EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2016*

Rekommendationssystem med begränsad data

Påverkan av gles data och cold start på rekommendationsalgoritmen Slope One

ANNA-KARIN EVERT

ALFRIDA MATTISSON

KTH
SKOLAN FÖR DATAVETENSKAP OCH KOMMUNIKATION

Limited Data in Recommender Systems: The Impact of *Sparse Data* and *Cold Start* on the Recommendation Algorithm Slope One

ANNA-KARIN EVERT
ALFRIDA MATTISSON

Degree Project in Computer Science, DD143X
Supervisor: Dilian Gurov
Examiner: Örjan Ekeberg

CSC, KTH. May 11, 2016.

Sammanfattning

I dagens överflöd av information och produkter på nätet, har rekommendationssystem blivit viktiga för att kunna presentera sådant som är intressant och relevant för varje enskild användare. Rekommendationssystem kan både förutsäga produktbetyg och ta fram ett urval av rekommenderade produkter för en användare.

Ett vanligt problem för rekommendationssystem är begränsad data, vilket kan försämra korrektheten i systemets rekommendationer och förutsägelser avsevärt. De vanligaste typerna av begränsad data är *gles data* och *cold start. Gles data* innebär att det finns en liten mängd produktbetyg i förhållande till antalet användare och produkter i systemet. *Cold start* är i stället då en ny användare eller produkt ska läggas till i systemet, och därmed saknar betyg.

Denna rapport har som syfte att studera hur korrektheten i rekommendationsalgoritmen *Slope Ones* förutsägelser påverkas av begränsad data. De situationer som undersöks är *gles data* samt *cold start*-situationerna *ny användare* och *ny produkt*. Rapporten undersöker även om situationen *ny användare* kan avhjälpas genom att låta nya användare betygsätta ett litet antal produkter direkt när de läggs till i systemet.

Sammanfattningsvis, visar rapportens resultat att *Slope One* är olika känslig för de olika typerna av begränsad data. I enlighet med tidigare forskning, kan slutsatsen dras att *Slope One* är okänslig för *gles data*. Vad gäller *cold start,* blir korrektheten avsevärt sämre, och dessa situationer kan således sägas vara problematiska för *Slope One*.

Abstract

In today's abundance of online information and products, recommender systems have become essential in finding what is interesting and relevant for each user. Recommender systems both predict product ratings and produce a selection of recommended products for a user.

Limited data is a common issue for recommender systems and can greatly impair their ability to produce accurate predictions and recommendations. The most prevailing types of limited data are *sparse data* and cold start. The data in a dataset is said to be sparse if the number of ratings is small compared to the number of users and products. Cold start, on the other hand, is when a new user or product is added to the system and therefore completely lacks ratings.

The objective of this report is to study the impact of limited data on the accuracy of predictions produced by the recommendation algorithm *Slope One*. More specifically, this report examines the impact of *sparse data* and the two cold start situations *new user* and *new product*. The report also investigates whether asking new users to rate a small number of products, instantly after being added to the system, is a successful strategy in order to cope with the problem of making accurate predictions for new users.

In summary, it can be deduced from the results of this report that the sensitivity of *Slope One* varies between the types of limited data. In accordance with previous studies, *Slope One* appears insensitive to *sparse data*. On the other hand, for cold start, the accuracy is seriously affected and cold start can thus be said to be problematic for *Slope One*.

Innehållsförteckning

1. Introduktion

1.1 Bakgrund

Rekommendationssystem har som uppgift att klassificera och filtrera data för att förutsäga vilka produkter eller vilken information en användare är intresserad av. I dagens informationsöverflöd och enorma utbud av varor och tjänster på nätet, blir rekommendationssystem därmed ett viktigt verktyg i att välja vad som presenteras för vilken användare [1]. Användningsområdena för dessa system är åtskilliga, med rekommendationer av alltifrån film hos Netflix, till artiklar hos Google och produkter hos Amazon.

Rekommendationssystem kan använda sig av och producera olika typer av data. Den indata som rekommendationssystem använder kan vara antingen explicit eller implicit. Explicit data är data som användaren ombeds lämna, medan implicit data samlas in under användningen av tjänsten [2]. På Netflix kan explicit data till exempel vara betyg användaren ger en film, och implicit data hur länge eller vilken tid på dygnet en användare tittar på en film. Även den data som produceras kan delas in i två huvudtyper, vilka är förutsägelser av betyg respektive ett urval av rekommenderade produkter [3]. Exempel på båda dessa kan även ses hos Netflix, som både förutsäger filmbetyg och ger förslag på filmer en användare tros gilla.

Rekommendationsalgoritmer, vilka är del av rekommendationssystem, kan använda sig av två tekniker. Dessa är *content based filtering* (CBF) och *collaborative filtering* (CF). CBF använder egenskaper hos produkter för att kunna hitta liknande produkter att rekommendera [1]. CF tittar i stället på relationer mellan antingen användare och användare, eller användare och produkter, eller en kombination av båda. Detta för att utifrån likhet ta fram en modell med vilken rekommendationer beräknas. Användare som liknar varandra antas alltså gilla samma produkter [3]. Den största delen av den aktuella forskningen berör CF-algoritmer, så även denna rapport. Rapporten behandlar de betygsförutsägande CF-algoritmerna *Slope One*, NMF och SVD++, med fokus på *Slope One.*

Ett ofta förekommande problem för rekommendationssystem är bristfällig data, något som till viss del förbises i den forskning som mäter korrekthet i rekommendationer. Begränsad data försämrar dock korrektheten avsevärt och bör, då det är ett vanligt problem, tas hänsyn till. Data kan vara begränsad på olika sätt, och två vanligt förekommande sådana hos rekommendationssystem är *cold start* och *gles data*. *Cold start* är då en ny användare eller en ny produkt ska läggas till i systemet [4]. *Gles data* innebär att antalet betyg är litet i förhållande till antalet användare och produkter [5]. I båda dessa fall leder avsaknaden av data till att rekommendationssystemet inte kan bygga en fullgod modell och att noggrannheten i betygsförutsägelserna således minskar.

1.2 Syfte

Denna rapport har som syfte att se hur känslig rekommendationsalgoritmen *Slope One* är för begränsad data. I tidigare studier har *Slope One* visat sig hantera *gles data* utan större svårigheter [6], vilket rapporten ämnar bekräfta. Rapporten kommer även behandla påverkan av *cold start*-situationerna *ny användare* och *ny produkt,* för att få en mer heltäckande bild av hur algoritmen klarar begränsad data. En populär strategi för att förbättra precisionen i rekommendationer för nya användare, är att låta dem lämna ett litet antal betyg på produkter i systemet. Rapporten har därför även som syfte att se om detta är en lämplig strategi för *Slope One*.

1.3 Frågeställning

Hur påverkas rekommendationsalgoritmen *Slope One* vad gäller korrekthet i förutsägelser av betyg, då den utsätts för *gles data* respektive *cold start*?

1.4 Avgränsning

Denna rapport är avgränsad till att studera påverkan av *gles data* och *cold start* på korrekthet i betygsförutsägelser. *Cold start* testas med situationerna *ny produkt* och *ny användare*, varav *ny användare* med noll och ett betyg per användare. Samtliga situationer testas på rekommendationsalgoritmerna *Slope One,* NMF och SVD++. *Slope One* är rapportens huvudsakliga fokus, och NMF och SVD++ används som referensram. De är alla betygsförutsägande *collaborative filtering*-algoritmer.

Tre olika datamängder används som testdata. Dessa är hämtade från databaserna MovieLens [7][8] och FilmTrust [9]. Rapporten är avgränsad till att studera explicit data, vilket är betygsvärden med tillhörande användar-ID och produkt-ID.

Implementationerna av algoritmerna kommer från det GPL-licensierade rekommendationsbiblioteket LibRec [10].

2. Referensram

Detta avsnitt redogör, inledningsvis, för rekommendationssystem med fokus på tekniken *collaborative filtering*. Vidare beskrivs de tre rekommendationsalgoritmerna *Slope One*, NMF och SVD++, samt de olika situationer av begränsad data som testas i denna rapport. Slutligen berörs hur korrekthet kan mätas i betygsförutsägande rekommendationssystem.

2.1 Rekommendationssystem

Rekommendationssystem kan använda två huvudsakliga tekniker för att ta fram betygsförutsägelser eller rekommendationer. Dessa är *collaborative filterin*g (CF) och *content based filtering* (CBF), och det finns även hybrider som använder sig av båda teknikerna. Majoriteten av den senaste forskningen behandlar dock CF- eller hybridalgoritmer.

2.1.1 Collaborative filtering

CF-algoritmer beräknar betygsförutsägelser och tar fram rekommendationer utifrån likhet mellan användare. Detta skiljer sig från CBF-algoritmer, som i stället tittar på hur lika produkter är i sina attribut, såsom genre, användningsområde eller upphovsman [1][3]. I rekommendationssystem med stor tillgång på betygsdata, ger CF-system mer korrekta produktrekommendationer än CBF-system. Situationen är dock omvänd när användar- och produktdatan är begränsad [11].

I en rekommendationstjänst som använder sig av CF, tolkas en användares betyg på produkter som dennes smak. Användare som ger samma produkter snarlika betyg anses därför av systemet ha liknande smak. Systemet bygger således sina betygsförutsägelser och rekommendationer för en användare på hur liknande användare betygsatt produkter [12].

CF-algoritmer kan, fortsättningsvis, vara antingen minnes- eller modellbaserade. Minnesbaserade CF-algoritmer använder hela rekommendationssystemets databas varje gång ett produktbetyg ska beräknas [13]. Modellbaserade CF-algoritmer, å andra sidan, använder datautvinning och maskininlärning för att ta fram en modell med vilken betyg kan förutsägas. När ett betyg väl ska räknas ut används bara en mindre mängd data, som indata till modellen [12][13].

Medan minnesbaserade CF-algoritmer är långsammare i realtidssystem, är de modellbaserade mycket mer tidskrävande vid start och uppdatering, eftersom modellen i båda dessa fall måste byggas om från grunden [13]. Idag är dock uppdelningen i minnes- och modellbaserade

rekommendationssystem inte alltid självklar, då många system använder en blandning av de två för att komma ifrån respektive metods nackdelar [12].

2.1.2 Tidigare forskning

Majoriteten av forskningen kring rekommendationssystem använder kvantitativa metoder och utvärderar algoritmer utifrån graden av korrekthet i deras betygsförutsägelser. För att avgöra hur riktiga förutsägelser som görs, är det vanligt att dela upp datan i en träningsdel och en testdel. På så sätt får algoritmen bygga sin modell utifrån träningsdatan, för att förutsäga resterande värden, vilka sedan valideras mot testdatan [6][14][15].

Forskningen kring rekommendationsalgoritmer syftar, fortsättningsvis, till att ta fram nya algoritmer, utveckla redan existerande algoritmer eller föreslå hybrider av algoritmer, för att på så sätt öka korrektheten i rekommendationssystem [14][16][17]. Det är även vanligt att jämföra två eller fler rekommendationsalgoritmer som tillhör samma ansats, för att avgöra vilken som presterar bäst utifrån valda kriterier [6][14][15][17].

2.2 Algoritmer

I detta avsnitt presenteras rekommendationsalgoritmerna *Slope One,* NMF och SVD++. *Slope One* är rapportens huvudsakliga fokus, och tester på NMF och SVD++ används för att sätta *Slope One* i ett sammanhang.

2.2.1 *Slope One*

Slope One är en betygsförutsägande CF-algoritm. Den är enkel att implementera och snabb att uppdatera när ny data, såsom användarbetyg, ska läggas till i rekommendationssystemet. *Slope One* ger samtidigt en tillfredsställande korrekthet i jämförelse med andra mer komplicerade och, vid uppdatering, mer tidskrävande CF-algoritmer [6][18]. *Slope One* sägs även hantera *gles data* väl, vilket innebär att den tar fram tillräckligt korrekta förutsägelser även då tillgången på data är begränsad [6].

I korthet tittar *Slope One* på hur mycket betygen för en produkt och en användare i genomsnitt skiljer sig från andra produkters respektive andra användares betyg i databasen. Med hjälp av dessa genomsnitt, förutsäger *Slope One* sedan de betyg som ännu inte satts [18].

När *Slope One* ska förutsäga en användare A:s betyg på en produkt P1 utgår den från A:s betyg på andra produkter, till exempel en betygsatt produkt P2. Medelvärdet av differensen mellan betygen för P1 och P2 räknas ut för ett antal användare som betygsatt de båda

produkterna. Summan av detta medelvärde och A:s betyg på P2 blir algoritmens första gissning på A:s betyg på P1. Denna beräkning repeteras sedan för ett antal par bestående av P1 i kombination med en av användare A:s betygsatta produkter. Varje sådan beräkning ger alltså en betygsgissning på P1, vilka sedan viktas utifrån hur många användare som betygsatt båda filmerna i respektive par.

2.2.2 NMF och SVD++

De modellbaserade CF-algoritmerna NMF och SVD++ använder matrisfaktorisering [17][19]. Modeller som använder matrisfaktorisering har visat sig vara skalbara samt ge hög grad av korrekthet i förutsägelser i rekommendationssystem [20].

Singulärvärdesuppdelning (SVD) är, i rekommendationssystem, en grupp av algoritmer, som har blivit populära då de med god precision tar fram rekommendationer och förutsäger produktbetyg. SVD är, fortsättningsvis, en så kallad *latent factor*-modell, vilket innebär att användare och produkter profileras så att de blir direkt jämförbara [17]. SVD++ är en SVD-algoritm, som utökats med att även använda implicit användarhistorik i de fall det finns att tillgå [21][22].

NMF (*non-negative matrix factorization)* består av en grupp algoritmer, som används inom multivariat statistik och linjär algebra. Ett viktigt användningsområde för NMF är rekommendationssystem [23]. NMF tar fram sina rekommendationer genom en linjär modell $V \approx WH$ där V är betygsmatrisen, och W och H är matriser med användarnas respektive produkternas egenskaper [20]. NMF ger, då mängden data är stor och tät, goda förutsägelser i jämförelse med andra CF-algoritmer [24].

2.3 Begränsad data

Begränsad data är ett vanligt problem i rekommendationssystem och försämrar ofta systemens korrekthet avsevärt. Det är därför av stor betydelse att studera effekten av begränsad data. I denna rapport har två olika typer av begränsad data undersökts, vilka är *cold start* och *gles data*.

2.3.1 Gles data

Gles data innebär att det finns få betyg jämfört med antalet användare och produkter i ett rekommendationssystems databas. *Gles data* är mycket vanligt, då de flesta användare endast betygsätter ett fåtal av de många produkter som finns i ett rekommendationssystem [25].

Problemet består, som tidigare nämnt, i att CF-algoritmer överlag kräver en viss mängd data för att kunna göra precisa förutsägelser [11][20].

Rekommendationsalgoritmers förmåga att hantera *gles data* har i tidigare forskning testats genom att minska storleken på träningsdatan, och öka storleken på testdatan [15]. Systemet ska följaktligen med färre betyg förutsäga en större mängd betyg.

2.3.2 Cold start

Cold start innebär att en eller flera nya användare eller produkter ska läggas till i ett rekommendationssystem [11]. Båda dessa situationer är oundvikliga och är, trots att mycket forskning behandlat ämnet *cold start,* fortfarande ett problem [26]. Nedan beskrivs de två *cold start*-situationerna *ny användare* och *ny produkt* mer ingående.

2.3.2.1 Ny produkt

Ny produkt innebär att en eller flera produkter är nya i systemet och därför ännu inte betygsatts av några användare [4]. I *CF* görs rekommendationer baserat på vilka produkter liknande användare gillat, och där kan en ny produkt, som inte betygsatts, aldrig rekommenderas [26][27]. I tidigare forskning har redan existerande algoritmer utvecklats eller kombinerats, med syfte att förbättra korrektheten då de utsätts för *cold start*-situationen *ny produkt* [28][29].

2.3.2.2 Ny användare

Situationen *ny användare* uppstår när en ny användare läggs till i rekommendationssystemet, eller när en användare inte betygsatt tillräckligt många, om ens några, produkter. Avsaknaden av information om användaren leder till att systemet inte kan ta fram precisa betygsförutsägelser för användaren i fråga.

Systemet måste alltså ha en lämplig strategi för att kunna hantera att nya användare saknar betyg. Målet för dessa är att så fort som möjligt lära känna användaren för att kunna ta fram personliga rekommendationer. Detta sker ofta med olika former av datainsamling. Vanligast är att samla in explicit data, genom att be användaren ge *feedback* i form av betyg på produkter eller genom att göra inställningar i sin profil [27]. Fördelen med explicit data är att användaren har kontroll över den information som lämnas [11].

Att be nya användare lämna betyg i början av sin användning, är bara möjligt i tjänster där användaren har tidigare erfarenhet av några av produkterna. Generellt sett går det inte heller att förlita sig på att varken nya eller gamla användare lämnar betyg eller gör inställningar i sin

profil [26]. Således är målet vid insamling av information om nya användare, att det inte är för krävande för användaren men samtidigt ger en tillräcklig mängd data om dennes preferenser, för att öka systemets korrekthet [30][31].

När *cold start*-situationen *ny användare* har studerats i tidigare forskning, har syftet ibland varit att ta reda på hur mycket *feedback* det krävs från nya användare, för att rekommendationssystemets betygsförutsägelser ska bli korrekta nog. Detta har gjorts genom att undersöka om korrektheten bibehålls i större utsträckning om nya användare får betygsätta en, två eller flera produkter [31].

2.4 Korrekthetsmått

Korrekthet i förutsägelser är det vanligaste kriteriet för att mäta hur bra eller dåligt ett rekommendationssystem är, och mäts ofta i MAE eller RMSE. Kriterier såsom serendipitet, mångfald och nyhetsvärde i ett urval rekommenderade produkter är inte lika vanliga, men anses också vara viktiga för att användare ska bli nöjda [32].

MAE och RMSE är mått på fel i prognoser, det vill säga medelvärden på avståndet mellan en modells förutsägelser och motsvarande riktiga värden. De kan båda anta värden mellan noll och oändligheten, och ju mindre värdet är, desto mindre är felet i prognosen. MAE och RMSE beräknas med följande formler [33].

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|e_i\right|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}e_i^2}$$

RMSE är, i studier av rekommendationssystem, det vanligaste korrekthetsmåttet, men kombineras också ofta med MAE. Anledningen till detta är att det inte finns någon konsensus kring vilket av de två som är det mest riktiga måttet på korrekthet [33][34].

3. Metod

I detta avsnitt redogörs för studiens metod. Inledningsvis beskrivs dess generella utformning. Därefter definieras de fyra testfallen *gles data, ny produkt, ny användare (0 betyg)* och *ny användare (1 betyg).* Slutligen beskrivs de datamängder och den programvara som används.

3.1 Studiens utformning

Begränsad data har i denna studie testats med testfallen *gles data*, *ny produkt* samt *ny användare* med noll och ett betyg. Då studiens syfte är att mäta effekten av begränsad data på korrekthet, har det även funnits ett behov av att definiera ett så kallat *normalfall.* Detta för att testa hur algoritmerna presterar i vanliga fall och kunna avgöra hur känsliga de är för begränsad data.

Rekommendationsalgoritmerna *Slope One*, NMF och SVD++ har körts på varje kombination av testfall och datamängd. *Slope Ones* resultat är studiens fokus, men motsvarande resultat från körningar med NMF och SVD++ används i jämförelser.

Varje kombination av algoritm, testfall och databas utgör ett test, som ger ett resultat i form av ett MAE-värde och ett RMSE-värde. Det är vanligt att kombinera de två måtten för att få en mer representativ bild av korrektheten [34]. Varje test repeteras tio gånger och ett medelvärde räknas ut för MAE och RMSE. Tio repetitioner var lämpligt ur tids- och omfattningssynpunkt.

Det slutgiltiga resultatet är hur mycket större eller mindre felet är i fallet av begränsad data jämfört med *normalfallet*. Denna skillnad mäts i procent.

3.2 Testfall

I detta avsnitt beskrivs de fem testfallen *normalfallet, gles data, ny produkt, ny användare (0 betyg)* och *ny användare (1 betyg)*.

3.2.1 Normalfallet

Normalfallet innebär att en betydande del av databasen används som träningsdata, och endast en liten del av datan ska förutsägas av rekommendationssystemet. *Normalfallet* ska motsvara en situation då datan inte är begränsad, och fungerar därför som en referens mot vilken situationerna av begränsad data kan jämföras. I studien testas *normalfallet* genom att

produktbetygen i datamängden delas upp i 80 procent träningsdata och 20 procent testdata. Detta är i enlighet med tidigare forskning [6][14][15].

3.2.2 Gles data

Problemsituationen *gles data* innebär att det finns en stor mängd användare och produkter, men en förhållandevis liten mängd betyg. Detta testas, i enlighet med tidigare forskning, genom att låta algoritmen öva på en mindre andel data än i normalfallet [15]. I denna studie används 20 procent av betygen som träningsdata, och 80 procent som testdata.

3.2.3 Ny produkt

Cold start-situationen *ny produkt* har i tidigare forskning testats genom att för en viss andel produkter, ta bort alla betyg. Dessa används sedan som testdata, och resterande betyg som träningsdata. Hur stor andelen träningsdata är varierar mellan studier, men vanligast är någonstans mellan 10 och 20 procent [28][29]. I enlighet med detta, används i denna studie 20 procent av produkterna som testdata, och 80 procent som träningsdata.

3.2.4 Ny användare (0 betyg)

Ny användare (0 betyg) innebär att nya användare ska läggas till i systemet och att dessa helt saknar betyg. Detta testfall testas i denna studie, genom att använda 20 procent av användarna som testdata, och 80 procent som träningsdata. För de användare som tillhör testdatan, tas samtliga betyg bort. De borttagna betygen ska sedan förutsägas med hjälp av träningsdatan. Liknande uppdelning kan ses i tidigare forskning [27][30][31].

3.2.5 Ny användare (1 betyg)

I testfallet *ny användare (1 betyg)* simuleras att varje ny användare ombeds betygsätta en produkt innan denne börjar använda tjänsten [27][30]. I likhet med fallet *ny användare (0 betyg)* delas datamängdens användare upp i 20 procent testdata och 80 procent träningsdata. I *ny användare (1 betyg)* behålls dock ett betyg för varje användare som tillhör testdatan.

3.3 Datamängder

De datamängder, som används i denna studie, är MovieLens (100k) [7], MovieLens (1M) [8] och FilmTrust [9]. Den data som används är explicit och består av betygsvärden med tillhörande användar-id och produkt-id.

	MovieLens (100k)	MovieLens (1M)	FilmTrust
Antal betyg	~100 000	~1 200 000	~35 000
Skala	[1-5]	[1-5]	[0.5-4.0]
Täthet	6,30%	4,47%	1.14%

Tabell 1: Egenskaper hos datamängderna MovieLens (100k), MovieLens (1M) och FilmTrust.

3.4 Programvara

I studien används programvaran LibRec, som är ett GPL-licensierat java-bibliotek för rekommendationssystem [10]. I rapportens tester har LibRecs implementationer av *Slope One,* SVD++ och NMF använts. *Normalfallet* och *gles data* kunde testas med LibRec, genom att ange procentsatser för test- och träningsdata. Testfallen *ny produkt* och *ny användare* krävde däremot ändringar och tillägg i koden, vilka finns i Bilagor, under avsnitt 7.2.

4. Resultat

I detta avsnitt presenteras resultaten från körningar av algoritmerna *Slope One,* NMF och SVD++ på testfallen *gles data, ny produkt*, *ny användare (0 betyg)* och *ny användare (1 betyg).* Varje sådan kombination av algoritm och testfall har körts på de tre databaserna MovieLens (100k), MovieLens (1M) och FilmTrust. Resultaten från körningarna består, som tidigare har skrivits, av felmåtten MAE och RMSE. Dessa mätvärden kan hittas i Bilagor, under avsnitt 7.1. De bearbetade resultaten utgörs av skillnaden i procent mellan *normalfallet* och respektive fall av begränsad data, och det är denna skillnad som redogörs för nedan. Om skillnaden är positiv, är felet i algoritmens förutsägelser större i fallet med begränsad data, än i *normalfallet*. Om skillnaden är negativ har felet, på samma sätt, blivit mindre än i *normalfallet*.

4.1 Gles data

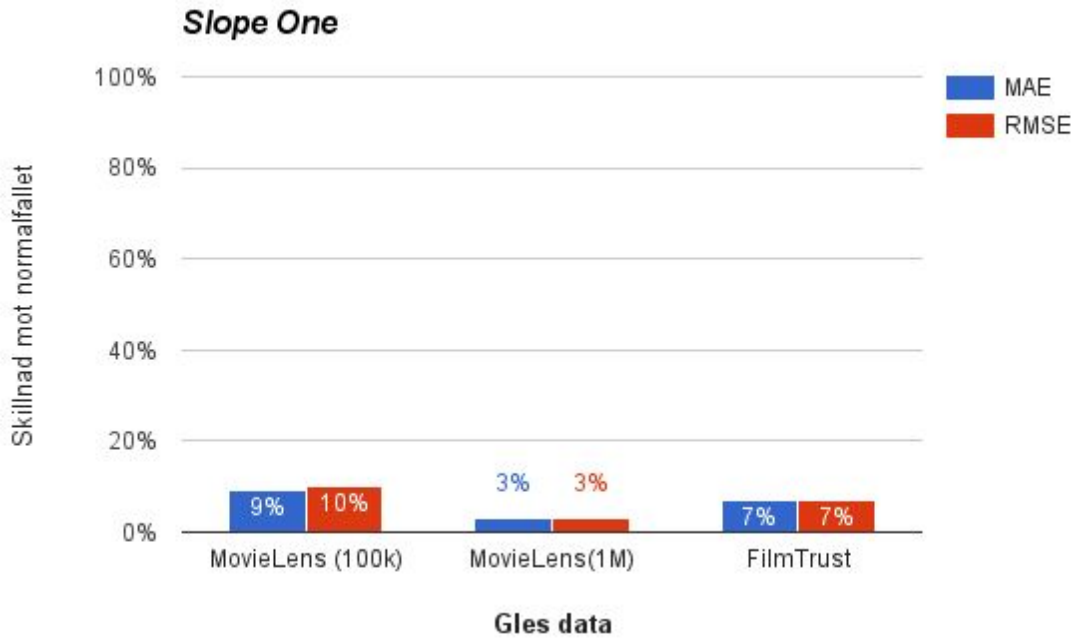

Diagram 1: Den procentuella skillnaden i MAE och RMSE mellan gles data *och* normalfallet *för* Slope One *med respektive databas.*

Resultaten från körningar av *Slope One* på en gles datamängd visar, för samtliga databaser, en försämring av korrektheten (Diagram 1). Minst försämring jämfört med *normalfallet* ses för datamängden MovieLens (1M), med 3 procent i både MAE och RMSE. För MovieLens (100k) är försämringen störst, med 9 procent och 10 procent i MAE respektive RMSE. Hur

mycket större eller mindre felet blir jämfört med *normafallet* varierar alltså beroende på datamängd.

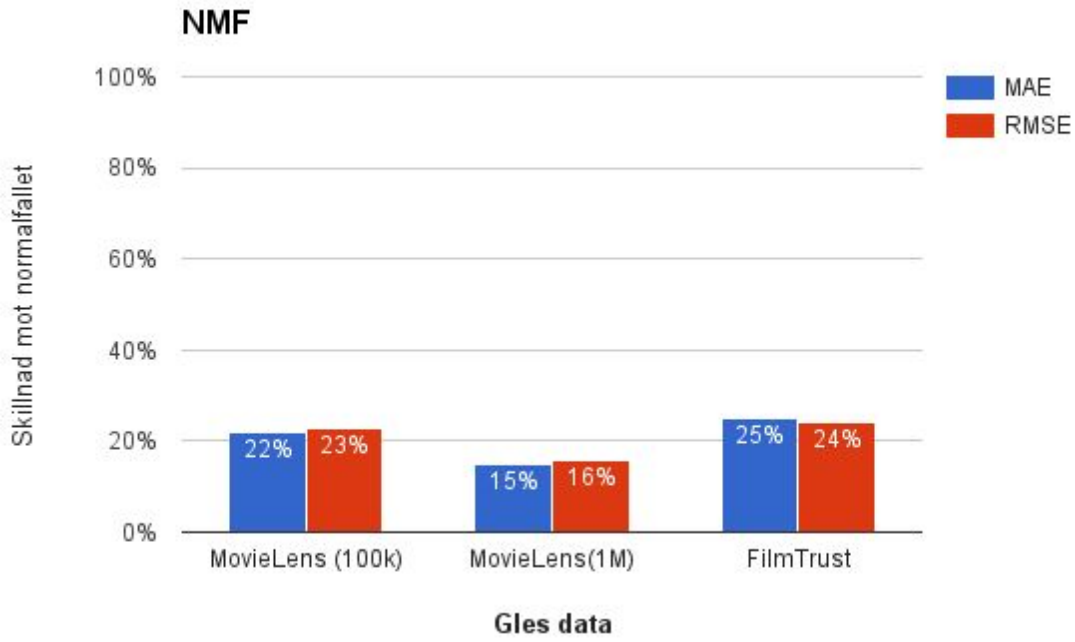

Diagram 2: Den procentuella skillnaden i MAE och RMSE mellan gles data *och* normalfallet *för NMF med respektive databas.*

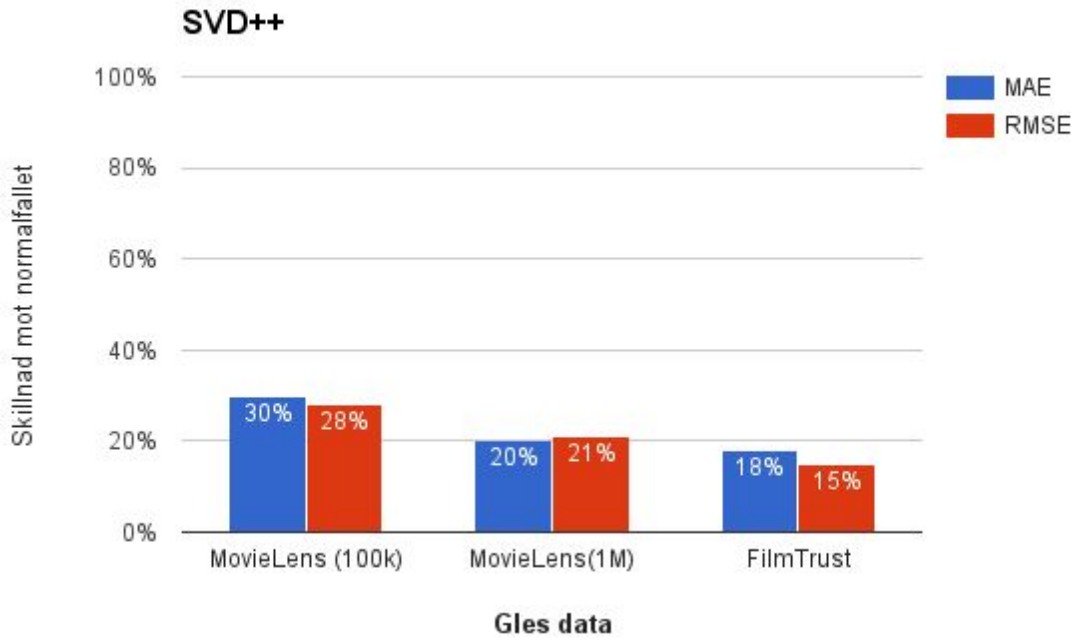

Diagram 3: Den procentuella skillnaden i MAE och RMSE mellan gles data *och* normalfallet *för SVD++ med respektive databas.*

Enligt resultaten påverkas *Slope One* mindre av *gles data,* än NMF och SVD++. Den minsta försämringen som ses i resultaten för NMF och SVD++, i detta testfall, är 15 procent i både MAE och RMSE (Diagram 2 och 3). I MAE är således det sämsta resultatet för *Slope One,* 5 procentenheter bättre än det bästa resultatet för NMF och SVD++. Motsvarande värde för MAE är 6 procentenheter. För alla tre algoritmer ger de olika databaserna olika stora fel. Det finns dock inget mönster i vilken databas som ger störst eller minst fel i detta testfall.

4.2 Ny produkt

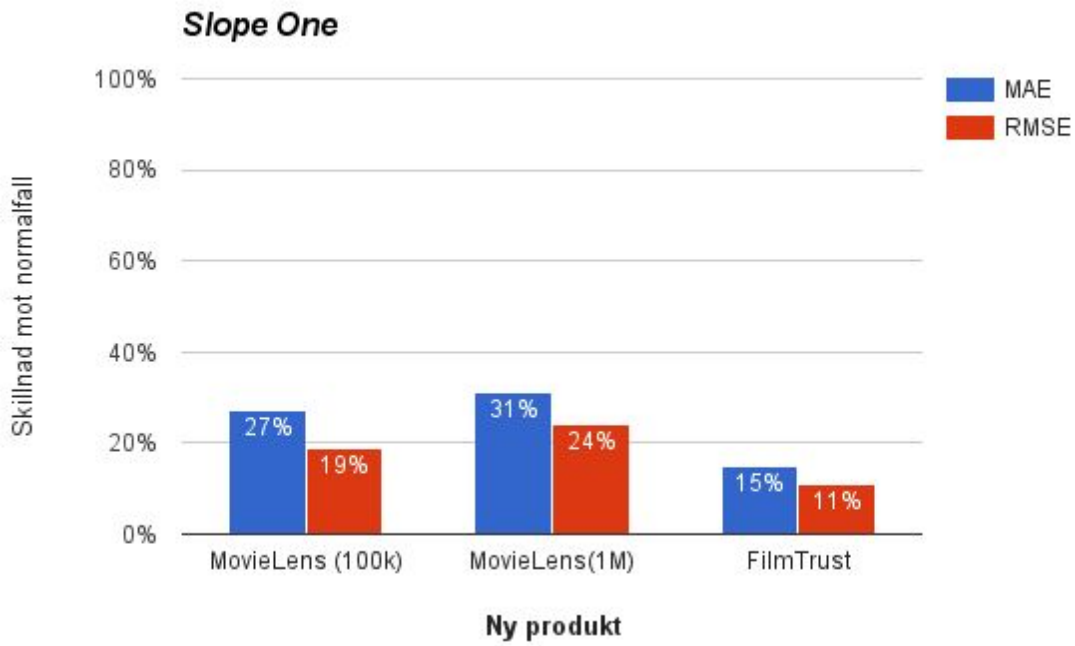

Diagram 4: Den procentuella skillnaden i MAE och RMSE mellan ny produkt *och* normalfallet *för* Slope One *med respektive databas.*

I *cold start*-situationen *ny produkt* får *Slope One* en betydande försämring av korrektheten jämfört med *normalfallet* (Diagram 4). Störst försämring sker med MovieLens (1M) där MAE är 31 procent och RMSE 24 procent. Bäst resultat fås med FilmTrust med 15 procent i MAE och 11 procent i RMSE. Det finns alltså även i detta testfall en stor variation i resultaten för de olika databaserna.

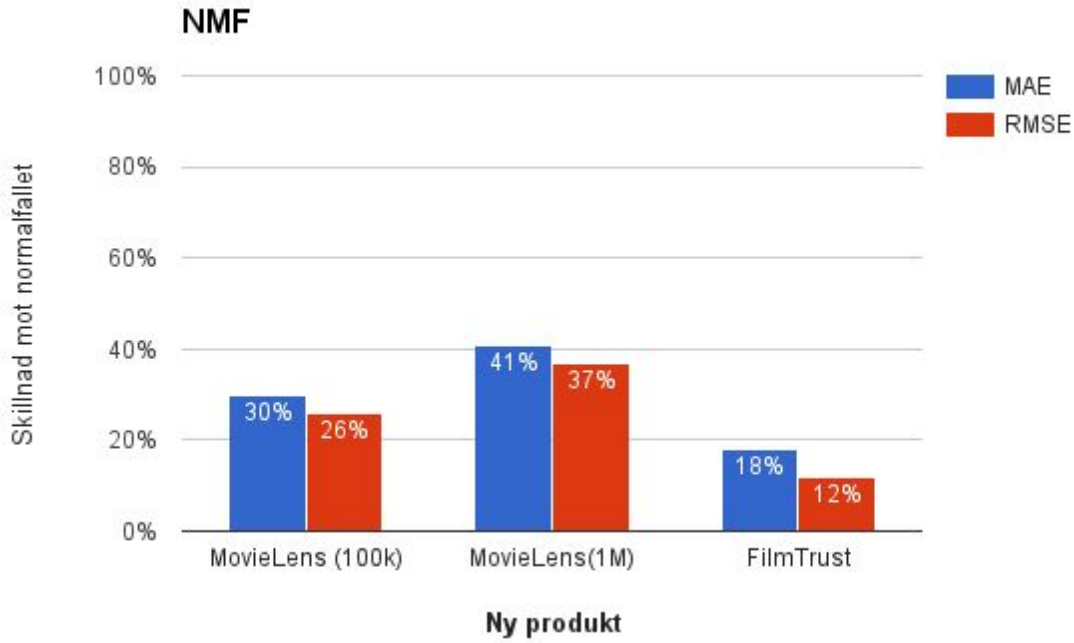

Diagram 5: Den procentuella skillnaden i MAE och RMSE mellan ny produkt *och* normalfallet *för NMF med respektive databas.*

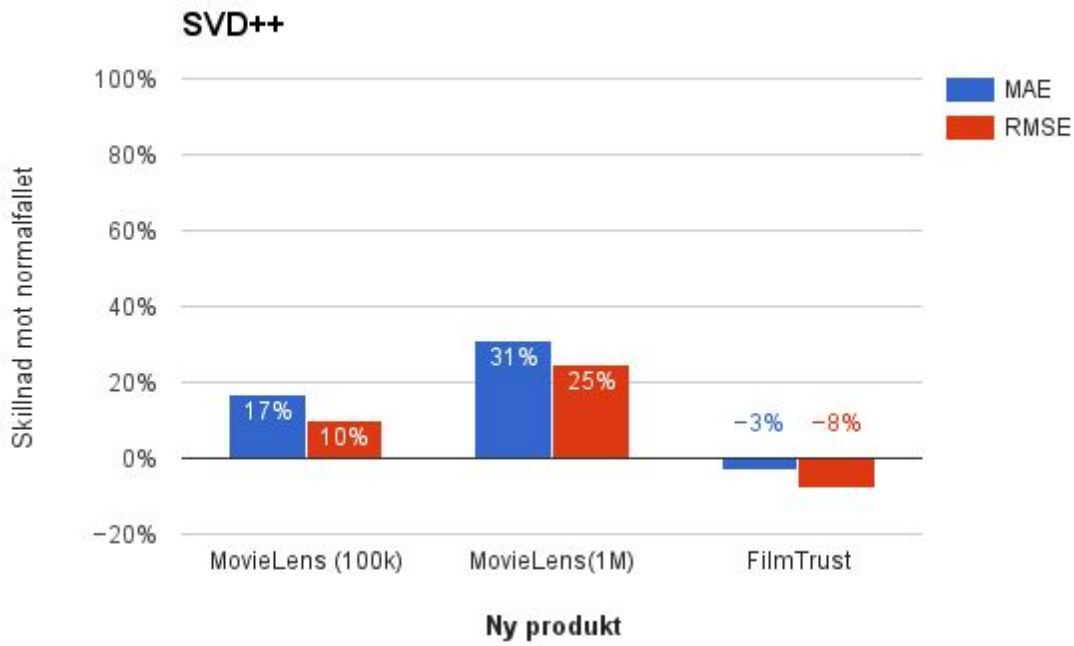

Diagram 6: Den procentuella skillnaden i MAE och RMSE mellan ny produkt *och* normalfallet *för SVD++ med respektive databas.*

Slope One påverkas genomgående något mindre av situationen *ny produkt* än NMF (Diagram 5), medan motsvarande resultat för SVD++ (Diagram 6) är för spridda för att kunna jämföras med *Slope Ones*.

4.3 Ny användare (0 betyg)

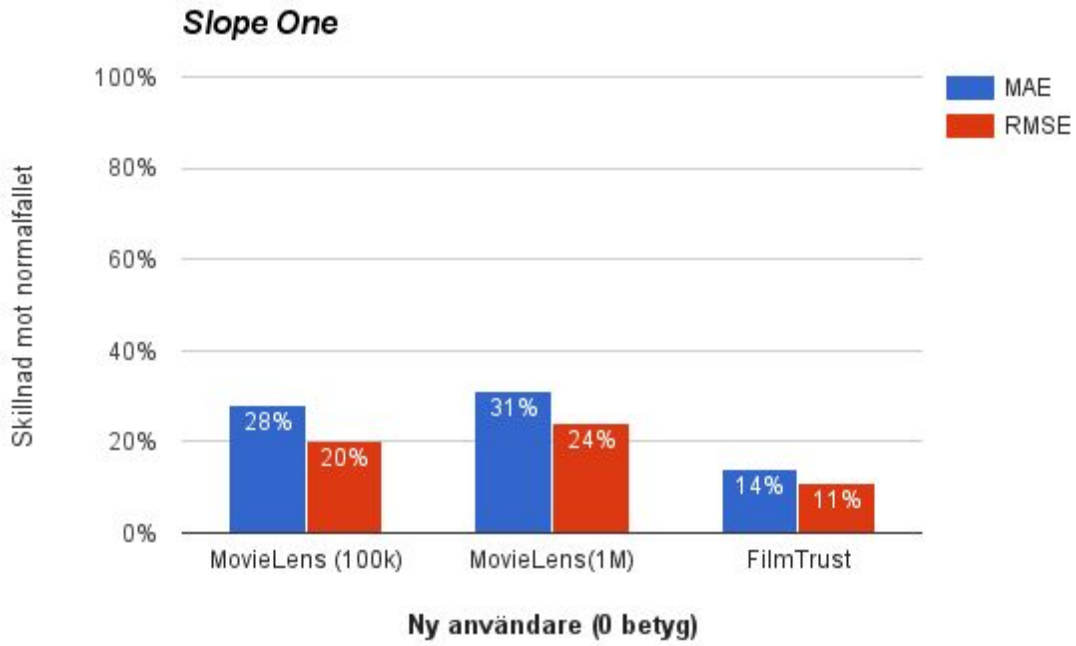

Diagram 7: Den procentuella skillnaden i MAE och RMSE mellan ny användare (0 betyg) *och* normalfallet *för* Slope One *med respektive databas.*

Testresultaten för *Slope One* i *cold start*-situationen *ny användare (0 betyg)* visar på en betydande försämring av korrektheten jämfört med normalfallet (Diagram 7). För både MAE och RMSE är försämringen minst för databasen FilmTrust med 14 respektive 11 procent. MovieLens (1M) ger störst försämring med 31 procent i MAE och 24 procent i RMSE.

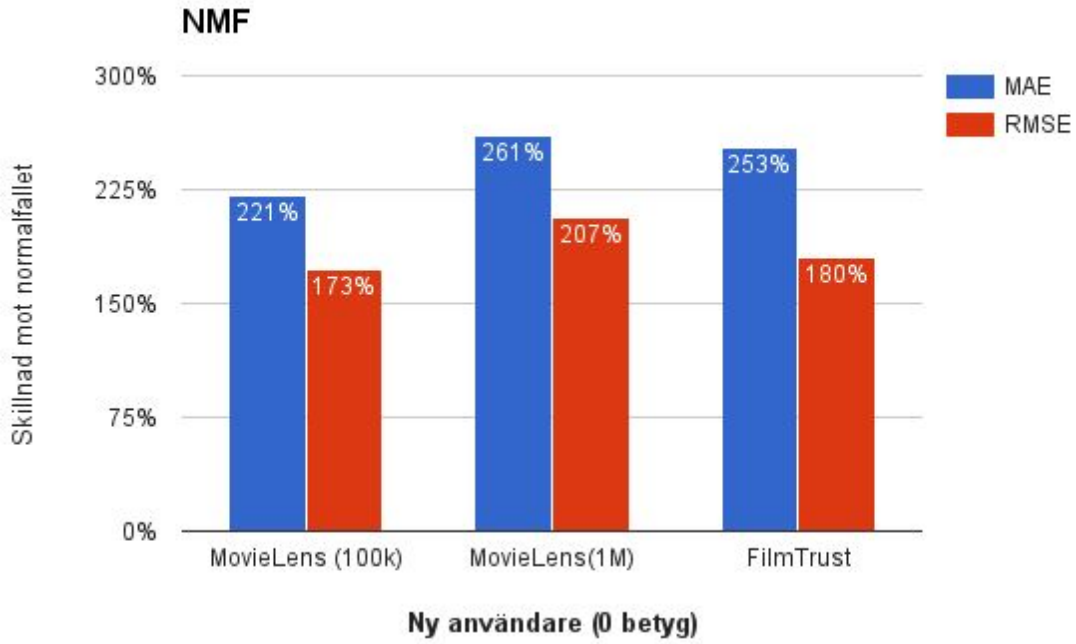

Diagram 8: Den procentuella skillnaden i MAE och RMSE mellan ny användare (0 betyg) *och* normalfallet *för NMF med respektive databas. Notera att skalan går upp till 300 procent.*

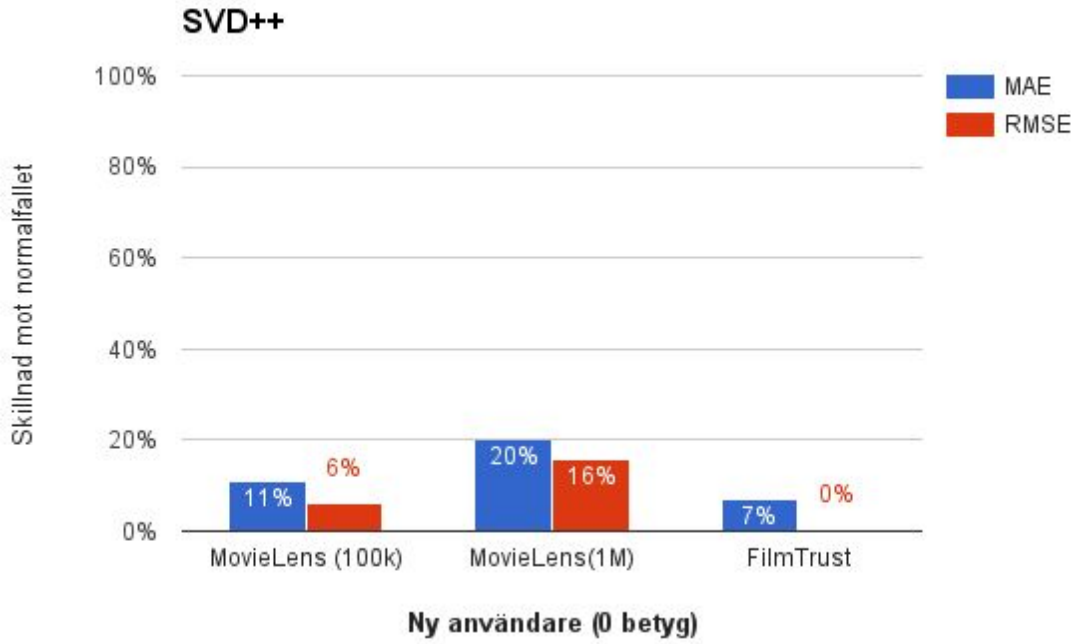

Diagram 9: Den procentuella skillnaden i MAE och RMSE mellan ny användare (0 betyg) *och* normalfallet *för SVD++ med respektive databas.*

I *cold start*-situationen *ny användare (0 betyg)* är försämringen för NMF väldigt stor i jämförelse med både *Slope One* och SVD++ (Diagram 7, 8 och 9). SVD++ är för detta testfall, på samtliga databaser, bättre än *Slope One* i att bibehålla korrekthet.

4.4 Ny användare (1 betyg)

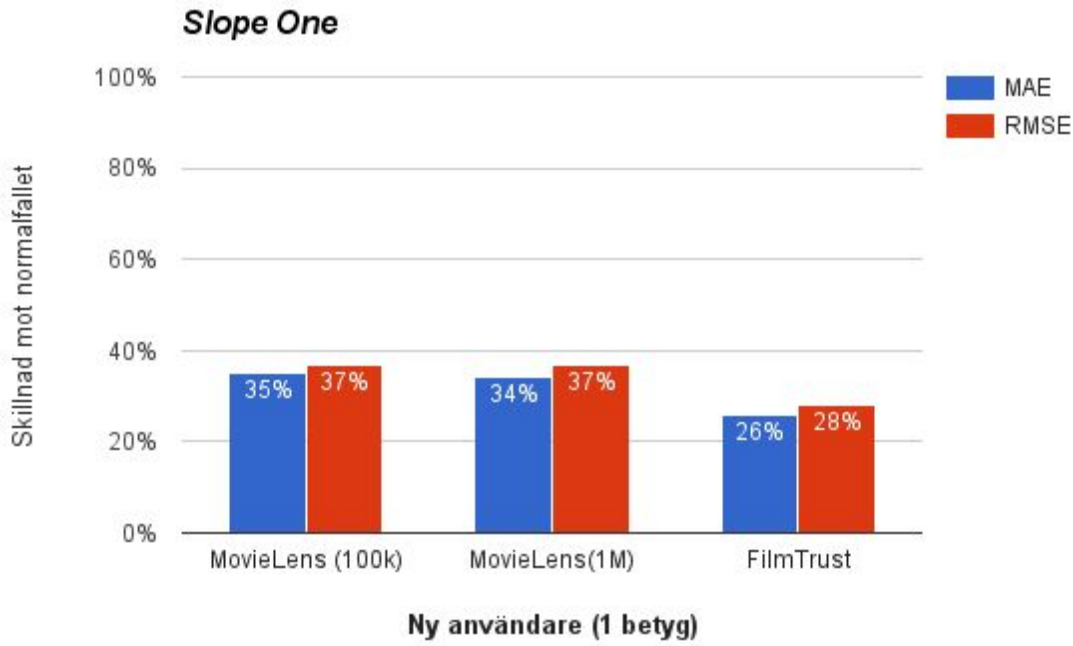

Diagram 10: Den procentuella skillnaden i MAE och RMSE mellan ny användare (1 betyg) *och* normalfallet *för* Slope One *med respektive databas.*

I *cold start*-situationen *ny användare (1 betyg)* får *Slope One* en tydlig försämring av korrektheten jämfört med *normalfallet*. Största försämringen ses hos MovieLens (100k) med 37 procent i RMSE och 35 procent i MAE. Även i det bästa resultatet är försämringen stor, med 26 respektive 28 procent i MAE och RMSE. Värt att notera är att *Slope One* enligt dessa resultat presterar sämre då vi behåller ett betygsvärde per ny användare (Diagram 10), än när de nya användarna helt saknar betyg (Diagram 7). Detta gäller för alla tre databaser.

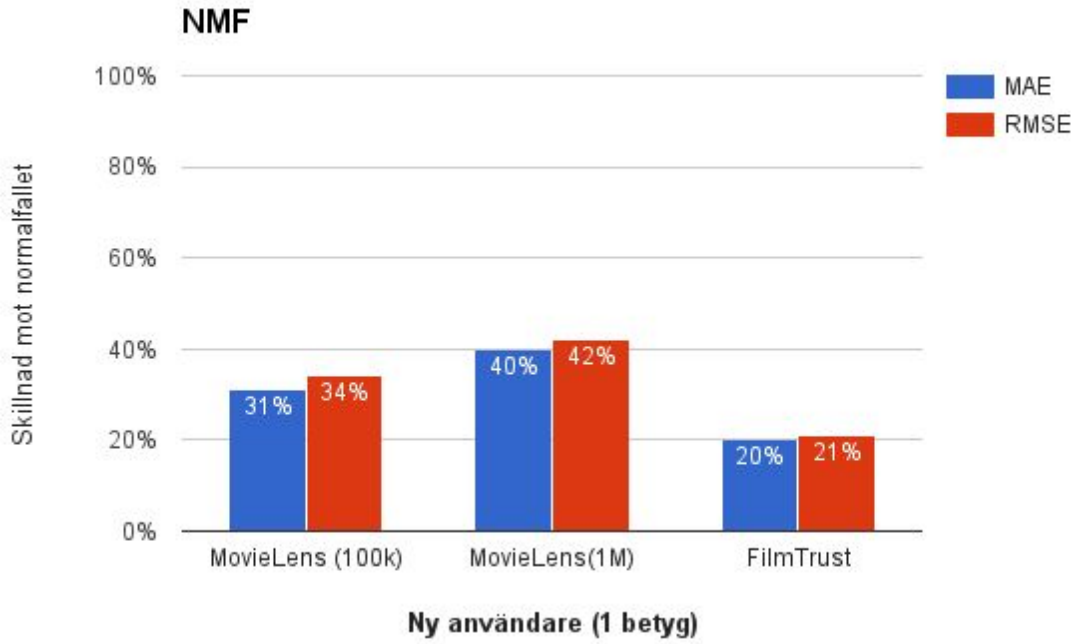

Diagram 11: Den procentuella skillnaden i MAE och RMSE mellan ny användare (1 betyg) *och* normalfallet *för NMF med respektive databas.*

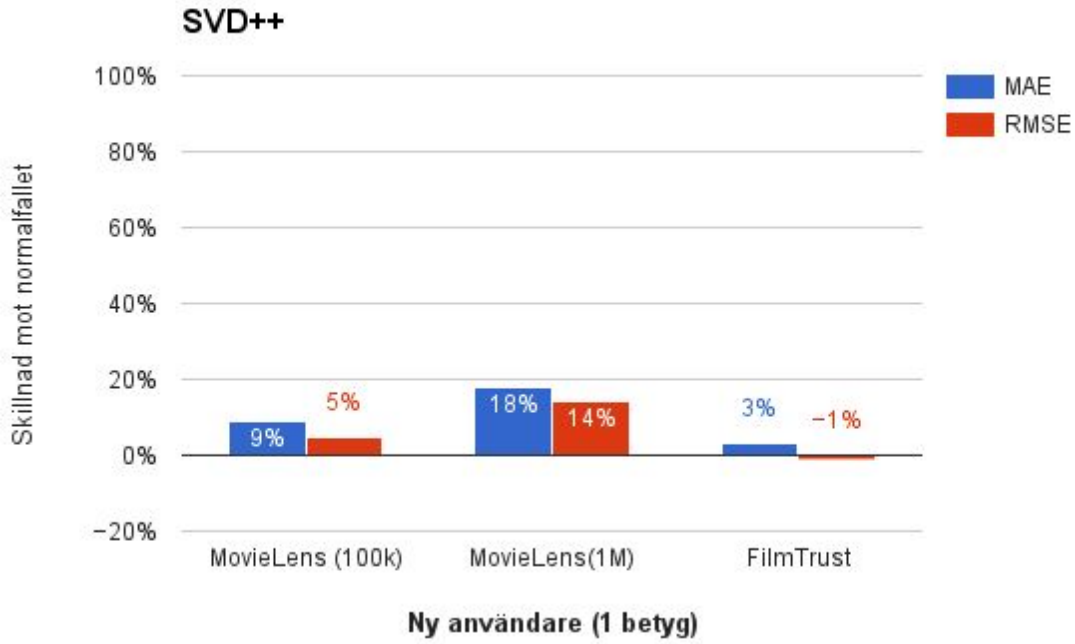

Diagram 12: Den procentuella skillnaden i MAE och RMSE mellan ny användare (1 betyg) *och* normalfallet *för SVD++ med respektive databas.*

Vid jämförelse av de två fallen *ny användare (0 betyg)* och *ny användare (1 betyg)* fås för NMF ett betydligt bättre resultat i det senare testfallet (Diagram 8 och 11). Korrektheten

försämras alltså mycket mindre för NMF, då ett betyg behålls för varje användare. För SVD++ ses en genomgående, om än liten, förbättring i resultaten i fallet *ny användare (1 betyg)* jämfört med *ny användare (0 betyg)* (Diagram 9 och 12). *Slope One* är således den enda av de tre algoritmerna, som inte påverkas positivt av att ett betyg behålls för nya användare.

5. Diskussion och slutsatser

I detta avsnitt diskuteras, inledningsvis, rapportens mest betydelsefulla resultat. Därefter följer metodkritik och till sist en sammanfattning av de viktigaste slutsatserna.

5.1 Diskussion

Slope One får i samtliga fall av begränsad data en försämring av korrektheten jämfört med *normalfallet*. Omfattningen av försämringen skiljer sig dock mellan fallen av begränsad data.

Försämringen av korrekthet i testfallet *gles data* kan för *Slope One* sägas vara ringa i jämförelse med motsvarande resultat för algoritmerna NMF och SVD++. Detta gäller även i förhållande till *Slope Ones* resultat för *cold start*. I enlighet med tidigare forskning förefaller *Slope One* således vara okänslig för *gles data*.

Slope Ones korrekthet påverkas däremot påtagligt av *cold start*. Graden av försämringen är näst intill identisk i de två testfallen *ny produkt* och *ny användare (0 betyg)*. Anledningen till detta är, troligtvis, att användar- och produktbetyg väger lika tungt i *Slope Ones* beräkning av betygsförutsägelser.

För *ny användare (1 betyg)* blir resultaten ytterligare lite sämre än för *ny användare (0 betyg)*. Detta är överraskande, då tillägg av betyg är en vanlig strategi i rekommendationssystem, för att förbättra korrektheten för nya användare. Korrektheten blir dessutom bättre för både NMF och SVD++ då ett betyg behålls för varje ny användare. Orsaken till att detta inte är en tillräcklig strategi för *Slope One* är, sannolikt, att ett enda betyg ger en mer felaktig bild av användarens smak än när inga betyg alls finns.

Från resultaten kan slutsatsen dras, att fokus vid eventuell utveckling av *Slope One* bör ligga på *cold start*. Trots att strategin att behålla ett betyg per ny användare inte är framgångsrik, är det möjligt att tillägg av fler betyg kan öka korrektheten i detta *cold start*-fall. Test av detta och ytterligare strategier lämnas till framtida forskning.

För samtliga algoritmer och testfall finns en stor variation mellan resultaten för de olika databaserna, vilket begränsar vilka slutsatser som kan dras. Då databaserna som används har olika storlek och gleshet, förefaller deras egenskaper vara en viktig faktor för korrektheten i ett rekommendationssystem. Det kan därför vara intressant att i framtida forskning göra en liknande studie, men istället använda databaser som är mer lika vad gäller storlek och gleshet. Det kan även vara av intresse att studera vid vilken grad av gleshet som korrektheten för rekommendationsalgoritmer försämras, det vill säga att definiera när en databas kan ses som gles.

Vad gäller jämförelser mellan de tre algoritmerna mäts i denna studie bara skillnaden mot det egna normalfallet, och inga slutsatser dras således om vilken av algoritmerna som ger bäst korrekthet på en viss datamängd eller i en viss situation.

5.2 Metodkritik

En begränsning i denna studie är att de implementationer av algoritmerna som används, är de som LibRec tillhandahåller. De slutsatser vi kan dra gäller således endast för just dessa implementationer. Ytterligare en begränsning var den tidsåtgång som krävdes för att träna algoritmerna SVD++ och NMF på stora databaser. Fler körningar hade kunnat ge mer representativa mätvärden, men detta var tvunget att vägas mot studiens begränsning i tid.

Då påverkan av egenskapen *gles data* på korrekthet i rekommendationssystem ämnats studeras, hade det kunnat vara mer lämpligt att köra testfallen på datamängder med samma gleshet. Då hade starkare slutsatser om effekten av *gles data* kunnat dras.

Att de enda resultat som diskuteras är hur mycket större felet blir mot normalfallet, kan ge en missvisande bild av algoritmernas egentliga korrekthet i situationer med begränsad data. Orsaken till detta är att korrektheten i normalfallet kan skilja sig mycket mellan de olika algoritmerna. På så sätt blir känsligheten ett sekundärt kriterium som, vid utvärdering av algoritmer, bör kombineras med den faktiska korrektheten i normalfallet.

5.3 Slutsatser

I enlighet med tidigare forskning, visar sig *Slope One* vara okänslig för *gles data*. När *Slope One* utsätts för *cold start*-situationerna *ny produkt* och *ny användare* blir korrektheten däremot betydligt försämrad. Vidare, är strategin att låta nya användare betygsätta en produkt innan användning, inte tillräcklig för att få en bättre korrekthet i situationen *ny användare.*

Skillnaden mellan resultaten för olika databaser och samma testfall, innebär att det är svårt att säga exakt hur mycket större felet blir i de olika situationerna av begränsad data. Däremot är det tydligt att databasernas egenskaper är en avgörande faktor i rekommendationssystems korrekthet.

6. Källor

[1]
Rajaraman A, Ullman JD. Mining of massive datasets. New York, USA: Cambridge University Press; 2012.

[2]
Lampropoulos AS, Tsihrintzis GA. Machine Learning Paradigms: Applications in Recommender Systems [Internet]. New York, USA: Springer International Publishing; 2015. [citerad 4 maj 2016]. Hämtad från: http://libris.kb.se/bib/18126693#more_info

[3]
Hofmann T. Collaborative filtering via gaussian probablistic latent semantic analysis. In: Proceedings of the 26th annual international ACM SIGIR conference: Research and development in informaion retrieval; 28 juli-1 augusti, 2003; Toronto, Canada. New York, USA: ACM; 2003. Hämtad från: ACM Digital Library Proceedings.

[4]
Schein AI, Popescul A, Ungar LH, Pennock DM. Methods and Metrics for Cold-Start Recommendations. In: Proceedings of the 25th annual international ACM SIGIR conference: Research and development in information retrieval; 11-15 augusti, 2002; Tampere, Finland. New York, USA: ACM; 2002. Hämtad från: ACM Digital Library Proceedings.

[5]
Robillard MP, Maalej W, Walker RJ, Zimmermann T. Recommendation Systems in Software Engineering [Internet]. Berlin, Tyskland: Springer Berlin Heidelberg; 2014. [citerad 14 april 2016]. Hämtad från: http://libris.kb.se/bib/16541809

[6]
Lemire D, Maclachlan A. Slope One Predictors for Online Rating-Based Collaborative Filtering. In: SIAM International Conference on Data Mining; 21-23 april, 2005; Newport Beach (CA), USA. New York, USA: Cornell University; 2005. Hämtad från: Cornell University Library.

[7]
MovieLens (100k) [Internet]. Minnesota: GroupLens, University of Minnesota; 1998-. [citerad 16 februari 2016]. Hämtad från: http://grouplens.org/datasets/movielens/

[8]
MovieLens (1M) [Internet]. Minnesota: GroupLens, University of Minnesota; 2003-. [citerad 16 februari 2016]. Hämtad från: http://grouplens.org/datasets/movielens/

[9]
FilmTrust [Internet]. Okänd utgivningsort; 2011-. [citerad 16 februari 2016]. Hämtad från: http://www.librec.net/datasets.html

[10]
LibRec. Librec: A Java library for Recommender Systems, ver. 1.3 [programvara]. Utgivningsort: Guibing Guo; 2016. [citerad 16 februari 2016]. Hämtad från: http://librec.net/

[11]
Park, S., Chu, W. Pairwise preference regression for cold-start recommendation. In: Proceedings of the third ACM conference: Recommender systems; 22-25 oktober 2009; New York (NY), USA. New York, USA: ACM; 2009. Hämtad från: ACM Digital Library.

[12]
Wikipedia [Internet]. St. Petersburg (FL): Wikimedia Foundation, Inc; 2004 - . Collaborative filtering; [uppdaterad 2 maj 2016; citerad 6 maj 2016]. Hämtad från: https://en.wikipedia.org/wiki/Collaborative_filtering

[13]
Carleton College Department of Computer Science. Recommender Systems [Internet]. Ottawa, Canada: Carleton College Department of Computer Science; 2007. [citerad 6 maj 2016]. Hämtad från: http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/about.html

[14]
Yang B, Lei Y, Liu D, Liu J. Social collaborative filtering by trust. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence: IJCAI '13; 3-9 augusti 2013; Beijing, China. New York, USA: ACM; 2013. Hämtad från: ACM Digital Library Proceedings.

[15]
Guo G, Zhang J, Yorke-Smith N. TrustSVD: Collaborative Filtering with Both the Explicit and Implicit Influence of User Trust and of Item Ratings. The AI Magazine. 2015; 36(2):sidor 90-101.

[16]
Pennock DM, Horvitz EJ, Lawrence S, Giles CL. Collaborative Filtering by Personality Diagnosis: A Hybrid Memory and Model-based Approach. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence; 30 juni-3 juli, 2000; Stanford, CA, USA. NY, USA: Cornell University; 2000. Hämtad från: Cornell University Library.

[17]
Koren, Y. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In: Proceedings of the 14th ACM SIGGDD International Conference: Knowledge Discovery and Data Mining; 24-27 augusti, 2008; Las Vegas (NV), USA; 2008. New York, USA: ACM; 2008. Hämtad från: ACM Digital Library.

[18]
Wikipedia [Internet]. St. Petersburg (FL): Wikimedia Foundation, Inc; 2007 - . Slope One; [uppdaterad 15 april 2015; citerad 5 maj 2016]. Hämtad från: https://en.wikipedia.org/wiki/Slope_One

[19]
Lee, D.D., Seung, S.H. Algorithms fro Non-Negative Matrix Factorization. In: Proceedings of Neural Information Processing Systems: Natural and Synthetic; 3-8 december 2001; Vancouver, Kanade. Pitsburg (PA), USA: CMU; 2001. Hämtad från: http://luthuli.cs.uiuc.edu/~daf/courses/optimization/papers/lee01algorithms.pdf

[20]
Zeng W, Zeng A, Shang M-S, Zhang Y-C. Information Filtering in Sparse Online Systems: Recommendation via Semi-Local Diffusion. Sánchez A, redaktör. *PLoS ONE*. 2013;8(11):e79354. doi:10.1371/journal.pone.0079354

[21]
Wikipedia [Internet]. St. Petersburg (FL), USA: Wikimedia Foundation, Inc; 24 september 2011 - . SVD++; [uppdaterad 30 maj 2012; citerad 5 maj 2016]. Hämtad från: http://www.recsyswiki.com/wiki/SVD%2B%2B

[22]
Amatriain, X. What's the Difference Between SVD and SVD++? [Internet]. Mountain View (CA), USA: Quora; 2015; [uppdaterad december 2015; citerad 5 maj 2016]. Hämtad från: https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++

[23]
Wikipedia [Internet]. St. Petersburg (FL): Wikimedia Foundation, Inc; 2006 - . Non-negative matrix factorization; [uppdaterad 25 april 2016; citerad 5 maj 2016]. Hämtad från: https://en.wikipedia.org/wiki/Non-negative_matrix_factorization

[24]
Zhang, S., Wang, W., Ford, J.C., Maheden, F. Learning from Incomplete Ratings Using Non-Negative Matrix Factorization. In: Proceedings of the sixth SIAM international conference: Data Mining; 20-22 april, 2006; Bethesda (MD), USA. Philadelphia (PA), USA: SIAM; 2006. Hämtad från: dplp.

[25]
Guo G. Resolving Data Sparsity and Cold Start in Recommender Systems. I: Masthoff J, Mobasher B, Desmarais MC, Nkambou R, redaktörer. Resolving Data Sparsity and Cold Start in Recommender Systems. Berlin: Springer Berlin Heidelberg; 2012. s. 361-364.

[26]
Zhang, M., Tang, J., Zhang, X., Xue, X. Addressing Cold Start in Recommender Systems: a Semi-Supervised Co-Training Algorithm. In: Proceedings of the 37th International ACM SIGIR Conference: Research & Development in Information Retrieval; 6-11 juli, 2014; Gold Coast, Australia. New York, USA: ACM; 2014. Hämtad från: ACM Digital Library.

[27]
Sahebi S, Cohen W. Community-Based Recommendations: a Solution to the Cold Start Problem. In: Proceedings of Workshop on Recommender Systems and the Social Web (RSWEB); 23 oktober, 2011; Chicago (IL), USA. Pittsburgh, USA: University of Pittsburgh; 2011. Hämtad från: http://d-scholarship.pitt.edu/13328/

[28]
Sharma M, Zhou J, Hu J, Karypis G. Feature-based factorized Bilinear Similarity Model for Cold-Start Top-n Item Recommendation. In: Proceedings of the 2015 SIAM International Conference on Data Mining; 30 april-2 maj, 2015; Vancouver, Canada. Philadelphia (PA), USA: SIAM; 2015. Hämtad från: SIAM.

[29]
Sohlberg H. Recommending new items to customers [Internet]. Stockholm: School of Computer Science and Communication (CSC), KTH; 2015. Kanditatuppsats. [citerad 18 april 2016]. Hämtad från:
http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A839780&dswid=1214

[30]
Li F., Lebanon G, Lee J, Sun M, Zha H, Zhou K. Learning Multiple-Question Decision Trees for Cold-Start Recommendation. In: Proceedings of the sixth ACM international conference on Web search and data mining; 4-8 februari, 2013; Rom, Italien. New York, USA: ACM; 2013. Hämtad från: ACM Digital Library.

[31]
Zhou K., Yang S., Zha H., Functional matrix factorization for cold-start recommendation. In: Proceedings of the 34th international ACM SIGIR conference: Research and Development in Information Retrieval; 24-28 juli 2011; Beijing, Kina. New York, USA: ACM; 2011. Hämtad från: ACM Digital Library.

[32]
Zhang YC, Séaghdha DÓ, Quecia D, Jambor T. Auralist: Introducing Serendipity into Music Recommendation. In: Proceedings of the fifth ACM international conference: Web search and data mining; 8-12 februari 2012; Seattle, Washington. New York, USA: ACM; 2012. Avaiable from: ACM Digital Library Proceedings.

[33]
Wikipedia [Internet]. St. Petersburg (FL), USA: Wikimedia Foundation, Inc; 11 december 2006 - . Mean absolute error; [uppdaterad 5 april 2016; citerad 4 maj 2016]. Hämtad från: https://en.wikipedia.org/wiki/Mean_absolute_error

[34]
Chai T, Draxler RR. Root mean square error (RMSE) or mean absolute error (MAE)?.Geoscientific Model Development Discussions. 2014; 7(1):sidor 1525-1534.

7. Bilagor

7.1 Mätvärden

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	0,7390	0,8076	0,9418	0,9484	0,9950
MovieLens (1M)	0,7112	0,7321	0,9337	0,9340	0,9515
Filmtrust	0,6314	0,6785	0,7244	0,7217	0,7939

Tabell 7.2.1: Resultat i MAE för *Slope One* avrundade till fyra decimaler

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	0,9397	1,0374	1,1217	1,1285	1,2879
MovieLens (1M)	0,9015	0,9323	1,1187	1,1161	1,2368
Filmtrust	0,8355	0,8955	0,9304	0,9243	1,0699

Tabell 7.2.2: Resultat för RMSE för *Slope One* avrundade till fyra decimaler

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	0,7903	0,9663	1,0262	2,5372	1,0386
MovieLens (1M)	0,7197	0,8259	1,0166	2,5978	1,0083
Filmtrust	0,7064	0,8836	0,8355	2,4900	0,8450

Tabell 7.2.3: Resultat för MAE för NMF avrundade till fyra decimaler

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	1,0152	1,2473	1,2813	2,7748	1,3556
MovieLens (1M)	0,9197	1,0664	1,2642	2,8256	1,3099
Filmtrust	0,9508	1,1786	1,0659	2,6581	1,1484

Tabell 7.2.4: Resultat för RMSE för NMF avrundade till fyra decimaler

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	0,7626	0,9895	0,8930	0,8473	0,8276
MovieLens (1M)	0,6734	0,8082	0,8808	0,8089	0,7912
Filmtrust	0,6793	0,8027	0,6582	0,7292	0,6966

Tabell 7.2.5: Resultat för MAE för SVD++ avrundade till fyra decimaler

	Normalfall	Gles matris	Ny produkt	Ny användare (0 betyg)	Ny användare (1 betyg)
MovieLens (100k)	0,9950	1,2760	1,0951	1,0504	1,0402
MovieLens (1M)	0,8668	1,0516	1,0826	1,0045	0,9906
Filmtrust	0,9142	1,0519	0,8450	0,9177	0,9014

Tabell 7.2.6: Resultat för RMSE för SVD++ avrundade till fyra decimaler

7.2 Kod

7.2.1 Metod getGivenNNewItems

Placeras i: librec/src/main/java/librec.data/DataSplitter.java

```
public SparseMatrix[] getGivenNNewItems(int ratio) throws
Exception {

            assert ratio > 0;

            int nrOfItems = rateMatrix.numColumns();
            int tr = (100-ratio)*nrOfItems/100;
            System.out.println(tr);
            System.out.println(nrOfItems);

            // Initierar två matriser
            SparseMatrix trainMatrix = new SparseMatrix(rateMatrix);
            SparseMatrix testMatrix = new SparseMatrix(rateMatrix);

            int[] itemIndices = Randoms.nextIntArray(tr, 1,
            nrOfItems);
            ArrayList<Integer> indices = new ArrayList<Integer>();
            for(int a = 0; a < itemIndices.length; a++){
                  indices.add(itemIndices[a]);
            }


            for(int item = 0; item < rateMatrix.numColumns();
            item++){
                  // Hämtar kolumnens rader som inte är tomma
                  List<Integer> users = rateMatrix.getRows(item);
                  // Antalet användare är då #betygsatta
                  int numRated = users.size();
                  if(indices.contains(item)){
                        for(int j = 0; j < numRated; j++){
                              trainMatrix.set(users.get(j), item,
                              0.0);
                        }
                  }else{
                        for(int j = 0; j < numRated; j++){
                              testMatrix.set(users.get(j), item, 0.0);
```

```
                    }
                }

            }

            // remove zero entries
            SparseMatrix.reshape(trainMatrix);
            SparseMatrix.reshape(testMatrix);

            debugInfo(trainMatrix, testMatrix, -1);

            return new SparseMatrix[] { trainMatrix, testMatrix };
        }
```

7.2.2 Ny metod getGivenNNewUsersZero

Placeras i: librec/src/main/java/librec.data/DataSplitter.java

```
public SparseMatrix[] getGivenNNewUsersZero(int ratio) throws
Exception {
            assert ratio > 0;


            int nrOfUsers = rateMatrix.numRows();
            int tr = (100-ratio)*nrOfUsers/100;
            //Double testUsers = (1-r)*nrOfUsers;
            System.out.println(tr);
            System.out.println(nrOfUsers);

            SparseMatrix trainMatrix = new SparseMatrix(rateMatrix);
            SparseMatrix testMatrix = new SparseMatrix(rateMatrix);

            // Randomises the specified number of user indices (ids)
            int[] userIndices = Randoms.nextIntArray(tr, 1,
            nrOfUsers);
            ArrayList<Integer> indices = new ArrayList<Integer>();
            for(int i = 0; i < userIndices.length; i++){
                indices.add(userIndices[i]);
            }

            Random rand = new Random();
            for(int u = 0; u < rateMatrix.numRows(); u++){
                List<Integer> items = rateMatrix.getColumns(u);
```

```
                int numRated = items.size();
                if(indices.contains(u)){
                      for(int i = 0; i < numRated; i++){
                            trainMatrix.set(u, items.get(i), 0.0);
                      }
                }else{
                      for(int i = 0; i < numRated; i++){
                            testMatrix.set(u, items.get(i), 0.0);
                      }
                }

          }

          // remove zero entries
          SparseMatrix.reshape(trainMatrix);
          SparseMatrix.reshape(testMatrix);

          debugInfo(trainMatrix, testMatrix, -1);

          return new SparseMatrix[] { trainMatrix, testMatrix };
     }
```

7.2.3 Ny metod getGivenNNewUsersOne

Placeras i: librec/src/main/java/librec.data/DataSplitter.java

```
public SparseMatrix[] getGivenNNewUsersOne(int ratio) throws
Exception {
          assert ratio > 0;


          int nrOfUsers = rateMatrix.numRows();
          int tr = (100-ratio)*nrOfUsers/100;
          //Double testUsers = (1-r)*nrOfUsers;
          System.out.println(tr);
          System.out.println(nrOfUsers);

          SparseMatrix trainMatrix = new SparseMatrix(rateMatrix);
          SparseMatrix testMatrix = new SparseMatrix(rateMatrix);

          // Randomises the specified number of user indices (ids)
          int[] userIndices = Randoms.nextIntArray(tr, 1,
          nrOfUsers);
```

```
        ArrayList<Integer> indices = new ArrayList<Integer>();
        for(int i = 0; i < userIndices.length; i++){
              indices.add(userIndices[i]);
        }

        Random rand = new Random();
        for(int u = 0; u < rateMatrix.numRows(); u++){
              List<Integer> items = rateMatrix.getColumns(u);
              int numRated = items.size();
              if(indices.contains(u)){
                    int j = rand.nextInt(numRated);
                    for(int i = 0; i < numRated; i++){
                          if(i!=j){
                                trainMatrix.set(u, items.get(i),
                                0.0);
                          }
                    }
              }else{
                    for(int i = 0; i < numRated; i++){
                          testMatrix.set(u, items.get(i), 0.0);
                    }
              }

        }

        // remove zero entries
        SparseMatrix.reshape(trainMatrix);
        SparseMatrix.reshape(testMatrix);

        debugInfo(trainMatrix, testMatrix, -1);

        return new SparseMatrix[] { trainMatrix, testMatrix };
  }
```