
Filmanbefalinger til en gruppe ved collaborative filtering

- P2 projekt -

Projektrapport
Software, Gruppe B2-24

Aalborg University
Department of Computer Science



Software
Aalborg Universitet
<http://www.aau.dk>

AALBORG UNIVERSITET STUDENTERRAPPORT

Titel:

Recommender systems - filmanbefalinger til en gruppe ved collaborative filtering

Tema:

Et større program udviklet af en gruppe

Projektpериode:

Forårs Semester 2020, 03/02 - 27/05

Projektgruppe:

sw2b2-24

Deltager(e):

Christian Bager Bach Houmann
Christopher Colberg Jensen
Kristian Morsing Pedersen
Louise Amanda Jensen
Philip Christian Greve
Rune Holmgaard Andersen

Vejleder(e):

Ming Shen

Oplagstal: 1**Sidetal:** 67**Afleveringsdato:**

27/5-2020

Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Abstract:

Recommender systems is an increasingly used feature in modern commercial applications, but proposes several challenges. One of those is the recommendation of an item, such as a movie, to a group of individuals. This problem will be the main focus of the report, and through memory based collaborative filtering, more precisely item-item filtering, propose a solution for generating a movie recommendation for a small group. The method consists of Pearsons Correlation and an extensive algorithm that combines multiple individuals preferences, for which an additive utilitarian strategy combined with least misery strategy will be used. The output is a top 10 list of movies the group could watch. To back up the solution, an analysis of different collaborative filtering methods is presented alongside datahandling. This solution is implemented as an webapplication created using Javascript, NODE.js, CSS and HTML5.

Indhold

1 Indledning	2
1.1 Initierende problemstilling	3
2 Recommender Systems	4
2.1 Introduktion til Recommender Systems	4
2.2 Filtreringsmetoder	5
2.2.1 Content based filtrering	5
2.2.2 Collaborative filtrering	5
3 Analyse af nuværende anvendelser	9
3.1 Spotify	9
3.2 Netflix	10
3.3 Amazon	13
3.4 Opsamling og valg af film-domænet	15
3.5 Data i recommender systems	16
3.5.1 Eksplisitte og implicitte data	16
3.5.2 Kontekstuel information ift. film-recommendations	17
4 Gruppe recommender systems	19
4.1 Anbefalinger fra individ til gruppe	19
4.2 Collaborative filtrering tilgang	20
4.2.1 Memory-based tilgang	20
4.2.2 Grupperingsstrategier	22
5 Problemdefinering	24
5.1 Problemafgrænsning	24
5.2 Problemformulering	24
6 Design	25
6.1 En webapplikation som løsning	25
6.2 Kravspecifikationer	26
6.3 Anbefalingsalgoritmen	27

6.3.1	Matematisk definition	31
6.3.2	Plan for implementation	32
7	Implementation	34
7.1	Værktøjer	34
7.2	Overblik over projektressourcerne	35
7.3	Programflow	35
7.3.1	Server start	36
7.3.2	Film anbefalingssiden	36
7.3.3	Bruger-system	36
7.4	Datagrundlag og data	39
7.4.1	Databasevalg	39
7.4.2	Indlæsning af data	39
7.4.3	At bygge databaserne	40
7.4.4	Mangel i datasæt	41
7.5	Gruppeanbefalinger	42
7.5.1	At lave en gruppeanbefaling	42
7.5.2	Database og hukommelsesbrug	44
7.6	UI design	45
8	Test	47
8.1	Vores tilgang til test	47
8.2	Test framework	47
8.3	Udvalgte tests med Jest	48
8.3.1	Test af gruppeanbefaling	48
8.3.2	Test af data	50
8.4	Test processen	51
8.4.1	Højt RAM forbrug	51
8.4.2	Fejl i output	52
8.4.3	Poster fejl med Vue.js	53
8.5	Brugertest	53
9	Diskussion	55
9.1	Kravspecifikationer	55
9.2	Forbedringsmuligheder	57
9.3	Anbefalingsalgoritmen	57
9.4	Skaleringsmuligheder	58
9.5	Opfyldelse af læringsmål	59
9.5.1	Viden	59
9.5.2	Færdigheder	59
9.5.3	Kompetencer	59

10 Konklusion	61
11 Perspektivering	63
Bibliografi	65
Bilag	66
Bilag A: Kildekode	67
Bilag B: Programflow	67
Bilag C: Survey	67

Forord

Rapporten er et semesterprojekt for 2. semester udarbejdet af gruppen sw2b2-24 på Aalborg universitet. Projektet opstod, som følge af et projektkatalog og en generel interesse for recommender systems. Det synes derfor oplagt for gruppen at beskæftige sig med et ikke særlig betrådt område, som endte ud i et recommender system til grupper.

Projektet er udarbejdet samløbende med kurserne Sandsynlighedsteori og Lineær algebra (SLIAL), Algoritmer og datastrukturer (ALG) og Internettværk og webprogrammering (IWP). Der skal sendes et tak til Ming Shen, som har fungeret som vejleder for gruppen.

For maksimalt udbytte og forståelse henvises der til de elektroniske bilag.

Kapitel 1

Indledning

Recommender systems bliver i dag brugt i høj grad, og i mange forskellige kontekster. Hvad enten om det er Amazon, der anbefaler et produkt, du måske er interesseret i, eller om det er Netflix, som anbefaler dig en ny film eller serie. Recommender systems er meget interessante, fordi de i høj grad benyttes i forskellige situationer, såsom at anbefale produkter, film, musik, rejser og meget mere.

Denne rapport tager udgangspunkt i at undersøge recommender systems til grupper, specifikt med fokus på film som domæne. Produktet er en platform, hvor individuelle brugere kan rate film blandt et omfattende katalog, tilføje venner til deres venneliste og få en samlet gruppeanbefaling på baggrund af deres profiler.

Vi indleder rapporten med en beskrivelse af recommender systems i et teknisk og samfundsmaessigt kontekst, og ønsker herefter at behandle de forskelle tilgange til filtrering, som collaborative- og content-based filtering. I vores problemanalyse kigger vi på nuværende anvendelser af teknologien og afgør hvordan processen, at anbefale en film til en gruppe, kan udvikles.

Med projektet er forbundet en del studiemæssige mål og kompetencer vi skal erhverve, og recommender systems fungerer i denne sammenhæng som omdrejningspunkt for opfyldelse af disse mål og kompetencer. Det er fastlagt at der som løsning ønskes et større program, web-applikation, hvor programmeringen skal laves i JavaScript. Hertil bruges der egnede extensions og frameworks, som er researchet og valgt ifm. problemløsningen. Der vil også blive lagt vægt på test af produktet, samt opnåelse af færdigheder indenfor versionskontrol og det at styre et projekt.

Først beskriver vi i et designafsnit de krav, der stilles til vores produkt, samt det koncept vi sigter mod at udvikle. Centralt er den anbefalingsalgoritme, som vi både matematisk præsenterer og viser flowcharts til. Efterfølgende kommer et implementationafsnit der dækker eksempler og udvalgte dele af implementationen i JavaScript og NODE.js. Forinden konklusionen er der et diskussionsafsnit, hvor vi diskuterer det udviklede produkt ift. målene for det.

1.1 Initierende problemstilling

I dag benytter mange virksomheder og online services recommender systems til at anbefale brugerne flere af deres egne produkter, og services, for primært at skabe vækst, og tiltrække flere brugere og kunder. Recommender systems er en relevant teknologi, som kan skabe større omsætning for virksomheder, og mange erhvervsdrivende er villige til at investere penge i teknologien, for at optimere og forbedre effektiviteten af teknologien.

Et recommender system kan anvendes i mange forskellige kontekster, og teknologien er ikke kun afgrænset til én enkelt type af virksomhed eller service, og omfanget af teknologien er derfor også meget bred. Derfor vil vi i denne rapport gerne kigge nærmere på områder, hvor recommender systems i mindre grad er udviklet, og hvor der i høj grad er plads til forbedringer. Derfor har vi opstillet følgende initierende problemstilling:

Eksisterer der hidtil uløste problemer indenfor recommender systems, som kan optimeres?

Kapitel 2

Recommender Systems

2.1 Introduktion til Recommender Systems

Recommender systems er i høj grad blevet en dominerende "feature" til rigtig mange store og små virksomheders services. Mange mennesker benytter sig af dem indirekte hver eneste dag med hjemmesider som Youtube, Amazon og Netflix [4].

Recommender systems er i bund og grund algoritmer, som er skræddersyet til at anbefale specifikke ting (alt fra film til møbler) til en given bruger af en hjemmeside, app mm. Dog eksisterer der, for nu, ikke mange muligheder for at få anbefalet en film til en gruppe. Men ved hjælp af andre rapporter og en analyse af tilgange til problemet, kan en model fremstilles (se afsnit 4.2). Det er derfor interessant at belyse hvad der foregår bag skærmen og fører til de anbefalinger, man får på bl.a. Netflix og Spotify. Recommender systems er endda blevet så vigtig en del af de store firmaers strategi, at Netflix i 2009 holdt en konkurrence med en præmiesum på 1 million USD [24]. For at vinde konkurrencen skulle deltagerne lave den bedste collaborative filtering algoritme, som er en af flere typer recommender systems.

Det viser, at recommender systems både er et stærkt effektiviserings værktøj, men også i høj grad er en vigtig investering. Herudover hjælper recommender systems den pågældende tjeneste til at føles mere overskuelig, når man uden besvær finder frem til sine yndlingsserier [4].

Når man snakker om recommender systems, er det væsentligt at se på de to mest generelle kategorier. Der er først og fremmest collaborative filtering methods (se 2.2.2). Denne metode fokuserer på fortidens interaktioner mellem indhold og brugeren. Der forsøges derfor at forudsige interessante materiale til en given bruger ud fra brugeren valg af tidligere "materialer" [4].

Der er derudover den anden metode, hvilket er content-based methods (se 2.2.1). Content-based methods fokuserer på at opbygge en model ved hjælp af enten et klassifikations-system, eller ved hjælp af regression [4]. Der fokuseres her på brugeren "profil" og er derudover genstandsfokuseret, hvilket vil sige at man forsøger at forudsige om en bru-

ger - ud fra brugerens profil - kan lide en given genstand [4].

Disse to metoder har meget mere at byde på og derfor vil de blive forklaret senere i rapporten.

Som nævnt eksisterer der ikke mange muligheder for at få anbefalet en film til en gruppe, og det er netop denne aggregering af individuelle præferencer i et recommender system, som projektet her vil beskæftige sig med.

2.2 Filtreringsmetoder

Når et recommender system skal implementeres, er der to mulige metoder for filtrering; collaborative og content based filtrering. Denne sektion vil gå i dybden med teorien bag hver metode samt forklare fordele og ulemper ved brugen af hver af dem.

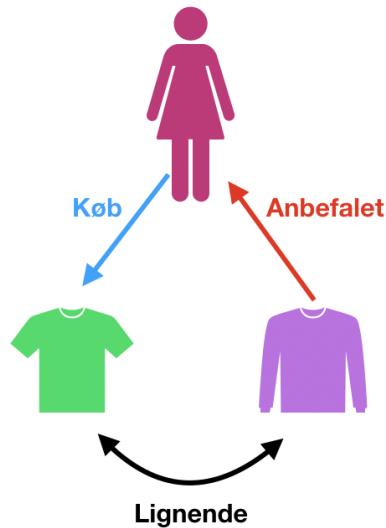
2.2.1 Content based filtrering

Content based filtrering anbefaler produkter på baggrund af funktionaliteter, som det enkelte produkt har. Hvert produkt har altså en "produkt profil", der gør det muligt at uddrage informationer om det enkelte produkt[12]. Ved denne metode benyttes brugers profil, hvor faktorer som køn, alder, profession og andre personlige informationer, kan drage konklusioner om en brugers præferencer. Derved kan produkter anbefales ud fra produkternes "profil". Ydermere kan den enkelte bruger også sammenlignes med brugere, der har lignende personlige informationer, da der for hvert køn eller aldersgruppe ofte er højt-ratede produkter, der kan anbefales[21]. I figur 2.1 ses et eksempel på en content based filtrering, ved historik, til at anbefale et lignende produkt ud fra hvad brugeren før har købt.

Content based filtrering har den fordel, at den kan anbefale et nyt produkt med det samme for nye brugere, da der tages udgangspunkt i brugerens profil. På den måde kan metoden skaleres op til et større antal brugere. Ydermere kan mindre populære produkter også anbefales, siden specifikke interesser for den enkelte bruger benyttes. Dog vil det være svært at anbefale et produkt, som ligger uden for brugerens interesser [21][8].

2.2.2 Collaborative filtrering

Ved Collaborative filtrering benyttes brugerens egen historik samt andre brugeres vurderinger til at drage konklusioner om deres præferencer. Den bygger derved på en antagelse om, at en bruger kan lide ting, som minder om forrige interesser i produkter, og lignende brugere med samme præferencer[9]. Collaborative filtrering kan ydermere opdeles i en memory based approach og en model based approach.

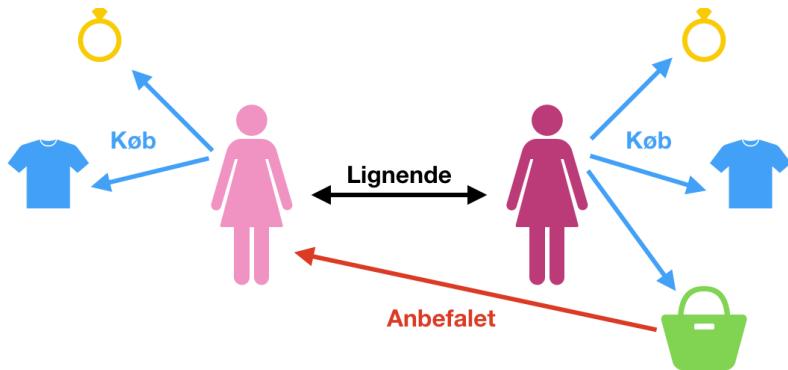


Figur 2.1: Content baseret filtrering - anbefalede produkter baseret på køb

Memory based approach

Memory based approach kan opdeles i user-item filtrering og item-item filtrering. En user-item filtrering tager udgangspunkt i at finde lignende brugere ved at sammenligne vurderinger af andre produkter, for derved at anbefale produkter som lignende brugere har vurderet godt[9]. Metoden finder den såkaldte "nearest-neighbor", som er defineret, som den mest lignende bruger.

I figur 2.2 ses et eksempel på et user-item system, hvor to brugere vurderes til at have de samme præferencer, da de har købt de samme produkter. Derved kan et produkt, den ene bruger har købt, blive anbefalet til den anden bruger, da de har lignende interesser. Denne form for anbefaling virker godt til relativt få antal brugere, men opskalerer ikke godt til mange millioner af brugere. Endvidere bygger denne form for anbefaling også på, at ens brugerpræferencer er statiske, hvilket er problematisk. En persons interesser og præferencer kan ændre sig dramatisk gennem tiden. F.eks kan man lide action film i sine teenage år, men i voksenalderen er man måske mere til dokumentarer. Derfor vil denne metode ikke være optimal, da brugernes interesser er dynamiske. Der er også "cold-start" problemet, hvilket går på, at systemet næsten ingen information har om de nye brugere, hvilket gør, at det er svært at sammenligne dem med andre brugere[20]. Da user-item filtrering har en del problematikker, blev item-item filtrering opfundet for at mindske nogen af disse. Ved item-item filtrering anbefales produkter, der er lignende de produkter, brugeren tidligere har vist interesse for. Ved lighed menes ikke nødvendigvis ens produkter, men produkter hvor brugere viser ens interesse eller giver lignende ratings[22]. I figur 2.3 ses et eksempel på en item-item filtrering, hvor to produkter vurderes lignende grundet at brugerne har givet en lignende interesse i produkterne, og vi



Figur 2.2: User-item metoden - To brugere ligner hinanden, fordi de har vist samme interesse i de samme produkter. Man kan dermed anbefale tasken til den anden bruger, da man går ud fra, at brugeren også vil have en interesse i tasken, fordi de generelt har samme interesser, i de samme produkter.

antager, at brugerne har "rated" tasken og trøjen positive.

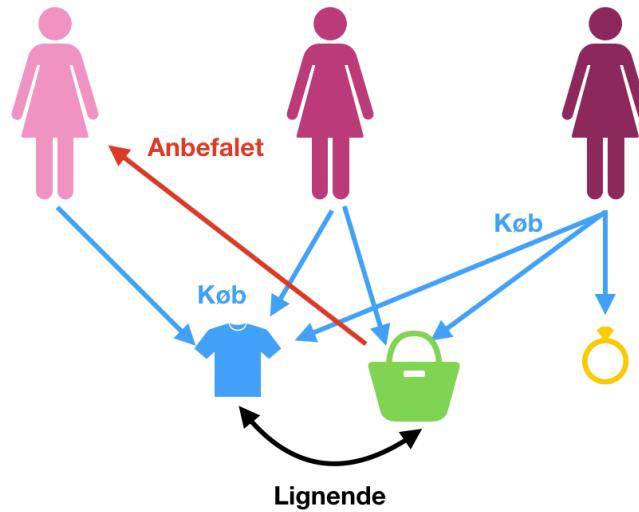
Denne metode optimerer den memory based approach og de problematikker, der var ved user-item, da man ikke længere har "cold-start" problemet. Årsagen til dette er, at systemet ikke afhænger af information om nye brugere, men kun om de enkelte "items". Generelt for både item-item og user-item approach er, at metoderne ikke er så gode, hvis der ikke er nok data[9]. Ydermere gælder det for begge metoder, at det er umuligt at anbefale noget til nye brugere, samt at flere produkter kan have for få interaktioner til nogensinde at kunne blive anbefalet[21].

Model based approach

Memory based recommender systems er ikke altid mulige at skalere store nok og ved store datasæt, er metoden ikke hurtig nok. Ved disse problematikker kan model based approach benyttes. I en model based approach benytter man typisk machine learning til at forudsige brugers vurderinger af ikke-vurderede ting. Dette kan gøres på flere forskellige måder, men den mest populære er matrix faktorisering. Ideen er at man har en user-item matrice, som består af to dimensioner, hhv. brugerne og de givne 'items'. Er matricen næsten tom, kan man forbedre ydeevnen af algoritmen ved at reducere dimensionerne på matricen. Her benytter man så matrix faktorisering[2].

Som navnet også antyder, bygger den model-baserede tilgang på, at bygge en model baseret på sit datasæt. Man benytter altså ikke alt data fra sit datasæt, men udvælger noget, og det gør metoden både hurtig og skalerbar. Der findes forskellige metoder til, hvordan man kan opstille sådan en model. Feltet Lineær algebra kan benyttes her, hvorfra der benyttes matricer og matriceoperationer.

Fordelene ved en model based approach er, at den kan skaleres meget bedre op til store



Figur 2.3: Item-item - To brugere har købt både en taske og en trøje, og en anden bruger har kun købt trøjen. Det antages, at alle brugere har været glad for deres køb. Eftersom at både tasken og trøjen har fået positive 'ratings' af brugerne, og har lignede egenskaber, og interesse fra brugerne, anbefales tasken til den bruger, som kun havde købt trøjen.

datasæt, uden at gå på kompromis med kørselstiden af modellen, samt er denne model også hurtigere til at komme med forudsigtigheder, end en memory based approach. Der er dog også nogle ulemper ved at benytte denne approach - f.eks er kvaliteten af forudsigtighederne ikke altid gode, eftersom modellen er bygget på en lille del af datasættet, og ikke hele datasættet. Det skal dog siges, at modellen kan opbygges således, at kvaliteten på forudsigtighederne er bedre, men det koster på kørselstiden af modellen. Desuden har denne approach også manglende fleksibilitet, da det at opbygge modellen tager både tid og ressourcer. Altså kan det være en svær opgave at tilføje mere data til et model-baseret system[5].

Hybrid filtrering

Da både content based og collaborative filtrering har deres ulemper, kan der med fordel benyttes en såkaldt hybrid filtrering, der er en kombination af de to metoder. Man kan ved at tage fordelene ved begge metoder, mindske nogen af de fejlkilder, der findes ved dem. Det gør altså kvaliteten og nøjagtigheden af anbefalingerne bedre. Der er forskellige måder på, hvordan man kan implementere den hybride metode. De populære går primært på, at man implementerer både content- og collaborative, og fremviser brugeren resultatet af begge metoder. En en anden er, at man sammenligner resultaterne fra begge metoder, og kun viser brugeren de ting, som begge metoder har anbefalet[16].

I det næste afsnit vil vi kigge på, hvordan filtreringsmetoderne bliver benyttet i praksis i industrien, hvor vi kigger på store internationale virksomheder som Amazon.

Kapitel 3

Analyse af nuværende anvendelser

Med en grundlæggende forståelse for hvordan recommender systems fungerer, når det gælder de to overordnede tilgange collaborative- og content-based filtering, er det muligt at analysere nogle konkrete cases. Dette vil de næste afsnit beskæftige sig med. Dette skal give en forståelse af, hvordan metoderne i dag anvendes i store virksomheder, til at anbefale indhold til brugere. Derudover skal det hjælpe med at aklare det domæne som vi vil indsætte os til ift. projektarbejdet.

3.1 Spotify

Hvad er Spotify?

Spotify er en digital musik streamingtjeneste. Af denne årsag er det i deres bedste interesse at anbefale relevant musik til deres brugere, og de bruger derfor recommender systems til at gøre netop dette.

Collaborative Filtering

Spotify bruger collaborative filtering da dette lader dem give et kvalificeret gæt på hvad en brugers præferencer er, set igennem historiske adfærdsmæssige-mønstre.

For eksempel, hvis to brugere lytter til det samme sæt af sange og kunstnere, så er deres musiksmag sandsynligvis sammenlignelig. Det vil derfor være en gedigen chance for, at man kan drage paralleller imellem de to lyttere og deres fremtidige forslag.

Spotify bruger collaborative filtering frem for content based filtering, da dette muliggør at lære fra implicit (se afsnit 3.5.1) feedback fra brugeren - dvs. at de lærer fra brugerens opførsel på deres service, for så at kunne forudse fremtidig opførsel - som den primære metode, men lærer også igennem explicit feedback som hvis en bruger 'liker' eller 'disliker' en sang.[18] En årsag til at bruge implicit feedback frem for eksplisit er at den implicitte data (click, sidevisninger, køb, medie streams) kan indsamles langt hurtigere og på større skala, uden endda at bede en bruger om eksplisit feedback.[11]

Der kan, når man arbejder med implicit feedback til collaborative filtering, antages at der arbejdes med ikke-negative sæt af feedback-værdier som er forbundet med hvert par af brugere og ting i ens domæne.

Det er vigtigt at pointere at sættet af bruger-ting observationsmatricen lader elementerne tilhøre de ikke-negative reelle tal, da dette lader Spotify lave kontekstuelle overvejelser og vægte forskellige aspekter.

For eksempel kan det være en fordel at vægte nyere sang-afspilninger frem for gamle, da en persons musiksmag ofte ændrer sig over tid.

Desuden kan et element i matricen være 0. Dette betyder ikke nødvendigvis at brugeren slet ikke ville bryde sig om det givne element, men at brugeren måske ikke kender til det. Så målet for Spotify er at finde den bedste sang til enhver bruger, som de endnu ikke har kendskab til. [11]

Audio Models & ConvNet

Spotify havde et problem med mangel af data i begyndelsen. Dette gjorde at de gav dårlige anbefalinger. For at overkomme dette problem brugte Spotify convolutional neural networks, eller ConvNet. De bliver trænet til at analysere rå lyddata for at identificere en sangs karakteristikker som længde, tone, tempo osv, hvilket lader netværket 'forstå' sangen så Spotify kan identificere og anbefale lignende sange til brugere.[18]

Natural Language Processing (NLP)

Denne metode bruger Spotify til at analysere en playliste som var det et dokument, og bruger så enhver sang-titel, kunstner, eller anden tekst til at analysere som en del af deres Machine Learning recommendation-algoritme.[18]

En sidste detalje som gør at Spotify kan give gode anbefalinger er at de også anvender deres machine learning til at finde 'Outliers'. Dette betyder at hvis man lytter til en sang som man ellers ikke ville have hørt - for eksempel hvis man lader en anden bruge ens Spotify konto - så vil Spotify ikke give anbefalinger baseret på denne, da den ikke er en del af ens sædvanlige præferencer.[18]

3.2 Netflix

Netflix er den største streamingtjeneste indefor for film og serier, hvilket gør dem interessante ift. hvordan de benytter recommender systems. Dette afsnit vil derfor dykke ned i hvilke tiltag, som Netflix benytter og hvorfor.

Når man streamer vælger man selv hvilke film og serier, som man har lyst til at se. Dette valg er ikke altid nemt, og det er derfor vigtigt for Netflix ikke at lade brugeren havne i den frustrerende proces, hvor man ikke ved hvad der er interessant at se[6].

Netflix's Algoritme

Netflix afholdte i 2006 en konkurrence, som blev kaldt "The Netflix Prize"[7]. Konkurrencen gik ud på at optimere Netflixs daværende platform kaldet "Cinematch". Der blev dermed bygget en platform med en samling af 107 algoritmer. De benyttede i deres løsning K-nearest neighbor algoritmen til post-processing af data[7]. Derudover benyttede de sig af Singular Value Decomposition (SVD) for at give en optimal dimensionelimplementering til brugerne af Netflix. De gjorde også brug af Restricted Boltzmann Machines (RBM) til at forbedre brugen af deres collaborative filtering model. En lineærkombination af disse to algoritmer gav et godt resultat [7]. Problemet ved denne løsning for Netflix var blot at den kun var implementeret med statiske algoritmer, hvilket vil sige, at de kun opererede med historisk data, og tog ikke dynamiske bruger-vurderinger og ændringer "real time". Dette måtte Netflix fikse og derefter implementere denne i sit recommender system.

Personalized video ranker: PVR

Netflix opererer med rækker under kategorier. Det vil sige at for eksempel så kan der stå som overskrift "suspenseful movies" og derunder vil der så være en hel række med film fra venstre til højre. Algoritmen der bruges her hedder som titlen: "Personalized video ranker" som vil blive nævnt ved forkortelsen "PVR" fremover.

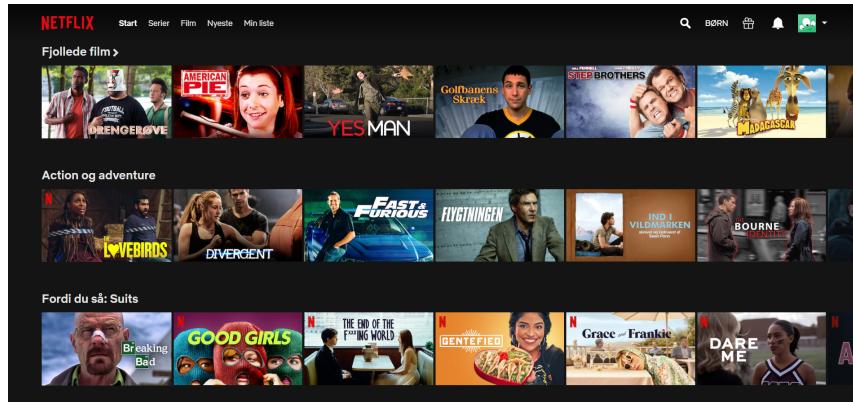
Denne algoritme sorterer ud fra delmængder valgt ved genre og anden filtrering ud fra personlige præferencer. Derudover så sorterer algoritmen også ud fra "general purpose relative rankings" som blander generelle "gode" film fra hele film kataloget ind. Dette gør den mindre personlig, men gør ifølge Netflix at det bliver god blanding [6].

Top-N Video Ranker

Netflix benytter også "Top N video ranker" algoritme, hvilket finder de få bedste personlige anbefalinger i hele filmkataloget for hver enkelt bruger. Top-N video ranker bliver derudover optimeret af data og algoritmer fra den øverste del af katalogets vurderinger i stedet for at bruge hele kataloget [6].

Trending now

Netflix har derudover en "trending now" feature, hvilket har 2 kategorier af trends. Den ene er short term, hvilket er begivenheder, som trender i kort tid, men også sker af og til, som for eksempel Valentine's day, hvor romantiske film er populære i et par dage[6]. Så er der "one-off short term" begivenheder, hvilket for eksempel kunne være, at der er en stor orkan i et område, hvilket vil give en stor interesse for dokumentarer omkring orkaner, film og generelt film og naturkatastrofer. Denne feature, samt de personlige "Top Picks" fremgår af Netflix' forside, som kan ses af figur 3.1.



Figur 3.1: Forsiden af Netflix, der har flere kategorier til diverse anbefalinger

Continue watching

En anden vigtig "video ranking algorithm" er continue watching algoritmen. Den sørger for at holde styr på serier, som man kan have set flere gange i perioder, men ikke set færdig. Denne algoritme laver en række af film under kategorien continue watching. Algoritmen skal bruge hvornår man har sluttet serien og hvor langt man har set i episoder og kan derfor foreslå at se videre fra hvor man slap. Den bruger også hvilken film man så senest for at kunne sortere rækken ud fra de senest sete[6].

Video-Video Similarity

Netflix benytter også en række til at anbefale film/serier som er anbefalet ud fra andre film/serier. En similarity "sims" algoritme bliver benyttet til at lave en upersonlig liste af videoer til hver video, som er de lignende videoer. Der bliver dog derefter lavet en "because you watched" række, hvilken er en personlig række. Rækken er en delmængde af de lignende videoer, som bliver genereret ud fra hvilke der bliver vurderet mindre om eller kunne være god for den givne bruger[6].

Opsummering

Alle disse algoritmer bliver benyttet til at lave gode rækker for den givne bruger, hvilket tilsammen er Netflixs recommender system i grove træk. Netflix bruger disse for at oprettholde konkurrencen overfor andre streamingtjenester og for at sørge for at brugeren får en god oplevelse[6]. Netflix har altså indført en lang række algoritmer, som hver især optimerer den enkelte brugers oplevelse, dog ses der ikke nogen algoritme, som anbefaler en film, hvis de er sammen med nogen. Det er altså muligt at optimere Netflix's applikation med et sådan aspekt.

3.3 Amazon

Hvad er Amazon?

Amazon er ikke blot en webshop, men tilbyder en række forskellige tjenester såsom musik og film. Dog vil dette afsnit fokusere på Amazons recommender system på deres online webshop.

Webshoppen startede originalt som en online boghandel, men blev udvidet til at sælge andet også blandt andet elektronik, tøj og møbler.[3]

Amazons Recommender System

Amazon anvender recommendation algoritmer til at gøre onlinebutikken personlig for enhver kunde. Den former sig altså efter brugeren, hvilket betyder at den viser relevante varer til den individuelle bruger. To vigtige målinger for Amazon er "click-through" og konverteringsfrekvens. Begge målinger øges markant ved brug af recommendation algoritmer, frem for reklamer som ikke retter sig direkte mod den individuelle kunde. Der er udfordringer som skal overkommes for store online butikker som Amazon. Der er meget data som skal behandles. Utdregninger skal foregå hurtigt så brugen kan bladre imens udregningerne udføres, og derudover skal butikken løbende tilpasse sig mens brugeren bladrer igennem. Med nye kunder kommer problemet med at der er for lidt information om dem. Altså der er kun få køb eller produktbedømmelser at gå ud fra. Med ældre brugere er der derimod en overflod af information. Når Amazon benytter anbefalinger laver de f.eks. et område til dagens anbefalinger. Dette kan ses af figur 3.2. Amazon bruger item-item collaborative filtering. Denne algoritme (modsat traditionel collaborative filtering) skalerer uafhængigt af mængden af kunder og mængden af varer i deres produktkatalog. Generelt finder collaborative filtering recommender systems et sæt kunder som har købt og bedømt produkter, som overlapper med en kundes købte og bedømte produkter. Derefter ophobes produkter fra disse lignende kunder, og der fjernes produkter som kunden allerede har købt eller bedømt. Amazon fokuserer i stedet på at finde lignende produkter frem for lignende kunder. De finder altså produkter som ligner de produkter en bruger har købt og bedømt, og anbefaler dem. Måden deres item-item collaborative filtering virker på, er ved at matche enhver brugers købte og bedømte produkter med lignende produkter, og samler så disse til en liste med anbefalinger. For netop at finde dette match, altså de mest lignende produkter, kigger Amazon på, hvilke produkter der ofte købes sammen. Deres algoritme forløber nogenlunde således for at opnå dette mål:

For hver genstand i produktkataloget (1), så findes enhver kunde (C) som har købt produkt (1). For hvert produkt (2), så registrer hvis kunden har købt (1) og (2). Sluttligt, for enhver (2), udregn ligheden mellem (1) og (2).



Figur 3.2: Anbefalinger af produkter på webshoppen Amazon

Amazon har netop valgt denne item-item collaborative filtering anbefalingsalgoritme for, at kunne overkomme de fornævnte udfordringer, hvilket de har konkluderet, at den lever op til at gøre[14].

3.4 Opsamling og valg af film-domænet

Ved at have studeret anvendelser nuværende anvendelser af recommender systems i kommercielle platforme, har vi fået et indblik i hvordan tilgangene med filtrering fungerer i praksis.

Med baggrund i den viden der er opnået gennem analyse af de forskellige platforme, har vi sporet os ind på film-domænet, som det område vi ønsker at arbejde med under recommender systems. Hertil har vi flere begrundelser for valget. Først og fremmest er det med det samme nemt at se et problem, da studier f.eks. viser at Netflix-brugere i gennemsnit bruger 18 minutter på at vælge det de skal se, hvilket er dobbelt så meget som ved almindeligt TV[23].

Udover at anbefaling af film er en af de største anvendelser af recommender systems, er der derudover godt med inspiration og litteratur. Da vi i projektet er interesseret i at finde og definere et teknisk problem, ser vi ikke behovet for at bevæge os ud på et niche domæne, og 'opfinde' eller beskæftige os med noget nyt.

3.5 Data i recommender systems

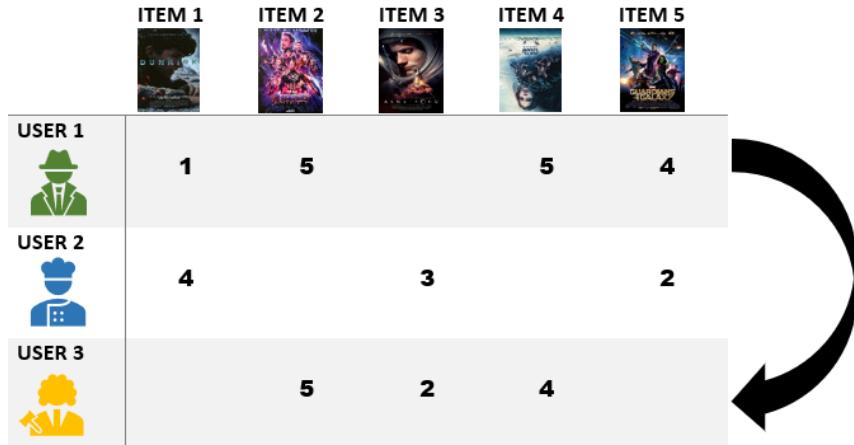
Recommender systems bliver, som tidligere belyst, benyttet mange steder, men da det er besluttet, at det er problemstillinger inden for film, der vil blive undersøgt, vil der i resten af rapporten benyttes film til at exemplificere begreber og teorier. Heri vil film dog stadig blive beskrevet som et item, da dette er den almene terminologi, når der snakkes om interaktion mellem forskellige elementer (f.eks. user). Der vil nu, for en bedre forståelse, følge et afsnit der forklarer de forskellige typer af data, man benytter i recommender systems.

3.5.1 Eksplicitte og implicitte data

Der findes to typer data inden for recommender systems; eksplicit og implicit. Begge med fordele og ulemper, men også en gennemgående mangel på kontekstuel sammenhæng, som vil blive beskrevet yderligere i afsnit 3.5.2.

Eksplicit data

Eksplicit data opstår af en direkte rating af et item fra en bruger. Det sker f.eks. ved at tildele et antal stjerner til et item eller en thumbs-up/thumbs-down, som det f.eks. ses på Netflix. Det betyder derfor, at man både får negativ og positiv data på forskellige items. Det er til gengæld svært at indsamle denne slags data, da det kræver en direkte interaktion fra brugeren. Der foreligger derudover også et psykisk aspekt i denne type feedback. Det opstår, da en person som regel kun anbefaler en film, hvis de enten syntes, at den var ekstremt god eller ekstremt dårligt. Oven i dette rater individer også forskelligt. Altså kan 3 stjerner fra en person have en anden betydning end 3 stjerner fra en anden person[15]. Figur 3.3 viser en tabel, som repræsenterer datafortolkningen ved brug af eksplicit data. Figuren illustrerer de ratings de respektive brugere har givet til diverse film. Her ses en tydelig korrespondence mellem user 1 og user 3 og derfor vil item 5 blive anbefalet til user 3.



Figur 3.3: Det ses, at bruger 1 og bruger 3 har lignende præferencer, og derfor anbefales film 5 til bruger 3

Implicit data

Den anden type af data er den implicitte data. Det er data, som opsamles når brugeren bruger en applikation, eller tjeneste, og bevæger sig rundt på den. Der indsamles f.eks. data om en bruger, når de klikker på et item, eller ud fra hvor lang tid de bruger på det givne item (evt. hvor mange procent af en film de har set), og er derfor nemt at indsamle. Det er dog, modsat eksplisit data, noget brugeren skal give lov til, hvilket typisk sker, når man accepterer terms of use. Ved denne slags data er det kun positiv feedback der opfanges. Altså analyserer algoritmen alt interaktion som positivt, og det er derfor svært, at sige hvad en given bruger ikke kan lide. Det medfører også, at hvis brugeren kigger flere gange på et item uden interesse for dette, vil det skabe confidence omkring dette item, og eventuelt give en forkert anbefaling. Der opstår også muligheder for fejlanbefalinger, når en bruger browser efter en gave til en ven, eller et andet individ låner kontoen[15].

3.5.2 Kontekstuel information ift. film-recommendations

I de fleste recommender systems ser man, at de anbefalinger der laves, kun benytter brugерens bedømmelser (ratings). Information vedrørende konteksten af anbefalingerne bliver derved ikke taget i betragtning. Dette er i særdeleshed tydeligt ift. film-anbefalinger, hvor man ser på, hvordan brugeren før har ratet en film (eksplisit), eller ser bagliggende brugstendenser hos seeren (implicit).

Pointen er, at en brugers nydelse af en anbefalet film ikke alene er afhængig af, hvordan denne person tidligere har bedømt andre film. At se en film er en mere kompleks aktivitet, der rummer en bred vifte af information som tidspunkt, lokation og det selskab anbefalingen laves i forbindelse med. Man kan også kigge på brugerkonteksten, hvor ting som humør, demografi og social netværk kan spille en rolle.

Der findes flere årsager til, at denne slags kontekstuel information bliver nedprioriteret. Først og fremmest er det meget nemmere, at benytte sig af konkrete ratings og implicitte brugermønstre, da der findes en overflod af disse, og derudover er det nemt at indsamle. Kontekstuel information er selvfølgelig sværere at indsamle, og bliver derfor ofte sat som videreudvikling.[13]

Trods den nemmere tilgang til et recommender system, der har til formål at anbefale det enkelte individ en film, er dette ikke at foretrække, når en gruppe skal se en film. Da dette kapitel har vist en interesse samt mangel på et recommender system, der påtænkes en hel gruppe, vil vi specifikt kigge nærmere på den sociale aktivitet, der er indeholdt i, at se en film som en gruppe. Altså skal der tages højde for flere brugeres præferencer, når der skal laves en samlet film-anbefaling.

Kapitel 4

Gruppe recommender systems

4.1 Anbefalinger fra individ til gruppe

Nu har vi afgrænset os til at behandle problemdomænet at anbefale en film til en gruppe. Vi har førhen beskrevet collaborative filtrering. Denne metode er favorabel frem for content-based filtrering til at løse vores problem, da den lader os tage udgangspunkt i brugernes eksisterende film-præferencer. I collaborative filtrering kan man benytte enten en memory-based eller en model-based tilgang som før beskrevet.

Selvom en anbefaling til en gruppe af brugere måske virker som en simpel generalisering af anbefalingen til et individ, er det en noget mere kompleks opgave, hvilket f.eks. fremgår af [1]. En årsag til dette er, at det naturligvis er mere besværligt at tilfredsstille en gruppe af personer med forskellige individuelle præferencer.

Når man arbejder med en memory-based tilgang, skal man beslutte sig for, hvor man vil foretage en aggregering, altså hvor data fra de forskellige brugere sammenflettes. En anden mulighed er at danne et virtuelt individ, som repræsenterer gruppen, hvilket vi vil gøre i vores løsning. Dette gøres, da man ved hjælp en en række metoder (se 4.2.1), kan vurdere hvor lighederne mellem individerne i en gruppe findes. Dernæst ønskes det, at danne en samlet matrice på de fundne ligheder. Herudover er det også en svære og større proces - hvis man ikke benytter machine learning - at regne på flere forskellige matricer end en samlet.

Generelt ved recommendations er der store huller i de data, man arbejder med, da en bruger oftest kun har været forbruger af en lille delmængde af det tilgængelig materiale. Denne data sparsity bliver kun værre, når der er flere restriktioner, i dette tilfælde, når der skal anbefales til flere brugere[19].

4.2 Collaborative filtrering tilgang

4.2.1 Memory-based tilgang

Memory-based tilgang benytter, at en given brugers præferencer for film skal sammenlignes med andre brugeres præferencer, hvilket er en central del af collaborative filtrering. Når sådan en sammenligning laves, bruger den en given formel for at finde korrelationen. Det kan f.eks. være ved brug af 'Pearsons korrelation' eller 'cosinus ligthed', som vil blive beskrevet senere i rapporten.

Det er nødvendigt at sammenligne den data, vi indsamler med en større mængde data, netop for at undgå cold start problemet. Til dette har vi fundet et datasæt fra MovieLens, som indeholder brugere samt deres bedømmelser af film. Denne data bruges til at sammenligne de individuelle film.

Hvordan kan der gives samlede anbefalinger ud fra flere individers præferencer?

Der er flere social choice strategies tilgængelige til at opnå brugernes præferencer og give anbefalinger derpå.

Nogle alment brugte strategier er

- *Utilitarian Strategy*: Summerer alle ratings fra de items som er givet, hvorfra den med den højeste score vælges.
- *Most Pleasure Strategy*: Finder den maksimale rating givet til et item givet af alle brugere og vælger den, som har den største.
- *Least Misery Strategy*: Finder den laveste givne rating på et item fra alle brugere og anbefaler items baseret på den største af dem.

Dog mener Vineet et al.[1] ikke at valget af en enkelt af disse ovenstående strategier er tilstrækkeligt. Derfor foreslår de RTL strategien.

RTL strategien fjerner den laveste rating givet til ethvert item, og anvender den additive utilitarian strategy til resten af bedømmelserne. Derfra anbefaler den de items med højest utility værdi til en gruppe brugere.[1]

Da rapporten ikke giver nok grundlag for, hvorfor at RTL strategien fjerner den dårligste rating, vil vi ikke benytte dette aspekt. Dog har andre succesfulde algoritmer benyttet sig af, at fjerne meget afvigende værdier. Dette kan vise sig fordelagtigt i større systemer, men da vores system forventes at blive benyttet til 2-5 personer, vurderes det, at det ikke er fordelagtigt at fjerne data, da det vil skabe for stor en afvigelse for resultatet.

Forinden at denne data, kan regnes på, skal den indsamles og fortolkes. Til dette formål fremstilles to metoder:

Cosinus lighed

I denne teknik er n -brugere repræsenteret af et n -dimensionelt vektorrum og ligheden er målt som cosinus distancen mellem to bedømmelsesvektorer. Her er brugerne og deres bedømmelser arrangeret i vektorformat. Distancen mellem to vektorer (to brugere) kan findes ved at dividere prikproduktet af de to vektorer med produktet af deres Euklidske distance.

Denne metode vil ikke blive brugt, da det er mere præcist at måle på vinkler mellem vektorer end distancen. Netop denne metode bliver brugt ved Pearson korrelation. [1]

Pearson korrelation

Denne metode udregner ligheden mellem to variable. Derfra fås en værdi mellem -1 og 1. Positive værdier betyder, at de har en positiv association, eller at der eksisterer en lighed. Negative værdier betyder, at der er modsætninger mellem dem, eller negativ association. [1]

Man kan finde ligheden mellem to brugere, u og v ved følgende formel:

$$sim(u, v) = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

Ligheden udregnes baseret på bedømmelses-informationer fra brugerne.

$r_{u,i}$ repræsenterer bedømmelsen givet af bruger u til item i .

Ligeledes repræsenterer \bar{r}_u samt \bar{r}_v gennemsnittet af de to repræsenterede brugeres anbefalinger. Ved at erstatte $\sum_{i \in I}$ med $\sum_{i \in I_{u \cap v}}$ i formlen anskues kun de items(i), som er bedømt eller brugt af begge brugere. Dette giver mere akkurate forudsigelser.

Ovenstående er et eksempel på en user-user collaborative filtrering teknik. Det er muligt at finde ligheden mellem to *items* i og j . Dette giver item-item collaborative filtrering:

$$sim(i, j) = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

Ved ovenstående formel for item-item filtrering er to items lignende, hvis de har næsten lige bedømmelser fra en bruger u , og på baggrund af disse ligheder vil items'ne blive anbefalet til den specifikke bruger.

Resultaterne heraf bruges til at danne en mængde (S) af de film (i) som mest ligner de film, som brugeren allerede kan lide. Mængden filtreres så det kun indeholder film som brugeren ikke har set, hvorfra der vil udregnes en forudsigelse af, hvorvidt brugeren vil kunne lide den ikke-sete film eller ej.

For at gøre dette anvendes følgende ligning. [1]

$$p_{u,i} = \frac{\sum_{j \in S} sim(i, j)r_{u,j}}{\sum_{j \in S} |sim(i, j)|}$$

En anden tilgang er en model-based tilgang, hvilken benytter sig af en machine-learning algoritme, der tilføjer ekstra parametre til den benyttede model. Det vil derfor ikke blive benyttet, da machine-learning er udenfor scopet af projektet.

På baggrund af de ovenstående analyser, vil der i dette projekt blive brugt en memory based tilgang.

Da dette besvarer det første spørgsmål, nemlig hvordan det er muligt at anbefale en film ud fra en brugeres præferencer, vil der nu kigges på:

4.2.2 Grupperingsstrategier

I afsnittet 4.2.1 blev RTL-strategien diskuteret.

Skulle man følge RTL-strategien, var antagelse, at det gavner anbefalingen, at fjerne brugere der ikke ligner hinanden. Dette vil vi dog ikke gøre, da vi arbejder med en lille gruppe personer, der skal se en film sammen. Selvom at det synes oplagt, at fjerne 1 ud af 5 brugere, hvis de fire andre har større korrelation, ser vi en anden tilgang, som vi mener er bedre. Dette er en kombination af den additive utilitarian strategi og least misery strategien. På denne måde vil vi, i stedet for at fjerne brugere, fjerne film, hvis en eller flere brugere har tildelt filmen under 3 i rating. Herved undgås det, at en eller flere brugere er utilfredse med filmanbefalingen, men dette sætter også en grænse for, hvor mange brugere, der kan tages i betragtning. Vi vil derfor, som følge af at bruge denne strategi, maksimalt tage fem brugere i betragtning til en anbefaling, hvilket også fremgår som værende det bedste antal ifølge [1]. For et mere præcist resultat implementeres et vægtningssystem. Vægtningssystemet baseres på en brugers rating. Det vil sige, at hvis en bruger har ratet en film 4, vil korrelationen for denne film med en anden given film blive ganget med faktor 0,8. Heraf følger en mere præcis model:

Rating	Vægt
1	0,2
2	0,4
3	0,6
4	0,8
5	1

Dette vægtningssystem er vigtigt af flere grunde. En af grundene er, at der er mange film, som deler de samme genre. Dette betyder, at mange film vil have den samme korrelation som andre film. Hvilket medfører manglende præcision i resultatet. For at sortere i denne mængde, og give nogle bedre anbefalinger, benytter vi brugerens ratings, til at i højere grad at udvælge film, som skal anbefales til brugeren. Endvidere er det nævnt i vores definition på item-item filtrering i afsnit 2.2.2, at man ikke kun anbefaler items ud fra, at de ligner hinanden, men også brugernes interesse i dem. Dette foregår oftest igennem ratings, hvor man kan vurdere, om en bruger har haft en positiv eller

negativ oplevelse med det givne item. Derfor ser vi det vigtigt, at vores anbefalinger afhænger af både genre, men også brugerens givne ratings.

Kapitel 5

Problemdefinering

5.1 Problemafgrænsning

Recommender systems bliver i høj grad brugt i praksis af store virksomheder, og er en teknologi, som der er mange penge i. Det kunne ses i vores case studies, at nogle af verdens største virksomheder som Netflix, Amazon og Spotify, gør brug af recommender systems. Teknologien er altså ikke noget nyt, men det er en teknologi, der er i udvikling og kan forbedres, eller anvendes andre hidtil usete steder. Vi afgrænsrer os derfor til kun, at håndtere det kontekstuelle aspekt. Mere specifikt når et recommender system skal anbefale en film til en gruppe. Det indebærer f.eks. hvor mange personer man er, hvilke præferencer af film genre de forskellige personer har, alder, køn og situation. Det er blevet belyst i et research paper[13], at netop disse kontekstuelle spørgsmål i høj grad bliver overset, når man vil anbefale film til en gruppe. Derfor vil vi i denne rapport tage hånd om nogle af disse spørgsmål. Vi har valgt, at yderligere afgrænse det til, kun at fokusere på, hvilke personer man vil se en film sammen med. På denne måde, får vi kombineret forskellige brugeres præferencer, og derved er det også et overkommeligt problem, set i lyset af projektets størrelse. Dette recommender system vil blive skrevet i JavaScript, og vi vil benytte en webplatform som grænseflade til brugerne. Dertil kommer en problemformulering:

5.2 Problemformulering

Hvordan kan man udvikle en context-based film-recommender system vha. collaborative filtering som webapplikation i JavaScript, hvor en gruppe af brugere kan gives en samlet film-anbefaling?

Kapitel 6

Design

Dette kapitel vil beskæftige sig med først at opstille kravspecifikationer for det program, der skal udvikles. Dernæst bliver måden, hvorpå systemet designes, diskuteret. Der vil være et særligt fokus på hvordan anbefalingsalgoritmen skal laves, altså vores problemstilling i at kombinere flere individuelle brugers præferencer til et gruppeanbefalingssystem. Som løsning til dette opstilles en web-applikation.

6.1 En webapplikation som løsning

På baggrund af den opstillede problemformulering, kan disse samles til en beskrivelse af en web-applikation som ville være en løsning på problemformuleringen.

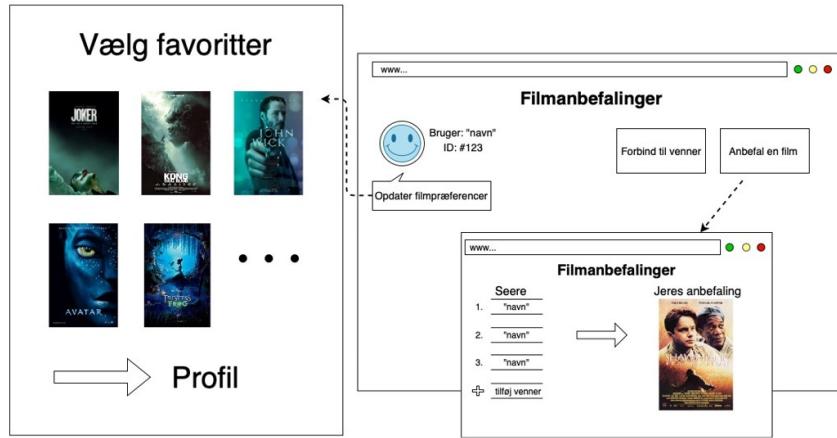
En sådan applikation vil have nogle kernefunktionaliteter:

- Mulighed for at opdatere sin profil. Her menes tilføje og fjerne ratings
- Oprette kontakt til venner
- En nem og intuitiv måde at få sin anbefaling på
- Fejlmeddelser hvis brugeren prøver at udføre ulovlige handlinger
- Navigationsbar, som er tilgængelig fra hele web-applikationen

På figur 6.1 ses ideen af, hvordan platformen skal se ud. Ved at trykke sig ind på **Update preferences** kan man justere på de præferencer, som man tilkendegav under oprettelsen af profilen.

Connect with others giver muligheden for at brugeren kan søge i databasen over andre brugere og tilføje andre som 'forbindelser/venner'.

Get a group movie recommendation åbner et interface, hvor man kan tilføje et antal brugere fra sin 'venneliste', og få en rangerede række film, eller én enkel film.



Figur 6.1: Første skitse over konceptet med web-applikationen

6.2 Kravspecifikationer

Forinden en webapplikation, som beskrevet i forrig afsnit 6.1, kan laves, skal der opstilles en række kravspecifikationer. Disse er udarbejdet i gruppen med henblik på helhed og struktur i programmet, samt et mål om et nemt og intuitivt GUI. Tager vi udgangspunkt i vores problemformulering og figur 6.1, kan vi opstille følgende krav:

1. Et bruger-system, hvor hver bruger har en personlig konto på websiden. Herunder er følgende punkter nødvendige for at opfylde kravet:
 - Brugerne skal kunne oprette en individuel konto
 - Serveren skal tjekke om den nye konto allerede eksisterer
 - Kontoen skal opbevare information vedr. filmpræferencer, et unikt bruger ID og et brugernavn
 - Brugerne skal kunne ændre sin filmpræferencer.
 - Brugerne skal kunne forbinde deres konti, så der oprettes et "vennesystem" for hver bruger
 - Login-in systemet skal verificere input på klient- samt server-siden
2. Webapplikationen skal kunne anbefale film til en gruppe af venner, herunder kræves følgende:
 - Evnen til at give en god anbefaling til en gruppe på op til 5 personer
 - Outputtet skal være en liste med vores top 10 anbefalede film, der er sorteret i listen efter bedste match
3. En intuitiv og brugervenlig grænseflade med følgende krav:

- Brugeren skal præsenteres for film når de logger ind
 - Rent og minimalistisk layout, der derved gøre den mere intuitiv
 - Korte, men beskrivende fejlmeddelelser
 - Film skal præsenteres via plakater
4. På det tekniske plan, vil vi udvikle vores webapplikation i overensstemmelse med projektkravene, dertil kan vi opstille følgende krav:
- Webapplikation skal skrives i JavaScript, med NODE.js som backend samt HTML og CSS som frontend
 - Opbygge og gemme en database over brugere i en separat fil
 - Filmanbefalingen skal benytte item-item filtrering på en samlet gruppematrix
 - Der skal benyttes en modificering af en kombination af den additive utilitarian strategi og least misery strategi
 - Pearson korrelation benyttes, til at finde korrelationen mellem to film
 - Der vil gennem programudviklingen blive benyttet testfunktioner. Mere specifikt vil Jest blive benyttet, samt egne funktioner, hvor det findes passende

Vi ønsker at opfylde disse krav, og vi vil i slutningen af rapporten referere til disse for at vurdere, om vi har opfyldt kravene tilfredsstillende.

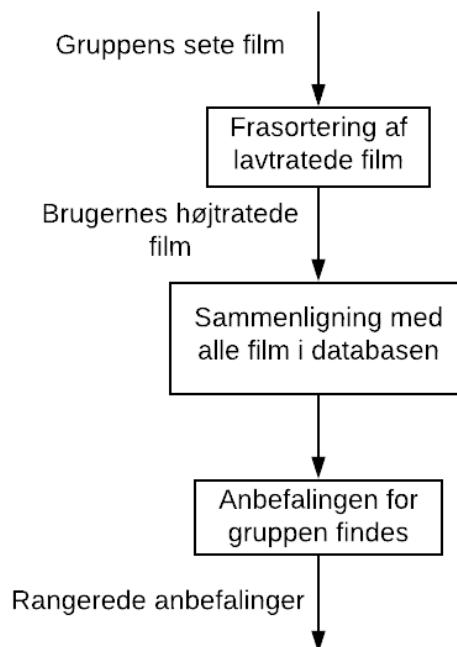
6.3 Anbefalingsalgoritmen

For at opfylde kravene, opstilles en fremgangsmåde for en mulig algoritme:

1. Start med én liste per bruger, der indeholder deres sete film
2. Saml alle brugeres film og frasorter film, der ligger under en given grænseværdi. Dette sker ved følgende:
 - (a) Lav en samlet liste (R_G) hvor de sete film får en værdi over eller under en given grænseværdi.
 - (b) Lav en liste med alle film, der ligger under grænseværdien
 - (c) Fjern alle film, der både ligger over den givne grænseværdi og under den givne grænseværdi (her benyttes least misery strategien)
3. Find Pearson korrelation mellem genre for hver film i listen R_G med alle film i databasen
4. Lav en liste med alle brugerne, som indeholder brugernes ratede film og en korrelation for hver ratet film med hver film i filmdatabasen

5. Adder hver brugers korrelationer med én film i databasen med hensyn til deres vægt, for alle film i databasen (her benyttes den additive utilitarian strategi)
6. Find gennemsnitskorrelation for hver film og tilføj til resultatlisten
7. Sorter listen, så de bedste korrelationer kommer først, og returner den til brugeren

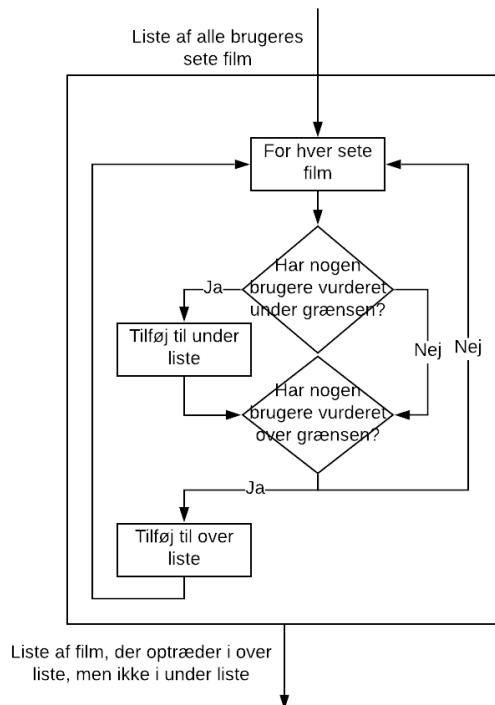
Denne fremgangsmåde illustreres med følgende modeller. På figur 6.2 ses et overblik over fremgangsmåden. Som det ses på figuren, tages gruppens sete film som input, for derefter at frasortere de lavt ratede film. For mere præcist at forklare hvordan dette gøres, ses der på figur 6.3. Output af denne procedure er brugerens højtratede film, der anvendes til at sammenligne med alle film i databasen. Proceduren for denne sammenligning ses dybere forklaret på figur 6.4. Herefter kan anbefalingen for gruppen findes, og outputtet af dette er de rangerede anbefalinger. Denne procedure vurderes dog så triviel, at den ikke er uddybet på en figur, men blot forklaret til sidst.



Figur 6.2: Model over overblikket af recommender systemet

Det fremgår af punkt 2 og 5 i fremgangsmåden, at least misery strategien og additive utilitarian strategien ønskes implementeret. For at forklare, hvordan dette gøres, betragtes figur 6.3, hvor det ses, at film, der er vurderet under den valgte grænseværdi, bliver tilføjet til én liste, og dem, der er vurderet over tilføjes til en anden. Det bemærkes dog også at en film godt kan optræde i begge lister, i det tilfælde at én bruger har givet filmen en høj rating, og en anden har givet filmen en lav rating. Af figuren ses det, at efter

alle film er gennemløbet, og tilføjet til en liste, overføres de film, der er i listen med film over grænseværdien, men ikke de film, som er under grænseværdien. Derved benyttes least misery, som er en startfiltrering. Dette giver både et mere præcist resultat og gør algoritmen mere effektiv, da den senere hen skal sammenligne alle film der er tilbage i listen.

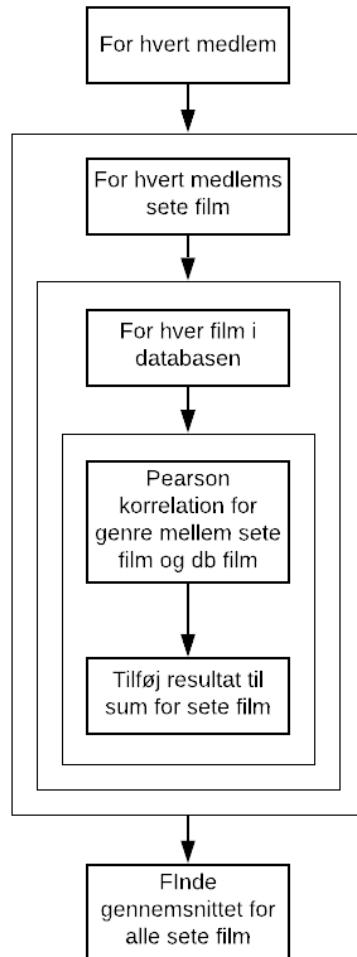


Figur 6.3: Frasortering af lavtratede film

Da der nu er brugernes højtratede film tilbage, kan disse sammenlignes ved brug af collaborative filtrering ved item-item filtrering på genre (se afsnit 9.3 for forklaring), for at finde korrelationerne mellem 2 film. Der er udover genre mange andre parametre at måle på, såsom skuespillere, årstal, ratings mm., men da det tidligere er besluttet at benytte Pearson korrelation, kan der kun måles på én parameter. Af denne grund, er der også implementeret et vægtningssystem, som er beskrevet i afsnit 4.2.2, for at få et mere præcist resultat.

På figur 6.4 ses det, at algoritmen for hvert medlem i gruppen sammenligner deres sete film med hver film i databasen, ved at anvende hvert af medlemmets sete film. De to film, der sammenlignes, bliver sammenlignet på genre ved hjælp af Pearson korrelation, og derefter tilføjes resultatet af hver Pearson korrelation til en samlet sum for hver film. Afslutningsvis findes gennemsnittet for hver film, for at kunne lave den endelige

anbefaling.



Figur 6.4: Sammenligning med alle film i databasen

Den samlede anbefalingen for gruppen findes ved, at sortere gennemsnittet af korrelationerne med den højeste korrelation først.

6.3.1 Matematisk definition

Følgende afsnit har til formål at introducere det matematiske grundlag for vores algoritme.

Når web-applikationen åbnes, er der i databasen indeholdt to mængder, som er interresante at kigge på. Vi lader B være mængden af alle brugere i databasen og F være alle film i databasen. Sættene kan derved defineres ved:

$$B = \{b_1, b_2, \dots, b_x\} \quad (6.1)$$

$$F = \{f_1, f_2, \dots, f_n\} \quad (6.2)$$

Hvor x angiver bruger og n angiver film. Disse to mængder er ikke repræsenteret på figur 6.2, da de kommer forud for algoritmen, og figurene blot havde til formål at beskrive designet af denne. Det følgende er en matematisk beskrivelse af den algoritme, der er beskrevet på figurene. Som det også kan ses på figur 6.2, eksisterer der en mængde S for enhver given bruger hvori alle brugerens sete film er indholdt. Vi lader S_x være mængden af alle sete film for en bruger x og f_n være en film indeholdt i mængden:

$$S_x = \{f_1, f_2, \dots, f_n\} \quad (6.3)$$

Yderligere gælder det at $\alpha \leq x \leq \beta$, hvilket betyder, at algoritmen kan tage minimum α og maksimum β brugere i betragtning. For hver mængde S_x overføres enhver film f_n til en ny mængde G der indeholder to mængder U og O :

$$G = \{\{O\}, \{U\}\} \quad (6.4)$$

Mængden O får tildelt alle film f_n som har $\geq \lambda$ i rating, hvilket er grænseværdien beskrevet i afsnit 4.2.2. Grænseværdien vil fremover blive beskrevet som λ . Mængden U får tildelt alle film $f_n < \lambda$.

Da det er muligt at en bruger, x , har ratet en film $f_n < \lambda$, mens en anden bruger, y , har ratet den samme film $f_n \geq \lambda$ kan det ske, at $f_n \in O \wedge f_n \in U$. Der foretages derfor en sorterings så:

$$O = O \setminus U \quad (6.5)$$

I mængden O er mængden M_x indeholdt for enhver bruger x . Lad mængden M_{n_r} indeholde den n' te ratede film samt den givne rating for hver film, og F_{x_k} er en mængde, hvilket der indeholder to nye mængder, der senere forklares. I mængden M_x er der derved indeholdt to mængder $M_{n_r} \wedge F_{x_k}$:

$$M_x = \{\{M_{n_r}\}, \{F_{x_k}\}\} \quad (6.6)$$

For yderligere at forstå opbygningen af denne, ses der nu på mængden F_{x_k} . I mængden F_{x_k} er korrelationen for hver film i databasen f_x med den n' te ratede film f_n . Dette giver os:

$$F_{x_k} = \{k_{f_1_1}, k_{f_1_2}, \dots, k_{f_n_x}\}, \quad (6.7)$$

Hvor $k_{f_n_x}$ er korrelationen mellem den $n'te$ ratede film og den $x'te$ film i databasen

Heraf fåes at mængden ser ud således:

$$O = \{\{M_x\}\} \quad (6.8)$$

Her bemærkes det at $O = O \setminus U$ som tidligere i ligning 6.5. O beskriver altså figur 6.3, hvor de lavratede film frasortes. På samme måde beskriver M_x det bagudliggende for figur 6.4. Efter korrelationerne er fundet oprettes et nyt sæt K . I sættet ligger elementerne f_x som indeholder summen af gruppens korrelationerne for en given film så:

$$K = f_1, f_2, \dots, f_x \quad (6.9)$$

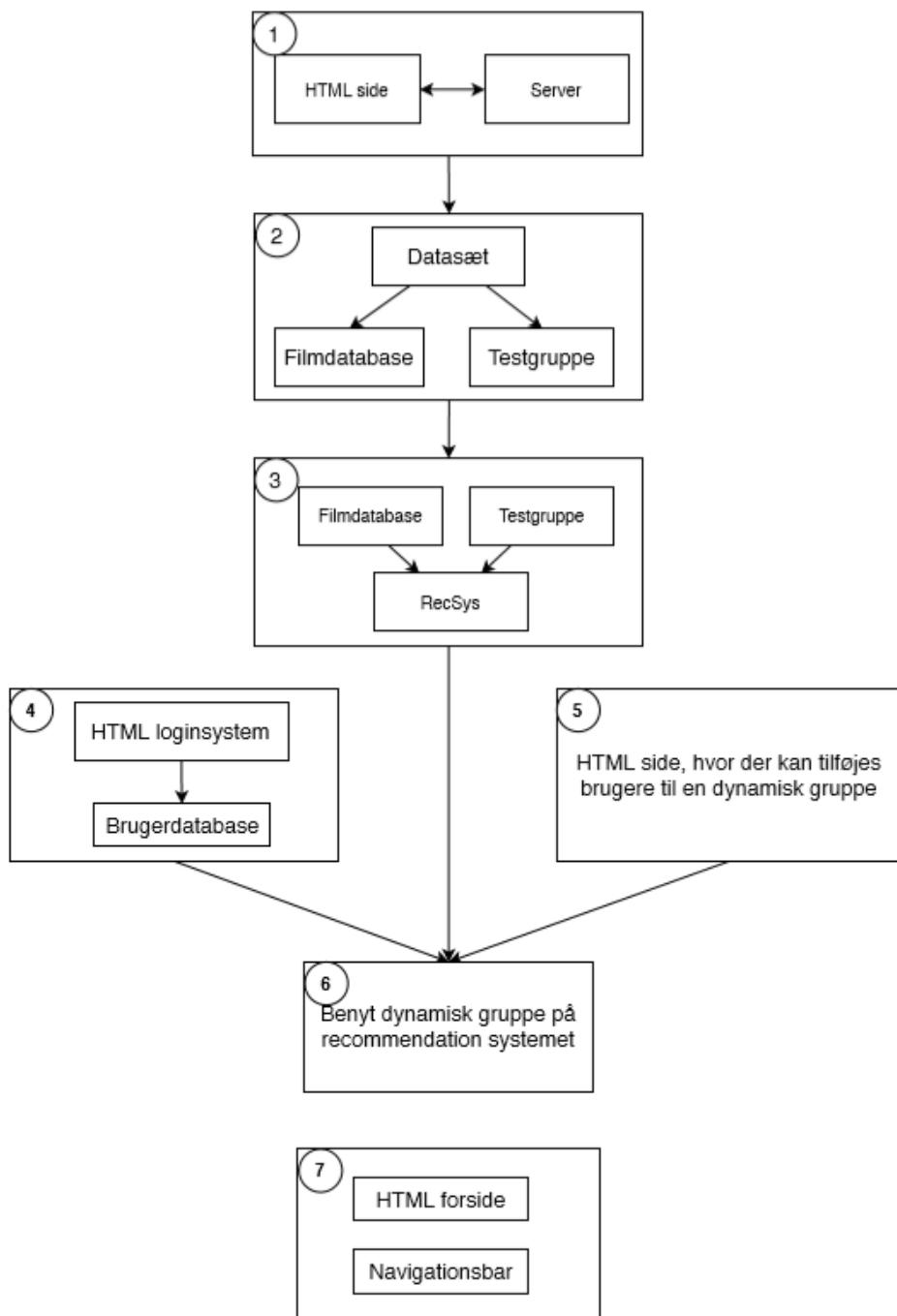
$$f_x = \sum_{n=1}^l k(f_{n_x}) \quad (6.10)$$

hvor $k(f_{n_x})$ er funktionen k , der beskriver korrelation for film x med brugeren n , og l beskriver antallet af brugere. Slutteligt findes gennemsnittet for hver korrelation, og giver resultatet

$$\frac{f_x}{|O|} \quad (6.11)$$

6.3.2 Plan for implementation

Inden at anbefalingsalgoritmen kan implementeres, kræves det, at der allerede eksisterer en række filer og funktioner. Derfor fremstilles figur 6.5, som beskriver hvilken rækkefølgen for opgaver i projektet, samt afhængigheder af separate filer. På figuren ses der flere bokse, som indkapsler andre bokse. Tallet i den yderste boks, beskriver hvilket nummer i rækkefølgen alt indhold ønskes implementeret.



Figur 6.5: Rækkefølge af implementation af recommendation systemet

Kapitel 7

Implementation

Det følgende kapitel omhandler hvordan det fremlagte design realiseres som en teknisk løsning, som gruppen har udviklet. Der vil løbende i afsnittet indsættes snippets af udvalgte dele af koden, dog kan hele projektets kildekode ses i bilag A.

Der skal udvikles et større program i JavaScript, hvor det er mest oplagt at lave en webapplikation. De teknologier og værktøjer vi benytter os af i implementationen beskrives i afsnit 7.1, og begrundes.

Projektet rummer en del komponenter, så afsnit 7.2 nedbryder hvilke mapper og filer der er, og hvad deres ansvar hver især er.

Kapitlet vil efterfølgende beskrive mere detaljeret nogle enkelte dele af programmet. Eftersom mange filer afhænger af hinanden indbyrdes, vil vi ikke gennemgå alle funktioner, men blot nogle udvalgte signifikante funktioner.

7.1 Værktøjer

Projektets programmeringsmæssige udgangspunkt er at udvikle en webapplikation. Til front-end benyttes der den klassiske triade HTML, CSS og JavaScript.

Til at håndtere backend delen af webapplikationen har vi valgt NODE.js og Express. Valget heraf forekommer, da vores frontend teknologier, samt NODE.js, er en væsentlig del af pensummet dette semester. Det er altså oplagt at kunne udnytte JavaScript færdigheder på både front- og backend.

Derudover valgte vi at bruge Express frameworkt til at komplementere vores brug af NODE.js ved at udnytte frameworkt for at gøre mere, hurtigere.

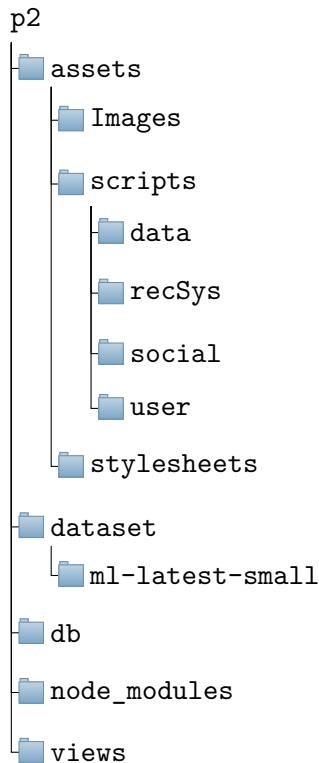
Når der udvikles, er det vigtigt at sikre sig, at ens funktioner gør det, de skal. Især i omfanget af en større applikation. Af denne årsag har vi valgt at bruge test-frameworket Jest. Jest understøtter NODE.js, hvilket derfor gør det velegnet til vores brug.[10]

For at holde styr på ændringer anvender vi Git som versionskontrol. Hertil anvendes GitHub. Disse to værktøjer gør det lettere for os at samarbejde om programmeringen af

webapplikationen.

7.2 Overblik over projektressourcerne

Projektet er et NODE.js projekt, der er opdelt med fem mapper; **assets**, **dataset**, **db**, **node_modules** og **views**. Opdelingen af mapper og undermapper ses nedenfor.



Mapperne er opdelt og navngivet alt efter, hvilken funktion filerne har. Herudover ligger der fire filer i **p2** mappen, som håndterer backend.

- **app.js** Starter serveren
- **initialize.js** Indlæser databasen
- **routes.js** Eksisterer for ikke at have alle routes i **app.js**
- **utility** Indeholder hjælpefunktioner

7.3 Programflow

Det udviklede program indeholder en lang række forskellige filer og mapper, for hvilke man kan se et overblik over i afsnit 7.2, men for en dybere forståelse af programmets

opbygning præsenteres figur 7.1. Figuren fylder mere end hvad er overskueligt, og den er derfor også vedhæftet i bilag B, hvis en dybere forståelse af figuren ønskes.

7.3.1 Server start

Hvis man anskuer figur 7.1, ses det at vores løsning starter ved `app.js`. Filen står for generel opsætning af vores backend samt at starte serveren, således at den lytter på den definerede port, som afhænger af, hvorfra serveren køres. Årsagen til dette er, at vi bruger Heroku til at hoste vores webapplikation - og Heroku skal have mulighed for at definere hvilken port, der skal benyttes.

Serveren lytter på nogle routes som er defineret i `routes.js`. De meste centrale routes ses i figuren. Der er udtaget '/', '/connectWithFriends' samt '/movieRec', som alle er routes som gør, at brugeren får EJS dokumenterne med samme navn vist. Derudover ses '/createUser', '/getRecommendations' og '/loginUsername'. Disse tilgås ikke direkte af brugeren, men gennem HTTP anmodninger, hvori der medtages parametre. Disse sendes for eksempel når en bruger indtaster et brugernavn, og trykker 'Create Account'. Efter at serveren er startet og aktivt lytter på den valgte port, bygges databasen. Serveren tjekker om enhver databasefil eksisterer, og såfremt der er en som mangler, bygges alle - pånær brugerdatabase - igen. Hvis de, derimod, alle eksisterer, så indlæses de blot.

7.3.2 Film anbefalingssiden

Når en bruger tilgår '/movieRec' vises brugerens venneliste hvorfra brugeren kan danne en gruppe. Gruppen sendes vha. '/getRecommendations' til vores anbefalingssystem på backend-siden. Anbefalingssystemet behandler den data som tilsendes vha. vores algoritme og tilbagesender en liste af anbefalinger til brugeren. Denne liste vises på samme side, '/movieRec', som brugeren sender anmodningen fra. Måden dette forekommer på er, at så snart brugeren modtager anbefalingerne som respons på anmodningen om dem, dannes der nye elementer på siden, korresponderende med anbefalingernes titler og plakat.

7.3.3 Bruger-system

Ny bruger

Vi valgte at implementere en navigationsbar. Foruden at være navigationsbar, er det også her at brugeren opretter en profil, eller logger ind. Når en ny bruger vil registrere sig, valideres inputtet, så der ikke opstår kopier eller tomme navne. Brugernavnet sendes til '/createUser' og valideres ved at tjekke om navnet allerede er taget af en anden bruger. Hvis ikke, så gemmes brugeren til databasen, og der sendes en velkomst-besked til brugeren. Hvis brugernavnet derimod er optaget, så modtager brugeren en fejlbesked.

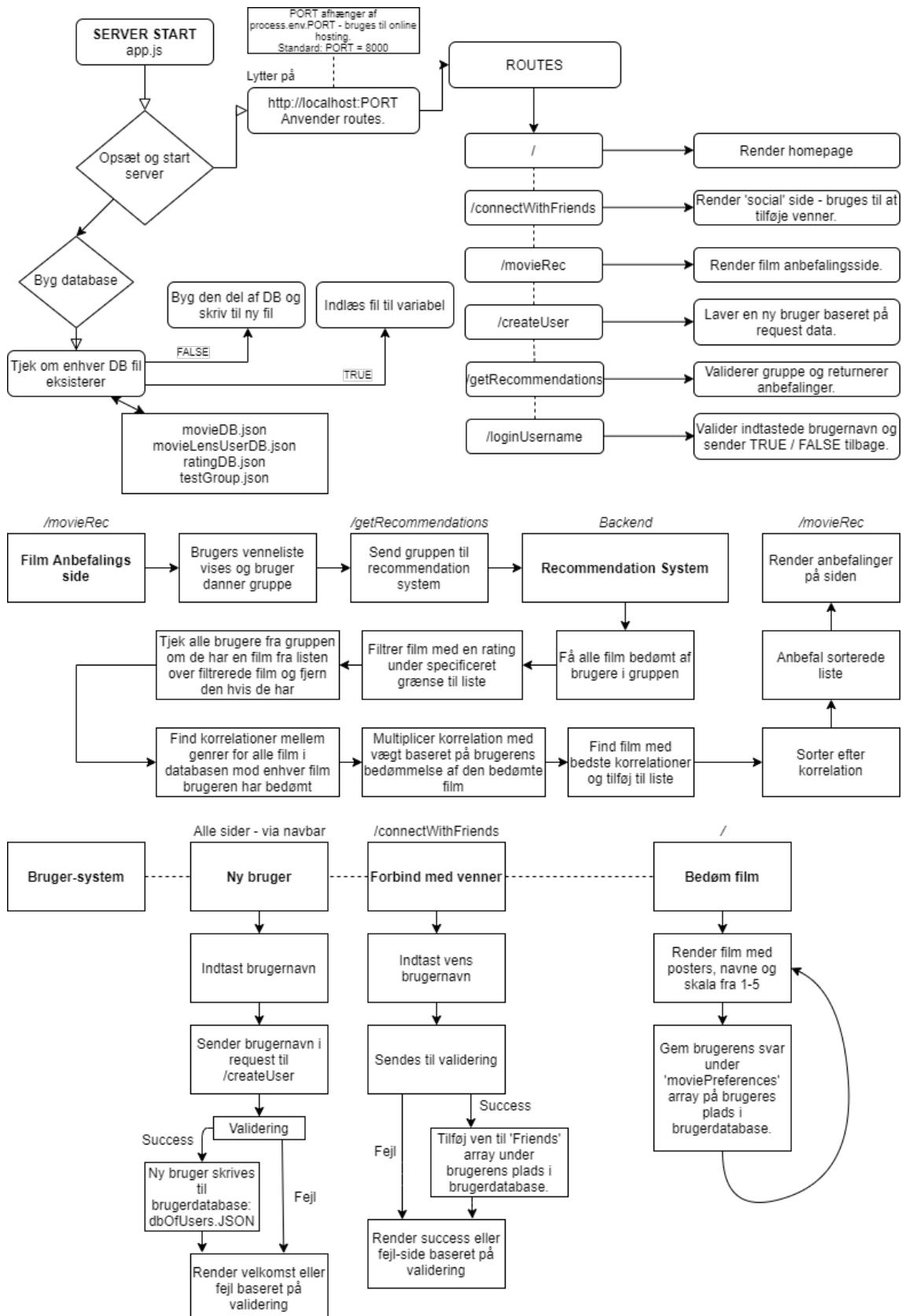
Forbind med venner

Når '/connectWithFriends' tilgås har brugeren mulighed for at tilføje venner. De indtaster blot deres vens brugernavn. Dette valideres - der tjekkes om det eksisterer - og

alt efter om valideringen slog fejl, eller var en success, så tilføjes vennen, og brugeren informeres om dette. Hvis det dog ikke var en success får brugeren en fejlbesked.

Bedøm film

På forsiden, '/', har brugere mulighed for at bedømme film. Der vises informationer om en tilfældig film fra databasen. Filmen kan brugeren bedømme fra 1-5, hvor 1 er værst, og 5 er bedst. Hvis brugeren ikke kender filmen, benyttes knappen "unknown". Derefter vises en ny film. Hvis brugeren bedømmer en film, gemmes svaret til deres plads i bruger databasen under deres 'moviePreferences', hvorefter brugeren vises en ny film de kan bedømme.



Figur 7.1: Flowchart over program (nogle routes udeladt)

7.4 Datagrundlag og data

For at kunne give filmanbefalinger, og samtidig undgå cold-start problemet, er der behov for et større datasæt af forskellige film. Der eksisterer givetvis flere muligheder til at hente sådanne data, men movielens datasættet^[17] er et anerkendt og brugt datasæt, der er udnyttet i recommender systems. Grundet scopet af projektet er der valgt at arbejde med det "lille" datasæt¹. Dette indeholder ca. 10000 ratings, ca. 9000 film og 600 brugere. De relevante data i CSV-filerne fra datasættet, bliver brugt til at opbygge vores databasefiler. Der skal både indlæses og efterfølgende behandles dataet, som vi vil berøre i det næste afsnit.

7.4.1 Databasevalg

Der eksisterer mange værktøjer, som tilbyder databaser bl.a. mongoDB og MySQL. Disse værktøjer er i dette projekt fravalgt, og i stedet fremgår databaserne som JSON filer. Dette har vi gjort, fordi det fremgik af IWP-kurset, og andre former fra databaser var ikke gennemgået i kurset.

7.4.2 Indlæsning af data

I **loadData.js** ligger en funktion, som benyttes til indlæsning; **getData()**, hvor man angiver filstien samt filtypen til filen der skal læses fra. Derved kan der håndteres særskilt om det er en CSV-fil eller en JSON-fil. Hertil returnes det læste data med hjælp af **neatCsv()** eller **JSON.parse()** afhængigt af den måde, dataet er formateret på.

```

1  async function getData(path, type, noLog = true) {
2      let result;
3      try {
4          let startTime = performance.now();
5          result = await fs.readFile(path);
6          if (noLog === false) utility.printTestAndTime(path, result,
7              startTime);
8      }
9      catch(error) {
10         utility.logError(`${error.name} in loading ${error.path}!`);
11     }
12     if (type === CSV_TYPE) return neatCsv(result);
13     if (type === JSON_TYPE) return JSON.parse(result);
14 };
15 }
```

Listing 7.1: getData funktion

Ved brug af eksportering af funktioner i Javascript kan **loadData.js** eksportere flere udgaver af **getData()** med de forskellige argumenter:

¹<https://grouplens.org/datasets/movielens/>

```

1 // Getting all data with type csv
2 module.exports.getMovieData = async() => { return await getData(
    MOVIES_CSV_PATH, CSV_TYPE) };
3 module.exports.getLinkData = async() => { return await getData(
    LINKS_CSV_PATH, CSV_TYPE) };
4 module.exports.getRatingData = async() => { return await getData(
    RATINGS_CSV_PATH, CSV_TYPE) };
5 // Getting all data with type JSON
6 module.exports.getTestGroupData = async() => { return await getData(
    TESTGROUP_PATH, JSON_TYPE) };
7 module.exports.getMovieDB = async() => { return await getData(MOVIE_DB_PATH
    , JSON_TYPE) };
8 module.exports.getRatingDB = async() => { return await getData(
    RATING_DB_PATH, JSON_TYPE) };
9 module.exports.getMovieLensUserDB = async() => { return await getData(
    MOVIELENS_USER_DB_PATH, JSON_TYPE) };
10 module.exports.getUserDB = async() => { return await getData(USER_DB_PATH,
    JSON_TYPE) };

```

Listing 7.2: Funktioner til at hente data fra vores diverse database-filer

7.4.3 At bygge databaserne

En helt essentiel funktion for programmet er **buildMovieDB**, som set i listing 7.3. Det er i denne funktion at filmdatabasen bliver opbygget. Her ses det hvordan funktionerne fra listing 7.2 bliver brugt, som moduler i en større funktion. Dataet fra MovieLens er ikke organiseret som ønsket til dette program, og det er derfor nødvendigt at udtrække bl.a. udgivelsesår fra titlen, og lægge dette over i egenskaben *year*, som tilhører objektet *movieDB*. For at forbinde den individuelle film, med den rigtige poster benyttes metoden **find()**, som returnerer filmplakaten, hvis den findes i TMDb (The Movie Database).

```

1 const buildMovieDB = async (ratingsDB, noLog = false) => {
2     let startTime = performance.now();
3     const movieDB = await loadData.getMovieData();
4     const links = await loadData.getLinkData();
5     // Adds essential data to each movie entry in the database
6     await movieDB.forEach(async movie => {
7         // Adds the year for each movie
8         movie.year = utility.getYearFromMovieString(movie.title)
9         // Finds movies TMDBIDs in the dataset links.csv file and adds it
10        to the movie. Used to receive data on client side about movie.
11        const movieLink = links.find(link => link.movieId === movie.movieId
12        );
13        movie.tmdbId = movieLink.tmdbId;
14        // Finds genres from movie and adds them to both and object and
15        // array (array is used in group recommendation system)
16        movie.genres = dataHandler.getGenresFromMovie(movie);
17        if (movie["genres"]["genres"][("no genres listed")] === 1) {movie.
18            skip = true} else {movie.skip = false};

```

```
15     // Adds ratings given by datasets users
16     movie.ratings = await dataHandler.getRatingsForMovieID(movie.
17         movieID, ratingsDB);
18     // Calculates average rating based on above ratings
19     movie.averageRating = dataHandler.getAverage(movie.ratings);
20 });
21 if (noLog === false) utility.printTestAndTime("MovieDB, Ratings, &
22     Average Ratings", movieDB, startTime);
23 return movieDB;
24 };
25 };
```

Listing 7.3: oprettelse og opbygning af filmdatabasen

7.4.4 Mangel i datasæt

Ved at se på og analysere vores datagrundlag, databasen fra MovieLens, kunne vi konstatere at noget data i filmdatabasen var mangelfuld. Konkret så eksisterede der film, hvor ingen genre var angivet. Dette er selvfølgelig et problem, og det ville ikke være optimalt, hvis flere film uden genre blev dømt lignende, bare fordi de begge havde "no genre listed". Dette ville blive et problem, når vi ønsker at lave genrekorrelation med en film der ingen genre havde. Der findes altså der ugyldig data i vores datasæt, som man kunne give sig til at skære fra. Problemet med at fjerne dataet er, at nogle brugere har rated disse film, og hvis filmen så ikke eksisterede i vores database, kunne det give nye problemer. Den valgte løsning hertil blev at implementere en egenskab *skip*, som blev sat til true, hvis ingen genre var angivet. Når løkken, der finder korrelationen, kører, bliver filmene altså tjekket for værdien *skip*, og hvis den er true, springer den filmen over.

7.5 Gruppeanbefalingen

Centralt i projektet er implementationen af en gruppeanbefaling af nogle film, altså vores tekniske problem. Dette afsnit vil beskrive hvordan designet og algoritmen beskrevet i afsnit 6.3 er implementeret. Det er ydermere nødvendigt at definere grænseværdierne fra afsnit 6.3.1. Det er besluttet at grænseværdien for ratings $\lambda = 3$, og at der minimum skal være 1 bruger, altså $\alpha = 1$ og et maksimum på 5 brugere, altså $\beta = 5$.

7.5.1 At lave en gruppeanbefaling

Kernefunktionen til at anbefale en række film er **makeGroupRecommendations()** i vores program, som har en gruppe som inputparameter. Denne gruppe repræsenteres som et array af objekter. Funktionen returnerer et array med 10 film, som anbefales til gruppen.

```

1 module.exports.makeGroupRec = async function makeGroupRecommendations(group
) {
2   // Group ratings array
3   let R_G = [];
4   // Array of movies that were rated below threshold
5   let badMovieList = [];
6
7   const movieDB = await loadData.getMovieDB();
8
9   // Pushes filtered ratings for each member to collected array of users.
10  group.forEach(member => { R_G.push(filterRatings(member)); });
11
12  // Adds every movie that the user rated that is under the set threshold
13  // to a list of 'bad' movies
14  R_G.forEach(member => { badMovieList.push(member[U_THRES]); });
15  badMovieList = utility.reduceArray(badMovieList);
16
17  // Check if any aboveThreshold list includes any of the movies in the
18  // list of 'bad' movies:
19  R_G.forEach(member => member[A_THRES].filter(entry => badMovieList.
20    includes(entry)));
21  const R_G_COR = findCorrelations(R_G, movieDB);
22
23  return getFinalRec(R_G_COR, movieDB);
24}
```

Listing 7.4: makeGroupRecommendations

En forudsætning for at der kan laves en anbefalingen er, at der per bruger er lavet en liste med seje film. Linje 10-17 i listing 7.4 beskæftiger sig med grænseværdi sortering, som er vores fremgangsmåde til at fjerne de film, der er ratet lavt af brugerne. Først løbes der altså over hvert medlem i gruppen for at lave R_G listen. Der ligger objekter

i R_G arrayet der angiver om en given film er over eller under vores grænseværdier, som henholdsvis repræsenteres ved A_THRES og U_THRES. I den såkaldte *badMovieList* indsættes de film der er under grænseværdien for enhver bruger. Eftersom en film kan være under grænseværdien for én bruger, men over grænseværdien for en anden bruger, skal der laves et særligt tjek der sørger for, at R_G til slut ikke indeholder film som nogen fra gruppen ikke synes godt om.

Mere interessant er den del af funktionen, der for den nuværende status af R_G finder korrelationer ved sammenligning med filmdatabasen. Dette er funktionen **findCorrelations()**, der sammensætter korrelationer fra flere brugere - **correlationByMember()**-funktionen - og returnerer et array med disse. Koden til **correlationByMember()** fremgår af listing 7.5

```

1 function correlationByMember(group, movieDB, correlations, memID, upTo) {
2     if (memID === upTo) return correlations;
3     let eID = 0; // entry ID
4     correlations[memID] = [];
5
6     const memberRatings = group[memID]["aboveThreshold"];
7     correlations[memID]["memberRatings"] = memberRatings;
8     correlations[memID]["entries"] = [];
9
10    memberRatings.forEach(entry => {
11        // Find the movie from 'entry' in the MovieDB (uses binary search)
12        const entMov = dataHandler.findMovieByID(entry.movieID, movieDB);
13        correlations[memID]["entries"].push([]);
14        movieDB.forEach(movie => correlationByMovie(movie, entMov, memID,
15            eID, correlations));
16        eID++;
17    });
18
19    return correlationByMember(group, movieDB, correlations, ++memID, upTo)
20        ;
}

```

Listing 7.5: correlationByMember

Som det kan ses af koden er funktionen rekursivt defineret, hvor den tager et start memberID (*memID*) og en værdi *upTo* der angiver for hvor mange medlemmer der skal findes korrelationer til. Hver gang funktionen kalder sig selv, tælles memberID'et 1 op. I funktionen definerer vi *memberRatings* som alle de film en bruger har ratet, og som ligger over grænseværdien (A_THRES). I linje 10 itererer vi over disse film, hvor løkken finder entry-filmen i databasen med en hjælpefunktion der benytter binary search, og herefter sammenlignes denne entry film med alle andre film i databasen, for at finde korrelation ift. genre. Dette er funktionen **correlationByMovie()** ansvarlig for.

```

1 function correlationByMovie(movie, entMov, memID, eID, correlations) {
2     // Get correlation between the 'entry' movie and 'movie' - PEARSON
3     CORRELATION between two arrays.

```

```

3   const corVal = jStat.corrcoeff(entMov["genres"]["genreNumArray"], movie
4     ["genres"]["genreNumArray"]);
5
6   // If the correlation is above some set values and the average rating
7   // of the movie with the high correlation,
8   // then the correlation and movie should be added to the list of movies
9   // to recommend.
10  correlations[memID]["entries"][eID].push({corVal, movie});
11 }
```

Listing 7.6: correlationByMovie

Som set i listing 7.6 på linje 3, så er det netop her at Pearson korrelation bliver taget i brug, med henblik på at finde korrelationer mellem genre, når der sammenlignes input film brugerens har set, med andre film i movieDB-databasen. Der benyttes jStat biblioteket, hvor Pearson korrelation er repræsenteret med **corrcoeff()** funktionen.

7.5.2 Database og hukommelsesbrug

Som set af figur 6.2, indgår der store objekter og arrays, som bl.a. indeholder brugere, film og ratings. Især filmdatabasen viste sig større end først forventet. Dette skyldes, at der efter filtrering, er 9742 filmobjekter, som hver indeholder titel, genre, ratings mm. Hertil er det oplagt, at benytte sig af en database såsom MongoDB eller MySQL (diskuteret i afsnit 7.4.1). Dog blev dette valgt fra, til fordel for et array, som indeholder alle filmobjekterne. Selvom dette er mere krævende, er det den bedste løsning, når omfanget af projektet tages i betragtning. Eftersom, at alle filmobjekter er indeholdt i et array, påvirker det effektiviteten og køreevnen af programmet, hvilket også gav problemer, da dette array skulle gennemløbes. Et eksempel på dette er, at vi først forsøgte at sammenligne alle film i filmdatabasen med alle andre film i databasen. Dette gav i bedste fald en tidskompleksitet på $\Theta(n \cdot \frac{n}{2})$. Hvilket endte i 47.453.282 operationer. Hertil skal det sluttes, at konstanter fjernes, når man regner tidskompleksitet. Altså bliver den reelle køretid $\Theta(n^2)$. Dette endte samtidig i, at der blev indlæst for mange objekter ind i et nyt array, som betød, at det krævede mere end 18GB RAM, for at serveren kunne køre. På dette grundlag blev det besluttet, at korrelationerne ikke skulle findes under serverstart, men først når gruppen er valgt og indlæst. Denne beslutning bakkes yderligere op, af en antagelse om at en gruppe maksimalt har 100 film, som ligger over grænseværdien (se afsnit 4.2.2 for uddybelse). Hvis denne antagelse holder, udføres der maks 1.000.000 operationer sammenlignet med tidligere 47.453.282. Altså en klar forbedring. Derudover spares der også en masse hukommelse, da der ikke bliver overført et objekt for hver operation, men derimod kun udføres simple udregninger og sammenligninger.

7.6 UI design

Programmet er indtil videre blevet beskrevet kodemæssigt med fokus på: effektivitet, valg, kodestandard, mm. Denne sektion vil fokusere i højere grad på hvad dette program gør for brugeren, programnets virke til at være intuitivt, minimalistisk og designets valg. Ved første møde ved hjemmesiden, ser man forsiden, som vist på figur 7.2. intentionen var at lave et program, som er intuitivt, simpelt og brugervenligt. Der er

The screenshot shows the homepage of the P2.RecSys application. At the top, there is a navigation bar with links for 'Home', 'Get Movie Recommendation', and 'Add friends'. On the right side of the bar are 'Rune' and 'Logout' buttons. Below the navigation bar, there are two main sections: 'Your rated movies' on the left and a movie detail page for 'The Terminal' on the right.

Your rated movies:

Movie Title	Rating	Action
Spartacus (1960)	3/5	X
Avatar (2009)	3/5	X
Avengers: Age of Ultron (2015)	1/5	X
Howl's Moving Castle (Hauru no ugoku shiro) (2004)	5/5	X
Dr. No (1962)	4/5	X
Mary Poppins (1964)	4/5	X
Taken (2008)	4/5	X
Willy Wonka & the Chocolate Factory (1971)	3/5	X
The Imitation Game (2014)	5/5	X
Jaws (1975)	3/5	X
Conjuring, The (2013)	3/5	X
Kung Fu Panda (2008)	5/5	X
Kung Fu Panda 2 (2011)	5/5	X
Kung Fu Panda 3 (2016)	5/5	X

Delete all ratings

The Terminal (2004) Description:

Viktor Navorski is a man without a country; his plane took off just as a coup d'état exploded in his homeland, leaving it in shambles, and now he's stranded at Kennedy Airport, where he's holding a passport that nobody recognizes. While quarantined in the transit lounge until authorities can figure out what to do with him, Viktor sneaks around, living, and courts romance with a beautiful flight attendant.

Filter & Search:

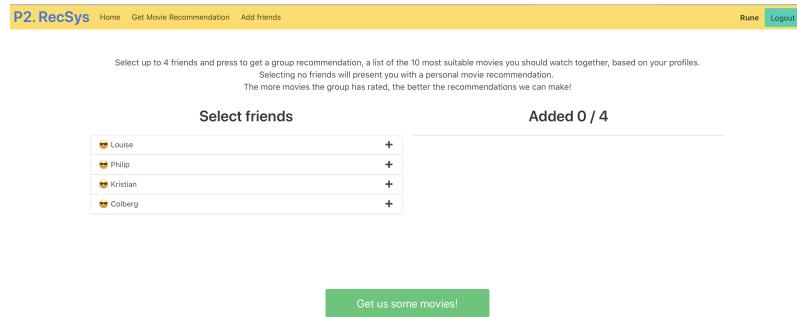
From year: 0 To year: 2020

Enter a movie name to search

Figur 7.2: Programmets forside - HTML

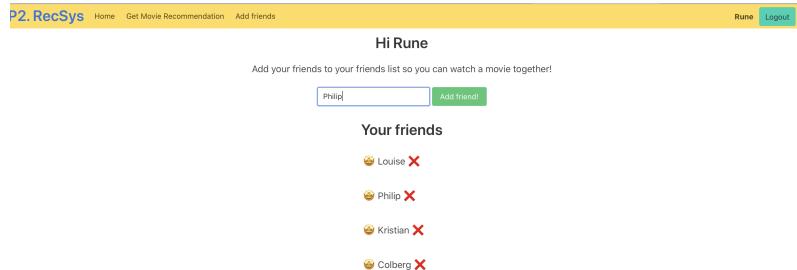
derfor benyttet farver, som kan hjælpe brugeren til at forstå knappernes funktion. Der er derfor brugt rød, hvis brugeren laver en drastisk ændring, som ikke kan gøres om. Dette er i denne sammenhæng på forsiden, hvis man sletter ratings. Grøn er brugt til de mest essentielle knapper, som at "rate" en film eller logge ud samt at logge ind. Gul er brugt til de mere neutrale kommandoer, som at skifte side eller skifte film fordi den ikke er kendt af brugeren.

Filmplakaten er i fokus og centreret, da det er vigtigt at brugeren lægger mærke til dette. Der er en navigationsbar i toppen, som skal gøre det nemt for brugeren at finde rundt på hjemmesiden. På figur 7.3 ses det, hvordan brugeren kan få anbefalet film samt tilføje venner til sin gruppe.



Figur 7.3: Side hvor man får en filmanbefaling - HTML

Denne side benytter den samme funktionalitet med farver til knapper og gør det derved simpelt og intuitivt at bruge. Man kan her tilføje en ven til sin "filmgruppe". Det er denne gruppe inklusiv sig selv, man vil få anbefalinger sammen med. Når filmgruppen er dannet, trykker man på knappen for at få en liste over 10 film, som man anbefales at se. Der er derudover også en side til at tilføje nye venner til sin venneliste. Siden fremgår af figur 7.4. Her kan man tilføje venner ved hjælp af en input tekstboks og en grøn knap.



Figur 7.4: Side til at tilføje venner - HTML

Der er også mulighed for at slette venner. Det er igen meget simpelt, og intuitivt sat op.

Kapitel 8

Test

At unit teste sit program er af enorm vigtighed. Disse tests gør det netop lettere at identificere eventuelle fejl, og derfra rette op på fejlene.

Ved unit testing er det nemt at isolere og reparere fejl under udviklingen, og derved undgå langsomme processer for at isolere hvor en fejl opstår. En fordel ved unit testing er desuden at det fremmer divide-and-conquer programmeringsstrategien, hvilket bidrager til at ens kode bliver bedre, og af højere kvalitet.

I dette kapitel vil vi gennemgå nogle af de tekniske test der er blevet udført for at sikre korrekthed af vores algoritme ift. forventninger, og mere generelt hvordan der er opnået enkelte af de opstillede kravspecifikationer.

8.1 Vores tilgang til test

For at kunne teste vores anbefalingsalgoritme, selvom den ikke var færdigudviklet, blev der lavet en testgruppe bestående af fem brugere fra MovieLens databasen. Dette gjorde det muligt at tjekke hvert enkelt led i koden igennem hele processen.

En del af vores krav giver det mening at brugerteste, og observere om funktionaliteterne har den ønskede virkemåde. Dette gælder f.eks. kravene vedrørende bruger-systemet, og de ting man kan med dette, såsom logge ind og ændre præferencer. De væsentlige tekniske test der menes at skulle udføres, er korrekt indlæsning af vores data, samt den centrale anbefalingsalgoritme. Vi stiller os et krav om at kunne generere 10 filmanbefalinger for enhver gruppe, hvor anbefalingerne i øvrigt er rangerede. Dette er betydningsfuldt at vurdere korrektheden af.

8.2 Test framework

Vi har valgt at teste vores kode ved hjælp af frameworket Jest. Andre mulige frameworks var eksempelvis Jasmine, Mocha og AVA. Dog valgte vi Jest grundet simpliciteten, hastigheden samt en udtømmende dokumentation, som har hjulpet os godt i gang.

Især simplicitet fyldte meget i valg af framework, da der eksisterer ingen, eller sparsom kendskab til det, at teste. Dette gjorde at vi kunne fokusere på at skrive gode tests, som bidrager til at vi kan sikre os, at vores program virker som det skal, da vi ikke skal takle en stejl læringskurve inden da.

8.3 Udvalgte tests med Jest

I de følgende afsnit vil vores whitebox test blive beskrevet, mens brugertest vil blive beskrevet i afsnit 8.5, processen generelt i afsnit 8.4, samt derudover vil dynamiske blackbox test blive forklaret i løbet af hele testafsnittet.

8.3.1 Test af gruppeanbefaling

Et af de væsentligere testområder er naturligvis korrektheden af anbefalingsalgoritmen til en gruppe. Hertil laves der flere testfunktioner, for at sikre at kravspecifikation 2 og 4 er opfyldt. I de følgende to eksempler på tests benyttes en funktion `getTestData()`, som asynkront læser fra en fil hvor vi har en testgruppe, og returnerer resultatet af vores `makeGroupRecommendations()` funktion. Outputtet fra dette skulle være et array bestående af de film, det anbefales at gruppen ser, hvor listen i øvrigt er sorteret, så den mest passende anbefaling er placeret først i arrayet.

Vores første test, som fremgår af listing 8.1, gik på blot at bekræfte det returnerede array bestod af det rigtige antal elementer, hvor 10 elementer var ønsket.

```

1 test('Correct finalArray', async () =>{
2     const finalArray = await getTestData()
3
4     expect(finalArray.length).toBe(10);
5
6 });

```

Listing 8.1: Test af størrelse på output-arrayet listen af anbefalede film

En anden test gik på at sørge for, at de elementer der var ordnet i arrayet, lå i den rigtige rækkefølge ift. korrelationer. Med andre ord, så skal den første indgang være den bedst passende film til gruppen, den næste indgang den næstbedste osv. Dette blev testet som vist i 8.2, hvor der itereres over hver indgang i arrayet, hvor den næstes korrelation skal være mindre eller lig med det forrige element.

```

1 test('Correct ordered correlations', async () => {
2     const finalArray = await getTestData();
3     for (let i = 0; i < finalArray.length - 1; i++){
4         expect(finalArray[i]["correlation"] >= finalArray[i+1] ["correlation"]
5             "").toBe(true);
6     }

```

Listing 8.2: Test af ordning på output-arrayet

I nogle tilfælde opleves det at indgange har fuldstændig ens korrelationer. Det gør det besværligt at fastsætte at én af anbefalingerne er bedre end en anden, da de vil fremstå som lige gode. Selvom denne situation kan opstå, så vil betingelsen om "mindre eller lig med" være opfyldt, og testen vil lykkedes.

Ved kørsel af Jest, opnår vi et output i terminalen der, ved succes, ser ud som i figur 8.1.

```
C:\git\p2>npm test
> p2 b2 2@61.0.0 test C:\git\p2
> jest --coverage

 PASS assets/scripts/recSys/groupRecommendation.test.js
 PASS assets/scripts/data/data.test.js (5.086s)

          File      % Stmts   % Branch   % Funcs   % Lines   Uncovered Line #
-----+-----+-----+-----+-----+
 All files        53.15    31.4     40.63    55.35
-----+-----+-----+-----+-----+
 p2
  initialize.js  25.15    0         3.23    30.08
  utility.js     19.47    0         0         21.78
  ...31-133,135-137,139,143-145,147,151-153,155,159-161,163,166,167,169-171,174,176,178
 p2/assets/scripts/data
  dataHandler.js 58.02    65         5.56    56.25
  loadData.js    48.76    61.54    31.03    47.71
  ...64-166,169,173-175,177,178,183,184,190-192,197-199,203,204,207-209,213-217,221,222
 p2/assets/scripts/recSys
  groupRecommendation.js 95.92  93.33    95.83    95.24
  ...165-168
 p2/assets/scripts/user
  user.js         33.33    0         0         38.89
  ...38.89
  user.js         33.33    0         0         38.89
  ...11-14,19,22,25,27-29,33

Test Suites: 2 passed, 2 total
Tests:       10 passed, 10 total
Snapshots:  0 total
Time:        7.285s
Ran all test suites.
```

Figur 8.1: Output ved kørsel af Jest Test

8.3.2 Test af data

Da vores løsning læner sig op ad data fra MovieLens datasættet, er det væsentligt at sikre sig, at dette data indlæses korrekt.

Derfor tester vi indlæsningen deraf gennem nogle testfunktioner. Til dette har vi valgt at bruge kendte konstanter ved dataet, så som antallet af film i datasættet, antallet af bedømmelser i datasættet, samt information om den første film i datasættet.

Vi tester indlæsningen af vores filmdata gennem testfunktionen som ses i listing 8.3.

```

1 const MOVIES_IN_MOVIE_DB = 9742;
2 const FIRST_MOVIE_IN_DB = "Toy Story (1995)";
3
4 test('Loads movie data', async () => {
5     let tempMovieDB;
6     await loadData.getMovieData().then(res => tempMovieDB = res);
7     expect(tempMovieDB.length).toEqual(MOVIES_IN_MOVIE_DB);
8     expect(tempMovieDB[0].title).toEqual(FIRST_MOVIE_IN_DB);
9 })

```

Listing 8.3: Test for indlæsning af filmdata

Her anskaffes dataet blot gennem vores `getMovieData()` funktion, og sammenlignes så med hvad vi ved om dataet. Det samme gøres når vi indlæser 'links.CSV' filen fra MovieLens datasættet, samt for at teste vores behandlede data over givne ratings fra datasættet.

Alle tre fornævnte tests er væsentlige ift. vores behandling af dataet, da det har en direkte påvirkning for præsentationen og outputtet for brugeren.

Vi har også valgt at teste om vores databehandlings-funktioner giver det korrekte output. Her har vi brugt den viden vi har om konstanterne i dataet, hvor vi sammenligner dem med de outputs vi får. Eksempelvis, så er det vigtigt at vi kan indlæse alle de ratings givet til en film fra brugere i datasættet. Derfor tester vi, om de funktioner vi har lavet til dette, giver det korrekte output. Et eksempel herpå ses i listing 8.4.

```

1 const MOVIE_ID_TO_TEST = "1";
2 const MOVIE_HAS_X_RATINGS = 215;
3
4 test('Gets ratings for movie, given an id', async () => {
5     let ratingsData = await loadData.getRatingDB();
6     let tempRatings = await dataHandler.getRatingsForMovieID(
7         MOVIE_ID_TO_TEST, ratingsData);
8     expect(tempRatings.length).toEqual(MOVIE_HAS_X_RATINGS)
9 });

```

Listing 8.4: Test for at filmens antal af ratings er korrekt

8.4 Test processen

I denne sektion vil der blive fokuseret på processen at teste vores funktioner ved output, som ikke var optimale.

8.4.1 Højt RAM forbrug

I programmet benyttes der en film database, som indeholder 9742 film med tilhørende genrer. Ud fra filmdatabasen lavede vi et objekt med de forskellige genrer indeholdt og en "boolsk" værdi. Opbygningen af objektet kan ses på listing 8.5.

```
1 module.exports.getGenresFromMovie = function getGenresFromMovie(movieEntry) {
2   // All the genres that exist in the movieDB
3   let genres = {
4     'Adventure': 0, 'Animation': 0,
5     'Children': 0, 'Comedy': 0,
6     'Fantasy': 0, 'Romance': 0,
7     'Drama': 0, 'Action': 0,
8     'Crime': 0, 'Thriller': 0,
9     'Horror': 0, 'Mystery': 0,
10    'Sci-Fi': 0, 'War': 0,
11    'Musical': 0, 'Documentary': 0,
12    'IMAX': 0, 'Western': 0,
13    'Film-Noir': 0, '(no genres listed)': 0
14  };
15}
```

Listing 8.5: Genre objekt

Objektet herover blev pushet til et array af film. Dette var ikke hensigtsmæssigt, da objektet blev pushet ca. 45 millioner gange, og dermed optog omkring 18 GB RAM. Derfor blev det istedet omstrukturert til et array af tal, som kan ses i listing 8.6.

```

1 const genreString = movieEntry.genres;
2
3 // In the CSV file movies are written in the form e.g.: Comedy|Crime|
4 // Thriller
5 const splitString = genreString.split(GENRE_SEPARATOR);
6 // Each of the elements obtained through the splitting
7 splitString.forEach(genre => { genres[genre]++; });
8
9 let genreNumArray = Object.keys(genres).map((value) => {return genres[
10   value]} );
11
12 return {genres, genreNumArray};
13 }
```

Listing 8.6: Omstrukturering af genre objektet til array af tal.

Denne fejl blev ikke fundet ved hjælp af Jest eller trinvise `console.log()`. Den blev derimod fundet ved at prøve at køre programmet, hvilket kørte meget ustabilt og førte til et crash for nogle af gruppens pc'er. Gruppen prøvede at lave en funktion til test af 'Memory usage,' da teorien lå i at der muligvis blev brugt meget. Funktionen kom desværre ikke rigtig op at køre, da programmet for det meste crashede. Funktionen kan ses i listing 8.7.

```

1 module.exports.logMemoryUsage = () => {
2   const used = process.memoryUsage();
3   for (let key in used) {
4     console.log(`${key} ${Math.round(used[key] / 1024 / 1024 * 100) /
5       100} MB`);
6   }
6 }
```

Listing 8.7: Memory Usage funktion

Der blev derfor benyttet en task manager for at se om der blev brugt meget CPU-kraft eller RAM, hvilket så viste at der blev brugt omkring 18 GB RAM.

8.4.2 Fejl i output

En anden fejl, som programmet har været igennem er fejl i output. I programmet bliver filmene sorteret med en grænseværdi, som siger at en given film skal have en minimum gennemsnitsrating på 3. Der var til gengæld film i det sidste endelige array med film under 3 i rating, hvilket vækkede undren. Heller ikke her var en tilstrækkelig testfunktion opstillet, men ved en gennemlæsning af programmet blev det opdaget, at fejlen skete, fordi at det indeks, som indeholdt den højeste korrelation blev fjernet, da det blev overført til et andet array, og derved blev alle efterfølgende indekser skubbet til position før deres oprindelige indeks. Det betød, at når vi senere forsøgte at hente det givne indeks, som nu var fjernet, modtog programmet en forkert film. Dette blev fikset ved at kopiere arrayet og fjerne elementer fra kopien ved brug af funktionen `splice()`, i stedet for det originale array. En præsentation af dette ses af listing 8.8

```

1 let index = [];
2 let i = 0;
3 const bTop = topArray.slice();
4
5 // Adds the index of the top-correlation to index[] and removes it from
6 // the topArray
7 for (i = 0; i < 10; i++) {
8     index[i] = bTop.indexOf(Math.max(...bTop));
9     bTop.splice(index[i], 1, 0);
}

```

Listing 8.8: Kopiering af originalt array samt sletning af elementer fra kopien

8.4.3 Poster fejl med Vue.js

I projektet er der benyttet Vue.js til flydende on-click events i HTML delen. Dette har været rigtig nyttigt, men det skabte én fejl. I programmets slutning er det målet at få anbefalet en håndfuld af de mest relevante og bedste film til brugeren. Dette skulle gøres ved hjælp af visning af en titel og en plakat til brugeren for enhver film, der anbefales. Dette kunne gøres intuitivt ved hjælp af en **v-for** og JavaScript funktion, men dette skabte en del problemer. Men plakaterne blev aldrig vist på siden. Dette problem blev aldrig løst ved hjælp af Vue.js, og blev derfor i stedet fikset ved hjælp af en **forEach()** løkke, som bliver aktiveret af en vue @click.

8.5 Brugertest

For at sikre programmets funktionalitet er der udført en masse whitebox testing, som fremgår af de foregående afsnit, men det er også nødvendigt at sikre, at det ikke kun virker i teorien, men også i praksis. Det er derfor nødvendigt at blackbox teste i form af brugertest. Vi har derfor udspurgt en gruppe på 7 personer, om feedback på programmet. Feedbacken blev givet i form af en survey, som er lavet med Google Sheets. Hvis det ønskes at læse surveyen samt svar, findes disse i bilag C. Foruden at teste programmets praktiske kvalitet er dette spørgeskema udformet, så vi er sikre på, at vi har opfyldt krav 1, 2 og 3.

For at tydeliggøre resultaterne opstilles en liste med hovedpunkter fra svarene:

- Forsiden er meget simpel og nem, men dog også lidt kedelig. Mangler en smule finesse for at være lækker
- Alle brugerne havde gode oplevelser med at oprette en bruger. Generelt er kommentarene, at den er hurtig, responsiv og uden tydelige fejl
- Søgefunktionen virker, og især det faktum, at filmene dukker op som forslag, inden hele titlen er skrevet falder i god jord. Dog er der problemer med at ikke

alle film eksisterer, samt at hvis titlen indeholder "the", skal dette stå til sidst

- Når folk indtaster ulovligt input i de år man vil søge i, kommer der en fejlmeddeelse. Dog mangler folk en dedikeret søgeknap, da det ikke er intuitivt, at man skal trykke på "unknown" for at aktivere filtreringen af årstal
- Der er stor tilfredshed med anbefalingerne for både gruppe og individ. Dette er specielt en vigtig observation, da det er kernen i vores projekt
- Der opstod kun en fejl undervejs, hvilket var, at der ikke kom en ny film, hvis man forsøgte at rate en film, man allerede har ratet. Derudover virker programmet ikke på Microsoft Edge eller Internet Explorer. Dette er ikke en fejl, men afhænger derimod af om browserne er kompatible med de brugte frameworks
- Generelt er der stor tilfredshed med programmet, med flere der gerne så hjemmesiden lanceret. Dog mangler der en smule nice-to-have features, for at optimere brugeroplevelsen

Af svarene er det tydeligt, at programmet findes overskueligt og intuitivt, dog er grænsefladen en smule kedelig, hvilket kunne fikses med mere farve og dynamik. Alle brugere giver udtryk for tilfredshed med brugeroprettelse, og ingen forsinkelse ved brug af hjemmesiden. Der er derudover vist begejstring for søgefunktionen, men en mangel hertil er en aktiv søgeknap, da man på nuværende tidspunkt skal benytte knappen "unknown", hvilket ikke er intuitivt. Dette leder til næste punkt, som er brugervejledning. Dette var en generel kommentar fra testpersonerne, som alle ønskede en lille bruger vejledning ifølge med programmet, evt. ved en lille note i siden eller pop-up besked. Af testpersonernes svar, ses det, at alle funktionaliteterne virker i praksis og reagerer korrekt både på lovlige og ulovlige handlinger.

Helt centralt for projektet er anbefalingerne, og korrektheden af disse. Testpersonerne er derfor blevet spurgt om filmene er tilfredstillende, og her svarer alle, at det er film de gerne ville se. Både som gruppe og som individ. Dog viste resultaterne, at mange var endnu mere tilfredse med deres individuelle anbefalinger ift. en gruppeanbefaling. Dette tyder på, at der er en klar korrespondance mellem brugerens præferencer og resultatet. Ydermere viser det også dilemmaet, som ligger i en gruppeanbefaling. Alligevel mener vi at kravet om en god anbefaling er opfyldt, da alle angiver tilfredshed med resultatet.

Slutteligt er gruppen blevet spurgt om deres overordnede tanker om programmet. Her bryder en gennemgående tilfredshed svarene, samt flere der viser interesse for, at få dette implementeret som en reel platform.

Kapitel 9

Diskussion

9.1 Kravspecifikationer

I afsnit 6.2 opstillede vi nogle kravspecifikationer til det færdige program, og det er nu relevant at kigge på, om vores produkt overholder de givne krav. Første hovedpunkt, 1, er formuleret således:

1: Et bruger-system, hvor hver bruger har en personlig konto på websiden. [...]

Kigger vi underpunkterne til første hovedpunkt, står det hurtigt klart, at de alle er opfyldt. Det er muligt, at oprette en bruger på hjemmesiden, samt er det muligt at ændre sine filmpræferecer, ved at rate nye film, eller fjerne gamle ratings. Det er endvidere muligt at tilføje andre brugere på siden som 'venner', hvilket gør det muligt at få anbefalet film sammen. Ved brugersystemet er der både tjek på server- og klient siden, samt opbevarer en brugers konto filmpræferencer, brugernavn ID og venner. Dette bliver gemt i en JSON-fil, så det er let tilgængeligt. Det kan altså siges, at vi i høj grad opfylder det første hovedpunkt i vores krav.

Hovedpunkt 2 i kravspecifikationerne er formuleret således:

2: Systemet skal kunne anbefale film til en gruppe af venner [...]

Her må det også siges at være opfyldt i tilfredsstillende grad. Vi har implementeret en algoritme, som giver en gruppe på op til fem personer en anbefaling. Outputtet er en liste med de 10 film, som vores algoritme har beregnet til at være de ti bedste film, til den givne gruppe. At vurdere om en anbefaling er god, kan være meget subjektivt, og derfor har vi i afsnit 8.5, lavet et spørgeskema, og indsamlet noget konkret feedback. Heri var konklusionen, at brugerne generelt fik gode anbefalinger, og det var alle sammen film, som de gerne ville se. Derfor mener vi, at vores anbefalinger kan anses som

værende gode. Det skal dog også nævnes, at systemet kan have svært ved at anbefale film, som alle i gruppen kan lide, hvis gruppen har meget forskellige præferencer. Dette er også en af grundene til, at der er sat en maksimal gruppstørrelse på fem, således at man mindsker risikoen for, at gruppen er alt for divers.

Tredje hovedpunkt 3 omhandler den grafiske grænseflade:

3: En intuitiv og brugervenlig grænseflade med følgende krav: [...]

Her vurderes det også, at vi har opfyldt kravet. Websiden er ren og minimalistisk, uden overflødige knapper, grafik og funktioner. Der er endvidere relevante fejlbeskeder og pop-ups, til at hjælpe brugerne til at benytte siden korrekt. Dette kan ses illustreret i afsnit 7.6, som indeholder flere figurer, og har en dybere gennemgang af grænsefladen. I afsnit 8.5 fik vi også feedback på grænsefladen, og her var konsensus at den var funktionel, ligetil og brugbar. Der blev også nævnt, at siden godt kunne have noget mere flair, men dette ligger uden for vores projektkrav.

Fjerde, og sidste hovedpunkt omhandler de tekniske krav:

4: På det tekniske plan, vil vi udvikle vores webapplikation i overensstemmelse med projektkravene, dertil kan vi opstille følgende krav: [...]

Her må det også siges, at kravene er opfyldt i høj grad. Vi har programmeret i JavaScript (ES6), og benyttet NODE.js som backend, samt en kombination af HTML5 og CSS til frontend. Dette kan ses bl.a. i afsnit 7.4.2, hvor vi indlæser data med brug af JavaScript, og i afsnit 7.4.3, hvor vi bygger vores database med JavaScript. Endvidere kan det også ses tydeligt i afsnit 7.5.1, hvor vi gennemgår nogle af de kerne funktioner, som er skrevet i JavaScript, for at kunne løse vores tekniske problem. Vi benytter os også af Pearson korrelationen i en item-item sammenhæng, hvor vi har beregnet det ud fra en givens films genrer. Dette ses i afsnit 7.5.1, hvor vi i listing 7.6 viser koden, hvor vi benytter Pearson korrelationen. For at forbedre vores anbefalinger, har vi også inddraget brugernes egne ratings af filmene, i et vægtningsssystem, beskrevet i afsnit 4.2.2, så korrelationerne passer bedre med brugernes præferencer. Opbygningen med NODE.js som backend ses bl.a. i afsnit 7.2 og afsnit 7.3, samt figur 7.1. Løbende som vi har skrevet programmet, har vi også benyttet Jest, til at teste vores funktioner. Dette bliver gennemgået i afsnit 8.

I afsnit 6.1 tog vi disse kravspecifikationer, og kom med et udkast til, hvordan vi forestillede os, at vores website skulle fungere, og ende med at se ud. Her opstillede vi også kort nogle krav, som vi også har opfyldt. Det er muligt at tilføje og fjerne både ratings og venner, samt er det opstillet på en nem og intuitiv måde, at få sin anbefaling på. Dette kan ses demonstreret i afsnit 7.6. Der er også fejlmeldelser, hvis brugeren prøver at

udføre ulovlige handlinger, samt en dynamisk navigationsbar, som giver hurtig adgang til hjemmesidens forskellige sider- og funktionaliteter. I figur 6.1 i afsnit 6.1 ses vores første skitse af hjemmesiden, som vi forestillede os på daværende tidspunkt.

Der er kun enkelte ting, som i vores tidligere udkast i afsnit 6.1 adskiller sig fra vores endelige produkt. Dette er dog grafiske detaljer, mens indholdet og funktionaliteterne er de samme. Derfor kan vi konkludere, at vi i høj grad har opfyldt de givne kravspecifikationer, som blev opstillet i afsnit 6.1, og afsnit 6.2

9.2 Forbedringsmuligheder

Der findes en række forbedringsmuligheder for programmet, hvoraf nogle allerede er nævnt. Især et loginsystem med password og kryptering heraf synes, at være højeste prioritet for en større skalering. Derudover ville implementation af machine learning også være i højsædet, for en bedre anbefaling. En anden overvejelse er, at hvis en eller flere brugere har mange flere ratings end resten af gruppen, vil de dominere anbefalingerne, da der ikke er taget lige hensyn til alle.

Med henblik på brugeroplevelsen foreligger der også en række muligheder. Her ville det især være fordelagtigt at implementere funktioner som omtalt i brugertest afsnittet (8.5), samt notifikationer, venneanmodninger, linke sin bruger med Facebook, indbyrdes kalender og langt mere der optimerer brugeroplevelsen.

9.3 Anbefalingsalgoritmen

I dette afsnit vil vi diskutere designet af den algoritme der er blevet anvendt i løsningen ift. andre muligheder, samt hvor godt implementationen af algoritmen har været.

I designet på algoritmen er der i vores løsning blevet benyttet genrekorrelationer, som den primære tilgang til at bestemme ligheder mellem forskellige film. Dette var ikke den originale tilgang, hvor tanken i stedet var at benytte ratings. Ratings helt for sig selv er dog problematisk, da man vil opnå det resultat at man anbefales andre gode ratede film, men det kan være vidt forskellige film ift. genre og indhold. Havde tilgangen været en anden, og vi ikke kiggede på ligheder mellem film, men mellem brugere, så kunne man have udviklet en algoritme der ville identificere lignende brugere og anbefale indhold der var højt ratet. Dette ville være en user-user tilgang, frem for vores item-item og kunne nemt resultere i long tail- og cold start problemet.

I udviklingen af et recommender system til en gruppe, er grupperingsstrategien en væsentlig faktor. I en anden rapport med lignende problemstilling, fjernede man brugere der ikke lignede hinanden, hvor vi derimod, fjernede lave ratings - se afsnit 4.2.2. Generelt er der mange måder hvorpå de individuelle præferencer kan kombineres.

Et problem som opstår som følge af grupperingen er, at en bruger ville kunne dominere de anbefalinger som gives ved selv at have bedømt mange film. En mulig løsning ville

være at give brugere med få anbefalinger korrelationer en større vægt, og derved er der større chancer for, at film som de også kan lide ville fremgå på anbefalingslisten. Denne vægt ville være baseret på den procentvise forskel på brugerenes antal af bedømte film. Der skulle dog være en sikkerhedsmekanisme i form af en begrænsning derpå, da brugeren med de få bedømmelser blot ville dominere outputtet i stedet - hvilket ville være endnu værre end det oprindelige problem, da de få bedømmelser er inden for et langt mere koncentreret område.

I implementationen af algoritmen i JavaScript er der blevet benyttet almene arrays som vores datastruktur. Dog ved benytelse af matricer ved at trække på et relevant matematik bibliotek, kunne det lette beregneligheden af programmet. Det ville kræve en opsætning af user-item matricer, så i stedet for at have arrays med objekter der angiver brugerens ratings til givne film, samles disse i en matrix. Fordelen ved en vektoriseret implementation er at kunne foretage numeriske beregninger frem for den iterative tilgang.

9.4 Skaleringsmuligheder

Gennem udviklingsprocessen er der blevet tænkt på, at gøre programmet skalerbart. Der er derfor blevet brugt modulære funktioner, så det nemt kan ændres til det, man ønsker. Der er dog gennem processen opstået dilemmaer, som har ført til manglende skalerbarhed. Et af disse dilemmaer opstår ved spørgsmålet om hvordan databasen skulle konstrueres. Dette blev, som beskrevet i afsnit 7.4.1, ved brug af JSON-filer og er derfor en mindre optimal løsning. Ydermere er datasættet fra MovieLens statisk og programmet vil derfor være delvist forældet efter en årrække, medmindre administratoren henter et nyt datasæt. Det ønskes derfor i fremtidig udvikling at lave dette datasæt dynamisk, eller benytte mere selvstændige og separate kilder. Et andet problem opstod tidligt i fasen, da vi ønskede at bruge machine learning. Dette blev dog hurtigt afvist, da vi ikke så det muligt i omfanget af dette projekt. Det er derfor ikke den bedste løsning på problemet, men hvis løsningen skulle implementeres på f.eks. Netflix, ville det være fordelagtigt, at benytte machine learning i stedet for Pearson korrelation. Trods disse to faktorer, vurderes det, at programmet kan skaleres, hvis dette ændres. Til gengæld er der gennem hele processen blevet opstillet testfunktioner, som er med til gøre programmet mere skalerbart. Foruden disse features er der en række nice-to-have funktioner, som ville være fordelagtige at implementere i fremtiden. Især en gennemgående tvivl, der går på, om programmet kun skal vise nye film, har fyldt meget i gruppens diskussioner, men er endt med, at vi vurderer at, det er bedst at vise den bedste film, også selv om den er gammel. Dette dilemma kunne i fremtidig udvikling løses, ved at lade brugeren bestemme et interval, for hvilke år de vil have anbefalet film.

En parameter, som vi ikke kan ændre, der går ud over skalerbarheden er programsproget. Det opstår, da JavaScript er single threaded, hvilket kan resultere i, at hvis flere brugere forsøger, at få en anbefaling samtidigt, sænkes hastigheden af programmet, og derved opstår der længere ventetid.

Slutteligt ønsker vi, at implementere et krypteret loginsystem, da det på nuværende stadien ikke er nødvendigt med en personlig kode, for at tilgå en profil, hvilket er et kæmpe problem, hvis et produkt som dette skal lanceres.

9.5 Opfyldelse af læringsmål

Vi vil her beskrive hvordan læringsmål menes at være opfyldt, på nogle udvalgte punkter.

9.5.1 Viden

Der er i programmet implementeret forskellige algoritmer såsom binary search. Der er derudover også udviklet en anbefalingsalgoritme af gruppen, som har worst-case tidskompleksitet $\mathcal{O}(n^2)$ hvor n betegner antal af film i databasen. At dette sker er dog meget usandsynligt, da dette kun opnåes, hvis alle film er ratet over 2. Til implementation af dette er der benyttet JavaScript og tilhørende frameworks, som bedst muligt løste opgaven. Her er det især vigtigt at bemærke, at frameworket Jest er blevet benyttet til test, samt egne testfunktioner. Der er under processen blevet udført dynamiske og statiske blackbox test, samt både dynamisk og statisk whitebox test. Derudover er forskellige fejlsценarier og ulovlige handlinger blevet testet, altså såkaldt equivalente partitioning. Læringsmålene inden for viden kan derfor konkluderes at være opfyldt.

9.5.2 Færdigheder

Det er lykkedes at definere en velfaglært problemformulering, hvor løsningen dertil, hvis givet mere tid, kunne implementeres og udnyttes af store firmaer såsom Netflix. Der er indgået mange overvejelser omkring opbygning af løsningen og hvordan denne skulle moduleres, hvilket kan læses gennem rapporten. Gennem hele processen er GitHub blevet brugt til versionsstyring. Der er også gennem projektet blevet udført test, som sikrer at programmet opfylder alle krav. Derfor mener vi, at det er et større program af høj kvalitet og læringsmålene for færdigheder er opfyldt.

9.5.3 Kompetencer

Der er gennem projektet opstillet modeller, som beskriver hvordan det afgrænsede problem tænkes at blive løst. Dette har resulteret i en korrekt fremstilling af et kørbart program, som opfylder kravene og løser problemformuleringen. Beslutninger taget angående programmet er beskrevet i kapitel 7, samt er der udført test, som beskrevet i kapitel 8, hvortil der også er forklaret, hvilke mulige mangler der måtte opstå, og hvordan disse er løst. Det er herudover gennem en problemanalyse blevet belyst, hvorfor dette mangler i virkeligheden, og derved er et godt problem. Undervejs har vi diskuteret, hvilke løsninger der har fungeret bedst, samt diskuteret hvordan disse er bedst

implementeret. Der er derfor opnået en stor viden inden for research, frameworks og test, samt en bedre forståelse for virtuelt gruppearbejde.

Kapitel 10

Konklusion

Vi gjorde os en række afgrænsninger i problemanalysen, der endte med at styre vores P2 projekt mod, at undersøge hvordan der kunne udvikles et recommender system, der anbefaler film til en gruppe, hvor fokus endvidere er at præsentere løsningen gennem en webapplikation. Vi analyserede forskellige tilgange til filtrering, forskellige nuværende anvendelser af recommender systems i kommercielle platforme, og identificerede gruppe recommender systems som et felt, der ikke er belyst i stor grad. Typisk er algoritmerne anvendt til individuelle anbefalinger, så når man skal koble kontekstuel information til anbefalingen, skal grupperingsstrategien overvejes.

Med et ønske om at udvikle en platform hvor f.eks. en gruppe venner kan få anbefalet passende mulige film til deres filmaften, blev en problemformulering defineret, omkring hvordan der gennem en webapplikation, udviklet i Javascript, kunne laves et film-recommender system til grupper. Vi har derefter opstillet krav og koncepter, som vi har opfyldt ved at implementere en platform, hvor brugere kan logge ind, søge på og rate film, tilføje venner til sin venneliste, og kunne anmode om anbefalinger til film på baggrund af de individuelle præferencer i den dannede gruppe.

Vores løsning til dette problem ligger på en offentlig server¹

I forbindelse med udviklingen af programmet har der været en del diskussionspunkter, som den metodik vores algoritme arbejder på, og den måde den konkrete kodeimplementation har foregået, men også en del features og forbedringsmuligheder der kunne berige platformen. Et af de forbedringsmuligheder der kunne implementeres er hvorledes en vektoriseret implementation kunne være bedre end den valgte løsning, som ligger sig op ad SLIAL kurset på semesteret.

Som studieprojekt er der en del faglige mål forbundet med arbejdet, og det ønskes også dels at inddrage viden fra de kurser vi har haft kørende på semesteret. Disse ting mener vi at have opfyldt i stor grad, da der gennem projektet er udviklet en bedre viden og forståelse af, hvordan en virkelig webapplikation kan implementeres. Vi har gennem dette også udviklet vores kompetencer og færdigheder inden for programmering og

¹<https://p2-test.herokuapp.com/>

gruppearbejde som helhed.

- Der blevet udviklet et større program i JavaScript og NODE.js, hvor vi altså har trukket på klient-server teori fra IWP-kurset
- Ift. programmeringssproget har vi været omhyggelige med at overveje de begreber og faciliteter der gør den særlig, bl.a. ved at benytte de relevante array metoder, i stedet for for-løkker
- Test har også været væsentligt, hvor der er blevet brugt Jest-frameworket til at lave unit-tests, hvor vi har forsøgt at koble de mindre tekniske (low-level) tests til deres væsentlighed for det overordnede program
- Kurset ALG er blevet benyttet, når der skulle udtænkes en tidseffektiv algoritme til anbefalingen, ydermere er der blevet brugt binary search, som er kendt fra dette kursus
- SLIAL er blevet brugt, da lineær algebra i form af vektorregning har spillet en central rolle i vores anbefalingsalgoritme
- Mere generelt ift. problemet, har vi defineret en afgrænset problemformulering og argumenteret undervejs for hvordan vores platform er en løsning hertil. Denne problemformulering vurderes på baggrund af rapporten også at være løst i en tilfredsstillende grad

Kapitel 11

Perspektivering

Vi har nu både diskuteret og konkluderet i vores rapport, og som det kunne ses, lever vores produkt i høj grad op til de kravspecifikationer, som vi opstillede i afsnit 6.2. I indledningsfasen af projektet, beskrev vi de mange steder, hvor et recommender system kunne benyttes. Særligt i afsnit 2.1, hvor vi kommer med konkrete eksempler fra virkeligheden, hvor teknologien benyttes. Derfor er det også relevant, og interessant, at kigge på, hvor vi kunne se vores recommender system blive benyttet i den virkelige verden. Som beskrevet i afsnit 2.1, er recommender system en stor del af mange virksomheders platform, samt bruger virksomheder mange penge på, at forbedre teknologien [24]. Der hvor vores program kunne have relevans, er f.eks hos streamingstjenster. Det kan f.eks være hos Netflix, som har deres eget system til individuelle brugere, men der mangler muligheder for, at kunne få anbefalet en film som gruppe. Her kunne vores projekt være en åbenlys løsning til dem, som samles som venner eller familie, til at se en film sammen i stuen på Netflix. Dette kunne gøres med et samarbejde, hvor programmet implementeres direkte i Netflix's service, eller som en sideløbende hjemmeside, hvor de film man får anbefalet, er film i Netflix's filmkatalog.

Biografportaler kunne også have glæde af et projekt som vores, hvor det kunne være muligt at få anbefalet film, som går i biografen. En biografoplevelse er typisk gjort i en gruppe, derfor er vores projekt en oplagt kandidat til, at hjælpe med at vælge hvilken film, man skal se. Hertil skal der heller ikke foretages de store ændringer i programmet, for at kunne implementere disse muligheder. Det kunne være så simpelt, at sørge for, at de film man anbefales, også går i biografen på daværende tidspunkt.

En tredje oplagt mulighed for vores program, er at lave det som app til sin mobil. Dermed kan man hurtigt hive telefonen frem, når man sidder hele gruppen foran TV'et i stuen, og finde en film. Dermed kan man få filmanbefalinger på farten, når man ikke er i nærheden af en computer med internetadgang.

Der findes altså mange muligheder hvortil vores program kunne implementeres, eller benyttes i den virkelige verden. Som feedback fra afsnit 8.5 indikerer, så giver vores recommender system brugbare film, som folk kan lide. De steder, hvor der kunne fortages flest umiddelbare forbedringer, iflg. vores adspurgte, var grænsefladen, hvor der kunne pyntes med mere interessante farver og dynamik.

Bibliografi

- [1] Abinash Pujahari, Vineet Padmanabhan. *Group Recommender Systems: Combining user-user and item-item Collaborative filtering techniques*. Besøgt 09-03-2020. <https://ieeexplore-ieee-org.zorac.aub.aau.dk/stamp/stamp.jsp?tp=&arnumber=7437606>, 2015.
- [2] Abhinav Ajitsaria. *Build a Recommendation Engine With Collaborative Filtering*. Jul. 2019. URL: <https://realpython.com/build-recommendation-engine-collaborative-filtering/> (bes. 20.02.2020).
- [3] Amazon (company) - Wikiwand. URL: https://www.wikiwand.com/en/Amazon_company (bes. 20.02.2020).
- [4] Baptiste Rocca, Joseph Rocca. *Introduction to recommender systems*. Besøgt 12-02-2020. <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>, 2019.
- [5] Carleton College. *Model-based recommendation systems*. URL: http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html (bes. 26.02.2020).
- [6] CARLOS A. GOMEZ-URIBE, NEIL HUNT. *The Netflix Recommender System: Algorithms, Business Value, and Innovation*. Besøgt 18-02-2020. https://beta.vu.nl/nl/Images/werkstuk-fernandez_tcm235-874624.pdf, 2015.
- [7] DATAFLAIR TEAM. *Data Science at Netflix A Must Read Case Study for Aspiring Data Scientists*. Besøgt 20-02-2020. <https://data-flair.training/blogs/data-science-at-netflix/>, 2019.
- [8] Google. *Content-based Filtering Advantages & Disadvantages*. Nov. 2018. URL: <https://developers.google.com/machine-learning/recommendation/content-based/summary> (bes. 20.02.2020).
- [9] Prince Grover. *Various Implementations of Collaborative Filtering - Towards Data Science*. Dec. 2017. URL: <https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0> (bes. 13.02.2020).
- [10] Jest is a Delightful JavaScript Testing Framework with a focus on simplicity. URL: <https://jestjs.io/en/> (bes. 05.03.2020).

- [11] Christopher C Johnson. *Logistic Matrix Factorization for Implicit Feedback Data*. Tekn. rap. URL: <https://www.netflix.com>.
- [12] Elena Kirzhner. *Machine Learning. Explanation of Collaborative Filtering vs Content Based Filtering*. Mar. 2018. URL: <https://codeburst.io/explanation-of-recommender-systems-in-information-retrieval-13077e1d916c> (bes. 18.02.2020).
- [13] Miklas S. Kristoffersen. *A Dataset for Inferring Contextual Preferences of Users Watching TV*. 2018.
- [14] Greg Linden, Brent Smith og Jeremy York. *Amazon.com Recommendations*. Tekn. rap. URL: <https://ieeexplore.ieee.org/document/1167344>.
- [15] Manish Barnwal. *Types of data in recommender systems*. Besøgt 26-02-2019. http://manishbarnwal.com/blog/2018/09/27/types_data_recommender_system/, 2018.
- [16] Geetha Mohan m.fl. "A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System". I: *Journal of Physics: Conference Series* 1000 (apr. 2018), s. 012101. doi: 10.1088/1742-6596/1000/1/012101.
- [17] movielens. <https://movielens.org/>.
- [18] Music To My Ears: De-Blackboxing Spotify's Recommendation Engine | CCTP-607: "Big Ideas": AI to the Cloud. URL: <https://blogs.commons.georgetown.edu/cctp-607-spring2019/2019/05/06/music-to-my-ears-de-blackboxing-spotifys-recommendation-algorithm/> (bes. 18.02.2020).
- [19] Fernando Ortega. *Recommendation to Groups of Users Using the Singularities Concept*. Jul. 2018. URL: <https://ieeexplore-ieee-org.zorac.aub.aau.dk/document/8404036>.
- [20] Carlos Pinela. *Recommender Systems User-Based and Item-Based Collaborative Filtering*. Nov. 2017. URL: <https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f> (bes. 17.02.2020).
- [21] Baptiste Rocca. *Introduction to recommender systems*. Jun. 2019. URL: <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada> (bes. 18.02.2020).
- [22] Ankur Tomar. *ITEM-ITEM Collaborative filtering Recommender System in Python*. Okt. 2017. URL: <https://medium.com/@tomar.ankur287/item-item-collaborative-filtering-recommender-system-in-python-cf3c945fae1e> (bes. 17.02.2020).
- [23] Tony Maglio. *Netflix Users Spend 18 Minutes Picking Something to Watch, Study Finds*. Besøgt 19-02-2019. <https://www.thewrap.com/netflix-users-browse-for-programming-twice-as-long-as-cable-viewers-study-says/>, 2016.
- [24] Wikipedia. *Netflix Prize*. Besøgt 13-02-2020. https://en.wikipedia.org/wiki/Netflix_Prize, 2009.

Bilag

Bilag A: Kildekode

Se elektronisk bilag.

Bilag B: Programflow

Se elektronisk bilag.

Bilag C: Brugerfeedback

Se elektronisk bilag.