Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Igor Avbelj

Hibridizacija priporočilnih sistemov temelječih na skupinskem izbiranju

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00041/2012
Datum: 13.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **IGOR AVBELJ**

Naslov: **HIBRIDIZACIJA PRIPOROČILNIH SISTEMOV TEMELJEČIH NA SKUPINSKEM IZBIRANJU**

HYBRIDISATION OF RECOMMENDATION SYSTEMS BASED ON COLLABORATIVE FILTERING

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Naloga je implementirati izbrane metode priporočanja v priporočilnih sistemih in preučiti možnosti za kombiniranje njihovih priporočil v »hibridno« priporočilo. Med implementiranimi metodami naj bodo:
- priporočanje na osnovi podobnosti (npr. k-NN z evklidsko razdaljo)
- Matrična faktorizacija po metodi BRISMF
- Matrična faktorizacija po metodi BPRMF

Izvedeni naj bodo poskusi z več metodami hibridizacije, kot sta povprečje in linearna regresijska kombinacija, in zbirkami podatkov o filmih ITV in MovieLens.

Mentor:

akad. prof. dr. Ivan Bratko

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Igor Avbelj, z vpisno številko **63090006**, sem avtor diplomskega dela z naslovom:

Hibridizacija priporočilnih sistemov temelječih na skupinskem izbiranju

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 20.9.2012 Podpis avtorja:

Zahvaljujem se mentorju, laboratoriju za umetno inteligenco, staršem in Urši.

Kazalo

Povzetek

V diplomi je predstavljena uporaba različnih tehnik za priporočanje. Uporabimo metodo najbližjih sosedov in dve različni matrični faktorizaciji. Napovedi zgenerirane s temi metodami potem še združimo (hibridizacija).

Za metodo najbližjih sosedov uporabimo več mer podobnosti, *Evklidsko podobnost*, *kosinusno podobnost* in *Pearsonov korelacijski koeficient*. Prva izmed metod, ki temeljijo na matrični faktorizaciji, je *BRISMF*. Pri tej faktorizaciji minimiziramo *RMSE* (koren povprečne kvadratne napake). Druga matrična faktorizacija je *BPRMF*, pri kateri optimiziramo vrstni red napovedi (rang korelacijo)

Nato napovedi iz teh posameznih metod še združimo (hibridizacija) z uporabo povprečja in linearne regresije. Z linearno regresijo dobimo uteži za napovedi iz posameznih metod, ki jih nato uporabimo pri kombiniranju napovedi z uteženo vsoto. Nato poskušamo še ugotoviti, kdaj se hibridizacija splača.

Napovedi delamo za uporabnike v podatkovnih zbirkah *iTIVI* in *MovieLens*. Napovedujemo njihove ocene za filme. Preizkušamo različne mere podobnosti in različne parametre, da bi dobili najbolj točne rezultate. Na koncu preizkusimo še hibridizacijo napovedi zgeneriranih z različnimi metodami in ugotovimo, katere metode skupaj dajejo najbolj točne napovedi.

Ugotovimo, da se na majhnih podatkovnih zbirkah od posameznih metod najbolje odnese metoda *k najbližjih sosedov*, na velikih zbirkah pa je točnost napovedi najboljša z metodo *BRISMF*. Metoda *BPRMF* sama po sebi ne daje zelo dobrih rezultatov, zelo pa izboljša rang korelacijo napovedi, če jo

uporabimo za hibridizacijo. S hibridizacijo se rezultati še izboljšajo.

Abstract

In this thesis, we present various techniques for recommender systems. We implement the *k-Nearest Neighbours* method and two matrix factorizations. Recommendations generated with these methods are then combined (hybridization).

For the *k-Nearest Neighbours* method we implemented several similarity measures: Euclidean similarity, cosine similarity and the Pearson correlation coefficient. The first of the matrix factorization methods is BRISMF. It minimizes RMSE (root mean squared error). The second matrix factorization technique is BPRMF, which optimizes rank correlation.

Recommendations from these individual methods are combined using average and linear regression blending. Using linear regression, we get weights which are then used in weighted sums. In the end, we were trying to find out in which cases hybridization improves final results.

We generate recommendations for users in datasets *iTIVI* and *MovieLens*. Both datasets contain data about movies. We experiment with different similarity measures and parameters in order to get the most accurate recommendations. We use hybridization of individual methods, to see which methods together give the best results.

On small datasets the most accurate recommendations are generated using the *k-Nearest Neighbour* method. On big datasets *BRISMF* gives better results. *BPRMF* does not give very good results by itself, however when used in hybridization it improves Spearman's rank correlation coefficient significantly. Using hybridization techniques improves results on most datasets.

Seznam uporabljenih kratic

Tabela 1: Seznam kratic uporabljenih v diplomi

m10k	Podatkovna zbirka *movielens* z 10000 ocenami
m100k	Podatkovna zbirka *movielens* s 100000 ocenami
m1m	Podatkovna zbirka *movielens* z milijon ocenami
itivi	Podatkovna zbirka *iTIVI*
k-NN	Metoda k najbližjih sosedov
ISMF	Incremental Simultaneous Matrix Factorization
RISMF	Reguarized Incremental Simultaneous Matrix Factorization
BRISMF	Biased Regularized Incremental Simultaneous Matrix Factorization
BPRMF	Bayesian Personalized Ranking Matrix Factorization

Poglavje 1

Uvod

1.1 Problem, motivacija in pregled rezultatov

Področje priporočilnih sistemov se je predvsem s porastom števila spletnih trgovin začelo zelo hitro razvijati. Leta 2006 so pri *Netflixu* ugotovili, da bi se jim, če bi uspeli točnost napovedi svojega priporočilnega sistema izboljšati za 10%, število strank zelo povečalo. Svoje podatke o uporabnikih in filmih so dali v javnost in razpisali nagrado, milijon dolarjev za tistega, ki prvi izboljša njihov priporočilni sistem za več kot 10 odstotkov. Ta cilj je bil leta 2009 dosežen [2].

To tekmovanje je motiviralo veliko raziskovalcev in tako je nastalo veliko metod za priporočanje. Posamezne metode so sicer dobre, a se da še boljše rezultate doseči s kombiniranjem teh metod.

Med študijem je področje priporočilnih sistemov začelo zanimati tudi mene. Še posebej so mi zanimivi sistemi temelječi na skupinskem izbiranju (*collaborative filtering*). Takšni problemi so v praksi zelo pogosti, saj je podatke, ki jih za tovrstne sisteme potrebujemo enostavno pridobiti. Pri skupinskem izbiranju ponavadi govorimo o uporabnikih in stvareh. Stvar je lahko film, glasba, knjiga, prodajni artikel, skratka kar koli. Poznati moramo le ocene, ki so jih posameznim stvarem dali posamezni uporabniki. Podatke

lahko pridobimo tudi implicitno. Lahko namreč privzamemo, da je uporabniku stvar, ki jo je kupil, všeč.

Naloga obsega implementacijo treh metod za generiranje priporočil. To so:

- metoda k najbližjih sosedov,
- BRISMF,
- BPRMF.

Nato bomo napovedi združili (hibridizacija). Tu bomo uporabili dve metodi, povprečje ter uteženo vsoto, kjer bomo uteži pridobili z linearno regresijo. Nato bomo poskusili ugotoviti, kdaj se hibridizacija najbolj splača. Točnost napovedi bomo preverili z merama RMSE ter Spearmanovo korelacijo ranga.

V tej diplomi bomo vsa testiranja izvajali na podatkih projekta *movielens* in na podatkih *iTIVI*. Pri obeh zbirkah podatkov so stvari filmi. Podatkovna zbirka *movielens* [1] vsebuje tudi dodatne podatke, na primer podatke o žanrih, ki pa jih pri pripravi priporočilnega sistema ne bomo uporabljali. Podatkovna zbirka *iTIVI* vsebuje samo podatke o ocenah, ki so jih filmom dali posamezni uporabniki.

Priporočilni sistem je možno relativno enostavno uporabiti tudi na drugih podatkih. Podatke je potrebno spraviti v obliko, ki jo sistem zna prebrati in ponovno pognati metode za optimizacijo parametrov. Skoraj vse metode za generiranje napovedi imajo neke parametre, od katerih je odvisna točnost napovedovanja. Več parametrov kot jih metoda ima, bolj je prilagodljiva in boljše rezultate daje, seveda le če ne pride do pretirane prilagoditve učnim podatkom (*overfitting*). To je še posebej očitno če so bili učni podatki zajeti najprej, testni pa kasneje in so se preference uporabnikov med tem spreminjale. Glede na to, smo podatke za testiranje razdelil na dva načina. Najprej smo testne podatke naključno jemali iz množice vseh podatkov, nato pa smo kot testne podatke vzeli najnovejših nekaj podatkov. Tako smo lahko primerjali rezultate in opazen je vpliv časovnih sprememb na točnost napovedi.

Sam priporočilni sistem smo implementirali v programskem jeziku *Python 2.7* [1]. Python je lahko berljiv predmetno usmerjen visoko nivojski jezik. Glavna slabost Pythona je hitrost izvajanja. Koda se med izvajanjem prevede v kodo za navidezni stroj, zaradi česar je Python hitrejši od jezikov, ki kodo direktno tolmačijo, ne more pa se kosati z jeziki ki se prevedejo direktno v strojno kodo. Vendar pa se da zahtevnejše preračunavanje izvesti v Cju, kar izvajanje zelo pohitri. Tako združimo enostavnost in berljivost Pythona in hitrost Cja. S samo uporabo Cja se nam v bistvu niti ni treba veliko ubadati, saj je veliko modulov napisanih v Cju in program pohitrimo že s tem, da jih uporabljamo. Eden izmed takih modulov (ki smo ga veliko uporabljali) je *NumPy* [2].

Ugotovili smo, da se na danih podatkih za metodo *k najbližjih sosedov* najbolje obnese kosinusna podobnost med uporabniki. Napovedi na majhnih zbirkah podatkov so s slednjo bolj natančne od napovedi zgeneriranih z BRISMFjem. BPRMF daje slabše rezultate kot BRISMF, tudi pri rang korelaciji, kar se nam zdi nenavadno, pričakovali smo namreč obratno. Vendar pa daje BPRMF dobre rezultate pri hibridizaciji. Vedno, kadar smo najboljšo rang korelacijo dosegli s hibridizacijo, so bile vanjo vključene tudi napovedi zgenerirane z metodo BPRMF. Iz tega lahko sklepamo, da BPRMF poda neko dodatno informacijo, ki pride do izraza šele ob hibridizaciji z ostalimi metodami.

1.2 Sorodno delo

V članku [6] je opisana metoda, kako združiti napovedi na podobnosti temelječih metod in BRISMF. Združevanje poteka sproti, med generiranjem napovedi. Mi smo najprej zgenerirali posamezne napovedi in jih nato združili. Avtorji članka [8] omenjajo, da njihov (takrat) najboljši priporočilni sistem za *Netflix* s hibridizacijo združuje več kot 100 posameznih sistemov, a ne ome-

[1] www.python.org
[2] numpy.scipy.org

njajo kako točno združujejo napovedi. V članku [9] najdemo nekaj različnih metod za hibridizacijo, med drugim tudi linearno regresijo.

Poglavje 2

Priprava podatkov

Za testiranje priporočilnega sistema smo uporabljali podatke projekta *movielens* in podatke *iTIVI*. Podatki so primerni za pristope temelječe na skupinskem izbiranju. Podatki projekta *movielens* sicer vsebujejo tudi dodatno znanje (npr. podatke o žanrih posameznih filmov), a jih v priporočilnem sistemu ne bomo uporabljali. *iTIVIjeva* podatkovna zbirka ne vsebuje podatkov o času, ko je bila ocena pridobljena.

2.1 MovieLens

Podatkovna zbirka je dostopna na [1]. Vsebuje podatke o ocenah, ki so jih dali uporabniki posameznim filmom. Poleg tega imamo na voljo tudi podatke o uporabnikih (spol, starost, izobrazba, poštna številka) in podatke o filmih (naslov, žanr). Te dodatne podatke smo zavrgli, saj metode, ki jih uporabljamo, temeljijo na skupinskem izbiranju in delujejo brez dodatnih podatkov. Natančnost napovedi bi lahko z upoštevanjem teh podatkov verjetno še izboljšali. Zelo enostavno bi bilo npr. pridobiti oceno uporabnika za žanr in nato priporočilo za film korigirati z ocenjeno všečnostjo žanra.

Podatki so, tako kot je praksa na tem področju, na voljo v CSV datoteki. Prvo polje predstavlja ID uporabnika, sledijo še ID filma, ocena ter časovni žig (v sekundah od 1.1.1970). Ponujajo podatkovne zbirke različnih velikosti.

Velikost je podana s številom ocen, se pa poleg tega spreminja tudi število uporabnikov in filmov, ki so zajeti v zbirki določene velikosti.

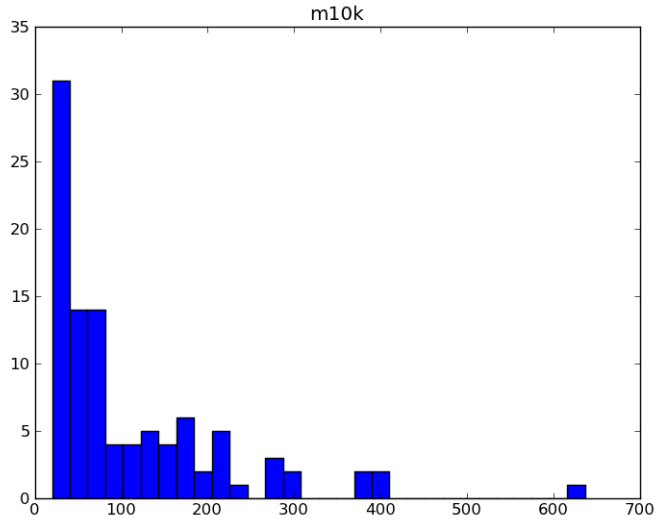
Najmanjša zbirka (*dataset*) obsega sto tisoč ocen. Večja jih ima en milijon, največja zbirka pa obsega deset milijonov ocen. Podrobnejše ocene so v tabeli 2.1. Poleg teh zbirk smo si iz najmanjše zgenerirali še eno manjšo, za sprotno hitro testiranje delovanja priporočilnega sistema. Posamezne zbirke smo glede na velikost poimenovali s kraticami, ki bodo uporabljane v diplomi in v programski kodi.

Tabela 2.1: Tabela velikost podatkovnih zbirk movielens

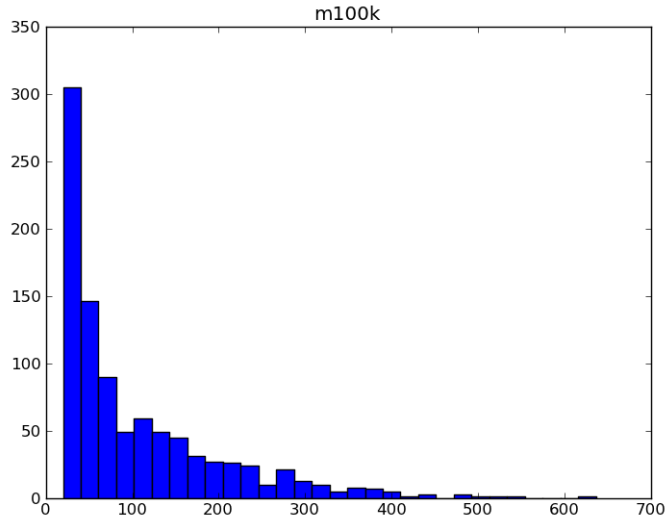
Ime	Število ocen	Število uporabnikov	Število filmov
m10k	11016	100	1238
m100k	100000	943	1682
m1m	1000000	6040	3706

Ker je za določitev nekaterih parametrov dobro poznati število ocen, ki so jih dali posamezni uporabniki smo za vsako podatkovno zbirko naredili tudi histograme s temi podatki (slike 2.1, 2.2, 2.3).

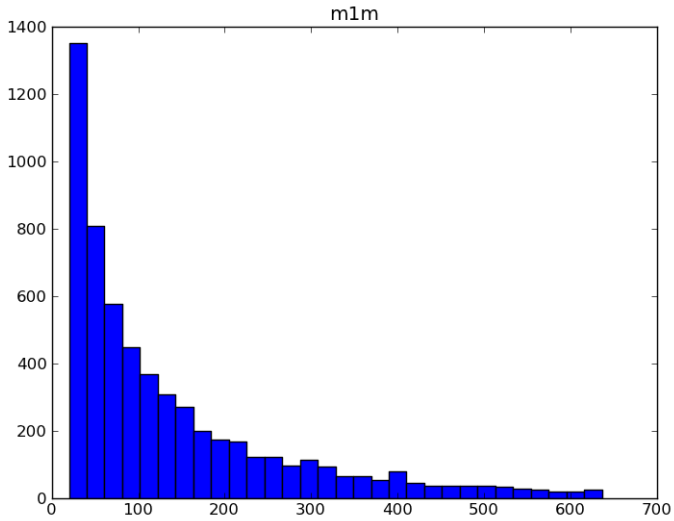
Slika 2.1: Histogram števila uporabnikov, ki so dali določeno število ocen v zbirki m10k



Slika 2.2: Histogram števila uporabnikov, ki so dali določeno število ocen v zbirki m100k



Slika 2.3: Histogram števila uporabnikov, ki so dali določeno število ocen v zbirki m1m

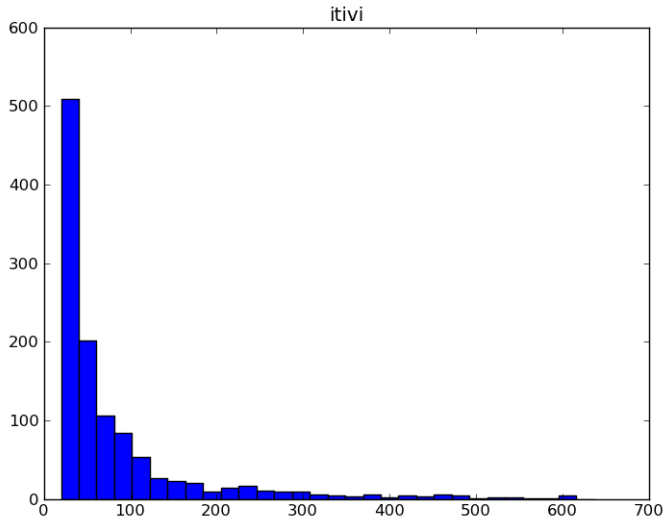


2.2 iTIVI

Podatke smo dobili v Laboratoriju za umetno inteligenco Fakultete za računalništvo in informatiko Univerze v Ljubljani. So v podatkovni bazi Firebird [1]. Podatke smo najprej prebrali in jih zapisali v CSV formatu, podobnem tistemu od podatkovnih zbirk movielens. *iTIVI* je bil spletni servis za izposojo filmov. Podatkov o tem, kdaj je bila ocena pridobljena ni. Vsebujejo 135131 ocen o 5760 filmih, ki jih je prispevalo 6975 uporabnikov.

Na sliki 2.4 je histogram števila uporabnikov, ki so dali določeno število ocen. Vidimo, da je velika večina uporabnikov prispevala manj kot 50 ocen.

Slika 2.4: Histogram števila uporabnikov, ki so dali določeno število ocen v zbirki iTIVI



2.3 Deljenje podatkov na učne in testne

Ker želimo vedeti kako dobro naš priporočilni sistem deluje, moramo imeti neko množico podatkov, na kateri bomo delovanje preverili. Če želimo pri

[1] http://www.firebirdsql.org

priporočanju uporabljati metode, ki za delovanje potrebujejo določene parametre, ki jih ne poznamo moramo imeti še neko množico podatkov, ki jo bomo uporabili za nastavitev teh parametrov. Če želimo dobiti napovedi s hibridizacijo, moramo spet imeti neko množico podatkov, na kateri se bomo naučili, kako združiti napovedi. V tem delu bomo opisali, kako smo podatke razdelili na potrebne učne in testno množico.

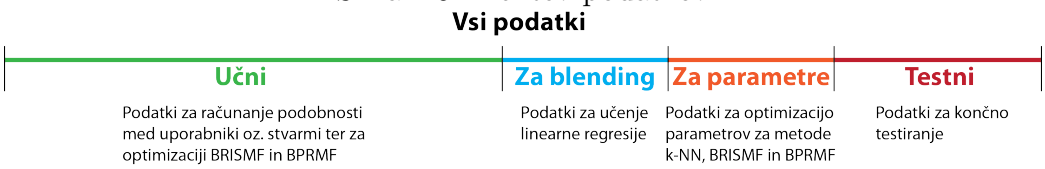
Po premisleku smo se odločili, da bomo podatke najprej razdelili na testne in učne. To smo naredili na dva načina, naključno in časovno urejeno. Tako smo lahko primerjali rezultate. V praksi je cilj doseči čim večjo točnost na podatkih, ki so bili pridobljeni za učnimi podatki.

Naključno delitev smo naredil tako, da smo vse podatke premešali, nakar smo 20% podatkov vzeli stran in jih shranili kot testne podatke za končno testiranje. Pri časovno urejeni delitvi smo podatke uredili glede na čas, ko je bila ocena pridobljena, nato pa smo vzeli najnovejših 20% podatkov in jih shranili kot testne podatke. Ti podatki so se uporabljali samo za končna testiranja, ne pa tudi za kakršno koli učenje. Tovrstne napake so sicer pri pripravi priporočilnih sistemov pogoste, v praksi pa privedejo do slabših rezultatov zaradi pretirane prilagojenosti testnim podatkom.

Iz preostalih podatkov smo nato spet vzeli 20% podatkov, ki smo jih uporabljali za optimizacijo parametrov. V primeru časovno urejene delitve so bili izbrani podatki pridobljeni takoj pred testnimi, sicer pa so bili naključno izbrani iz množice podatkov. S pomočjo teh podatkov, bomo optimizirali parametre metod BRISMF, BPRMF in *k-NN*.

Nato smo isto količino podatkov kot za optimizacijo parametrov še enkrat vzeli iz ostalih podatkov. Ti podatki so tudi pri časovno urejeni delitvi vzeti iz istega časovnega intervala kot učni. Te podatke smo uporabili za učenje hibridizacije (*linearna regresija*). Preostale podatke smo uporabili za generiranje samih napovedi oziroma učenje modelov, torej za računanje podobnosti oziroma obe matrični faktorizaciji. Delitev podatkov je grafično ponazorjena na sliki 2.3.

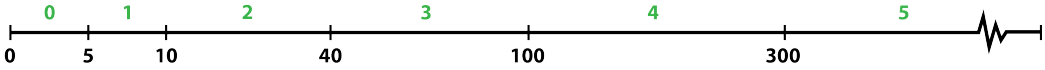
Slika 2.5: Delitev podatkov



2.4 Delitev uporabnikov za fokusirano evalvacijo

Uporabnike bomo glede na število ocen v učni množici razdelili v gruče. Nato bomo RMSE ter Spearmanovo rang korelacijo izračunali za vsako od teh gruč posebej. Tako bomo videli, v kateri gruči hibridizacija povzroči največje spremembe. Glede na števila ocen, ki jih je dal posamezen uporabnik smo gruče uporabnikov določili tako, kot je prikazano na sliki 2.6.

Slika 2.6: Gruče uporabnikov glede na število njihovih ocen



Poglavje 3

Pregled metod

3.1 Metoda k najbližjih sosedov

Metoda se v literaturi imenuje tudi *k-NN (k-Nearest Neighbours)*. Ideja metode je preprosta. Če želimo napovedati oceno uporabnika u za film i, poiščemo uporabniku u podobne uporabnike *(userbased)*. Nato pogledamo kako so oni ocenili film i in glede na njihovo podobnost po enačbi (3.1) napovemo kakšno oceno bi dal uporabnik u temu filmu [5].

Naj bo $\text{sim}(u1, u2)$ funkcija, ki vrne podobnost med uporabnikoma *u1* in *u2*. $r_{u,i}$ je ocena, ki jo je uporabnik u dal stvari i. $\overline{r_u}$ je povprečje ocen uporabnika u. U je množica k najbolj podobnih uporabnikov, ki so ocenili stvar i.

$$\text{pred}(u, i) = \overline{r_u} + \frac{\sum_{b \in U} \text{sim}\,(u, b) \cdot (r_{b,i} - \overline{r_b})}{\sum_{b \in U} \text{sim}\,(u, b)} \tag{3.1}$$

Zaradi izboljšanja rezultatov smo enačbo (3.1) malce spremenil, dodali smo ji še naivno ($\text{pred}_{\text{naive}}$) komponento, ki je povprečje povprečij uporabnika in stvari. S tem so se rezultati napovedovanja opazno izboljšali. Enačba (3.2) predstavlja spremenjen obrazec za napovedovanje. K je utež s katero utežimo naivno napoved.

$$\text{pred}(u, i) = \overline{r_u} + \frac{\sum_{b \in U} \text{sim}\,(u, b) \cdot (r_{b,i} - \overline{r_b}) + K \cdot \text{pred}_{\text{naive}}(u, i)}{\sum_{b \in U} \text{sim}\,(u, b) + K} \tag{3.2}$$

Napovedi pa smo generirali tudi na podlagi podobnosti med stvarmi *(itembased)*. To se pogosto izplača predvsem s stališča porabe pomnilnika (stvari je ponavadi manj kot uporabnikov). Tedaj smo uporabili enačbo (3.3). Množica I predstavlja stvari i najbolj podobne stvari. Moč množice je k.

$$\mathrm{pred}(u,i) = \overline{r_u} + \frac{\sum_{b \in I} \mathrm{sim}\,(i,b) \cdot (r_{u,b} - \overline{r_u}) + K \cdot \mathrm{pred}_{\mathrm{naive}}(u,i)}{\sum_{b \in I} \mathrm{sim}\,(i,b) + K} \tag{3.3}$$

Podobnost lahko izračunamo z različnimi merami podobnosti. Podobnosti smo vedno računali na prekrivni množici med dvema uporabnikoma oz. stvarema. Med dvema uporabnikoma poiščemo stvari, ki sta jih ocenila oba in iz njunih ocen izračunamo podobnost med njima. Isto velja če računamo podobnost med stvarmi, tedaj poiščemo uporabnike, ki so ocenili obe stvari in na podlagi teh ocen izračunamo podobnost.

3.1.1 Evklidska podobnost

Evklidska podobnost med dvema vektorjema je izpeljana iz evklidske razdalje. Razdaljo izračunamo po formuli (3.4).

$$\mathrm{dist}(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.4}$$

Kot vidimo iz enačbe (3.4) velja $\mathrm{dist}(x,y) = \mathrm{dist}(y,x)$.

Evklidsko podobnost izračunamo iz evklidske razdalje po enačbi (3.6). Takšna evklidska podobnost vrača vrednosti z intervala $[0,1]$ pri čemer 1 pomeni maksimalno podobnost, 0 pa pomeni da podobnosti ni. Vektorja x in y predstavljata presek množic ocen dveh uporabnikov oziroma presek množic uporabnikov dveh stvari. Ti podatki so bili pred računanjem podobnosti

normalizirani.

$$\text{sim}(x, y) = \frac{1}{1 + \text{dist}(x, y)} \tag{3.5}$$

$$\text{sim}(x, y) = \frac{1}{1 + \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}} \tag{3.6}$$

3.1.2 Kosinusna podobnost

Kosinusna podobnost med dvema vektorjema je definirana kot kosinus vmesnega kota. Vemo da se skalarni produkt med vektorjema lahko izračuna po enačbi (3.7).

$$x \cdot y = \|x\|\|y\| \cos\theta \tag{3.7}$$

Podobnost lahko torej izračunamo po enačbi (3.8).

$$\text{sim}(x, y) = \cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|} \tag{3.8}$$

$$\text{sim}(x, y) = \cos(\theta) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3.9}$$

Kosinusna podobnost vrača podobnosti na intervalu $[-1, 1]$. Ker delamo z različnimi metodami, želimo imeti vse podobnosti na območju $[0, 1]$. V priporočilnem sistemu smo torej kosinusno podobnost izračunali po enačbi (3.10). Kosinusno razdaljo smo vedno računali na normaliziranih podatkih.

$$\text{sim}(x, y) = \frac{\frac{x \cdot y}{\|x\|\|y\| + 0.1} + 1}{2} \tag{3.10}$$

3.1.3 Pearsonov korelacijski koeficient

Pearsonov korelacijski koeficient (3.11) meri linearno korelacijo med vektorjema x in y. Vrača vrednosti z intervala $[-1, 1]$ pri čemer 1 pomeni, da obstaja linearna enačba, ki opisuje odnos med vektorjema x in y. Če je korelacija 0, pomeni da med vektorjema x in y ni linearne povezanosti. Vrednost -1 kaže

na obratno korelacijo. Korelacijo smo računali na normaliziranih podatkih.

$$\mathrm{r}(x,y) = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})}\sqrt{\sum_{i=1}^{n}(y_i-\overline{y})}} \tag{3.11}$$

Tudi Pearsonov korelacijski koeficient smo naskalirali na območje $[0,1]$ po formuli (3.12).

$$\mathrm{sim}(x,y) = \frac{\mathrm{r}(x,y)+1}{2} \tag{3.12}$$

3.2 ISMF

ISMF (*Incremental Simultaneous Matrix Factorization*) je metoda za regeneriranje osnovne preferenčne matrike (matrike ocen). Podatke najprej normaliziramo (odštejemo jim povprečje uporabnika). Celotno učno množico imamo predstavljeno z (ponavadi) redko matriko R (preferenčna matrika) velikosti $u \times i$, pri čemer je u število uporabnikov, i pa število stvari. Matriko R lahko predstavimo kot dve manjši matriki z k latentnimi spremenljivkami. Ti dve matriki, P in Q, inicializiramo z naključnimi majhnimi števili. Nato se iterativno, z gradientno metodo približujemo pravim vrednostim. Opis metode je povzet po [6].

$$R_{ui} \approx P_{uk} \cdot Q_{ki} \tag{3.13}$$

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1i} \\ r_{21} & r_{22} & \cdots & r_{2i} \\ \vdots & \vdots & \ddots & \vdots \\ r_{u1} & r_{u2} & \cdots & r_{ui} \end{bmatrix} \approx \begin{bmatrix} p_{11} & \cdots & p_{1k} \\ p_{21} & \cdots & p_{2k} \\ \vdots & \ddots & \vdots \\ p_{u1} & \cdots & p_{uk} \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1i} \\ \vdots & \vdots & \ddots & \vdots \\ q_{k1} & q_{k2} & \cdots & q_{ki} \end{bmatrix}$$

Pri tej metodi izhajamo iz tega, da želimo optimizirati RMSE (koren povprečne kvadratne napake). Naj τ predstavlja vse trojke (u,i,r). Če je neka trojka v τ to pomeni, da v imamo v učni množici oceno uporabnika u za stvar i, to oceno označimo z r_{ui}. p_u predstavlja vrstico v matriki P, ki

ustreza uporabniku u. q_i predstavlja stolpec v matriki Q, ki ustreza stvari i.

$$\begin{aligned}
\hat{r}_{ui} &= \sum_{k=1}^{k} p_{uk} q_{ki} = p_u q_i \\
e_{ui} &= r_{ui} - \hat{r}_{ui} \text{ for } (u,i) \in \tau \\
e'_{ui} &= \frac{1}{2} e_{ui}^2 \\
\text{SSE} &= \sum_{(u,i)\in\tau} e_{ui}^2 \\
\text{SSE}' &= \frac{1}{2}\,\text{SSE} = \sum_{(u,i)\in\tau} e'_{ui} \\
\text{RMSE} &= \sqrt{\frac{\text{SSE}}{|\tau|}}
\end{aligned} \tag{3.14}$$

$$(P*, Q*) = \arg\min_{(P,Q)} \text{SSE}' = \arg\min_{(P,Q)} \text{SSE} = \arg\min_{(P,Q)} \text{RMSE} \tag{3.15}$$

Iz enačb (3.14) sledi, da če minimiziramo napako e_{ui} s tem minimiziramo tudi RMSE. Če pa želimo minimizirati e_{ui} moramo izračunati odvode.

$$\begin{aligned}
\frac{\partial}{\partial p_{uk}} e'_{ui} &= -e_{ui} \cdot q_{ki} \\
\frac{\partial}{\partial q_{uk}} e'_{ui} &= -e_{ui} \cdot p_{uk}
\end{aligned} \tag{3.16}$$

3.2.1 ISMF

Matriki P in Q bomo nato posodobili v obrati smeri od odvoda. Tako bomo popravili vrednosti v P in Q da bo kvadrat napake v vsaki iteraciji manjši. Glede na odvode (enačba (3.16)) bomo popravili vrednosti p_{uk} in q_{ki} po

enačbah (3.17).

$$p'_{uk} = p_{uk} + \eta \cdot e_{ui} \cdot q_{ki} \qquad (3.17)$$
$$q'_{ki} = q_{ki} + \eta \cdot e_{ui} \cdot p_{uk}$$

V enačbah (3.17) parameter η predstavlja hitrost učenja *(learning rate)*.

3.2.2 RISMF

Če matriki popravljamo po enačbah (3.17) hitro pride do pretiranega prilagajanja učnim podatkom. Zato uvedemo regularizacijski faktor. Enačbi po katerih popravljamo P in Q sta sedaj (3.18). Ta metoda se imenuje RISMF (*Regularized ISMF*).

$$p'_{uk} = p_{uk} + \eta \left(e_{ui} \cdot q_{ki} - \lambda \cdot p_{uk}\right) \qquad (3.18)$$
$$q'_{ki} = q_{ki} + \eta \left(e_{ui} \cdot p_{uk} - \lambda \cdot q_{ki}\right)$$

Omeniti moramo, da sedaj matematično gledano ne optimiziramo več RMSEja.

3.2.3 BRISMF

Nekateri uporabniki vse stvari ocenjujejo bolje ali slabše kot povprečni uporabnik. To lahko enostavno upoštevamo, tako da prvi stolpec matrike P fiksno nastavimo na vrednost 1, isto naredimo tudi z ne–prvo vrstico matrike Q. Takšna metoda se imenuje BRISMF (*Biased RISMF*). Ko sedaj računamo $\hat{r}_{ui}$ bo en faktor odvisen samo o preferenc uporabnika, drugi pa samo od filma.

Ta metoda daje od vseh *ISMF metod najboljše rezultate, zato smo v priporočilnem sistemu implementirali le to.

3.3 BPRMF

BPRMF pomeni *Bayesian Personalized Ranking Matrix Factorization*. Tudi ta metoda temelji na matrični faktorizaciji, od prejšnje pa se razlikuje v tem, da tu optimiziramo vrstni red napovedi, in ne RMSEja. Če uporabnik od nas želi napovedi za i_1, i_2 in i_3, mu želimo podati pravilni vrstni red všečnosti teh stvari. Ne želimo torej da je ocena, ki jo napovemo številsko pravilna, želimo da je pravilen vrstni red v katerem mu stvari priporočamo. Tak problem je v realnosti pogostejši, saj uporabniku v praksi priporočamo nek seznam stvari, ki mu bodo všeč, ne podamo mu seznama vseh stvari in napovedi njegovih ocen.

Metoda kot kriterij za optimizacijo predlaga BPR-Opt. Kriterij je zapisan v enačbi (3.19) in je odvedljiv. Njegov odvod predstavlja enačba (3.21). Množica D_S vsebuje vse trojke (u, i, j), za katere velja, da uporabnik u preferira i pred j. θ predstavlja parametre modela, $\hat{r}_{uij}(\theta)$ je ocena ali uporabnik u preferira i pred j. Avtorji članka [7] predlagajo da to oceno pridobimo po enačbi (3.20). $\hat{r}_{ui}$ je napoved ocene uporabnika u za stvar i. λ_θ predstavlja regularizacijske parametre modela.

$$\text{BPR-Opt} = \sum_{(u,i,j)\in D_S} \ln \sigma\left(\hat{r}_{uij}(\theta)\right) - \lambda_\theta \|\theta\|^2 \tag{3.19}$$

$$\sigma(x) := \frac{1}{1+e^{-x}}$$

$$\hat{r}_{(uij)}(\theta) = \hat{r}_{ui}(\theta) - \hat{r}_{uj}(\theta) \tag{3.20}$$

Ker metoda temelji na gradientni metodi, moramo tudi tu poznati odvod enačbe, ki jo optimiziramo (3.19). Odvod je zapisan v enačbi (3.21).

$$\frac{\partial\, \text{BPR-Opt}}{\partial\theta} = \sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{r}_{uij}}}{1+e^{-\hat{r}_{uij}}} \cdot \frac{\partial}{\partial\theta}\hat{r}_{uij} - \lambda_\theta\theta \tag{3.21}$$

Avtorji v članku [7] za optimizacijo modela predlagajo stohastični gradi-

entni spust (*stochastic gradient descent*). Ta pristop se od običajnega gradientnega spusta razlikuje v tem, da pri slednjem iteriramo preko celotne učne množice, pri stohastičnem gradientnem spustu pa v vsaki iteraciji iz vseh podatkov izberemo enega in glede na tega korigiramo parametre modela. V vsaki iteraciji torej izvedemo enačbo (3.22).

$$\theta \leftarrow \theta - \alpha \left(\frac{e^{-\hat{r}_{uij}}}{1 + e^{-\hat{r}_{uij}}} \cdot \frac{\partial}{\partial \theta} \hat{r}_{uij} + \lambda_\theta \theta \right) \tag{3.22}$$

Odvod $\frac{\partial}{\partial \theta} \hat{r}_{uij}$ je zapisan v enačbi (3.23). V enačbi P_u predstavlja vrstico iz matrike P, ki pripada uporabniku u, Q_i pa stolpec iz matrike Q, ki pripada stvari i.

$$\frac{\partial}{\partial \theta} \hat{r}_{uij} = \begin{cases} (q_i - q_j) & \text{če } \theta = p_u, \\ p_u & \text{če } \theta = q_i, \\ -p_u & \text{če } \theta = q_j, \\ 0 & \text{sicer} \end{cases} \tag{3.23}$$

V vsaki iteraciji torej v matriki P popravimo vrstico uporabnika u, v matriki Q pa stolpca stvari i in j.

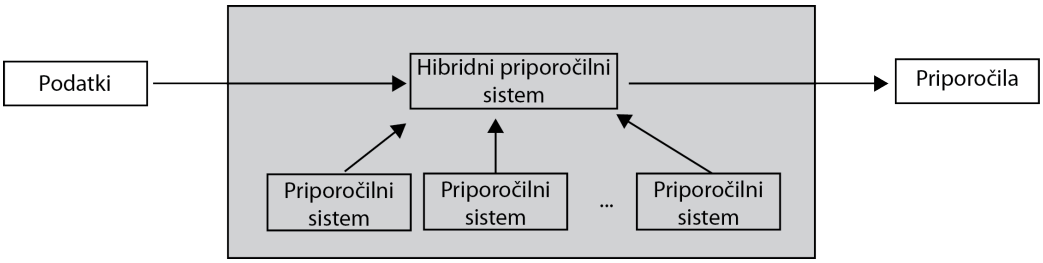
3.4 Hibridizacija

Hibridizacija priporočilnih sistemov pomeni združevanje različnih priporočilnih sistemov, tako da končni sistem daje boljše ocene od posameznih. Poznamo več načinov hibridizacije:

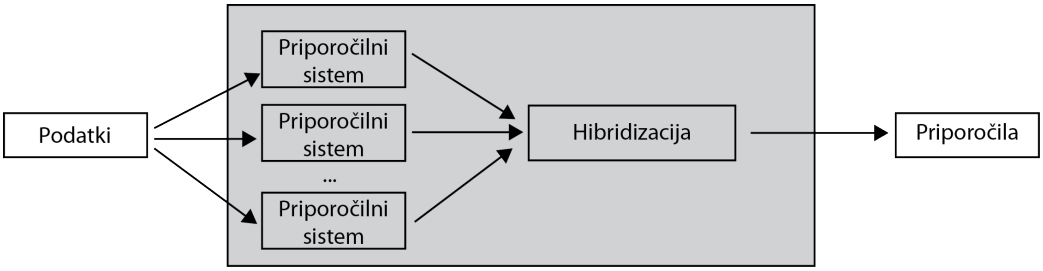
- monolitna,
- paralelna,
- cevovodna.

V tej diplomi smo uporabili paralelno metodo hibridizacije. To pomeni, da posamezna priporočila pridobimo posebej in jih nato združimo. Združili

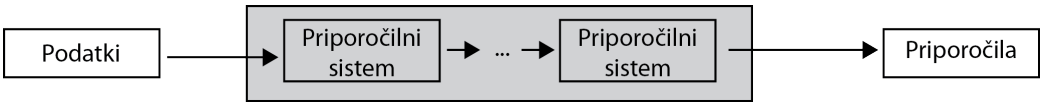
smo jih tako, da smo priporočila pridobljena s posameznimi metodami uteženo sešteli. Pri hibridizaciji s povprečenjem so bile vse uteži enake, pri hibridizaciji z linearno regresijo pa se sistem na učni množici za učenje hibridizacije nauči najboljše uteži.



Slika 3.1: Monolitni hibridni sistem



Slika 3.2: Paralelni hibridni sistem



Slika 3.3: Cevovodni hibridni sistem

3.5 Linearna regresija

Linearno regresijo smo uporabljali pri hibridizaciji napovedi. Z linearno regresijo na n točk postavimo premico ki se jih po metodi najmanjših kvadratov kar najbolje prilega. Z Linearno regresijo pridobimo uteži, s katerimi moramo utežiti posamezne napovedi, da bo na koncu napaka čim manjša. Za učenje teh uteži potrebujemo tudi posebno učno množico namenjeno učenju hibridizacije. Na tej učni množici izdelamo napovedi s posameznimi priporočilnimi sistemi. Te kot stolpce združimo v matriko R velikosti $n \times m$, pri čemer je n velikost učne množice za hibridizacijo, m pa število različnih napovedi, ki jih želimo združiti. Stolpčni vektor t predstavlja dejanske ocene na učni množici. Da dobimo uteži za posamezne napovedi rešimo sistem (3.24) po metodi najmanjših kvadratov.

$$Rx = t \tag{3.24}$$

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ r_{31} & r_{32} & \cdots & r_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix}$$

Sedaj ocene izračunamo po enačbi (3.25).

$$r_i = \sum_{j=1}^{m} x_j r_{nj} \tag{3.25}$$

3.5.1 Tikhonova regularizacija

Sama linearna regresija se podatkom pogosto preveč prilagodi. Če rešujemo problem $Ax = b$ z metodo najmanjših kvadratov, optimiziramo napako $\|Ax - b\|^2$. Tikhonova regularizacija deluje tako, da minimizira enačbo (3.26) [4].

$$\|Ax - b\|^2 + \|\Gamma x\|^2 \tag{3.26}$$

Γ je pogosto kar identiteta, torej $\Gamma = I$. Tedaj regularizacija preferira rešitve, s čim manjšo normo.

Z Tikhonovo metodo lahko problem rešimo po enačbi (3.27).

$$\hat{x} = \left(A^T A + \Gamma^T \Gamma\right)^{-1} A^T b \tag{3.27}$$

3.6 RMSE

RMSE (*root mean sqared error - koren povprečne kvadratne napake*) je mera natančnosti napovedi. Napovedi testiramo na testni množici. Za vsako kombinacijo uporabnika in stvari v testni množici izdelamo priporočilo, nakar izračunamo razliko med priporočilom ter dejansko oceno. Tako dobimo posamezne napake, ki jih kvadriramo, izračunamo njihovo povprečje ter nato korenimo. Kvadriranje poveča vpliv velikih napak in zmanjša vpliv majhnih.

V enačbi (3.28) (povzeto po [6]) je N število vseh testnih primerov, r_{ui} je ocena ki jo je dal uporabnik u stvari i. $\hat{r_{ui}}$ je napoved te ocene. τ predstavlja testno množico, na kateri računamo RMSE.

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{r_{ui} \in \tau} (r_{ui} - \hat{r_{ui}})} \tag{3.28}$$

3.7 Spearmanova rang korelacija

Spearmanova korelacija *(rang korelacija)* je mera korelacije med dvema slučajnima spremenljivkama. Ta korelacija oceni, kako dobro lahko odnos med obema spremenljivkama opišemo z monotono funkcijo. Rang korelacijo smo uporabili kot eno izmed mer natančnosti napovedi. Na testnih podatkih smo izračunali rang korelacijo napovedi za vsakega uporabnika in jo nato povprečili.

Spearmanovo korelacijo izračunamo po enačbi (3.29). Dejanske vrednosti X_i in Y_i naprej pretvorimo v njihove range (uredimo po velikosti in

oštevilčimo). Enačba je povzeta po [3].

$$\rho = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2 \sum_{i=1}^{n} (y_i - \overline{y})^2}} \tag{3.29}$$

Poglavje 4

Implementacija

Celoten priporočilni sistem je napisan v Pythonu. Razdeljen je na več modulov, glavni modul je `recommender.py`. V njem je razred `Recommender`, ki vsebuje vso kodo samega priporočilnega sistema. Nekaj kode za testiranje, optimizacijo, generiranje napovedi in razdeljevanje podatkov se nahaja v drugih modulih.

Vse dolgotrajnejše izračuna se izračuna enkrat, nakar se rezultate shrani na disk. Naslednjič se jih z diska prebere in uporabi. Slaba lastnost tega pristopa je predvsem pri metodi *k najbližjih sosedov* velika poraba pomnilnika (matrika podobnosti med uporabniki ali stvarmi). S tem pa se pridobi večjo hitrost generiranja samih napovedi, saj tedaj ni potrebno izračunavati podobnosti ampak imamo sezname najbolj podobnih uporabnikov oz. stvari že na voljo. Prav tako se za vsakega uporabnika oz. stvar podobne uporabnike oz. stvari išče le enkrat in ne vsakokrat ko generiramo napoved zanj.

Modul za delovanje potrebuje module: `bisect`, `bsddb`, `cPickle`, `collections`, `csv`, `itertools`, `marshal`, `math`, `numpy`, `os`, `random`, `scikits`, `scipy`, `sys`, `time`, `uuid`.

4.1 Pregled metod

Točen pregled metod je v dodatku B.

4.2 Primeri

V tem poglavju je predstavljenih nekaj izsekov kode, s katerimi lahko izvedemo najbolj tipične akcije.

4.2.1 Delitev podatkov

Kako smo podatke razdelil na učno in testno množico je opisano že v poglavju 2.

```
from split_data import split_data

split_data("m100m", train=0.8, train_params=0.2, \
    train_blending=0.2, random_seed = 123, timebased=True)
```

4.2.2 Računanje matrike podobnosti

Matriko podobnosti med uporabniki oziroma stvarmi se izračuna vnaprej. Nato se jo vedno samo prebere z diska. Tako se pohitri sama faza priporočanja. V tem primeru izračunamo in shranimo na disk kosinusne podobnosti med uporabniki in stvarmi za naključno razdeljene podatke podatkovne zbirke *iTIVI*.

```
import data
from recommender import Recommender

d = data.data("random")["itivi"]
r = Recommender(d["train"])

# kosinusna podobnost
r.item_similarity = r.sim_cosine_item
r.user_similarity = r.sim_cosine_user
```

```
r.fill_user_similarities_buffer()
r.dump_user_similarities_buffer(d["sims"]["cos"]["user"])
r.fill_item_similarities_buffer()
r.dump_item_similarities_buffer(d["sims"]["cos"]["item"])
```

4.2.3 Izdelava napovedi z metodo *k-NN*

V tem primeru predpostavimo da imamo že izračunano matriko podobnosti med uporabniki za podatkovno zbirko *m1m*, časovno razdeljeno. Matriko preberemo z diska, sestavimo seznam vsakemu uporabniku podobnih uporabnikov in izdelamo napovedi za testno množico. Uporabili bomo največ 100 najbolj podobnih uporabnikov, od teh morajo imeti podobnost vsaj $0,8$.

```
import data
from recommender import Recommender

d = data.data("timebased")["m1m"]
r = Recommender(d["train"])

r.fill_user_similarities_buffer(d["sims"]["cos"]["user"])
r.fill_similar_users_buffer()

rmse, spearman, recommendations = r.make_recommendations( \
      d["test"], r.recommend_sim_user, N=100, min_sim_margin=.8, \
      K=10)

# Napovedi so v recommendations v obliki
# [(uid, iid, rating, recommendation), ...]
```

4.2.4 Izdelava napovedi z metodo BRISMF

Napovedi naredimo za dataset *m100k*. Naprej naključno inicializiramo matriki P in Q, nakar ju optimiziramo z metodo BRISMF. Na koncu še naredimo napovedi na testni množici.

```
import data
from recommender import Recommender

d = data.data("random")["m100k"]
r = Recommender(d["train"])

r.fill_pq (K=300, m=0.001) # Nakljucna inicializacija
r.brismf(learn_rate = 0.01, lamb=0.001, max_iterations=50, \
    iteration_stop_history_length=5, part_train_data=0.1)
rmse, spearman, recommendations = \
    r.make_recommendations(d["test"], r.recommend_pq)
```

4.2.5 Izdelava napovedi z metodo BPRMF

Napovedi naredimo za časovno razdeljeno zbirko *m1m*. Privzamemo da imamo optimizirani matriki P in Q že shranjeni in ju samo preberemo z diska. Če ju tam še nebi imeli, bi ju morali najprej optimizirati podobno kot v primeru za metodo BRISMF.

```
import data
from recommender import Recommender

d = data.data("timebased")["m1m"]
r = Recommender(d["train"])

r.fill_pq (data=d["pq"]["bprmf"]) # PQ preberemo z diska
```

```
rmse, spearman, recommendations = \
    r.make_recommendations(d["test"], r.recommend_pq)
```

4.2.6 Hibridizacija napovedi

Najprej zgeneriramo napovedi na učni množici za hibridizacijo ter na testni množici. Nato napovedi na testni množici združimo.

```
import data
from recommender import Recommender

d = data.data("random")["m100k"]
r = Recommender (data=d["train"])

# Dummy
train1 = r.make_recommendations(d["train_blending"], \
    r.recommend_dummy)[-1]
test1 = r.make_recommendations(d["test"], \
    r.recommend_dummy)[-1]

# Average of user and item averages
train2 = r.make_recommendations(d["train_blending"], \
    r.recommend_avg_avg)[-1]
test2 = r.make_recommendations(d["test"], \
    r.recommend_avg_avg)[-1]

# BRISMF
r.fill_pq(data=d["pq"]["brismf"])
train3 = r.make_recommendations(d["train_blending"], \
    r.recommend_pq)[-1]
test3 = r.make_recommendations(d["test"], \
```

```
    r.recommend_pq)[-1]

# BPRMF
r.fill_pq(data=d["pq"]["bprmf"])
train4 = r.make_recommendations(d["train_blending"], \
    r.recommend_pq)[-1]
test4 = r.make_recommendations(d["test"], \
    r.recommend_pq)[-1]

train_recs = [train1, train2, train3, train4]
test_recs = [test1, test2, test3, test4]

print "LIN REG",
rmse, spearman, recommendations = r.blend(r.blend_lin_reg, \
    test_recs,train_recs, train_data = d["train_blending"])
```

4.2.7 Evalvacija po gručah uporabnikov

V primeru je privzeto, da uporabnik že ima na voljo izdelane napovedi `recommendations` in inicializiran priporočilni sistem `r`.

```
from divided_graph import evaluate_graph
import data

# r je instanca razreda Recommender
# recommendations so napovedi, ki smo jih zgenerirali

# Za izpis v konzolo
print "RMSEji po grucah: %s" % \
    r.eval_RMSE_divided(data.user_bins, recommendations)
print "Spearman po grucah: %s" % \
```

```
    r.eval_spearman_divided(data.user_bins, recommendations)
# Izrise graf
evaluate_graph (r, recs)
```

Poglavje 5

Rezultati

V tem poglavju bodo podani in komentirani doseženi rezultati na raznih testnih množicah.

5.1 Metoda k najbližjih sosedov

Rezultati so podani za optimizirane parametre. Optimizacija je potekala tako, da smo za `K`, `min_sim_margin` in `N` podali možne vrednosti nakar se je v vsaki iteraciji izbrala ena kombinacija vrednosti. Z izbranimi parametri se je nato naredilo napovedi. Potem se je parametre skupaj z RMSEjem shranilo. Po nekaj časa smo optimizacijo ustavili, kot najboljše parametre pa smo vzeli tiste, pri katerih je bil RMSE najmanjši. Rezultati testiranja so podani v tabelah 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 ter 5.7.

Na naključno razdeljenih podatkih smo najboljši RMSE vedno dosegli s kosinusno podobnostjo. Najboljšo rang korelacijo smo trikrat dosegli z kosinusno, enkrat pa z evklidsko podobnostjo. Pri časovno razdeljenih podatkih so bili najboljši RMSEji prav tako največkrat doseženi s kosinusno podobnostjo, tesno za njo je bil Pearsonov korelacijski koeficient. Najboljša rang korelacija je bila vedno dosežena z *userbased* metodo, z vsako od mer podobnosti enkrat. Glede na rezultate lahko trdimo, da na teh podatkih najboljše rezultate dosežemo z kosinusno razdaljo.

Tabela 5.1: Rezultati metode *k-NN* na naključno razdeljeni zbirki *m10k*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	14	0.000	89	1.013	**0.241**
evklidska itembased	44	0.111	1	1.012	0.233
kosinusna userbased	10	0.556	45	1.012	0.228
kosinusna itembased	48	0.778	237	**1.008**	0.220
pearson userbased	7	0.333	56	1.011	0.232
pearson itembased	5	0.778	119	1.014	0.211

Tabela 5.2: Rezultati metode *k-NN* na časovno razdeljeni zbirki *m10k*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	4	0.444	8	**1.119**	0.250
evklidska itembased	44	0.000	1	**1.119**	0.253
kosinusna userbased	6	0.333	68	**1.119**	0.243
kosinusna itembased	45	0.778	228	1.120	0.250
pearson userbased	9	0.000	53	**1.119**	0.247
pearson itembased	20	0.556	1024	1.120	**0.268**

Tabela 5.3: Rezultati metode *k-NN* na naključno razdeljeni zbirki *m100k*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	3	0.000	943	0.972	0.366
evklidska itembased	48	0.889	1	0.988	0.374
kosinusna userbased	10	0.556	524	**0.960**	**0.382**
kosinusna itembased	16	0.556	1042	0.975	0.379
pearson userbased	5	0.111	524	0.961	0.375
pearson itembased	48	0.222	348	0.988	0.380

Tabela 5.4: Rezultati metode *k-NN* na časovno razdeljeni zbirki *m100k*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	10	0.111	1	1.051	**0.347**
evklidska itembased	12	0.000	1	1.051	0.344
kosinusna userbased	22	0.556	209	**1.050**	0.346
kosinusna itembased	26	0.667	338	1.052	0.344
pearson userbased	18	0.111	278	**1.050**	0.344
pearson itembased	11	0.111	1	1.051	0.344

Tabela 5.5: Rezultati metode *k-NN* na naključno razdeljeni zbirki *m1m*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	12	0.000	6040	0.937	0.393
evklidska itembased	25	0.444	1	0.957	0.397
kosinusna userbased	7	0.667	5369	**0.918**	**0.411**
kosinusna itembased	14	0.222	803	0.922	0.407
pearson userbased	6	0.111	2014	0.920	0.408
pearson itembased	49	0.222	2006	0.957	0.405

Tabela 5.6: Rezultati metode *k-NN* na časovno razdeljeni zbirki *m1m*

Metoda	Parametri			RMSE	Spearman
	`K`	`min_sim_margin`	`N`		
evklidska userbased	17	0.000	4463	0.974	0.390
evklidska itembased	10	0.111	1	0.973	0.396
kosinusna userbased	25	0.556	1984	**0.968**	**0.400**
kosinusna itembased	11	0.000	791	0.969	0.397
pearson userbased	15	0.222	1984	**0.968**	0.399
pearson itembased	11	0.000	791	0.978	0.388

Tabela 5.7: Rezultati metode *k-NN* na naključno razdeljeni zbirki *iTIVI*

Metoda	Parametri			RMSE	Spearman
	K	min_sim_margin	N		
evklidska userbased	43	0.000	5408	1.003	0.277
evklidska itembased	22	0.667	1	1.006	0.285
kosinusna userbased	9	0.000	3605	**0.991**	**0.300**
kosinusna itembased	14	0.556	4190	0.997	0.280
pearson userbased	3	0.111	3004	0.996	0.285
pearson itembased	14	0.000	599	1.002	0.290

5.2 BRISMF

BRISMF v podatkih odkriva bolj globalne trende, zato je predvsem na večjih zbirkah podatkov boljši od na sosednosti temelječih metod. Tako kot pri *k-NN* metodah je jasno viden vpliv časovno razdeljenih podatkov v primerjavi z naključno razdeljenimi. Rezultati (RMSE in Spearman) so pri časovno razdeljenih podatkih vedno slabši kot pri naključno razdeljenih podatkih.

Vrednost parametrov, ki smo jih optimizirali so podani v tabelah ob rezultatih, vrednosti ostalih parametrov pa so bile takšne:

- `part_train_data = 0.1`,
- `iteration_stop_history_length = 5`,
- `max_iterations: 150`.

Tabela 5.8: Rezultati metode BRISMF na naključno razdeljenih zbirkah

Podatki	K	m	λ	η	RMSE	Spearman
m10k	500	0.009	0.010	0.018	1.016	0.241
m100k	900	0.003	0.010	0.003	0.937	0.399
m1m	5000	0.003	0.010	0.005	0.873	0.446
iTIVI	900	0.010	0.010	0.020	1.002	0.296

Tabela 5.9: Rezultati metode BRISMF na časovno razdeljenih zbirkah

Podatki	K	m	λ	η	RMSE	Spearman
m10k	100	0.006	0.009	0.007	1.121	0.220
m100k	100	0.009	0.007	0.014	1.045	0.348
m1m	1000	0.000	0.009	0.007	0.957	0.419

5.3 BPRMF

Glede na rezultate v tabelah 5.10 in 5.11 opažamo, da metoda ne daje boljših rezultatov od drugih metod. Predvsem se nam zdi zanimivo, da ni Spearmanova korelacija nič boljša kot pri ostalih metodah, še posebej glede na to da metoda optimizira rang in ne RMSEja. Pričakovali smo, da bo RMSE slabši, Spearmanova korelacija pa boljša kot pri ostalih metodah, a temu ni tako.

Parametri, ki smo jih skozi optimizacijo spreminjali so prikazani v tabelah, vrednosti ostalih parametrov pa so:

- `part_train_data = 0.1`,
- `iteration_stop_history_length = 3`,
- `test_iters: 5000`,
- `max_iterations: 10000000`.

Tabela 5.10: Rezultati metode BPRMF na naključno razdeljenih zbirkah

Podatki	K	m	λ	η	RMSE	Spearman
m10k	5000	0.008	0.001	0.005	1.040	0.231
m100k	100	0.010	0.003	0.016	0.996	0.331
m1m	1000	0.008	0.008	0.001	0.971	0.360
iTIVI	1500	0.004	0.000	0.003	1.025	0.251

Tabela 5.11: Rezultati metode BPRMF na časovno razdeljenih zbirkah

Podatki	K	m	λ	η	RMSE	Spearman
m10k	1500	0.001	0.000	0.018	1.123	0.267
m100k	500	0.010	0.008	0.001	1.054	0.329
m1m	500	0.007	0.004	0.005	0.997	0.372

5.4 Hibridizacija

V tabelah 5.12, 5.13, 5.14, 5.15, 5.16 so podani rezultati hibridizacije. Vse hibridizacije smo naredili s povprečenjem napovedi iz posameznih metod in z uporabo linearne regresije. Hibridizacijo smo zaradi časovne stiske naredili le na časovno ter naključno razdeljenih podatkovnih množicah *m10k* in *m1m*.

5.4.1 5 najboljših metod BRISMF

Rezultati hibridizacije napovedi zgeneriranih s 5 najboljšimi parametri z metodo BRISMF. Točni parametri so podani v dodatku A.1.1.

Tabela 5.12: Rezultati hibridizacije najboljših 5 napovedi BRISMF

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.008	0.247	1.089	0.210
m10k časovna delitev	1.120	0.232	1.130	0.241
m1m naključna delitev	0.870	0.449	0.870	0.449
m1m časovna delitev	0.956	0.422	0.960	0.421

5.4.2 5 najboljših metod BPRMF

Rezultati hibridizacije napovedi zgeneriranih s 5 najboljšimi parametri z metodo BPRMF. Točni parametri so podani v dodatku A.2.1.

Tabela 5.13: Rezultati hibridizacije 5 najboljših napovedi BPRMF

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.034	0.269	1.135	0.274
m10k časovna delitev	1.124	0.251	1.188	0.254
m1m naključna delitev	0.971	0.363	1.010	0.358
m1m časovna delitev	0.997	0.377	1.015	0.376

5.4.3 20 naključnih metod BRISMF

Rezultati hibridizacije napovedi zgeneriranih z 20 naključno določenimi množicami parametrov z metodo BRISMF. Točni parametri so podani v dodatku A.1.2.

Tabela 5.14: Rezultati hibridizacije 20 naključnih napovedi BRISMF

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.009	0.236	1.165	0.205
m10k časovna delitev	1.120	0.237	1.174	0.238
m1m naključna delitev	0.872	0.446	0.875	0.452
m1m časovna delitev	0.956	0.418	0.957	0.420

5.4.4 20 naključnih metod BPRMF

Rezultati hibridizacije napovedi zgeneriranih z 20 naključno določenimi množicami parametrov z metodo BPRMF. Točni parametri so podani v dodatku A.2.2.

Tabela 5.15: Rezultati hibridizacije 20 naključnih napovedi BPRMF

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.031	0.243	1.161	0.269
m10k časovna delitev	1.124	0.250	1.256	0.249
m1m naključna delitev	0.971	0.362	1.030	0.355
m1m časovna delitev	0.997	0.376	1.107	0.373

5.4.5 *k-NN*, BRISMF, BPRMF metode

Rezultati hibridizacije napovedi zgeneriranih s *k-NN*, BRISMF, BPRMF metodami. Za metodo *k najbližjih sosedov* smo uporabili kosinusno razdaljo med uporabniki z za vsako podatkovno zbirko najboljšimi parametri (glede na rezultate v razdelku 5.1). Poleg teh napovedi smo uporabili še najboljšo napoved zgenerirano z BRISMF ter najboljšo napoved zgenerirano z metodo BPRMF.

Tabela 5.16: Rezultati hibridizacije *k-NN*, BRISMF in BPRMF napovedi

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.008	0.274	1.029	0.277
m10k časovna delitev	1.120	0.248	1.122	0.239
m1m naključna delitev	0.900	0.426	0.881	0.446
m1m časovna delitev	0.964	0.414	0.955	0.420

5.4.6 Vse metode

Rezultati hibridizacije vseh 6 *k-NN* metod z najboljšimi parametri (vse mere podobnosti po uporabnikih in stvareh z najboljšimi parametri za določen *dataset*), najboljših 5 BRISMF ter BPRMF in naključnih 20 BRISMF ter BPRMF.

Tabela 5.17: Rezultati hibridizacije vseh najboljših *k-NN*, najboljših 5 BRISMF ter BPRMF, naključnih 20 BRISMF ter BPRMF

	Povprečenje		Linearna regresija	
Podatki	RMSE	Spearman	RMSE	Spearman
m10k naključna delitev	1.008	0.259	1.113	0.209
m10k časovna delitev	1.121	0.241	1.360	0.252
m1m naključna delitev	0.901	0.425	0.896	0.454
m1m časovna delitev	0.966	0.413	0.956	0.425

5.4.7 Pregled rezultatov

Hibridizacijo smo izvajali na časovno in naključno razdeljenih podatkih *m10k* ter *m1m*. Od posameznih metod je na manjšem *datasetu* najboljše rezultate dala metoda *k najbližjih sosedov*. Na večji podatkovni zbirki je dala najboljše rezultate metoda BRISMF. S *k-NN* metodo je bil dvakrat dosežen najboljši RMSE. Z metodo BRISMF smo dvakrat dosegli najboljši RMSE ter trikrat najboljšo rang korelacijo. Z metodo BPRMF smo enkrat dosegli najboljšo rang korelacijo.

Pri vseh podatkih, razen pri časovno razdeljeni zbirki *m10k* so bili najboljši rezultati doseženi z hibridizacijo. Pri slednji je bil najmanjši RMSE dosežen s *k-NN* metodo, največja rang korelacija pa z BPRMF. Glede na to da je bila večina najboljših rezultatov dosežena s hibridizacijo, lahko rečemo, da je hibridizacija izboljšala natančnost napovedi.

Napovedi z metodo BPRMF so same po sebi relativno slabe. Ko smo analizirali rezultate hibridizacije, pa smo ugotovili da je bila najboljša rang korelacija vedno dosežena pri hibridizaciji ostalih napovedi z napovedmi zgeneriranimi z metodo BPRMF, razen pri časovno razdeljeni zbirki *m10k*,

Prav tako moramo omeniti da so bili rezultati (RMSE in rang korelacija) hibridizacije 5 najboljših napovedi z metodo BRISMF vedno boljši kot rezultati zgenerirani samo z najboljšimi pramateri BRISMF.

Najmanjši RMSE je bil dvakrat dosežen s posameznimi metodami, šestkrat pa z eno od hibridizacij. Največja rang korelacija je bila enkrat dosežena z BPRMF metodo, trikrat pa z hibridizacijo.

Podrobnejši rezultati so v tabeli 5.18. Kot najboljša *k-NN* metoda je bila za časovno razdeljeno zbirko *m10k* izbrana kosinusna podobnost po stvareh, za naključno razdeljeno zbirko *m10k* je bila izbrana evklidska podobnost med stvarmi. Za obe delitvi podatkov *m1m* je bila izbrana kosinusna podobnost po uporabnikih.

Tabela 5.18: Končni rezultati hibridizacije

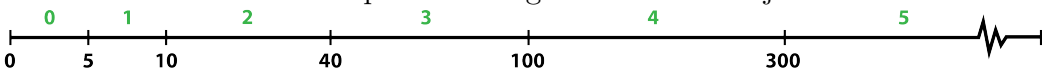
	m10k nak.		m10k čas.		m1m nak.		m1m čas.	
	RMSE	rang	RMSE	rang	RMSE	rang	RMSE	rang
Najboljša *k-NN*	**1.008**	0.220	**1.119**	0.253	0.918	0.411	0.968	0.400
BRISMF	1.016	0.241	1.121	0.220	0.873	0.446	0.957	0.419
BPRMF	1.040	0.231	1.123	**0.267**	0.971	0.360	0.997	0.372
POVP. najboljših 5 BRISMF	**1.008**	0.247	1.120	0.232	**0.870**	0.449	0.956	0.422
LIN. REG. najboljših 5 BRISMF	1.089	0.210	1.130	0.241	**0.870**	0.449	0.960	0.421
POVP. najboljših 5 BPRMF	1.034	0.269	1.124	0.251	0.971	0.363	0.997	0.377
LIN. REG. najboljših 5 BPRMF	1.135	0.274	1.188	0.254	1.010	0.358	1.015	0.376
POVP. naključnih 20 BRISMF	1.009	0.236	1.120	0.237	0.872	0.446	0.956	0.418
LIN. REG. naključnih 20 BRISMF	1.165	0.205	1.174	0.238	0.875	0.452	0.957	0.420
POVP. naključnih 20 BPRMF	1.031	0.243	1.124	0.250	0.971	0.362	0.997	0.376
LIN. REG. naključnih 20 BPRMF	1.161	0.269	1.256	0.249	1.030	0.355	1.107	0.373
POVP. najboljše *k-NN*, BRISMF, BPRMF	**1.008**	0.247	1.120	0.248	0.900	0.426	0.964	0.414
LIN. REG. najboljše *k-NN*, BRISMF, BPRMF	1.029	**0.277**	1.122	0.239	0.881	0.446	**0.955**	0.420
POVP. vse	**1.008**	0.259	1.121	0.241	0.901	0.425	0.966	0.413
LIN. REG. vse	1.113	0.209	1.360	0.252	0.896	**0.454**	0.956	**0.425**

5.5 Fokusirana evalvacija

Zanima nas, v katerih gručah uporabnikov hibridizacija najbolj izboljša rezultate. Za evalvacijo smo izbrali tiste zbirke podatkov, pri katerih smo najboljše rezultate dosegli z uporabo hibridizacije. To so naključno deljena *m10k* ter na oba načina razdeljena *m1m*.

Velikost kroga predstavlja število uporabnikov v trenutni gruči (logaritemska skala). Z namenom lažjega branja naslednjih grafov še enkrat prilagam grafični prikaz razdelitve uporabnikov po gručah (slika 5.1).

Slika 5.1: Gruče uporabnikov glede na število njihovih ocen

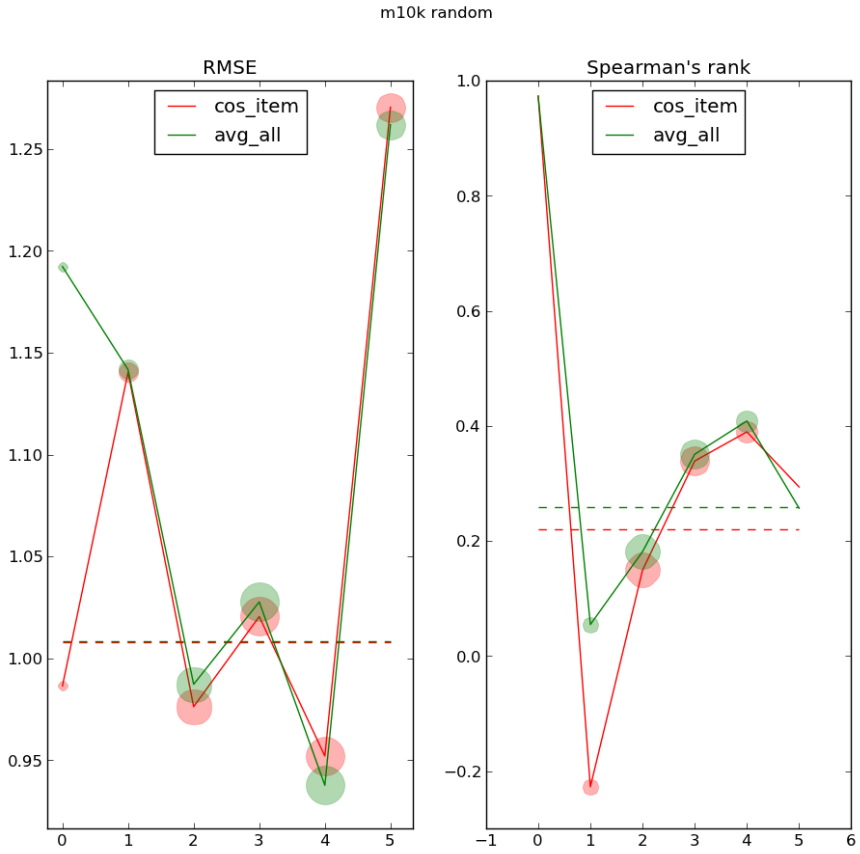


Ugotovili smo, da so nekatere metode boljše pri majhnih gručah, druge pa pri velikih. Metode, pri katerih je doseženi RMSE slabši pogosto dajo dobre rezultate pri rang korelaciji.

5.5.1 Naključno razdeljena množica *m10k*

Graf gibanja RMSE ter rang korelacije za naključno razdeljeno množico *m10k* je prikazan na sliki 5.2. Primerjamo *k-NN* s kosinusno razdaljo po stvareh in hibridizacijo s povprečenjem vseh metod (vse *k-NN*, najboljših 5 BRISMF ter BPRMF in naključnih 20 BRISMF ter BPRMF). Opazimo, da je RMSE obeh metod približno enak. V gručah uporabnikov z manj kot 100 ocenami je boljši RMSE dosegla *k-NN* metoda, v preostalih dveh gručah prevladuje hibridna metoda. Obratno je pri rang korelaciji. Tu je skoraj v vseh gručah, razen v zadnji boljša hibridna metoda. Slabši rezultat v tej gruči na končno rang korelacijo skoraj ne vpliva, saj je število uporabnikov zelo majhno. Še posebno velika razlika v korist hibridni metodi je v gruči 1.

Slika 5.2: Fokusirana evalvacija naključno razdeljene množice *m10k*



5.5.2 Naključno razdeljena množica *m1m*

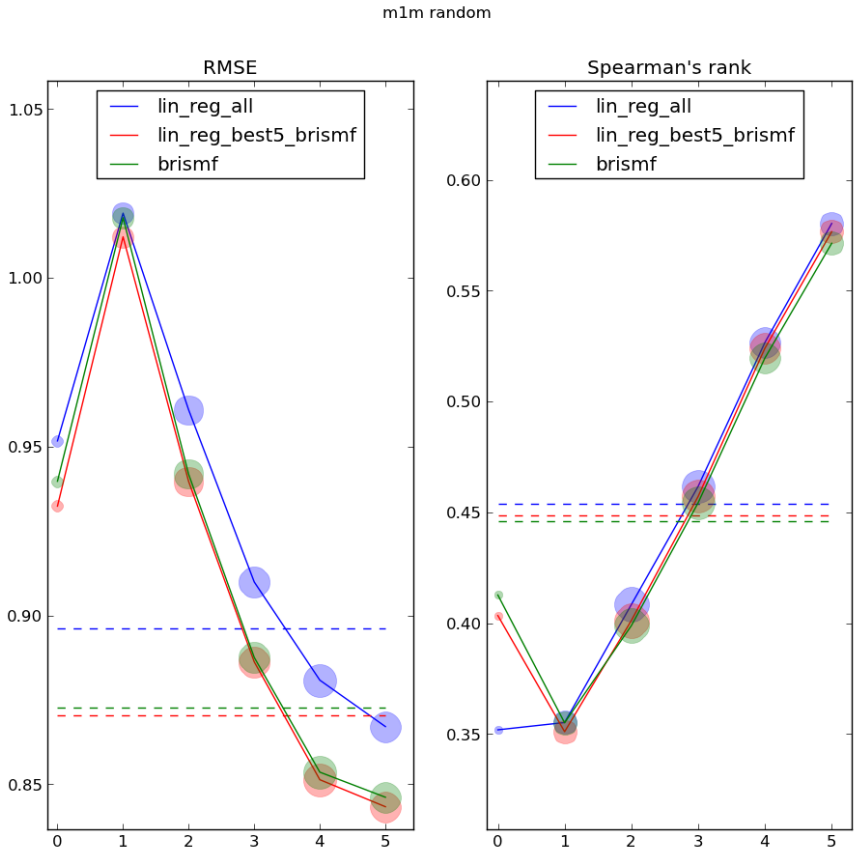
Graf gibanja RMSE ter rang korelacije za naključno razdeljen *dataset m1m* je prikazan na sliki 5.3. Primerjamo:

- najboljšo BRISMF metodo,
- **H1**: hibridizacija (linearna regresija) najboljših 5 BRISMF metod,
- **H2**: hibridizacija (linearna regresija) vseh napovedi.

Opazimo, da smo s hibridno metodo **H1** vedno dosegli boljši RMSE kot samo z najboljšo BRISMF metodo. **H2** ima najslabši RMSE. Pri rang korelaciji

je **H1** v gručah z več kot 10 ocenami na uporabnika boljša od najboljše BRISMF metodo. V vseh gručah, razen v gruči 0 pa je najboljša metoda **H2**. Gruča 0 ima zelo malo uporabnikov, tako da na končni rezultat nima velikega vpliva.

Slika 5.3: Fokusirana evalvacija naključno razdeljene množice *m1m*



5.5.3 Časovno razdeljena množica *m1m*

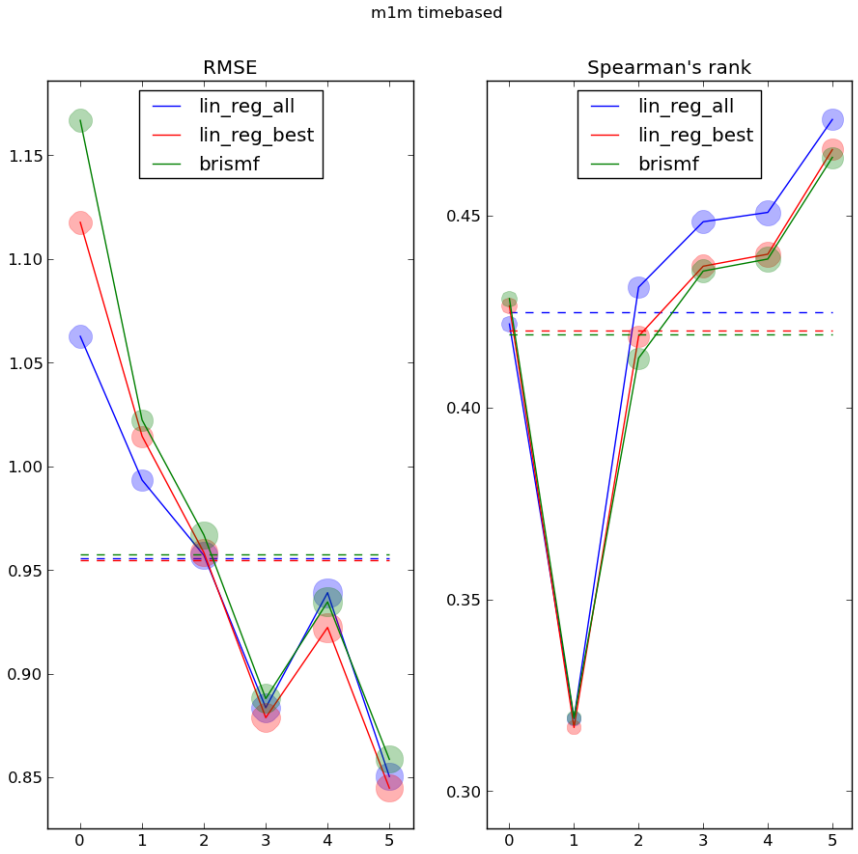
Graf gibanja RMSE ter rang korelacije po gručah uporabnikov za časovno razdeljen *dataset m1m* je prikazan na sliki 5.4. Primerjamo 3 metode:

- najboljšo BRISMF metodo,

- **H1**: najboljša *k-NN*, najboljša BRISMF ter najboljša BRISMF napoved hibridizirane z linearno regresijo,
- **H2**: hibridizacija (linearna regresija) vseh napovedi.

Najboljša BRISMF metoda da vedno, razen v gruči 4 najslabše rezultate. Posledično je tudi povprečni RMSE dosežen z to metodo najslabši. Boljša od te je **H2**. V gručah 0, 1, 2 daje najboljše rezultate. Najboljši RMSE dosežemo z metodo **H1**, ki daje najboljše rezultate v gručah z veliko ocenami na uporabnika, to so 3, 4 in 5. V vseh gručah da boljši rezultat kot najboljša BRISMF metoda. Pri rang korelaciji je najboljša metoda **H2**. V gruči 0 je najslabša, v gruči 1 je med ostalima metodama. V ostalih gručah je veliko boljša od ostalih dveh metoda. Te gruče imajo veliko uporabnikov, zato je tudi končna rang korelacija dosežena s to metodo najboljša. Na drugem mestu glede na rezultat je metoda **H1**, najslabšo rang korelacijo pa smo dobili samo z uporabo najboljše BRISMF metode. V gruči 0, je bila ravno slednja najboljša.

Slika 5.4: Fokusirana evalvacija časovno razdeljene množice *m1m*



Poglavje 6

Zaključek

V diplomskem delu smo implementirali tri metode za učenje priporočanja. To so bile *metoda k najbližjih sosedov (k-NN)*, BRISMF in BPRMF. Od posameznih metod smo najboljše rezultate na majhnih množicah podatkov dobili z metodo *k-NN*, na večjih množicah pa z BRISMFjem. Pri večini podatkovnih zbirk smo rezultate s hibridizacijo še nekoliko izboljšali. Ugotovili smo tudi, da se napovedi zgenerirane z metodo BPRMF zelo dobro združujejo z ostalimi napovedmi. Vedno, ko smo najboljšo rang korelacijo dosegli s hibridizacijo, so bile med metodami, ki so bile vključene v hibridizacijo, tudi napovedi zgenerirane z *BPRMF*jem. Glede na to lahko sklepamo, da *BPRMF* poda neko dodatno informacijo o rangu, ki pride do izraza šele ob hibridizaciji z ostalimi metodami. Verjetno je metoda uporabna pri hibridizaciji zato, ker podatke obravnava na drugačen način (optimizira rang korelacijo in ne točnosti tako kot ostale metode). V spodnji preglednici so najboljši rezultati doseženi na posamezni zbirki podatkov.

6.1 Nadaljnje delo

Predlagamo, da se preizkusi še kakšno metodo hibridizacije. Podatke se lahko razdeli drugače, tokrat so bili podatki za učenje hibridizacije vzeti iz istega časovnega obdobja kot učni podatki za *k-NN*, *BRISMF* oz. *BPRMF* metodo.

Tabela 6.1: Najboljši doseženi rezultati

	RMSE	Spearmanova rang korelacija
m10k naključna delitev	1.008	0.277
m10k časovna delitev	1.119	0.268
m100k naključna delitev	0.937	0.399
m100k časovna delitev	1.045	0.348
m1m naključna delitev	0.870	0.454
m1m časovna delitev	0.955	0.425
iTIVI naključna delitev	0.991	0.300

Morda bi bilo bolje učne podatke za hibridizacijo vzeti iz istega časovnega obdobja kot za učenje parametrov. Prav tako predlagamo zmanjšanje testne množice (predvsem pri velikih učnih množicah), saj bi bila končna evalvacija dovolj natančna tudi na manjši množici, obenem pa bi pridobili še nekaj podatkov za učenje. Lahko bi se uporabilo tudi preostale podatke (o žanru, uporabnikih ...), ki jih ponuja zbirka *movielens*, tako bi lahko točnost napovedi verjetno še izboljšali. Lahko bi se tudi naučili katera metoda daje v neki gruči uporabnikov najboljše rezultate, in nato napovedi generirali glede na to. Tako bi bila točnost napovedi še boljša.

Literatura

[1] MovieLens Data Sets. Dostopno na (30.7.2012):
http://www.grouplens.org/node/73/

[2] Netflix Prize. Dostopno na (17.9.2012):
http://www.netflixprize.com/

[3] Wikipedia: Spearman's rank correlation coefficient. Dostopno na (8.8.2012):
http://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

[4] Wikipedia: Tikhonov regularization. Dostopno na (18.8.2012):
http://en.wikipedia.org/wiki/Tikhonov_regularization

[5] D. Jannach & G. Friedrich: Recommender Systems Tutorial (prosojnice), 2011

[6] G. Takács, I. Pilászy, B. Németh, D. Tikk, "Scalable Collaborative Filtering Approaches for Large Recommender Systems", v zborniku: Journal of Machine Learning Research, zv. 10, str. 623–656, 12.1.2009.

[7] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, "BPR: Bayesian Personalized Ranking from Implicit Feedback", v zborniku: UAI '09 Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, str. 452–461, 2009.

[8] Y. Koren, R. Bell, C. Volinsky, "Matrix factorization techniques for recommender systems", v reviji: IEEE Computer, str. 42–49, avgust 2009.

[9] M. Jahrer, A. Töscher, R. Legenstein, "Combining Predictions for Accurate Recommender Systems", v zborniku "KDD '10 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining", str. 693–702, 2010.

[10] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich, "Recommender Systems: An Introduction". New York: Cambridge University Press, 2011. ISBN 978-0-521-49336-9.

[11] I. H. Witten, F. Eibe, M. A. Hall, "Data mining: Practical Machine Learning Tools and Techniques". Burlington: Morgan Kaufman, 2011. ISBN 978-0-12-374856-0.

Dodatek A

Rezultati optimizacije parametrov

A.1 BRISMF

A.1.1 Najboljših 5

Tabela A.1: Rezultati optimizacije BRISMF, najboljših 5

Podatki	K	m	λ	η
m10k naključna delitev	5000	0.008	0.009	0.012
	1000	0.006	0.002	0.012
	2500	0.000	0.002	0.005
	3000	0.010	0.010	0.005
	500	0.009	0.010	0.018
m10k časovna delitev	300	0.009	0.000	0.020
	700	0.001	0.001	0.020
	5	0.004	0.004	0.005
	50	0.001	0.008	0.012
	100	0.006	0.009	0.007
m100k naključna delitev	2000	0.003	0.007	0.005
	700	0.002	0.001	0.003
	2500	0.006	0.009	0.007
	700	0.007	0.010	0.003
	900	0.003	0.010	0.003
m100k časovna delitev	900	0.003	0.010	0.020
	500	0.002	0.010	0.009
	3000	0.003	0.008	0.007
	700	0.007	0.007	0.016
	100	0.009	0.007	0.014
m1m naključna delitev	2000	0.004	0.000	0.005
	2000	0.009	0.007	0.005
	500	0.008	0.009	0.003
	5000	0.003	0.010	0.009
	5000	0.003	0.010	0.005
m1m časovna delitev	1000	0.010	0.006	0.014
	1500	0.000	0.004	0.016
	500	0.001	0.008	0.016
	50	0.003	0.007	0.005
	1000	0.000	0.009	0.007
iTIVI naključna delitev	20	0.003	0.009	0.005
	50	0.006	0.006	0.007
	100	0.003	0.004	0.014
	900	0.003	0.004	0.009
	900	0.010	0.010	0.020

A.1.2 Naključnih 20

Tabela A.2: Rezultati optimizacije BRISMF, naključnih 20, dataset m10k

Podatki	K	m	λ	η
m10k naključna delitev	500	0.001	0.006	0.014
	1500	0.002	0.007	0.001
	3000	0.006	0.010	0.001
	2500	0.002	0.000	0.005
	5000	0.000	0.009	0.007
	2500	0.007	0.001	0.016
	2500	0.001	0.010	0.014
	900	0.000	0.009	0.007
	3000	0.000	0.009	0.009
	1000	0.000	0.003	0.009
	500	0.003	0.004	0.003
	20	0.004	0.007	0.020
	50	0.003	0.006	0.020
	5000	0.009	0.009	0.009
	300	0.006	0.001	0.014
	500	0.004	0.008	0.001
	500	0.009	0.009	0.018
	1500	0.002	0.010	0.003
	5	0.003	0.000	0.016
	700	0.007	0.002	0.007
m10k časovna delitev	1500	0.010	0.001	0.018
	100	0.006	0.009	0.020
	1000	0.002	0.001	0.005
	5	0.009	0.009	0.018
	500	0.004	0.008	0.012
	700	0.007	0.006	0.018
	5000	0.008	0.001	0.001
	2500	0.007	0.007	0.018
	5000	0.002	0.008	0.018
	100	0.006	0.000	0.009
	3000	0.003	0.004	0.014
	10	0.007	0.009	0.016
	500	0.002	0.003	0.020
	700	0.000	0.008	0.018
	5000	0.003	0.003	0.007
	10	0.004	0.007	0.009
	1000	0.006	0.008	0.016
	700	0.006	0.001	0.001
	500	0.007	0.003	0.009
	2500	0.007	0.001	0.007

Tabela A.3: Rezultati optimizacije BRISMF, naključnih 20, dataset m1m

Podatki	K	m	λ	η
m1m naključna delitev	100	0.002	0.009	0.016
	500	0.000	0.006	0.012
	5000	0.006	0.004	0.016
	700	0.007	0.001	0.001
	2500	0.008	0.001	0.005
	20	0.007	0.009	0.001
	300	0.010	0.006	0.005
	5000	0.010	0.006	0.014
	900	0.008	0.007	0.007
	500	0.008	0.000	0.009
	5	0.006	0.000	0.007
	5	0.006	0.002	0.007
	500	0.003	0.008	0.003
	20	0.000	0.001	0.020
	50	0.010	0.007	0.003
	2500	0.000	0.006	0.020
	20	0.000	0.006	0.016
	700	0.010	0.000	0.020
	1000	0.007	0.008	0.007
	2000	0.007	0.000	0.001
m10k časovna delitev	50	0.008	0.002	0.014
	20	0.009	0.006	0.012
	900	0.010	0.007	0.012
	1000	0.001	0.000	0.003
	3000	0.002	0.001	0.016
	300	0.001	0.000	0.009
	50	0.008	0.001	0.012
	100	0.006	0.001	0.014
	1500	0.008	0.009	0.003
	5000	0.000	0.009	0.007
	3000	0.003	0.006	0.016
	900	0.010	0.010	0.016
	900	0.010	0.009	0.018
	5000	0.007	0.008	0.012
	5	0.008	0.001	0.001
	300	0.004	0.000	0.018
	5000	0.006	0.000	0.007
	50	0.009	0.001	0.007
	10	0.008	0.003	0.001
	300	0.004	0.010	0.014

A.2 BPRMF

A.2.1 Najboljših 5

Tabela A.4: Rezultati optimizacije BPRMF, najboljših 5

Podatki	K	m	λ	η
m10k naključna delitev	2000	0.004	0.003	0.003
	2000	0.000	0.002	0.020
	700	0.001	0.004	0.020
	5000	0.008	0.001	0.007
	5000	0.008	0.001	0.005
m10k časovna delitev	900	0.008	0.000	0.016
	300	0.000	0.000	0.001
	10	0.000	0.001	0.012
	3000	0.000	0.002	0.007
	1500	0.001	0.000	0.018
m100k naključna delitev	300	0.008	0.002	0.001
	50	0.007	0.007	0.001
	300	0.000	0.000	0.001
	2500	0.008	0.003	0.001
	100	0.010	0.003	0.016
m100k časovna delitev	500	0.008	0.007	0.001
	10	0.010	0.009	0.009
	2000	0.009	0.000	0.001
	2000	0.003	0.000	0.003
	500	0.010	0.008	0.001
m1m naključna delitev	900	0.006	0.002	0.005
	50	0.008	0.004	0.003
	1000	0.000	0.001	0.001
	500	0.003	0.004	0.001
	1000	0.008	0.008	0.001
m1m časovna delitev	1000	0.009	0.001	0.003
	100	0.003	0.004	0.005
	10	0.010	0.002	0.005
	300	0.009	0.001	0.003
	500	0.007	0.004	0.005
iTIVI naključna delitev	900	0.007	0.001	0.001
	500	0.006	0.000	0.001
	10	0.003	0.001	0.001
	300	0.007	0.001	0.003
	1500	0.004	0.000	0.003

A.2.2 Naključnih 20

Tabela A.5: Rezultati optimizacije BPRMF, naključnih 20, dataset m10k

Podatki	K	m	λ	η
m10k naključna delitev	900	0.004	0.008	0.005
	1000	0.008	0.010	0.001
	20	0.004	0.003	0.016
	50	0.001	0.007	0.014
	1000	0.009	0.004	0.005
	10	0.010	0.010	0.003
	5	0.004	0.002	0.003
	100	0.006	0.004	0.001
	100	0.003	0.010	0.014
	2500	0.000	0.001	0.020
	300	0.006	0.000	0.018
	900	0.001	0.006	0.016
	1000	0.006	0.008	0.007
	300	0.010	0.000	0.018
	2500	0.002	0.010	0.007
	2500	0.003	0.008	0.016
	1000	0.008	0.003	0.018
	1000	0.003	0.006	0.014
	5000	0.006	0.000	0.003
	500	0.003	0.003	0.020
m10k časovna delitev	100	0.000	0.002	0.016
	5	0.001	0.000	0.009
	2500	0.010	0.002	0.018
	100	0.004	0.002	0.005
	2500	0.006	0.001	0.014
	20	0.004	0.004	0.012
	1000	0.008	0.002	0.005
	2500	0.010	0.000	0.007
	1500	0.003	0.003	0.012
	100	0.008	0.007	0.001
	100	0.004	0.010	0.007
	2000	0.007	0.002	0.003
	100	0.000	0.006	0.001
	20	0.003	0.006	0.005
	5000	0.004	0.003	0.014
	900	0.000	0.002	0.003
	5000	0.006	0.003	0.016
	50	0.004	0.008	0.009
	700	0.009	0.009	0.012
	5	0.010	0.010	0.018

Tabela A.6: Rezultati optimizacije BPRMF, naključnih 20, dataset m1m

Podatki	K	m	λ	η
m10k naključna delitev	50	0.009	0.004	0.007
	10	0.008	0.010	0.018
	2500	0.004	0.008	0.005
	5000	0.010	0.008	0.007
	500	0.006	0.004	0.018
	10	0.002	0.001	0.016
	5000	0.002	0.004	0.020
	300	0.000	0.007	0.020
	500	0.006	0.010	0.020
	3000	0.000	0.007	0.001
	50	0.009	0.001	0.001
	50	0.000	0.008	0.001
	700	0.008	0.008	0.001
	5	0.002	0.000	0.001
	1000	0.000	0.007	0.003
	500	0.007	0.008	0.012
	10	0.007	0.003	0.018
	1000	0.000	0.010	0.014
	2500	0.001	0.006	0.001
	1500	0.009	0.009	0.007
m10k časovna delitev	20	0.008	0.009	0.003
	10	0.000	0.009	0.016
	10	0.001	0.000	0.016
	10	0.003	0.009	0.007
	50	0.009	0.001	0.016
	900	0.002	0.007	0.014
	1000	0.007	0.000	0.009
	1500	0.000	0.008	0.001
	1500	0.003	0.000	0.005
	900	0.007	0.000	0.012
	2500	0.009	0.006	0.009
	10	0.009	0.000	0.012
	100	0.009	0.008	0.007
	2000	0.003	0.003	0.018
	2000	0.001	0.003	0.014
	20	0.002	0.006	0.009
	1000	0.007	0.007	0.007
	2500	0.009	0.008	0.018
	100	0.004	0.010	0.016
	2000	0.006	0.003	0.020

Dodatek B

Dokumentacija

B.1 Razred `Recommender`

Razred `Recommender` vsebuje naslednje metode:

__init__(data=None, normalize=True)

Konstruktor razreda. Naloži podatke in jih po potrebi normalizira. Argumenti:

- `data` Podamo učno množico, lahko je seznam trojk, slovar slovarjev (1. ključ je uporabnik, 2. je stvar) ali pot do CSV ali *pickle* datoteke.
- `normalize` Če je `True` (privzeto) se podatke ob nalaganju normalizira. Nastavimo na `False`, če bomo instanco razreda uporabili za razdelitev podatkov.

__len__()

Vrne število učnih primerov.

blend(blend_function, test_recs, train_recs=None, train_data=None)

Združi različna priporočila. Argumenti:

- `blend_function` Funkcija ki se bo uporabila za dejansko združevanje. Lahko je `blend_lin_reg` ali `blend_avg`, lahko pa podamo svojo fukcijo, ki vzame kot argumente `train_recs`, `train_data` in `test_recs`.
- `test_recs` Seznam priporočil na testni množici, v formatu kot jih vrne `make_recommendations`.
- `train_recs=None` Seznam priporočil kot jih vrne `make_recommendations` za učno množico za *blending*. Če `blend_function` ne potrebuje teh podatkov, jih ni treba podati.
- `train_data=None` Učna množica, na kateri se učimo združevanja. Lahko je seznam trojk (uporabnik, stvar, ocena), slovar slovarjev (prvi ključ uporabnik, drugi stvar) ali datoteka *pickle*, ki vsebuje enega od zgornjih formatov. Če `blend_function` ne potrebuje teh podatkov, jih ni treba podati.

blend_avg(train_recs, train_data, test_recs)

Pomožna metoda, ki jo podamo metodi `blend`. Združi napovedi s povprečenjem. Podrobnejši opis argumentov je pri funkciji `blend`.

blend_lin_reg(train_recs, train_data, test_recs)

Pomožna metoda, ki jo podamo metodi `blend`. Združi napovedi z uporabo linearne regresije. Podrobnejši opis argumentov je pri funkciji `blend`.

bprmf(learn_rate, lamb, max_iterations, iteration_stop_history_length=None, part_train_data=0.1, test_iters=1000)

Metoda za poganjanje optimizacije BPR-Opt. Podpira samodejno ustavljanje; ko so trenutni rezultati slabši od zadnjih nekaj, se optimizacija konča, kot rezultat se vzame najboljše. Argumenti:

- `learn_rate` Hitrost učenja. Parameter α iz enačbe (3.22).
- `lamb` Regularizacijski faktor. Parameter λ iz enačbe (3.22).

- `max_iterations` Maksimalno število iteracij.
- `iteration_stop_history_length` Dolžina vrste v kateri se shranjujejo rezultati evalvacije, ki se jih uporabi za samodejno ustavljanje.
- `part_train_data` Kolikšen del podatkov se odvzame stran za sprotno evalvacijo za samodejno ustavljanje.
- `test_iters` Na vsakih koliko iteracij izvajamo evalvacijo.

brismf(learn_rate, lamb, max_iterations, iteration_stop_history_length=None, part_train_data=0.1)

Metoda za poganjanje optimizacije BRISMF. Podpira samodejno ustavljanje; ko so trenutni rezultati slabši od zadnjih nekaj, se optimizacija konča, kot rezultat se vzame najboljše. Argumenti:

- `learn_rate` Hitrost učenja. Parameter η iz enačb (3.18).
- `lamb` Regularizacijski faktor. Parameter λ iz enačb (3.18).
- `max_iterations` Maksimalno število iteracij.
- `iteration_stop_history_length` Dolžina vrste v kateri se shranjujejo rezultati evalvacije, ki se jih uporabi za samodejno ustavljanje.
- `part_train_data` Kolikšen del podatkov se odvzame stran za sprotno evalvacijo za samodejno ustavljanje.

calculate_RMSE(*args)

Vrne izračunan RMSE po enačbi (3.28). Če je argument en, se predpostavi da so to priporočila, ki jih vrne `make_recommendations`. Če sta argumenta dva, se privzame da sta oba seznama, v enem so napovedi v drugem pa dejanske ocene.

calculate_averages()

Pomožna metoda, ki se uporablja pri normalizaciji.

calculate_spearman(results_with_ratings)

Izračuna povprečno rang korelacijo po uporabnikih. Kot argument vzame priporočila, ki jih vrne `make_recommendations`.

dump_data_to_csv(path)

Shrani učno množico v CSV datoteko. Uporabno predvsem za delitev podatkov. Argument je pot do datoteke, kamor naj se shrani.

dump_data_to_pickle(path)

Shrani učno množico v *pickle* datoteko. Uporabno predvsem pri delitvi podatkov. Branje pickle datoteke je hitrejše od branja CSV datoteke. Argument je pot do datoteke, kamor naj se podatkih shranijo.

dump_item_similarities_buffer(path)

Shrani izračunano matriko podobnosti med stvarmi. Shrani jo v *marshal* datoteko, ki je sicer specifična glede na arhitekturo sistema, kjer je bila zgenerirana, je pa shranjevanje in branje z nje hitrejše kot pri *pickle* datotekah. Argument je pot do datoteke, kamor naj se shrani.

dump_pq(path)

Shrani matriki P in Q, ki se uporabljata pri matrični faktorizaciji v datoteko *pickle*. Argument je pot, kamor naj se matriki shranita.

dump_user_similarities_buffer(path)

Shrani izračunano matriko podobnosti med uporabniki. Shrani jo v *pickle* datoteko. Argument je pot, kamor naj se podobnosti shranijo.

eval_RMSE_divided(bins, recs)

Izračuna RMSE po skupinah uporabnikov, glede na število ocen, ki so jih posamezni uporabniki dali. Argumenti:

- `bins` Seznam z mejami med posameznimi skupinami. Če je zadnja vrednost -1, se ta izračuna avtomatično, tako da zajame vse učne primere.
- `recs` Seznam z priporočili, kot ga vrne `make_recommendations`.

eval_spearman_divided(bins, recs)

Izračuna povprečno rang korelacijo po gručah uporabnikov, glede na število ocen posameznega uporabnika.

- `bins` Seznam z mejami med posameznimi skupinami. Če je zadnja vrednost -1, se ta izračuna avtomatično, tako da zajame vse učne primere.
- `recs` Seznam z priporočili, kot ga vrne `make_recommendations`.

fill_item_similarities_buffer(path=None)

Napolni matriko s podobnostmi med stvarmi. Če podamo argument, potem predpostavi da je to pot do *marshal* datoteke, kjer je matrika shranjena. Sicer podobnost izračuna. Mero podobnosti nastavimo tako, da atribut `item_similarity` nastavimo na želeno metodo. Možne metode so `sim_euclidean_item`, `sim_cosine_item` in `sim_pearson_item`.

fill_pq(K=None, m=None, data=None)

Metoda napolni matriki P in Q za matrično faktorizacijo. Če argumenta `data` ne podamo, se matriki zgenerirata naključno, upoštevajoč argumenta `K` in `m`. $[-m, m]$ predstavlja interval s katerega se enakomerno naključno izbirajo vrednost za inicializacijo matrike. Argument `K` predstavlja število latentnih spremenljivk (glej enačbo (3.13)). Če podamo argument `data` se `K` in `m` ignorirata, matriki P in Q pa se naložita z diska. `Data` je pot do datoteke v datotečnem sistemu ali pa seznam, ki vsebuje dve matriki.

fill_similar_items_buffer()

Metoda zgenerira slovar, ki za vsako stvar hrani seznam najbolj podobnih stvari (glede na matriko podobnosti stvari, ki jo sestavi metoda `fill_item_similarities_buffer`).

fill_similar_users_buffer()

Metoda zgenerira slovar, ki za vsakega uporabnika hrani seznam njemu najbolj podobnih uporabnikov. Pri tem uporabi matriko podobnosti uporabnikov, ki jo sestavi metoda `fill_user_similarities_buffer`.

fill_user_similarities_buffer(self, path=None)

Napolni matriko s podobnostmi med uporabniki. Če podamo argu-

ment, potem predpostavi, da je to pot do *marshal* datoteke, kjer je matrika shranjena. Sicer podobnost izračuna. Mero podobnosti nastavimo tako, da atribut `user_similarity` nastavimo na želeno metodo. Možne metode so `sim_euclidean_user`, `sim_cosine_user` in `sim_pearson_user`.

generate_number_of_ratings_per_user()

Pomožna metoda, ki shrani slovar ocen za vsakega uporabnika.

load_data(data)

Pomožna metoda, ki jo kliče konstruktor. Vzame parameter `data`, ki je enak kot ga dobi konstruktor.

make_recommendations(test_data, recommend_function, prints, **kwargs)

Metoda, ki naredi priporočila na testni množici. Argumenti:

- `test_data` je seznam štirk `(uporabnik, stvar, ocena, čas)` ali pot do *pickle* ali CSV datoteke.
- `recommend_function` je metoda, ki naj se uporabi za pridobivanje posameznega priporočila (za uporabnika u in stvar i). To je lahko ena izmed teh metod:
  - `recommend_avg_avg`,
  - `recommend_dummy`,
  - `recommend_pq`.
  - `recommend_sim_item`,
  - `recommend_sim_user`,
- `prints` Če je `Flase`, metoda ne bo izpisovala stanja, sicer bo.
- `kwargs` Če funkcija za generiranje posameznega priporočila potrebuje dodatne parametre, jih podamo tu.

normalize_data()

Pomožna metoda, ki jo kliče konstruktor, če želimo da se podatki normalizirajo.

recommend_avg_avg(user, item)

Metoda vrne napoved ocene za kombinacijo uporabnika `user` in stvari `item`. Vrne povprečje povprečje ocene uporabnika in povprečne ocene stvari.

recommend_dummy(user, item)

Metoda vrne napoved ocene za kombinacijo uporabnika `user` in stvari `item`. Vrne povprečje stvari korigirano z pesimizmom/optimizmom uporabnika.

recommend_pq(user, item)

Metoda vrne napoved ocene za kombinacijo uporabnika `user` in stvari `item`. Vrne skalarni produkt uporabnikove vrstice iz matrike P in stolpca ki predstavlja stvar v matriki Q.

recommend_sim_item(user, item, N, min_sim_margin, K)

Metoda vrne napoved ocene za kombinacijo uporabnika `user` in stvari `item`, ki bazira na metodi *k-NN*. Uporabi matriko podobnosti med stvarmi. Argument `N` predstavlja koliko najbolj podobnih stvari se pri priporočanju upošteva, `min_sim_margin` je minimalna podobnost med stvarmi, da se podobna stvar upošteva. `K` je parameter K iz enačbe (3.2).

recommend_sim_user(user, item, N, min_sim_margin, K)

Metoda vrne napoved ocene za kombinacijo uporabnika `user` in stvari `item`, ki bazira na metodi *k-NN*. Uporabi matriko podobnosti med uporabniki. Argument `N` predstavlja koliko najbolj podobnih uporabnikov se pri priporočanju upošteva, `min_sim_margin` je minimalna podobnost med uporabniki, da se podoben uporabnik upošteva. `K` je parameter K iz enačbe (3.2).

sim_cosine_item(item1, item2)

Vrne kosinusno podobnost med stvarema `item1` in `item2`.

sim_cosine_user(user1, user2)
Vrne kosinusno podobnost med uporabnikoma `user1` in `user2`.

sim_euclidean_item(item1, item2)
Vrne evklidsko podobnost med stvarema `item1` in `item2`.

sim_euclidean_user(user1, user2)
Vrne evklidsko podobnost med uporabnikoma `user1` in `user2`.

sim_pearson_item(item1, item2)
Vrne Pearsonov korelacijski koeficient med stvarema `item1` in `item2`.

sim_pearson_user(user1, user2)
Vrne Pearsonov korelacijski koeficient med uporabnikoma `user1` in `user2`.

B.2 Modul `data`

Ker je raznih poti v datotečnem sistemu, ki jih rabimo za branje in shranjevanje podatkov veliko, smo napisali modul `data`. Vsebuje metodo `data(data_split=None)`, ki vrne slovar z potmi. Argument `data_split` ima lahko dve vrednosti:

`random` Poti kažejo na podatke, ki so bili razdeljeni naključno.

`timebased` Poti kažejo na podatke, ki so bili razdeljeni upoštevajoč čas, ko so bile ocene dane.

Slovar, ki ga metoda vrne je večdimenzionalen, točna uporaba je vidna v razdelku 4.2.

B.3 Modul `split_data`

V tem modulu se nahajajo metode za generiranje učnih in testnih množic. Vse potrebne poti se preberejo iz modula `data`.

generate_itivi()

Metoda se poveže na lokalno *Firebird* bazo in podatke o ocenah filmov iz zbirke *iTIVI* shrani na disk v CSV datoteko.

generate_m10k()

Metoda iz dataseta *m100k* vzame ocene naključnih 100 uporabnikov.

split_data(dataset, train=0.8, train_params=0.2, train_blending=0.2, random_seed = 123, timebased=True)

Metoda razdeli podatke na testne in učne množice. Argumenti metode:

- `dataset` Podatkovna zbirka, ki jo želimo razdeliti.
- `train` Delež podatkov, ki jih damo v učno množico. To množico nato delimo še naprej preostali podatki pa postanejo testni.
- `train_params` Delež podatkov znotraj učne množice, ki jih uporabimo za optimizacijo parametrov.
- `train_blending` Delež podatkov znotraj učne množice, ki jih uporabimo za učenje *blendinga*.
- `random_seed` Nastavimo seme psevdonaključnega generatorja števil.
- `timebased` Če ga nastavimo na `True`, bodo podatki razdeljeni glede na čas, ko so bile ocene pridobljene, sicer pa se bodo vse množice generirale naključno.