksisse kasutajate ja filmide vahelised sündmised.

4.2 PredictionIO keskkonna üles seadmine

TeliaTV API arenduskeskkond on üles ehitatud virtuaalserveri lahendusele (VirtualBox, Vagrant, Puppet), mistõttu oli loogiline samm ka PredictionIO sarnaselt ehitada. PredictionIO installeerimisjuhendist[1] leiab viite PredictionIO-Vagrant repositiooriumile[2], mis võimaldab soovitusmootori jaoks virtuaalse serveri loomist. Seal paiknev provision.sh skript tõmbab alla https://install.prediction.io/install.sh installeerimise skripti, mis omakorda paigaldab vajaliku tarkvara uude serverisse.

Lisaks Javale installeerib see vajalikud komponendid: Apache Hadoop, Apache Spark HBase, ElasticSearch andmebaas ja Event Server. Antud projekti raames sai välja jäetud serveri klastrisse ehitamine ning kõik sai paigaldatud ühe serveri peale. Vältimaks võimalikke probleeme, tuleks hiljem paigaldada kõik komponendid ükshaaval, kasutades selleks vastavat juhendit[3]. Viimase asjana on vaja alla tõmmata mootori mall Universal Recommendation vaikimisi seadetega.

4.3 TeliaTV API täiendused

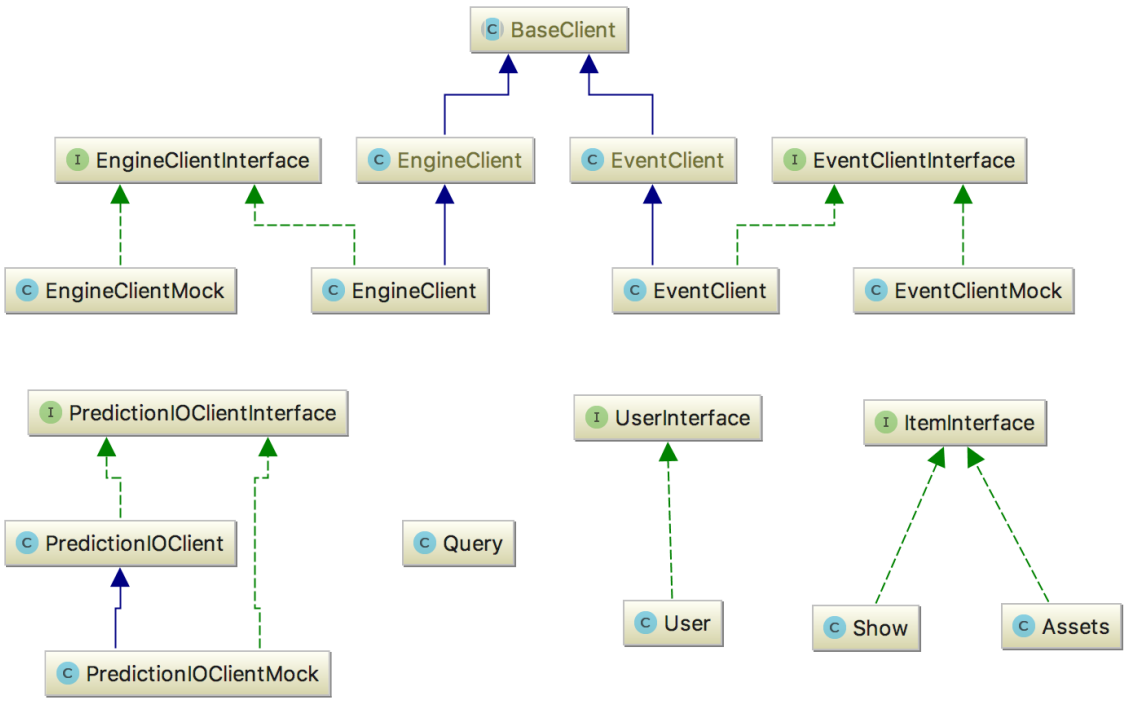
Selleks, et sündmuste serverisse sündmuseid saata, tuleb seda teha üle HTTP või otse failist. PredictionIO jaoks on loodud mitmeid SDK-sid (Java, PHP, Python, Ruby jne), mis võimaldavad sündmusi üle HTTP saata. Olemasolev PHP SDK[4] on sobilik TeliaTV API jaoks, kuid kasutusel oleva Yii2 raamistiku jaoks tuli ehitada sinna peale eraldi kiht abstraheerimaks suhtlust serveriga (Joonis 21). Heleda kirjaga klassid on PredictionIO SDK komponendid, tumedama kirjaga on uued klassid ja liidesed TeliaTV koodibaasis.

[1] http://predictionio.incubator.apache.org/community/projects/#docker-installation-for-predictionio

[2] https://github.com/PredictionIO/PredictionIO-Vagrant

[3] http://predictionio.incubator.apache.org/install/install-sourcecode/

[4] https://github.com/apache/incubator-predictionio-sdk-php



Joonis 21 – Telia TV API jaoks PredictionIO klassidiagramm

PredictionIO'ga suhtlemiseks kasutatakse kahte teenust – sündmuste serveriga suhtleb EventClient ja soovitusmootoriga EngineClient. BaseClient kasutab Guzzle[1] komponenti HTTP päringute tegemiseks. TeliaTV EngineClient ja EventClient laiendavad PredictionIO klasse, et need oleks Yii2 raamistiku poolt konfigureeritavad ning võimaldavad luua lisafunktsionaalsust. Mock tüüpi klassid on loodud automaattestide jaoks, et oleks lihtsam testida komponente, mis kasutavad neid liideseid. PredictionIOClient on vahekihiks, mis otsustab milliseid andmeid sündmuste serverisse salvestada ning kuidas serverist tulevaid tulemusi muuta õigeteks objektideks. API kontrollerid hakkavad kasutama just seda klassi kõikideks tegevusteks PredictionIO serveriga. Esimene selline tegevus on ajaloomaatriksi täitmine sündmuste serveris.

4.4 Andmete import

Sündmuste server võtab vastu erinevat tüüpi sündmusi. EventClientInterface kirjeldab need meetodid, millega saab sündmuste serveriga suhelda:

[1] http://docs.guzzlephp.org/

- setUser($uid, array $properties = [], $eventTime = null) – salvestab kasutaja omadused (properties) sündmuste serverisse. *EventTime* märgib kasutaja loomise aega, vaikimisi praegune ajahetk.

- setItem($iid, array $properties = [], $eventTime = null) – salvestab filmi/sarja omadused (properties) sündmuste serverise. *EventTime* märgib objekti loomise aega, vaikimisi praegune ajahetk.

- recordUserActionOnItem($event, $uid, $iid, array $properties = [], $eventTime = null) – salvestab sündmuse (*view* või *view_episode*) kasutaja ja objekti vahel. Vajadusel saab salvestada lisaparameetreid (näiteks kui palju kasutaja filmi protsentuaalselt vaatas) ning ette anda sündmuse toimumise aja, mis on tagantjärele importimisel vajalik.

4.4.1 Kasutajate import

Esimesena vaatleme kasutajate salvestamist. RECOMTPOC-7 kasutusloo ülesandeks sai määratud aktiivsete kasutajate andmete import sündmuste serverisse. Selle jaoks on vaja EventClient kaudu kutsuda välja setUser meetodit andes ette kasutaja identifikaatori. Universal Recommenderi koodi lähemalt uurides selgub, et kasutaja atribuudid ei mängi mingit rolli antud algoritmi jaoks, mistõttu muid andmeid ei ole vaja kasutaja kohta sündmuste serverisse saata. Kogu protsessi peab vahendama ja kontrollima PredictionIOClient ja vastav *interface*. Selleks lisame uue meetodi setUser (Joonis 22).

```
56      /**
57       * @inheritdoc
58       */
59      public function setUser(UserInterface $user)
60      {
61          $events = $this->getEventClient()->findEvents($user::ENTITY_TYPE, $user->getId());
62
63          if (!empty($events)) {
64              $event = reset($events);
65              return $event['eventId'];
66          }
67
68          return $this->getEventClient()->setUser($user->getId());
69      }
```

Joonis 22 – PredictionIOClient setUser meetodi kood

Lisame UserInterface klassile meetodi getId(), mis peab tagastama kasutaja unikaalse identifikaatori. Kuna sündmuste serverist on võimalik ka sündmuseid küsida parameetri

entityType ja entityId järgi, siis lisame UserInterface ning ItemInterface klassidele ka konstandi ENTITY_TYPE (Joonis 23). Sündmuste küsimine on võimalik sündmuste serverist, kuid predictionIO EventClient seda ei võimalda. Selleks lisame TeliaTV EventClient klassi meetodi findEvents, mis võimaldab seda päringut teha kasutades ülemklassi GuzzleHttp komponenti. See vajab EventClient klassis rohkem muudatusi, et sellele privaatsele komponentdile ligi saaks, kuid antud töö raames ei ole selle välja toomine autori arvates oluline.

```php
<?php

namespace TeliaTV\common\components\PredictionIO;

/**
 * Interface UserInterface
 * @package TeliaTV\common\components\PredictionIO
 */
interface UserInterface
{
    const ENTITY_TYPE = 'user';

    /**
     * @return string
     */
    public function getId();
}
```

Joonis 23 – UserInterface kood

Lõpuks tuleb modifitseerida TeliaTV User mudeli klassi, et see implementeeriks UserInterface'i ja lisame getId() meetodi, mis tagastab id parameetri väärtuse. Nüüd on võimalik luua skript, mis otsib TV andmebaasist välja kõik kasutajad impordib need kutsudes PredictionIOClientInterface kaudu välja meetodi setUser andes ükshaaval User objektid kaasa. RECOMTPOC-6 kande jaoks kulus autoril hinnatud ajast kõigest pool, mis on tingitud sellest, et enamus sai eelnenud töö käigus tehtud.

4.4.2 Filmide ja sarjade import

Peale kasutajate importimise võimekuse loomist saab asuda järgmise etapi juurde, mis on kirjeldatud kasutusloos RECOMTPOC-5. Sarnaselt kasutajatele loome PredictionIOClient ja PredictionIOClientInterface klassidesse uue meetodi setItem(ItemInterface $item). See meetod peab kutsuma välja EventClient küljes

meetodit setItem(iid, properties, eventTime). Identifikaatoriga on keerulisem kui kasutaja puhul – lisame getItemId meetodi ItemInterface liidesesse ning anname selle kaasa. Nii Show kui Assets mudelid tagastavad ID koos unikaalse prefiksiga, et need omavahel konflikti ei tekitaks. Properties parameetriga on asi veelgi keerulisem. Põhjalikuma analüüsi tulemusena salvestatakse sündmuste serverisse järgnevad parameetrid, mis on kirjeldatud allpool väljatoodud tabelis (Tabel 1).

Parameeter	**ItermInterface meetod**	**Võimalikud väärtused**	**Kirjeldus**
itemType	getType()	SHOW, ASSET	Määrab ära kas tegemist on filmi või sarjaga, võimaldab kasutajal filtreerida ühte või teist.
Adult	getIsAdult()	1, 0	Määrab ära kas filmil või sarjal on vanusepiirang.Vaikimisi filtreeritakse piiranguga sisu välja.
packetIds	getPacketIds()	p30, p36, ...	Määrab ära millistesse pakettidesse antud objekt kuulub. Võimaldab filtreerida kasutajal juba tellitud pakettide järgi sisu, mille eest ei pea eraldi tasuma. PredictionIOClient paneb ise prefiksid „p” ID’dele ette.
title_est title_eng title_rus	getTitle($lang)	Filmi/sarja pealkiri kolmes keeles	Võimaldab paremini leida sarnaseid filme pealkirja alusel. Lisaks võimaldab tulevikus rakendada otsingusõna filtrit.
description_est description_eng description_rus	getDescription($lang)	Filmi/sarja kirjeldus kolmes keeles	Sama eesmärgiga, mis pealkiri. Võimaldab leida sisu kirjelduse järgi, kuid väiksema kaaluga kui pealkiri.

publicFrom	getPublicFrom()	ISO8601 formaadis kuupäev	UR engine.json konfiguratsioonis tuleb määrata "availableDateName": "publicFrom". Automaatselt filtreeritakse välja tulevikus kehtima hakkavad filmid.
publicTo	getPublicTo()	ISO8601 formaadis kuupäev	UR engine.json konfiguratsioonis tuleb määrata "expireDateName": "publicTo". Automaatselt filtreeritakse välja kehtivuse kaotanud filmid.
classIds	getClassIds()	c22, c53, ...	Määrab millistesse kategooriatesse antud film/sari kuulub. Võimaldab kuvada soovitusi kategooriate kaupa. Prefiksi „c" lisab PredictionIOClient ise ette.

Tabel 1 – Filmide ja sarjade parameetrid sündmuste serveris

Tekstilised väärtused tuleb millegipärast salvestada massiivina, et filtreerimine tööle saada, kuid numbriliste väärtuste filtreerimine masiivina ei tööta. Samas kui number ei ole esitatud massiivina nagu adult parameetri puhul, siis on tulemused ootuspärased. See tuleneb sellest, et ElasticSearch päring pannakse alati kokku „terms"[1] märksõnaga ning väärtused tekstiliselt. Sellise anomaalia parandamiseks tuleks URAlgorithm.scala koodis ElasticSearch päringu koostamist muuta dünaamilisemaks. Antud hetkel sai probleemist jagu katse-eksitus meetodil.

Väljatoodud meetodid on vaja lisada Show, Assets ja ItemInterface klassidele lähtuvalt tabelis kirjeldatule (Tabel 1). PredictionIOClient peab setItem funktsioonis kõik loetletud parameetrid lisama *properties* muutujasse kutsutes välja vastavaid meetodeid ItemInterface'i küljest. Lõpuks tuleb luua taaskord import skript, mis otsib TV

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-terms-query.html

andmebaasist välja kõik aktiivsed filmid ja sarjad ning lisab need PredictionIOClient kaudu ajaloomaatriksisse. Samuti tuleb filmi või sarja andmete muutmisel seda teha.

Kasutajate importimise puhul oli faktoreid rohkem, millega esialgu ei olnud arvestatud ning filtreerimise tööle saamine vajas lähemat uurimist ja katsetamist. Seetõttu kulus selle peale hinnatud ajast kaks korda rohkem aega. Järgnevalt tuleb ajaloomaatriksi lõplikuks täitmiseks luua kasutajate ja filmide seosed.

4.4.3 Tellimuste import

RECOMTPOC-8 ülesandeks on tekitada viimaste tellimuste põhjal seosed sündmuste serveris. Arvesse tuleb võtta selliseid tellimusi, kus klient on vaadanud vähemalt 50% filmi või episoodi pikkusest. Selleks on EventClient klassil olemas recordUserActionOnItem($event, $uid, $iid, array $properties = [], $eventTime = null) meetod. Loome samasuguse meetodi PredictionIOClient klassi ilma andmete töötlemiseta ning kutsume otse EventClient'i küljest seda välja.

Seoste importimiseks otsime TV andmebaasist välja tellimused, kus filmi on vaadatud 50% selle kestusest ning kutsume PredictionIOClient küljest välja recordUserActionOnItem meetodit. Parameetritena tuleb kaasa anda kasutaja ID, filmi või sarja ID ning tellimuse aeg. Kuna tellimused on tehtud igale sarja episoodile eraldi, siis Assets mudelisse sai lisatud kontroll, et episoodi puhul tagastab getItemId hoopis sarja ID.

Selleks, et teada sündmuse nime, kas view või view_episode, sai lisatud ItemInterface klassile juurde getEventType meetod, mis Show objekti puhul on alati view_episode, kui Assets objekti puhul on vastavalt view (filmi korral) või view_episode (sarja episoodi korral). RECOMTPOC-9 raames lisame recordUserActionOnItem meetodi välja kutsumise filmi vaatamise lõpetamise päringusse. See tagab, et iga vaatamise järel salvestataks sündmus samuti PredictionIO'sse ning kasutajad ei näeks juba vaadatud filme soovituses. Lisaks mõjutavad need sündmused jooksvalt soovitusi, kuna kasutaja ajalukku tekivad uued indikaatorid.

Kõigi kolme skripti valmimise järel tuleb need käivitada ning sündmused arenduskeskkonnas sündmuste serverisse importida. Kui see on tehtud, tuleb

PredictionIO masinas käivitada järgnevad käsklused vastavalt juhendile (PredictionIO Docs):

- pio build – ehitab lähtuvalt engine.json konfiguratisoonist ja mootori malli failidest algoritmi mudeli. Seda peab tegema iga kord kui algoritmi kood või konfiguratsioon muutub
- pio train – kutsub algoritmil välja train() meetodit, mis ehitab indikaatormaatriksi ja populeerib andmed ElasticSearch andmebaasi.
- pio deploy & - käivitab treenitud soovitusmootori instantsi ja paneb protsessi taustale (ampersand).

Nüüd, kui soovitusmootor töötab, on võimalik hakata kasutama EngineClient komponenti, mis oli eelduseks mitmele kasutusloole.

4.5 API ja UI vaheline liidestus

UI (kasutajaliides) peab saama küsida API käest erinevaid soovitusi, mis on lahti löödud kasutuslugudeks RECOMTPOC-2 (isiklikud soovitused), RECOMTPOC-3 (sarnased filmid ja RECOMTPOC-4 (filmisoovitused). Järgnevaid arendusi vaadeldakse koos, kuna kõiki soovitusi saab küsida Universal Recommenderi käest samamoodi. Erinevus tuleb sisse sellest, kas anda kaasa user ID ja item ID või mitte.

Soovituste küsimiseks tuleb välja kutsuda EngineClient *sendQuery(array $query)* meetodit. Päringuna võtab see massiivi ning konverteerib sellest ElasicSearch päringu. Toetatud parameetrid on kirjeldatud Engine.scala Query klassis[1]. Selleks, et päringu koostamine oleks rakenduse poolel lihtsam sai loodud API poolel eraldi Query klass, mida PredictionIOClient getRecommendations meetod vastu võtab. Query klassile lisame meetodi toArray(), mis tagastab massiivi UR algoritmile sobival kujul. Query klassi tuleb lisada erinevad parameetrid nagu *count, userId, itemId, itemType, adult, classIds, packetIds, keyword*. Need võimaldavad päringu koostajal lisada erinevaid filtreid, määrata tulemuste arvu ja kasutaja ID ning filmi ID'd.

[1] https://github.com/pferrel/template-scala-parallel-universal-recommendation/blob/master/src/main/scala/Engine.scala#L32

PredictionIOClient meetodi getRecommendations (Joonis 24) väljakutsumisel päritakse andmed PredictionIO'st, käiakse tulemused üle ning vaadatakse iga tulemust, kus sisaldub objekti tüüp ja ID. Iga tüübi ID'd kogutakse kokku ning päritakse välja nende mudelid. Selleks on tarvis ItemInterface'le sh Show ja Assets klassidele lisada meetod createFromIds. Kõik objektid tagastatakse lõpuks samas järjekorras, milles nad PredictionIO'st tulid. Seega tagastab getRecommendations meetod alati masiivi ItemInterface tüüpi objektidest, mis teeb andmete töötlemise ülejäänud koodis lihtsamaks. Joonisel 24 on osa koodi peidetud ning veakäsitlus ja logimine eemaldatud vältimaks funktsiooni liigset pikkust.

```php
/**
 * @param Query $query
 * @return ItemInterface[]
 * @throws PredictionIOAPIError
 */
public function getRecommendations(Query $query)
{
    $response = $this->getEngineClient()->sendQuery($query->toArray());

    if (empty($response['itemScores'])) {
        return []; // No results were found
    }

    // Group item IDs by item type
    $itemIdsByType = [];
    foreach ($response['itemScores'] as $itemScore) {...}

    // Find all items of the given type in a single query
    $itemsByType = [];
    foreach ($itemIdsByType as $type => $itemIds) {
        /** @var ItemInterface $class */
        $class = $this->itemClassMap[$type];
        if (new $class instanceof ItemInterface) {
            foreach ($class::createFromIds($itemIds) as $item) {
                $itemsByType[$type][$item->getId()] = $item;
            }
        }
    }

    /** @var ItemInterface[] $items */
    $items = [];

    // Map ItemInterface objects to scored results to retain returned sorting
    foreach ($response['itemScores'] as $itemScore) {...}

    return $items;
}
```

Joonis 24 – PredictionIOClient getRecommendations meetodi näidiskood

Kui soovituste küsimiseks on eeldused loodud, tuleb lisada uus API päring, mis tagastab soovitusi. Olgu selleks RecommendationsController, mille lihtsustatud kood on allpool väljatoodud (Joonis 25). Esialgne versioon tagastab soovitusi kindlatest kategooriatest, mis on konfigureeritav (*recommendationsClassIds*), et luua tasulisse videolaenutusse „Telia soovitab" kõrvale uus sektsioon „Minu soovitused", kus on sarnane sisu personaalselt sorteeritud. Lisaks on välja filtreeritud täiskasvanute sisu *adult* parameetriga. Joonisel on näha kuidas Query objekti loomisel saab anda parameetrid ette ning vajadusel hiljem neid juurde lisada või muuta otse objekti küljest. Query konstruktor käib kõik masiivi elemendid läbi ning omistab nende põhjal objekti parameetrid. Tulemuse väljastamisel muudetakse tagastatud massiiv json formaati. Nii Asset kui Show objektid implementeerivad JsonSerializable ning otsustavad ise milliseid parameetreid UI'le tagastada.

```php
class RecommendationsController extends ApiController
{
    protected $pioClient;

    public function __construct($id, Module $module, PredictionIOClientInterface $predictionIO, array $config = [])
    {
        $this->pioClient = $predictionIO;
        parent::__construct($id, $module, $config);
    }

    /**
     * @return array
     * @throws HttpException
     */
    public function actionGet()
    {
        try {
            $query = new Query([
                'count' => $this->getRequest()->get( name: 'limit',  defaultValue: 25),
                'adult' => ItemInterface::ADULT_NO, // exclude adult content
                'classIds' => Yii::$app->params['recommendationsClassIds'], // include only configured classes items
            ]);

            // Add userId if session exists
            if ($session = $this->getSession()) {
                $query->userId = $session->user_id;
            }

            // Add itemId if given
            if ($itemId = $this->getRequest()->get( name: 'itemId')) {
                $query->itemId = $itemId;
            }

            $recommendations = $this->pioClient->getRecommendations($query);

            return [
                'items' => $recommendations,
            ];
        } catch (PredictionIOAPIError $e) {
            throw new HttpException( status: 503,  message: 'Service Unavailable');
        }
    }
}
```

Joonis 25 – RecommendationsController näidiskood

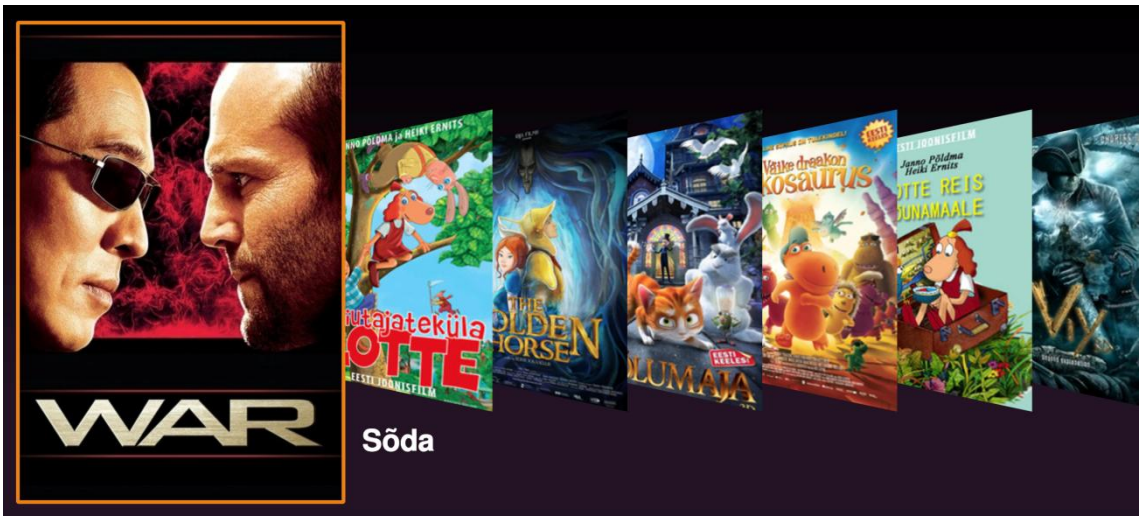
Valminud koodi saab käivitada näiteks päringuga:
GET /recommendations?limit=50&itemId=ASSET_1234

Järgmise sammuna ongi see päring kasutusele võtta UI koodis, et arendused valmis saaks lugeda.

4.6 UI täiendused

Nagu varasemalt sai mainitud, siis oli plaanis uus soovitusmootor esialgu kasutusele võtta uue sektsiooni näol tasulises videolaenutuses. Selleks tuli UI koodist üles otsida koht, kus kuvatakse „Telia Soovitab" kategooria filme (toimetajate poolt käsitsi märgitud) ning sinna kõrvale luua sarnaselt uus sektsioon, mille andmed võetakse recommendations päringuga GET /recommendations, mis tagastab vaikimisi 25 tulemust kasutaja eelistusi arvestades. Juhul, kui päringu vastuseks tuleb viga, siis kuvatakse kliendile teavitust, et soovitusi ei leitud ning kuvatakse Telia soovitusi. Kui kasutaja uuesti sinna navigeerib, üritatakse päringut korrata. Menüüpunktide nimed said kooskõlastatud ärijuhtidega ning muudetud ära „Uued filmid" ja „Filmisoovitused" (Joonis 26). Filmisoovituste avamisel kuvatakse nimekiri filmidest samamoodi nagu Uued filmid puhul (Joonis 27).

Joonis 26 – Filmisoovituste menüüpunkt



Joonis 27 – Filmisoovituste kategooria vaade

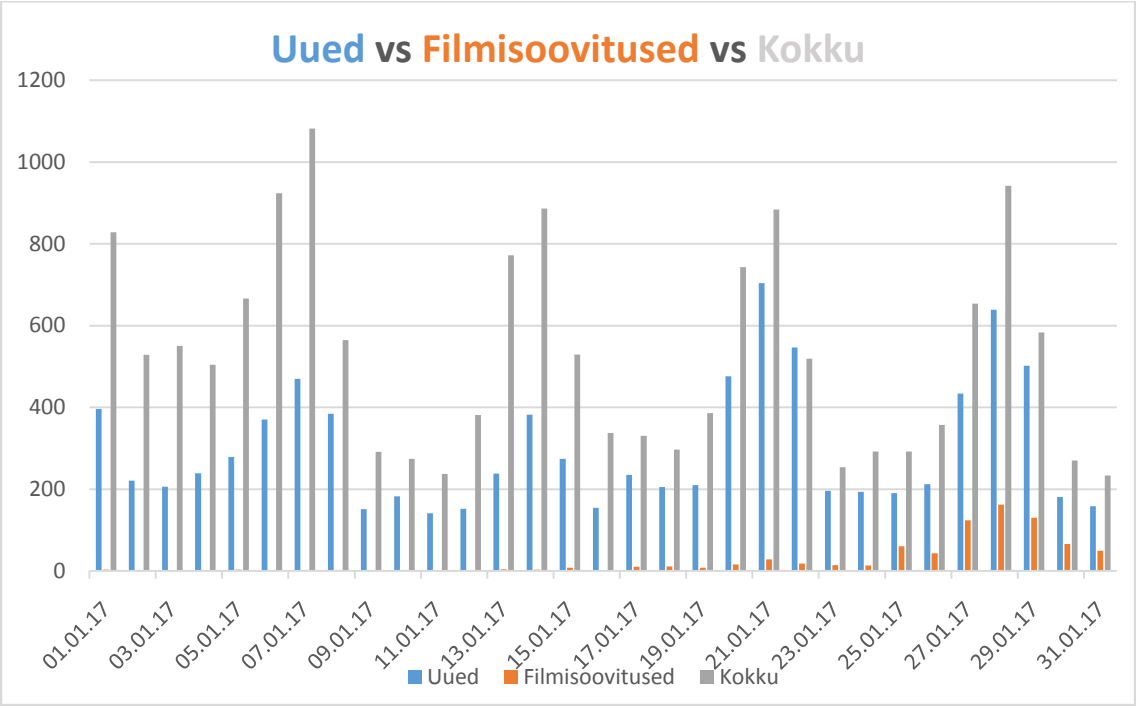
Kui kasutaja lõpetab filmi vaatamise, siis uuendatakse soovituste nimekiri sama päringuga. Soovituste nimekiri võib muutuda, kuna juba vaadatud filmid filtreeritakse välja ning indikaatorite nimekirja täienemise tõttu võib järjekord muutuda.

Pärast arenduste valmimist sai ärijuhtidele demonstreeritud lahendust ning tutvustatud konseptsiooni ja visiooni. Projekt loeti edukalt lõppenuks ning otsustati jätkata valminud lahenduse kasutamist ja edasiarendust.

4.7 Tulemused

Pärast klientidele uue funktsionaalsuse avalikuks tegemist sai võrreldud omavahel „Uued filmid" ja „Filmisoovitused" alt käivitamiste arvu. Esimese hooga käivitati umbes 65% uute filmide kategooriast, pea 20% filmisoovituste alt ning ülejäänud kategooriatest kõigest 15% (Tabel 2). See tähendab, et filmisoovitused on algusest peale efektiivsem viis sobiva filmi leidmiseks kui lihtsalt kategooriates ringi sirvides.

Kindlasti üks oluline featuur, mille järgi kasutajad puudust tunnevad, on sisu otsing. API poolel sai esmased arendused selle jaoks küll tehtud, kuid esimesest skoobist otsustati see välja jätta, kuna see vajab UI poolel suuremaid muudatusi ja disaineri kujundust. Järgmine samm ongi uue sisu vaate loomine, kus soovitusmootor mängib suuremat rolli ning võimaldab kasutaja jaoks olulist sisu paremini leida.



Tabel 2 – Filmide käivitamise statistika 2017 jaanuar

5 Kokkuvõte

TeliaTV on pikka aega vajanud lahendust sisu paremaks leidmiseks. Selleks uuris ja analüüsis autor erinevaid iseõppivate ja masinõppe süsteemide lahendusi. Sobiva süsteemi valiku tegemiseks sai kasutatud Saaty meetodi abil koostatud otsustusmudelit. Käesoleva lõputöö esimene peamine ülesanne oli, et saaks tehtud parim otsus kasutades selleks täppismeetodeid. Otsuse vastuvõtmine oli sellevõrra keerulisem, et IBM Watsoni ja Apache PredictionIO lahendused said võrdse tulemuse. Autor langetas otsuse PredictionIO kasuks, kuna see oli vabavaraline ja tasuta võrreldes Watsoniga.

Peale otsuse langetamist sai analüüsitud PredictionIO'd detailsemalt, valitud sobilik algoritm Universal Recommender ning tutvutud selle põhimõtetega. UR kasutab Ted Dunningu raamatu põhjal korrelatsioonide risttabelit soovituste ennustamiseks.

Pärast analüüsi sai PredictionIO server tööle pandud ning võetud TeliaTV API jaoks kasutusele EventClient ja EngineClient. Nendele lisaks sai loodud liidesed ja klassid, mis võimaldavad importida filmid, sarjad, kasutajad ning nendevahelised seosed sündmuste serverisse ja küsida soovitusmootorilt erinevat tüüpi soovitusi. Olemasolevasse TeliaTV videolaenutusse sai loodud uus kategooria „Filmisoovitused", mille andmed küsitakse API vahendusel soovitusmootorist. Kõik püstitatud eesmärgid tehnilise võimekuse tõestamiseks said täidetud peale otsingu lisamise, sest see oleks vajanud põhjalikumat analüüsi ja disaini. Projekti tulemus sai kaitstud TV valdkonna juhtide ees demonstreerides valminud lahendust ja tutvustades selle kontseptsiooni. Idee ning edasised arendused said heakskiidu ning PredictionIO lahendus jätkab soovitustega ka edaspidi. Peamine on see, et ka klientidele on uus funktsionaalsus korda läinud, nagu näitas filmide käivitamise statistika.

Kokkuvõttes võib lugeda projekti edukalt lõppenuks, kuna tehniline lahendus töötas, kasutuskogemus paranes ning juhtide poolt anti positiivset tagasisidet. Autorile oli see paljudes aspektides uueks kogemuseks, võimaldas rakendada ülikoolis õpitut ning arendada analüütilist mõtlemist.

Kasutatud kirjandus

Brownlee, J. (16. 06 2016. a.). *How to Download and Install the Weka Machine Learning Workbench*. Kasutamise kuupäev: 07. 03 2017. a., allikas Machine Learning Mastery: http://machinelearningmastery.com/download-install-weka-machine-learning-workbench/

Documentation of scikit-learn 0.18. (2016). Kasutamise kuupäev: 07. 03 2017. a., allikas Scikit Learn: http://scikit-learn.org/stable/documentation.html

Getting started with IBM Graph. (kuupäev puudub). Kasutamise kuupäev: 12. 03 2017. a., allikas IBM Bluemix Docs: https://console.ng.bluemix.net/docs/services/graphdb/index.html

Getting started with Machine Learning. (kuupäev puudub). Kasutamise kuupäev: 12. 03 2017. a., allikas IBM Bluemix Docs: https://console.ng.bluemix.net/docs/services/PredictiveModeling/index.html

Ofcom. (07 2016. a.). *TV viewing annex.* Kasutamise kuupäev: 01. 03 2017. a., allikas PSB Annual Research Report 2016 : https://www.ofcom.org.uk/__data/assets/pdf_file/0024/69720/annex-b.pdf

PredictionIO Docs. (kuupäev puudub). Kasutamise kuupäev: 08. 03 2017. a., allikas PredictionIO: http://predictionio.incubator.apache.org/index.html

Ted Dunning, E. F. (2014). *Practical Machine Learning: Innovations in Recommendation.* (M. Loukides, Ed.) United States of America: O'Reilly Media, Inc.

Television. (2017). Kasutamise kuupäev: 01. 03 2017. a., allikas Wikipedia: https://en.wikipedia.org/wiki/Television

The R Manuals. (kuupäev puudub). Kasutamise kuupäev: 08. 03 2017. a., allikas R Project: https://cran.r-project.org/manuals.html

Weka Documentation. (kuupäev puudub). Kasutamise kuupäev: 07. 03 2017. a., allikas Weka Machine Learning: http://www.cs.waikato.ac.nz/ml/weka/documentation.html