UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matic Perovšek

ENORAZREDNI PRIPOROČILNI SISTEMI

DIPLOMSKO DELO
NA INTERDISCIPLINARNEM UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Blaž Zupan

Ljubljana, 2011

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: 00030/2011
Datum: 05.09.2011

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

Kandidat: **MATIC PEROVŠEK**

Naslov: **ENORAZREDNI PRIPOROČILNI SISTEM**
SINGLE-CLASS RECOMMENDATION SYSTEMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Ob pregledovanju spletnih strani, izboru zanimivih knjig, izdelkov, stvari, oblikovanju skupin prijateljev na družabnih omrežjih in podobnih, za današnjo družbo značilnih spletnih aktivnostih, mnogokrat želimo, da nam uporabljeno okolje svetuje o primernih izbirah. Spletne aplikacije v te namene uporabljajo priporočilne sisteme, ki iz podatkov o preteklih uporabnikovih izbirah lahko izluščijo njegovo preferenčno znanje ter ga uporabijo pri računalniško-podprtem svetovanju. V nalogi izdelajte pregled tovrstnih sistemov, izbrane implementirajte in njihovo delovanje preverite na podatkovnem naboru spletnih strani s strokovnimi članki CiteULike. Osredotočite se na enorazredne sisteme, kjer je edina vrsta podatkov ta o pozitivni preferenci uporabnika za izbrano množico stvari.

Mentor:

prof. dr. Blaž Zupan

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

prof. dr. Andrej Likar

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 6

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Matic Perovšek,

z vpisno številko 63060233,

sem avtor/-ica diplomskega dela z naslovom:

Enorazredni priporočilni sistemi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom prof. dr. Blaža Zupana
- 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 27.09.2011 Podpis avtorja/-ice:

Zahvala

Za strokovno vodenje, nasvete in pomoč pri izdelavi diplomskega dela se iskreno zahvaljujem mentorju prof. dr. Blažu Zupanu. Zahvaljujem se tudi Laboratoriju za bioinformatiko, ki mi je omogočil izvajanje testnih poizkusov na njihovem računalniškem sistemu.

Zahvaljujem se tudi družini, ki me je med študijem podpirala tako finančno kot tudi moralno.

Kazalo

Povzetek

Priporočilni sistemi se danes pogosto uporabljajo v spletnih aplikacijah in trgovinah. Uporabniku skušajo ugoditi tako, da mu iz velike množice stvari pomagajo izbrati najustreznejše glede na njegove preference. Prikrite preferenčne kriterije se priporočilni sistemi učijo na podatkih, ki pogostokrat vsebujejo le enorazredne vrednosti (kliki na spletne povezave, spletni zaznamki ...). Tovrstni podatki zajemajo le pozitivne primere - predmete, ki so bili uporabniku všeč oziroma je zanje pokazal zanimanje. Za ostale predmete je preferenca neznana in je lahko pozitivna ali negativna. Raziskali smo nekaj priporočilnih algoritmov, ki se lahko soočijo s tovrstnimi podatki. V ta namen smo preverili dve vrsti algoritmov. Prvi, algoritmi RISMF in wALS, temeljijo na matrični faktorizaciji. Z njeno pomočjo identificirajo medsebojna razmerja med uporabniki in stvarmi, katera nato uporabijo za priporočanje. Preverili pa smo tudi algoritme, ki za napoved uporabljajo tehniko najbližjih sosedov: na uporabnikih in na stvareh temelječa algoritma k najbližjih sosedov. Vse algoritme smo implementirali ter jih med seboj primerjali na podatkovni bazi spletne strani CiteULike.org. Na izbranih podatkih smo najboljše rezultate dosegli z algoritmi wALS.

Ključne besede:

priporočilni sistemi, enorazredni podatki, filtriranje medsebojnih povezanosti, matrična faktorizacija, metode k najbližjih sosedov, CiteULike.org podatkovna baza

Abstract

The task of recommender systems is to recommend items that fit the user's preferences. Recommender systems are today often used in web applications and shops in order to help the user in selecting and purchasing items from an overwhelming set of choices. The data from where the hidden preference criteria can be learned often only contains single-class values (web links clicks, bookmarks ...) instead of elaborative ranking. Such data is comprised of only positive examples, listing items that the user has liked or has expressed interest for. For other items, the preference is unknown and may be positive or negative. In this work we study the recommender algorithms that can learn from such data. We examined two types of algorithms. First, RISMF and wALS algorithms, are based on matrix factorization which identifies and later recommends items based on relationships between users and items. We also proposed two types of nearest neighbours algorithms: user and item based. We implemented all of the algorithms and performed a comparison on CiteULike.org website's database. Results show that wALS algorithms give the best results on the selected data.

Key words:

Recommender systems, one-class data, collaborative filtering, matrix factorization, k-Nearest neighbours methods

Poglavje 1

Uvod

Priporočilni sistemi uporabljajo matematične in statistične tehnike za modeliranje uporabnikovih preferenc o stvareh. Glavna naloga priporočilnih sistemov je, da skušajo uporabnikom čimbolj točno napovedovati najustreznejše stvari. Poosebljeni predlogi so še posebej pomembni na trgih, kjer je izbira zelo velika. Področja, kjer se največkrat uporabljajo, so povezana z moderno kulturo (filmi, knjige, glasba), modo in restavracijami [1]. Uporabniki sistemov na teh področjih izbrane stvari ocenijo, na podlagi ocen pa jim priporočilni sistemi ponudijo nove stvari, ki bi jih utegnile zanimati.

Tabela 1.1: Primer matrike ocen, na kakršni se uporabijo priporočilni sistemi.

	Usual Suspects	Home Alone	Terminator	Love Actually	Batman
Miha	5	-	4	-	-
Rok	4	-	-	-	5
Angelika	-	4	-	5	-
Tatjana	1	-	-	4	-
Gorazd	-	3	5	-	4
Ana	-	2	3	5	-
Meta	1	5	-	4	3

Tabela 1.1 prikazuje enostaven primer matrike ocen, kakršne uporabljajo priporočilni sistemi. V podanem primeru uporabniki ocenjujejo filme z ocenami od 1 do 5, kjer je 5 najvišja ocena. Manjkajoči vnosi so označeni s pomišljajem. V tem primeru sta uporabniku *Rok* všeč filma *Usual Suspects* in *Batman*, uporabniku *Miha* pa je poleg *Usual Suspects* všeč film *Terminator*. Dober priporočilni sistem, bi opazil, da sta uporabnikom, kateri so bili navdušeni nad filmom *Usual Suspects*, všeč filma *Batman* in *Terminator*. Uporabniku *Rok* bi

tako bil priporočen film *Terminator*, uporabniku *Miha* pa film *Batman*. Podobno lahko opazimo, da uporabniki, katerim je všeč film *Love Actually* in ne marajo filma *Usual Suspects*, obožujejo film *Home Alone*. Uporabniku *Tatjana* bi tako bil priporočen film *Home Alone*.

V naslednjem poglavju bomo definirali splošen problem napovedovanja preferenc. V poglavju 1.2 pa bomo razložili, zakaj je problem z enorazrednimi podatki potrebno obravnavati drugače od splošnega.

1.1 Definicija problema

Problem, ki ga priporočilni sistemi rešujejo, se da zapisati z naključno trojico (U, I, R), kjer je:

- U identifikator uporabnika in lahko zavzema vrednosti med $\{1, ..., N\}$, N je število vseh uporabnikov in $N \in \mathbb{N}$,
- I identifikator stvari in lahko zavzema vrednosti med $\{1, ..., M\}$, M je število vseh stvari in $M \in \mathbb{N}$,
- R zavzema vrednosti $X \subset \mathcal{R}$. Vrednosti so lahko binarne($X = \{0, 1\}$), cela števila ($X = \{1, 2, 3, 4, 5\}$), tudi realna števila na zaprtem intervalu ($X = [-50, 50]$).

Trojica (u, i, r), kjer $u \in U$, $i \in I$ in $r \in X$, pomeni, da je uporabnik u stvari i dal oceno r.

Cilj preferenčnega modeliranja je, da se oceni R glede na (U, I) tako, da je kvadratni koren povprečne kvadratne napake (*angl. Root Mean Squared Error, RMSE*)

$$RMSE = \sqrt{E\{(\widehat{R} - R)^2\}} \tag{1.1}$$

minimalen [1]. $\widehat{R}$ so ocene priporočilnega sistema za R.

V praksi nam celotna porazdelitev (U, I, R) ni poznana. Podano imamo le končno množico uporabnikov ter njihovih ocen za različne stvari $\mathcal{T}' = (u_1, i_1, r_1), (u_2, i_2, r_2), ..., (u_t, i_t, r_t)$. Množico $\mathcal{T}'$ vzamemo za učno množico, pri tem pa predpostavimo, da lahko vsak uporabnik glasuje le enkrat za posamezno stvar. Definirajmo množico parov $\mathcal{T} = \{(u, i); \exists r : (u, i, r) \in \mathcal{T}'\}$. $|\mathcal{T}'| = |\mathcal{T}|$, tipično pa velja in $|\mathcal{T}| \ll N * M$, saj uporabniki večinoma podajo preferenčno znanje le za majhno število stvari. Množico je moč opisati z redko matriko R, ki ima na mestih $(u, i) \in \mathcal{T}$, označene z r_{ui}, oceno uporabnika u za stvar i, na ostalih mestih pa ima matrika nedefinirane vrednosti.

Cilj priporočilnih sistemov je torej zgraditi napovedne modele, ki čim bolj minimizirajo napako iz enačbe (1.1). Ker je porazdelitev (U, I, R) neznana, ne moremo izmeriti dejanske napake. Zato skušamo napako oceniti na testni množici, ki jo definiramo z $\mathcal{V}' \subset [1, ..., N] \times [1, ..., M] \times X$. Pari (u,i) so tako porazdeljeni med množici $\mathcal{T}'$ in $\mathcal{V}'$. Definiramo še $\mathcal{V} = \{(u, i); \exists r : (u, i, r) \in \mathcal{V}'\}$, zaradi prej omenjenih pogojev velja $\mathcal{T} \cap \mathcal{V} = \emptyset$. Če sta učna množica $\mathcal{T}'$ in testna množica $\mathcal{V}'$ generirani na isti porazdelitvi, se da približek za RMSE izračunati z:

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

1.2 Enorazredni podatki

Priporočilni sistemi se učijo na ocenah uporabnikov za različne stvari. Ker je uporabnikom pogosto odveč ocenjevati veliko stvari (in je zatorej težavno zbirati ocene uporabnikov), mnogo sistemov temelji na implicitno pridobljenih podatkih kot je npr. klik na določeno povezavo ali pa zaznamek določenega predmeta/strani. Ko imamo na razpolago tovrstne podatke, nastane problem, saj negativnih primerov (primeri, ko uporabniku stran/predmet ni bil všeč ter ga zato zavestno ni zaznamoval) in manjkajočih pozitivnih primerov (uporabnik določene stvari, ki bi mu lahko sicer bila všeč, ni sploh videl) ne more razlikovati. Domena množice možnih ocen je $X = \{1\}$.

Tabela 1.2: Primer matrike enorazrednih podatkov. Oznaka 1 pomeni, da je bil film uporabniku všeč.

	Usual Suspects	Home Alone	Terminator	Love Actually	Batman
Miha	1	-	1	-	-
Rok	1	-	-	-	1
Angelika	-	1	-	1	-
Tatjana	-	-	-	1	-
Gorazd	-	-	1	-	1
Ana	-	-	-	1	-
Meta	-	1	-	1	-

Tabela 1.2 prikazuje primer matrike ocen z enorazrednimi podatki. Dobljena je iz tabele 1.1. Oznaka 1 pomeni, da je bil film uporabniku všeč - v prvotni tabeli je uporabnik film ocenil z 4 ali 5, vse ocene med 1 in 3 pa so obravnavane kot, da uporabniku film ni bil všeč. Kot je značilnost enorazrednih podatkov

ne moremo medseboj razlikovati primerov, katerih uporabnik ni videl, ter primerov, kateri mu niso bili všeč. Primer: uporabnik *Meta* iz tabele 1.1 je filmu *Usual Suspects* dal oceno 1, v tabeli z enorazrednimi vrednostmi je oznaka za ta film enaka kot za film *Terminator*, katerega si ta uporabnik verjetno ni nikoli ogledal.

V praksi se za probleme z enorazrednimi podatki največkrat uporabljajo sistemi s tehniko, ki jo imenujemo enorazredna strategija filtriranja medsebojnih povezanosti (*angl. One-class Colaborative Filtering, OCCF*) [2]. V delu bomo predstavili ter opisali 7 različnih algoritmov za reševanje problemov priporočanja. Vse algoritme smo tudi implementirali v programskem jeziku Python ter jih v poglavju 3 eksperimentalno primerjali na podatkih pridobljenih iz spletnih strani CiteULike.org. Statistično ovrednotene rezultate smo podali v poglavju 3.4.4.

Poglavje 2

Priporočilni sistemi

Na priporočilne sisteme lahko gledamo kot na posebno vrsto aplikacij filtriranja informacij, katerih cilj je določiti mnogo manjšo in kompaktnejšo podmnožico stvari za določenega uporabnika. V literaturi se priporočilni sistemi delijo v dve kategoriji [3], kjer uporabnike in stvari povezujemo na podlagi njihovih vsebinskih opisov ali pa na podlagi že zbranih preferenčnih podatkov.

Pristop, ki temelji na vsebinskem opisu uporabnikov in stvari (*angl. Content based Approach*), profilira uporabnike in stvari, tako da se najprej zbere njihove karakteristične značilnosti (npr. demografske podatke o uporabnikih ter informacije o produktu za stvari) [4]. V algoritmih se nato ti profili uporabijo za povezovanje uporabnikovih interesov z opisi stvari. Ta strategija ima veliko pomanjkljivost: pogostokrat je težavno zbrati vse potrebne podatke o stvareh, kot tudi motivirati uporabnike, da delijo veliko osebnih podatkov.

Zato se v praksi večkrat uporablja strategija filtriranja na podlagi medsebojnih povezanosti (*angl. Collaborative Filtering, CF*) [1]. Ta ne potrebuje eksplicitnih profilov in se uči le s pomočjo uporabnikovih preteklih aktivnosti (pretekle transakcije, obiski strani, ocene izdelkov ...). CF se da uporabiti za priporočilne sisteme neodvisno od domene problema. Algoritmi, ki se pri tem uporabljajo, identificirajo medsebojna razmerja med uporabniki in stvarmi ter nato uporabijo vzorce pridobljene iz teh informacij za napovedovanje uporabnikovih preferenc. Vendar pa tehnika CF trpi za t.i. problemom hladnega zagona (*angl. cold start*) [4], saj za nove uporabnike (ali stvari) nima podlage preteklih izkušenj. S tega stališča je filtriranje na podlagi vsebine boljše. Pogostokrat se zato uporabljajo t.i. hibridne metode [3], ki so v bistvu kombinacija obeh principov.

V tem delu smo se ukvarjali s tehnikami filtriranja medsebojnih povezanosti. Dve glavni področji CF sta: metode matričnih faktorizacij (*angl. Matrix*

Factorization, MF) ter metode najbližjih sosedov (*angl. neighborhood methods*) [4]. Metode z matrično faktorizacijo skušajo karakterizirati uporabnike in stvari z npr. 20 do 200 značilnostmi (*angl. features*), ki jih povzamejo iz matrike ocen. V poglavju 2.2 bomo predstavili dva algoritma, ki temeljita na matrični faktorizaciji, RISMF in wALS. Metode najbližjih sosedov so osredotočene na izračun razmerij med uporabniki oziroma stvarmi. V poglavju 2.3 bomo opisali ter implementirali dva algoritma najbližjih sosedov; prvi temelji na odvisnostih med uporabniki, drugi pa med stvarmi.

2.1 Strategije reševanja enorazrednih problemov

V sistemih filtriranja medsebojne povezanosti je pogosto težko zbirati ocene uporabnikov. Zato mnogo sistemov temelji na implicitno pridobljenih podatkih kot so npr. kliki na določeno povezavo ali pa zaznamki določenega predmeta/strani. Za probleme z enorazrednimi podatki je značilno, da ne moremo ločiti med primeri, ko je nek uporabnik neko stvar označil za negativno ali pa je sploh ni videl.

Obstajata dve strategiji, kako obravnavati manjkajoče vrednosti: Prva strategija obravnava vse manjkajoče primere kot neznane (*angl. All Missing as Unknown, AMAU*) [2]. Ta strategija ignorira manjkajoče primere ter poda algoritmu le množico pozitivnih primerov. Velika pomanjkljivost tega je, da obstaja trivialna rešitev tega problema - označitev vseh testnih primerov za pozitivne. Druga strategija obravnava vse manjkajoče primere kot negativne (*angl. All Missing as Negative, AMAN*). Njena slabost je, da se nekateri manjkajoči pozitivni primeri (npr. strani, katere bi uporabnik ob ogledu zaznamoval) obravnavajo kot negativni. V naslednjem poglavju bomo predstavili algoritem RISMF, ki se uporablja predvsem za probleme z večjo domeno ocen. Ta algoritem bomo v poglavju testiranja 3 preverili tako s strategijo AMAU, kot tudi z AMAN. Porodila pa se je tudi ideja, da bi manjkajoče primere označili za negativne ter jih obtežili [2] ter jih tako naredili manj pomembne v primerjavi s pozitivnimi pri odločanju. Na tej ideji temelji algoritem wALS [2], ki bo predstavljen v poglavju 2.2.3.

2.2 Sistemi faktorizacije matrik

2.2.1 Algoritem ISMF

Faktorizacija matrik je ena izmed največkrat uporabljenih tehnik v strategijah filtriranja na podlagi medsebojnih povezanosti [1]. Tehnika je postala popularna, ko je Simon Funk na svojem spletnem dnevniku objavil matrično faktorizacijo s stohastičim gradientnim spustom [4], kasneje pa je sledilo mnogo modifikacij tega modela s strani drugih avtorjev. Ideja metod, ki uporabljajo matrično faktorizacijo je, da skušamo aproksimirati matriko R kot produkt dveh matrik:

$$R \approx PQ,$$

kjer je P dimenzije $N \times K$ in Q dimenzij $K \times M$. P imenujemo matrika značilnosti (*angl. features*) za uporabnike, Q matrika značilnosti za stvari, medtem ko je K število značilnosti v dani faktorizaciji. Če gledamo na matrike kot linearne transformacije, se da aproksimacijo interpretirati kot: matrika Q transformira prostor $S_1 = \mathcal{R}^M$ v $S_2 = \mathcal{R}^K$, medtem ko matrika P transformira S_2 v $S_3 = \mathcal{R}^N$. Tipično velja, da je K mnogo manjši od M in N, zato lahko ocenimo, da je število potrebnih parametrov za opis matrike R namesto $|\mathcal{T}|$ kar $NK + KM$. Matriki P in Q sta sestavljeni iz realnih števil tudi kadar R vsebuje le cela.

Aproksimacijski problem se da zapisati na sledeči način. Naj p_{uk} označuje element iz $P \in \mathcal{R}^{N \times K}$ in q_{ki} element iz $Q \in \mathcal{R}^{K \times M}$, medtem ko sta p_u vrstica iz P in q_i stolpec iz Q. Potem:

$$\widehat{r}_{ui} := \sum_{k=1}^{K} p_{uk} q_{ki} = p_u q_i, \tag{2.1}$$

$$e_{ui} := r_{ui} - \widehat{r}_{ui}; \text{ za } (u,i) \in \mathcal{T},$$

$$e'_{ui} := \frac{1}{2} e_{ui}^2, \tag{2.2}$$

$$SSE := \sum_{(u,i)\in\mathcal{T}} e_{ui}^2 = \sum_{(u,i)\in\mathcal{T}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} \right)^2,$$

$$SSE' := \frac{1}{2} SSE = \sum_{(u,i)\in\mathcal{T}} e'_{ui},$$

$$RMSE := \sqrt{\frac{SSE}{|\mathcal{T}|}},$$

$$(P*, Q*) = \underset{(P,Q)}{\operatorname{argmin}} SSE' = \underset{(P,Q)}{\operatorname{argmin}} SSE = \underset{(P,Q)}{\operatorname{argmin}} RMSE. \tag{2.3}$$

$\widehat{r}_{ui}$ je ocena modela, kako bi uporabnik u glasoval za stvar i. e_{ui} predstavlja zmoto učenja za primer (u, i), medtem ko je SSE vsota kvadratov zmot na vseh učnih primerih. Enačba (2.3) pogojuje, da optimalna P in Q minimizirata vsoto kvadratov napak le na znanih elementih iz R.

Minimizacija RMSE je ekvivalentna minimizaciji SSE'. Za slednjo lahko uporabimo metodo inkrementalnega gradientnega spusta, ki poišče SSE'-jev lokalni minimum. Vsak korak zmanjša kvadrat napake napovedi ene ocene, kar je ekvivalentno e'_{ui} oz e^2_{ui}.

Na minimizacijo RMSE lahko gledamo kot na obteženo aproksimacijo nizkega reda (*angl. weighted low rank approximation, wLRA*) [1], ki skuša minimizirati relevantno funkcijo $SSE_w = \sum_{u=1}^{N} \sum_{i=1}^{M} w_{ui} \cdot e^2_{ui}$, kjer so w_{ui} vnaprej definirane nenegativne uteži. Kadar uporabimo tehniko AMAU za reševanje problema z enorazrednimi podatki je utež w_{ui} enaka 1 za znane primere in 0 za neznane. V primeru tehnike AMAN pa so tako uteži za znane kot tudi neznane primere enake 1.

Pri metodi inkrementalnega gradientnega spusta, če gledamo učni primer (u, i), za katerega poznamo r_{ui} in njegovo aproksimacijo $\widehat{r}_{ui}$, izračunamo gradient od e'_{ui} z:

$$\frac{\partial}{\partial p_{uk}} e'_{ui} = -e_{ui} * q_{ki}, \qquad \frac{\partial}{\partial q_{ki}} e'_{ui} = -e_{ui} * p_{uk}$$

Uteži posodobimo v nasprotni smeri gradienta:

$$p'_{uk} = p_{uk} + \eta * e_{ui} * q_{ki},$$
$$q'_{ki} = q_{ki} + \eta * e_{ui} * p_{uk}.$$

Uteži v P in Q spremenimo z namenom zmanjšanja kvadrata napake in s tem posledično izboljšanje aproksimacije za r_{ui}. Spremenljivka η predstavlja stopnjo učenja (*angl. learning rate*). Po končanem učenju se da vsako vrednost R izračunati z enačbo (2.1), tudi za neznane vrednosti. S tem modelom - matrikama $P*$ in $Q*$ iz enačbe (2.3) - lahko torej napovemo, kako bi nek uporabnik ocenil poljubno stvar.

2.2.2 Algoritem RISMF

Matrična faktorizacija predstavljena v prejšnjem poglavju se lahko preveč prilagaja (*angl. overfit*) uporabnikom, ki so podali malo ocen (ne več kot K). Pogost način, kako se temu izogniti, je uporaba regularizacijskega faktorja na kvadratu evklidske norme uteži [1]. Ta tehnika se pogostokrat uporablja tudi v drugih metodah strojenega učenja (npr. metodah podpornih vektorjev). Rešujemo nov optimizacijski problem:

$$\widehat{r}_{ui} := \sum_{k=1}^{K} p_{uk} q_{ki} = p_u q_i, \tag{2.4}$$

$$e_{ui} := r_{ui} - \widehat{r}_{ui} \quad za\ (u,i) \in \mathcal{T},$$

$$e'_{ui} := \frac{1}{2}\left(e_{ui}^2 + \lambda * p_u * p_u^T + q_i^T * q_i\right), \tag{2.5}$$

$$SSE' := \sum_{(u,i)\in\mathcal{T}} e'_{ui},$$

$$(P*, Q*) = \underset{(P,Q)}{\operatorname{argmin}}\, SSE'. \tag{2.6}$$

$\lambda \geq 0$ je regulizacijski faktor. To metodo imenujemo regularizirana inkrementalna hkratna matrična faktorizacija (*angl. Regularized Incremental Simultaneous Matrix Factorization, RISMF*). Pri tej metodi velja, da je minimizacija SSE' ekvivalentna minimizaciji SSE le, ko je $\lambda = 0$. Takrat je to kar metoda ISMF.

Podobno kot pri metodi ISMF se gradient od e'_{ui} izračuna z:

$$\frac{\partial}{\partial p_{uk}} e'_{ui} = -e_{ui} * q_{ki} + \lambda * p_{uk}, \qquad \frac{\partial}{\partial q_{ki}} e'_{ui} = -e_{ui} * p_{uk} + \lambda * q_{ki}.$$

Sledi posodobitev uteži v nasprotni smeri gradienta:

$$p'_{uk} = p_{uk} + \eta * (e_{ui} * q_{ki} - \lambda * p_{uk}), \tag{2.7}$$

$$q'_{ki} = q_{ki} + \eta * (e_{ui} * p_{uk} - \lambda * q_{ki}). \tag{2.8}$$

Izvorna koda algoritma RISMF je priložena v dodatku A.

Časovna zahtevnost

Algoritem RISMF na vsakem koraku izvajanja za vsak podan primer z enačbama (2.7) in (2.8) posodobi vrstico matrike P ter stolpec matrike Q. Število podanih primerov je največ $N \cdot M$, dolžina stolpca oziroma vrstice pa je dolžine K. Časovna zahtevnost ene iteracije je tako:

$$O(KNM).$$

Ker se postopek ponovi v vsaki iteraciji, je celoten čas algoritma RISMF:

$$O(Kn_tNM),$$

kjer je n_t število iteracij.

2.2.3 Algoritem wALS

Metoda izmenjujočih najmanjših kvadratov (*angl. weighted Alternating Least Squares, wALS)* se nahaja nekje med strategijama AMAU in AMAN. Uporablja namreč obteženo aproksimacijo nizkega reda [2]. Ideja na kateri temelji je, da se uporabi manjša obtežitev za neznane primere. Pozitivnim primerom se tako priredi višji pomen pri učenju v primerjavi z neznanimi/negativnimi.

Za dano matriko $R = (R_{ij})_{N\times M} \in \{0,1\}^{N\times M}$ z N uporabniki in M stvarmi in ustrezno matriko nenegativnih uteži $W = (W_{ij})_{N\times M} \in \mathcal{R}_+^{N\times M}$, obtežena aproksimacija nizkega reda stremi k iskanju matrike nizkega reda $\widehat{R} = (\widehat{R}_{ij})_{N\times M}$, ki minimizira obteženo Frobeniusovo funkcijo napake (*angl. Frobenius loss function*):

$$\mathcal{L}(\widehat{R}) = \sum_{ij} W_{ij}(R_{ij} - \widehat{R}_{ij})^2 \tag{2.9}$$

V funkciji $\mathcal{L}(\widehat{R})$ enačbe (2.9) je $(R_{ij} - \widehat{R}_{ij})^2$ skupna kvadratična napaka (*angl. common square error*), ki jo pogostokrat zasledimo v aproksimacijah nizkega reda. V OCCF nastavimo $R_{ij} = 1$ in $W_{ij} = 1$ za pozitivne primere, za neznane primere pa predpostavimo, da je večino primerov negativnih, zato zanje nastavimo $R_{ij} = 0$, medtem ko uteži zanje znižamo, tako da $W_{ij} \in [0,1]$.

Naj bo $\widehat{R} = PQ^T$ faktorizacija, kjer $P \in \mathcal{R}^{N\times K}$ in $Q \in \mathcal{R}^{M\times K}$. Velja, da je število značilnosti $K \ll \min\{N, M\}$. Enačbo (2.9) lahko prepišemo v:

$$\mathcal{L}(\widehat{R}) = \sum_{ij} W_{ij}(R_{ij} - P_{ij}Q_{ij}^T)^2 \tag{2.10}$$

Da preprečimo preveliko prileganje podatkom, podobno kot pri algoritmu RI-SMF dodamo člena za regularizacijo:

$$\mathcal{L}(P,Q) = \sum_{ij} W_{ij}((R_{ij} - P_{ij}Q_{ij}^T)^2 + \lambda(\|P_{i.}\|_F^2 + \|Q_{j.}\|_F^2)) \tag{2.11}$$

V enačbi (2.11) $\|.\|_F$ označuje Frobeniusovo normo. Za učinkovito reševanje tovrstne aproksimacije nizkega reda se uporablja pristop izmenjujočih najmanjših kvadratov (*angl. Alternating Least Squares, ALS*) [4, 2]. Z namenom minimiziranja funkcije $\mathcal{L}$ iz enačbe (2.11) izračunamo parcialne odvode za vektorje iz P in Q. Dobimo:

$$\frac{1}{2}\frac{\partial\mathcal{L}(P,Q)}{\partial P_{ik}} = \sum_j W_{ij}(P_iQ_j^T - R_{ij})Q_{jk} + \lambda(\sum_j W_{ij})P_{ik}), \forall 1 \leq i \leq N, 1 \leq k \leq K. \tag{2.12}$$

Iz tega sledi:

$$\frac{1}{2}\frac{\partial\mathcal{L}(P,Q)}{\partial P_{i.}} = \frac{1}{2}\left(\frac{\partial\mathcal{L}(P,Q)}{\partial P_{i1}}, \ldots \frac{\partial\mathcal{L}(P,Q)}{\partial P_{iK}}\right) = P_i(Q^T\widetilde{W}_{i.}Q + \lambda(\sum_j W_{ij})I)) - R_{i.}\widetilde{W}_{i.}Q, \tag{2.13}$$

kjer je $\widetilde{W}_{i.} \in \mathcal{R}^{MxM}$ diagonalna matrika z elementi $W_{i.}$ na diagonali. Če fiksiramo Q in rešimo $\frac{\partial\mathcal{L}(P,Q)}{\partial P_{i.}} = 0$, dobimo:

$$P_{i.} = R_{i.}\widetilde{W}_{i.}Q(Q^T\widetilde{W}_{i.}Q + \lambda(\sum_j W_{ij})I)^{-1} \quad \forall 1 \leq i \leq m. \tag{2.14}$$

Podobno je na drugi strani. Če fiksiramo P, lahko Q izračunamo kot:

$$Q_{j.} = R_{.j}^T\widetilde{W}_{.j}P(P^T\widetilde{W}_{.j}P + \lambda(\sum_i W_{ij})I)^{-1} \quad \forall 1 \leq j \leq n, \tag{2.15}$$

Algoritem wALS deluje torej tako, da izmenično popravlja P oziroma Q z enačbama (2.14) oziroma (2.15) vse do konvergence napake ali omejitve števila iteracij. Izvorna koda algoritma je priložena v dodatku A.

Časovna zahtevnost

Če skušamo oceniti čas delovanja algoritma wALS: P je matrika velikosti $N \times K$, Q pa $M \times K$, pri čemer velja, da je $K \leq \min\{N, M\}$. Torej glede

na enačbi (2.14) in (2.15) vsak korak posodabljanja matrike P ali Q porabi $O(K^2NM)$ časa. Celoten čas delovanja wALS je:

$$O(K^2 n_t NM),$$

kjer je n_t število iteracij.

Sheme obteževanja wALS

Kot smo že omenili v poglavju 2.2.3, je izbira dobre matrike W ključnega pomena za čim boljše delovanje algoritma wALS. Ideja je, da dodelimo utež 1 le, ko smo popolnoma prepričani v dani primer. Drugače povedano, W_{ij} naj bo enak 1, če je $R_{ij} = 1$. Za manjkajoče podatke je zelo verjetno, da so negativni, saj se uporabniki ponavadi zanimajo le za manjši del celotne množice stvari. Zato veliko študij navkljub subjektivnosti pristopa vse manjkajoče primere označuje za negativne (AMAN) [2]. Zaupanje, da so manjkajoči primeri res negativni, ni tako visoko kot za podane primere, ki so gotovo pozitivni. Zato vpeljemo različne sheme uteži. Predstavili bomo tri obteževalne sheme, ki smo jih v nadaljevanju tudi preizkusili.

Prva shema je uniformna (algoritem uniform-wALS v poglavju testiranja 3). Ta obteževalna shema privzame, da imajo vsi manjkajoči primeri enako utež, ne glede na to za katerega uporabnika oziroma stvar gre. Vsem negativnim/manjkajočim primerom se tako dodeli ista utež $wf \in [0, 1]$.

Uporabniško usmerjena obteževalna shema predpostavlja, da uporabnikom, ki že imajo zaznamovanih veliko stvari, ostale stvari z večjo verjetnostjo ne ugajajo. V poglavju testiranja 3 to shemo uporablja algoritem user-wALS.

Tretja stvarno usmerjena shema obteževanja (algoritem item-wALS v poglavju testiranja 3) stvarem, ki imajo malo pozitivnih primerov, predpostavlja večjo verjetnost, da so manjkajoči primeri negativni. Faktor uteži wf, s katerim množimo pri vseh treh shemah, bomo opisali ter testirali v poglavju 3.4.1. Povzetek vseh treh shem je v tabeli 2.1.

Tabela 2.1: Različne obteževalne sheme algoritma wALS.

Shema	$R_{ij} = 1$	$R_{ij} = 0$
uniformna	$W_{ij} = 1$	$W_{ij} = wf$
uporabniško usmerjena	$W_{ij} = 1$	$W_{ij} = wf + (1 - wf)\frac{\sum_j R_{ij}}{\max_t \sum_j R_{tj}}$
stvarno usmerjena	$W_{ij} = 1$	$W_{ij} = wf\frac{(m - \sum_i R_{ij})}{\max_t (m - \sum_i R_{it})}$

2.3 Algoritmi najbližjih sosedov

Brez dvoma so algoritmi najbližjih sosedov (*angl. k-Nearest Neighbours, k-NN*) eni izmed najpopularnejših algoritmov v strojnem učenju. Algoritmi k-NN so enostavni za implementacijo in dosegajo dostojne rezultate na različnih množicah podatkov [5, 6]. V tem poglavju bomo predstavili na uporabnikih in stvareh temelječa algoritma najbližjih sosedov.

2.3.1 Na uporabnikih temelječa metoda *k* najbližjih sosedov

Na uporabnikih temelječ algoritem k najbližjih sosedov skuša posnemati priporočanje od ust do ust [7]. Domneva, ki jo ta tehnika privzame je, da bodo uporabniki s podobnimi preferencami stvari ocenjevali podobno. Manjkajoče ocene za uporabnika lahko napovemo tako, da najprej poiščemo soseščino podobnih uporabnikov (k najbolj podobnih uporabnikov) ter s pomočjo njihovih ocen ocenimo napoved. Za izračun podobnosti med uporabniki se uporabljajo različne metrike podobnosti. V naslednjem poglavju bomo opisali nekatere izmed popularnejših.

Metrike podobnosti med uporabniki

Soseščino (podobnih uporabnikov) se torej definira tako, da najdemo dano število k najbolj podobnih uporabnikov ali pa z vsemi uporabniki, ki imajo podobnost višjo od nekega praga (*angl. threshold*). Za obe tehniki je potrebno definirati mero podobnosti (*angl. similarity measure*), s katero se izračuna podobnost dveh uporabnikov.

Naj bo W matrika uteži, ki vsebuje podobnosti med uporabniki. Če je $W(i,j) = 1$ pomeni, da imamo za uporabnika i in j podano identično preferenčno znanje. W definiramo z enačbo:

$$W(i,j) = \begin{cases} sim(i,j) & ; \quad j \in K \\ 0 & ; \quad j \notin K \end{cases}, \tag{2.16}$$

kjer je K množica uporabnikov iz soseščine uporabnika i. Predlaganih je bilo več različnih metod računanja funkcije podobnosti $sim(i,j)$. Med najbolj priljubljene sodijo Minkowskijeva razdalja (*angl. Minkowski distance*), Pearsonov korelacijski koeficient (*angl. Pearson correlation coefficient*) in kosinusna podobnost (*angl. Cosine similarity*) [7].

Minkowskijeva razdalja je poznana kot standardna metrika za geometrijske probleme [7]. Definirana je z:

$$d_M = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}.$$

Poseben primer Minkowskijeve razdalje je evklidska razdalja, pri kateri je parameter p enak 2.

Pearsonova korelacija je v [8] definirana kot „mera razpona do točke, ko še obstaja linearno razmerje med dvema spremenljivkama“. Pearsonovo korelacijo se da izračunati z enačbo:

$$\rho = \frac{1}{n} \sum_{i=1}^{n} (\frac{X_i - \mu_x}{\sigma_x})(\frac{Y_i - \mu_y}{\sigma_y}).$$

Ko računamo **kosinusno podobnost** med dvema uporabnikoma, obravnavamo uporabnikove ocene kot enodimenzionalen vektor. Podobnost med dvema uporabnikoma izračunamo kot kosinusno podobnost dveh vektorjev ocen. Kosinusno podobnost se izračuna z enačbo:

$$sim(x, y) = cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \cdot ||\vec{y}||}.$$

Čeprav zgoraj omenjene tri metrike spadajo med popularnejše, se za problem z enorazrednimi podatki najpogosteje uporablja **Jaccardova podobnost** (*angl. Jaccard index*) [7]. Jaccardov koeficient izmeri razmerje med številom skupnih atributov vektorjev ocen in skupnim številom atributov. Jaccardova podobnost se izračuna z:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Izračun napovedi uporabnika za določeno stvar

Na izbiro sosedov uporabnikov lahko gledamo kot na fazo učenja k-NN algoritma. Korak napovedi se izvede v testni fazi. Od algoritma se pričakuje napoved za vse testne primere. Najenostavnejši pristop napovedi je, da so ocene vseh sosedov med seboj enakovredne, napoved za testni primer pa je kar povprečje ocen vseh sosedov:

$$\rho(u, N_u, i) = \frac{\sum_{n \in N_u} \delta(n, i)}{k},$$

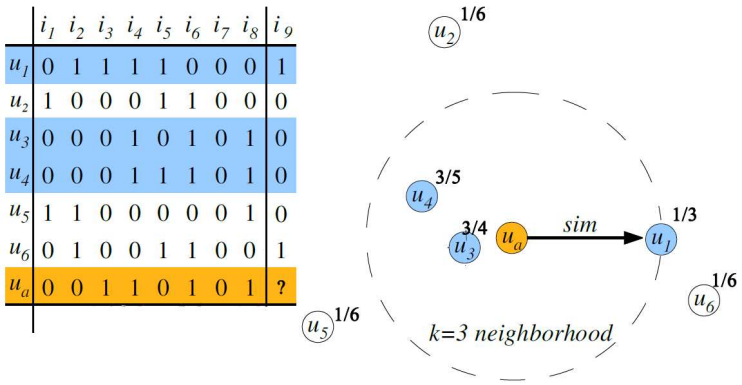
kjer je N_u seznam sosedov uporabnika u. $\delta(n, i)$ je enak 1, če je uporabnik u stvar i zaznamoval ter 0 če ni.

Drug pogost način napovedovanja z algoritmom k najbližjih sosedov vsakemu uporabniku iz soseščine aktivnega uporabnika dodeli utež, ki določa njegov vpliv pri učenju [6]. Ponavadi je to kar podobnost med aktivnim uporabnikom ter sosedom. Pri tem načinu končna enačba napovedi aktivnega uporabnika u za stvar i izgleda sledeče:

$$\rho(u, N_u, i) = \frac{\sum_{n \in N_u} \delta(n, i) W(u, n)}{\sum_{n \in N_u} W(u, n)}, \tag{2.17}$$

kjer je δ matrika učnih primerov, W pa je kvadratna matrika podobnosti med uporabniki, velikosti $N \times N$. Enačba (2.17) vedno vrne napoved med 0 in 1. Če želimo cela števila, uporabimo zaokroževanje.

Da se enačba (2.17) lahko uporabi, je potrebno vse manjkajoče primere označiti za negativne - uporaba tehnike AMAU. Čeprav je ta predpostavka nekoliko subjektivna, saj se nekateri manjkajoči primeri zaznamkov stvari iz učne množice napačno označijo za negativne primere, bo napaka majhna, saj se večina uporabnikov ponavadi zanima le za majhno množico stvari. To je tudi glavni vzrok za redkost podatkov (*angl. sparse data*) [9]. Celotna izvorna koda na uporabnikih temelječe metode k najbližjih sosedov je priložena v dodatku B.



Slika 2.1: Primer na uporabnikih temelječe metode k najbližjih sosedov. V tem primeru je $k = 3$.

Primer priporočanja, ko imamo na voljo enorazredne podatke in uporabljamo na uporabnikih temelječo metodo k najbližjih sosedov, prikazuje slika 2.1. Na levi strani je prikazana matrika R s 6 uporabniki in 9 stvarmi. Za manjkajoče

primere je bila uporabljena tehnika AMAU. Aktivni uporabnik u_a, za katerega želimo napovedati oceno za stvar i_9, je prikazan na dnu matrike. Da najdemo k najbližjih uporabnikov aktivnemu uporabniku, izračunamo podobnosti aktivnega uporabnika z vsemi ostalimi na prvih osmih stvareh. Uporabimo Jaccardovo metriko podobnosti. Desna stran slike 2.1 predstavlja dvodimenzionalno predstavitev podobnosti z aktivnim uporabnikom v središču. Vrednosti, ki so na sliki zapisane desno zgoraj od vsakega izmed uporabnikov, predstavljajo Jaccardovo podobnost z aktivnim uporabnikom. Izbrani trije najbližji sosedi so označeni tudi v matriki R. Da bi napovedali uporabnikovo u_a oceno za stvar i_9, s pomočjo ocen treh najbližjih sosedov z enačbo (2.17) izračunamo napoved:

$$\frac{1 \cdot \frac{1}{3} + 0 \cdot \frac{3}{4} + 0 \cdot \frac{3}{5}}{\frac{1}{3} + \frac{3}{4} + \frac{3}{5}} = 0{,}198,$$

kar je bližje 0 kot 1. Algoritem torej v tem primeru vrne 0.

Časovna zahtevnost

Časovno najbolj potraten del algoritma na uporabnikih temelječe metode najbližjih sosedov je izračun k najbližjih sosedov za vsakega od uporabnikov. Izračun podobnosti med dvema uporabnikoma se izvrši v $O(M)$ (v realnosti je bližje $O(1)$, saj uporabniki ponavadi zaznamujejo majhno število stvari), medtem ko za izračun napovedi z enačbo (2.17) algoritem potrebuje $O(kM)$ časa. Časovna zahtevnost napovedovanja za uporabnika je tako $O(NM) + O(kM)$, kjer je $k \ll N$. Izračun za vse uporabnike se torej izvrši v:

$$O(N^2 M).$$

Glavni slabosti na uporabnikih temelječega algoritma k najbližjih sosedov sta potreba po hranjenju celotne podatkovne baze uporabnikov v pomnilniku ter drago računanje podobnosti med aktivnim uporabnikom ter vsemi ostalimi [7].

2.3.2 Na stvareh temelječa metoda *k* najbližjih sosedov

Na stvareh temelječa metoda najbližjih sosedov je pristop CF, ki deluje tako, da na učni množici zgradi model, katerega kasneje uporablja za napovedovanje [6, 7]. Priporočanje temelji na razmerjih med stvarmi, ki jih izpelje iz matrike ocen. Logika na kateri ta pristop temelji je, da bodo uporabnikom, ki preferirajo določene stvari, všeč tudi njim podobne stvari. Korak gradnje modela vsebuje računanje matrike podobnosti (*angl. similarity matrix*), ki vsebuje vse podobnosti med različnimi stvarmi, dobljene z izbrano

mero podobnosti [10]. Kot smo omenili, se pri metodi najbližjih sosedov, ki temelji na uporabnikih, najpogosteje uporabljata meri Pearsonova korelacija ter kosinusne podobnosti, za enorazredne podatke pa Jaccardov koeficient. Za metodo k najbližjih sosedov, ki temelji na stvareh, sta leta 2004 Deschpande in Karypis predlagala pogojno na verjetnosti temelječo podobnost (*angl. Conditional probability-based similarity*) [11]. Ta je definirana z enačbo:

$$sim_{conditional}(i_1, i_2) = \frac{\text{Freq}(i_1 i_2)}{\text{Freq}(i_1)} = \hat{P}(i_2|i_1), \tag{2.18}$$

kjer i_1, i_2 pripadata množici stvari, Freq$(\cdot)$ pa je število uporabnikov, ki so zaznamovali dano stvar. Ta mera podobnosti je v bistvu približek pogojne verjetnosti, da je uporabnik, ki je zaznamoval stvar i_1, zaznamoval tudi stvar i_2. Slaba stran te mere je njena občutljivost na primere z nizko frekvenco i_1, saj enačba (2.18) zanje lahko vrača visoke podobnosti. Ta problem smo delno odpravili s predhodno obdelavo podatkov, več o tem v poglavju 3.1.

Vse parne podobnosti se shranjujejo v matriki podobnosti S velikosti $N \times N$, ki je normalizirana tako, da je seštevek vsake vrstice enak 1. Da se zmanjša velikost modela na $N \times k$, kjer je $k \ll N$, se lahko za vsako stvar shrani le seznam k najbolj podobnih drugih stvari skupaj z vrednostmi podobnosti. To močno izboljša prostorsko zahtevnost.

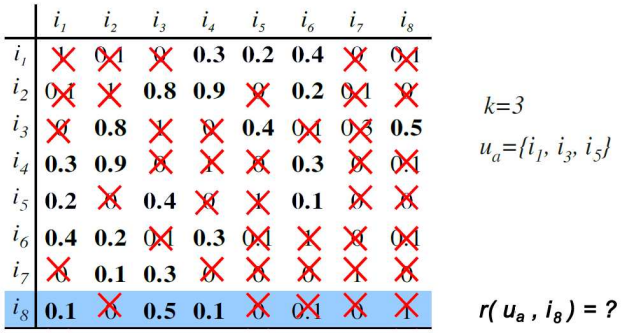
Izračun napovedi uporabnika za določeno stvar

Za priporočilo uporabnika u za stvar i, so tako potrebni trije koraki:

- identifikacija sosedov stvari i (označimo jih z N_i),
- označitev vseh manjkajočih primerov za negativne (tehnika AMAN),
- izračun ocene uporabnika u za i kot obteženo povprečje ocen sosednih stvari z enačbo:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_i} sim(i,j) r_{uj}}{\sum_{j \in N_i} sim(i,j)}. \tag{2.19}$$

Primer z 8 stvarmi ($M = 8$) na katerih uporabimo priporočanje z metodo najbližjih sosedov, ki temelji na stvareh, prikazuje slika 2.2. Za podano normalizirano matriko podobnosti med stvarmi S se v tem primeru shrani le $k = 3$ vnosov na vrstico, prečrtane vrednosti se zavržejo. Želimo napovedati zaznamek uporabnika u_a za stvar i_8, pri čemer je uporabnik predhodno zaznamoval stvari i_1, i_3 in i_5. Kot razberemo s slike 2.2 so i_1, i_3 in i_4 trije najbližji sosedi stvari i_8. Napoved uporabnika izračunamo z enačbo (2.19):



Slika 2.2: Primer matrike podobnosti med stvarmi pri na stvareh temelječi metodi najbližjih sosedov, kjer je $k = 3$.

$$\hat{r}_{u_a i8} = \frac{1 \cdot 0{,}1 + 1 \cdot 0{,}5 + 0 \cdot 0{,}1}{0{,}1 + 0{,}5 + 0{,}1} = 0{,}857,$$

kar je bližje 1 kot 0. Algoritem torej v tem primeru vrne 1.

Časovna zahtevnost

Časovno najbolj potraten del na stvareh temelječe metode k najbližjih sosedov je ravno izračun k najbolj podobnih sosedov za vsako izmed stvari v koraku učenja. Za izračun podobnosti se uporabljajo različne mere podobnosti. Kot smo omenili, je med pogostejšimi izračun s pogojno na verjetnosti temelječo podobnostjo iz enačbe (2.18). Na sliki 2.3 je prikazana psevdokoda izračuna soseščine vsake izmed stvari.

```
for i1 in items:
    for i2 in items:
        if i1 != i2:
            calculate_similarity_between(i1,i2)
    save_top_l_items_for(i1)
```

Slika 2.3: Izračun najbližjih sosedov vsake izmed stvari

Časovna zahtevnost izračuna je enaka $O(N^2 m)$, kjer je $m = min\{L1, L2\}$, $L1$ oziroma $L2$ pa sta število uporabnikov, ki je zaznamovalo stvar $i1$ oziroma $i2$. V najslabšem primeru je časovna zahtevnost enaka $O(N^2 M)$, v realnosti

pa je bližje $O(N^2)$, saj ponavadi posamezno stvar zaznamuje majhno število uporabnikov v primerjavi s številom vseh uporabnikov.

Izračun napovedi uporabnika za določeno stvar je hiter, saj je odvisen le od števila stvari, ki jih je uporabnik že zaznamoval. Algoritem na stvareh temelječe metode k najbližjih sosedov je zelo efektiven, saj je model (skrčena matrika podobnosti) relativno majhen ($N \times k$) in se ga da izračunati vnaprej. Za metodo k najbližjih sosedov, ki temelji na stvareh, je značilno, da dosega nekoliko slabše rezultate v primerjavi z metodo k najbližjih sosedov, ki temelji na uporabnikih, in modeli višjih redov [7]. Kljub temu pa se vseeno uporablja tudi v večjih priporočilnih sistemih (npr. Amazon.com) [12]. Izvorna koda na stvareh temelječe metode k najbližjih sosedov se nahaja v dodatku C.

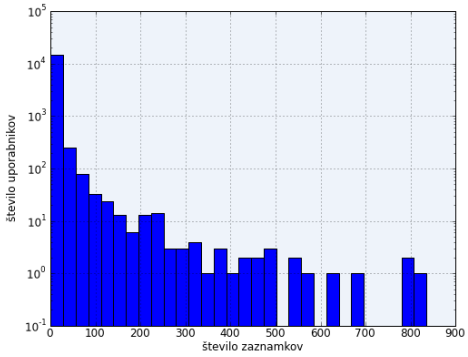
Poglavje 3

Vrednotenje priporočilnih tehnik na enorazrednem problemu

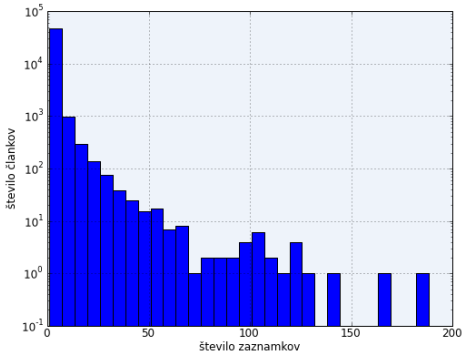
V prejšnjih poglavjih smo predstavili problem enorazrednih podatkov ter opisali nekaj tehnik in algoritmov spoprijemanja z njimi. Kot smo omenili v poglavju 1.2, je enorazredni problem težje rešljiv kot problemi z večjo domeno ocen. V naslednjem poglavju bomo predstavili uporabljeno podatkovno bazo enorazrednih podatkov ter opisali meri, kateri smo uporabili za ocenjevanje algoritmov. V poglavju 3.4 bomo najprej primerjali, kako se obnese algoritem RISMF s tehnikama AMAU in AMAN za različna števila značilnosti. Preverili smo tudi, kako se obnesejo algoritmi wALS, opazovali smo odvisnost rezultatov od različnih faktorjev uteži wf ter števila značilnosti k. Prav tako smo testirali, kako se obneseta obe v poglavju 2.3 opisani metodi najbližjih sosedov, preizkusili smo njuno odvisnost od števila sosedov k. Na koncu poglavja smo predstavili najboljše rezultate vseh implementiranih algoritmov ter v poglavju 4 podali sklepne ugotovitve.

3.1 Podatki

CiteULike.org je spletna stran, ki omogoča hranjenje, organiziranje in deljenje strokovnih člankov. Na njihovi spletni strani so na voljo dnevni posnetki njihovih baz. Z njihove spletne strani lahko prenesemo podatke, ki povedo katere članke je določen uporabnik zaznamoval. To bazo smo uporabili za testiranje v poglavju 2 opisanih priporočilnih sistemov.

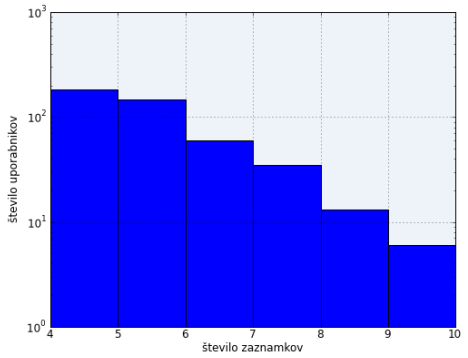


Slika 3.1: Število uporabnikov z določenim številom zaznamkov

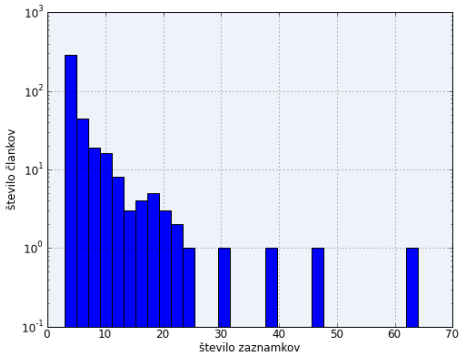


Slika 3.2: Število člankov z določenim številom zaznamkov

Baza zaznamkov CiteULike.org vsebuje 101.850 zaznamkov (parov uporabnik-članek), pri čemer je 15.178 unikatnih uporabnikov in 49.230 unikatnih člankov. Na sliki 3.1 je prikazana statistika, koliko uporabnikov je zaznamovalo določeno število člankov, medtem ko slika 3.2 prikazuje število zaznamkov za članke.



Slika 3.3: Število uporabnikov z določenim številom zaznamkov po preobdelavi podatkov



Slika 3.4: Število člankov z določenim številom zaznamkov po preobdelavi podatkov

Te podatke smo najprej obdelali, saj nam uporabniki, ki so naredili malo zaznamkov in članki, ki so bili malokrat zaznamovani, ne dajo veliko veljave pri bodočem odločanju. Zato smo izmenično odstranjevali uporabnike, ki so zaznamovali manj kot 4 ali pa več kot 10 člankov, ter članke, kateri so bili za-

znamovani s strani manj kot treh uporabnikov, dokler ni število obojih skonvergiralo do neke rešitve. Ostalo je 2236 zaznamkov, razporejenih med 445 uporabnikov ter 394 člankov. Ker je velika večina uporabnikov zaznamovala majhno število člankov, prav tako pa je bila večina člankov zaznamovana s strani malo uporabnikov, smo po preobdelavi dobili mnogo manjšo množico podatkov. Statistika števila zaznamkov za uporabnike in članke po obdelavi je prikazana na sliki 3.3 in sliki 3.4.

3.2 Implementirani postopki učenja

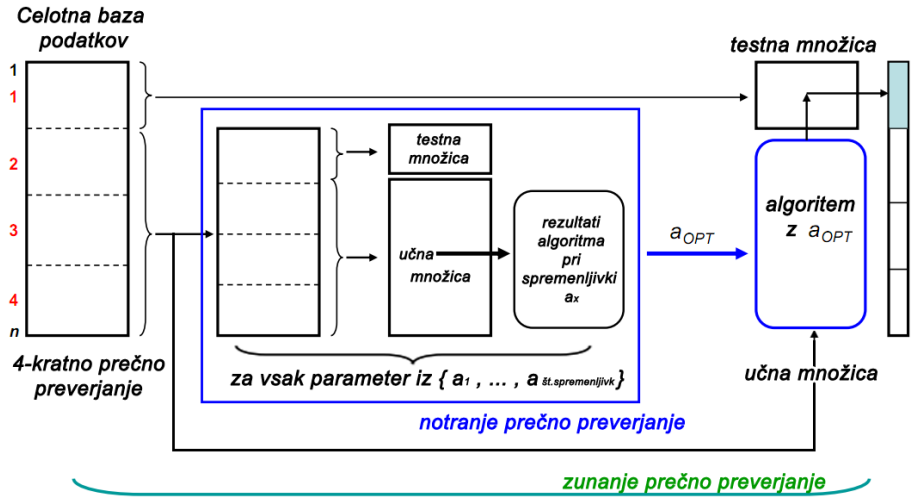
V nadaljevanju bomo za algoritme uporabljali določene oznake. V tabeli 3.1 so zapisani vsi algoritmi, testirani v tem poglavju, in njihove kratice. Izvorne kode vseh algoritmov so priložene v dodatku.

Tabela 3.1: Tabela implementiranih algoritmov in njihovih oznak.

Oznaka	Algoritem
RISMF, AMAU	Regularizirana inkrementalna hkratna matrična faktorizacija z neobravnavanjem manjkajočih primerov
RISMF, AMAN	Regularizirana inkrementalna hkratna matrična faktorizacija z označitvijo manjkajočih primerov za negativne
popularity	Metoda največje popularnosti
item-wALS	Metoda obteženih izmenjujočih najmanjših kvadratov s stvarno usmerjeno shemo obteževanja
user-wALS	Metoda obteženih izmenjujočih najmanjših kvadratov z uporabniško usmerjeno shemo obteževanja
uniform-wALS	Metoda obteženih izmenjujočih najmanjših kvadratov z uniformno shemo obteževanja
user-kNN	Na uporabnikih temelječa metoda k najbližjih sosedov
item-kNN	Na stvareh temelječa metoda k najbližjih sosedov

V strojnem učenju in področju odkrivanja znanj iz podatkov se za ocenjevanje delovanja algoritmov pogosto uporablja prečno preverjanje (*angl. Cross-validation*). Celotno podatkovno bazo (t.j. vse zaznamke) smo razdelili v 5 enakovrednih množic, od katerih smo vsakokrat eno uporabili za testno, ostale pa za učno množico. Učna in testna množica sta tako v vseh petih korakih bili v razmerju 80:20. Učna množica je sestavljena iz 80% znanih pozitivnih primerov, vse ostale pa obravnava kot neznane/negativne. Testna množica je vsebovala ostalih 20% pozitivnih primerov ter vse negativne/neznane primere. Torej so vsi znani pozitivni primeri v učni množici izvzeti iz testne množice.

Intuicija dobrega delovanja metode je, ko ima metoda visoko verjetnost, da rangira znane pozitivne primere preko neznanih, ki so ponavadi negativni.



Slika 3.5: Iskanje optimalne vrednosti parametra algoritma ter ocenjevanje uspešnosti

Za določitev parametrov pri vseh algoritmih je bilo potrebno na učni množici izvesti še iskanje parametra, ki naj bi na testni množici dosegel čim boljši rezultati. Parameter je lahko bodisi število značilnosti bodisi faktor uteži wf. Slika 3.5 prikazuje celoten postopek preverjanja uspešnosti algoritma. Iskanje optimalne vrednosti parametra se izvede tako, da se na učni množici za vsako podano vrednost izvede 5-kratno (razmerje med učno in testno množico je 80:20) prečno preverjanje. Najboljša vrednost parametra je tista, ki pri notranjem prečnem preverjanju na učni množici, doseže najvišjo mero ocenjevanja. Optimalno vrednost algoritem nato uporabi pri napovedovanju na testni množici.

3.3 Ocenjevanje uspešnosti

Delovanje algoritmov smo ocenili z merama srednje povprečne natančnosti in mero pozitivne napovedne vrednosti.

Mera srednje povprečne natančnosti (*angl. Mean Average Precision, MAP*) se ponavadi uporablja za ocenjevanje rangiranja dokumentov za množico poizvedb [9, 2]. Pri meri MAP se izračuna srednjo vrednost povprečnih

natančnosti vseh uporabnikov. Povprečno natančnost (*angl. Average Precision, AP*) za uporabnika u se izračuna z:

$$\mathrm{AP}_u = \frac{\sum_{i=1}^{N} \mathrm{prec}(i) \times \mathrm{zaznamek(i)}}{\text{število zaznamkov}}, \tag{3.1}$$

kjer je i pozicija v rangiranem seznamu vseh napovedanih zaznamkov za uporabnika u, N je število primerov označenih za pozitivne s strani algoritma, zaznamek(i) je binarni indikator, ki vrne 1, kadar je članek dejansko preferiran s strani uporabnika u, ter 0 sicer. Prec(i) je natančnost pravilno ocenjenih preferiranih člankov seznama prvih i elementov iz seznama napovedanih zaznamkov in je definirana z:

$$\mathrm{prec}(i) = \frac{\text{število pravilno napovedanih zaznamkov}}{\text{število napovedanih zaznamkov}}.$$

Mera MAP poleg same natančnosti upošteva tudi dejansko število pozitivnih primerov, za kar poskrbi imenovalec desne strani enačbe (3.1). Algoritmi faktorizacije matrik pri testiranju za vsakega od uporabnikov izberejo vse članke iz testne množice ter vse neznane primere. Za vsakega od njih s skalarnim produktom vrstice iz P ter stolpcem iz Q izračunajo oceno za r_{ij}. Napovedi se za enorazredni problem gibljejo med 0 do okoli 1. Pri algoritmih najbližjih sosedov se za izračun napovedi uporabita enačbi (2.17) in (2.19). V obeh primerih vrednosti nad 0,5 pomenijo, da jih je algoritem označil za pozitivne, manjše vrednosti pa negativne. Prednost mere MAP je tudi v upoštevanju ranga članka v urejenem seznamu ocen za uporabnika, katerega povprečno natančnost računamo. Primer: imamo algoritma, ki sta za pozitivne primere napovedovala vrednosti okrog 0,8. Prvi algoritem je nepravilno ocenjenim negativnim primerom, dajal vrednosti okrog 1, drugi pa je dajal vrednosti malo nad 0,5. Z mero MAP bo prvi algoritem dobil slabšo oceno kot drugi.

Druga mera uporabljena za ocenjevanje je **pozitivna napovedna vrednost** (*angl. Positive Predictive Value, PPV*) in je definirana z:

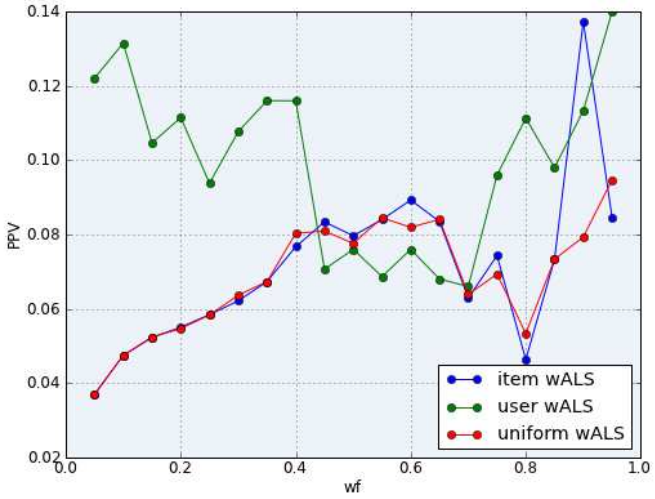
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$

kjer je TP število pravilno napovedanih pozitivnih primerov (*angl. True Positive, TP*) in FP število napačno napovedanih pozitivnih primerov (*angl. False Positive, FP*). Mera PPV ocenjuje natančnost napovedovanja pozitivnih primerov, torej koliko od primerov označenih za pozitivne je resnično pozitivnih.

3.4 Rezultati

3.4.1 Odvisnost algoritmov wALS od velikosti uteži

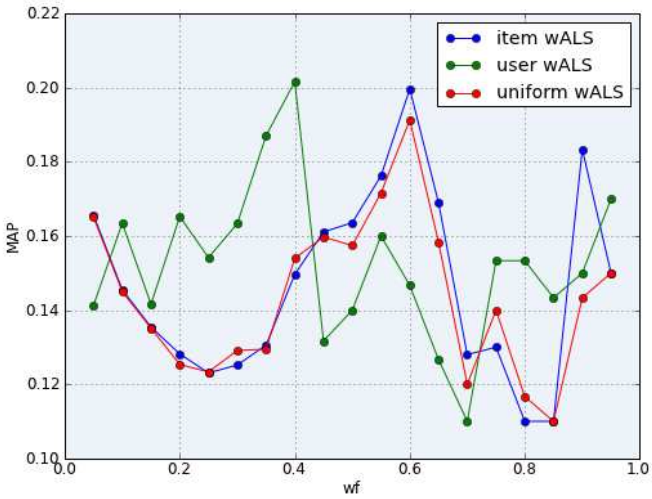
Ker ni določenega postopka, kako izračunati, kolikšno utež je pri algoritmih wALS potrebno dati negativnim/neznanim primerom, smo najprej preverili odvisnost algoritmov wALS od faktorja uteži. Na vsaki od učnih množic zunanjega prečnega izvajanja smo za različne vrednosti faktorja uteži *wf* izvedli notranje prečno preverjanje, tako kot prikazuje slika 3.5. Testirali smo vse tri obteževalne sheme opisane v poglavju 2.1.



Slika 3.6: Mera PPV za različne faktorje uteži neznanim primerom, K=20

Slika 3.6 prikazuje, kako se obnaša natančnost algoritmov iz družine wALS v odvisnosti od faktorja uteži *wf*. Uporabljena mera je PPZ. Vsaka točka na grafu predstavlja povprečje rezultatov vseh izvajanj notranjih prečnih preverjanj za določen faktor uteži. Število značilnosti je bilo nastavljeno na vrednost 20. Opazimo, da algoritma item-wALS ter uniform-wALS dosegata podobne rezultate. Mera PPZ pri obeh narašča, ko narašča velikost uteži. Pri obeh raste vse do vrednosti približno 0,65, ko sledi rahel padec. Relativno konstanto naraščanje lahko pripišemo dejstvu, da mera PPZ ne upošteva števila vseh pozitivnih primerov, ampak le natančnost primerov katere algoritem vrne. Večja kot je utež, večji poudarek algoritem daje negativnim primerom in s tem posledično tudi vse manj primerov označuje za pozitivne. Če bi utež postala prevelika (večja kot 1), bi se negativnim primerom dajalo prevelik pomen pri

učenju. Algoritmi bi takrat vse primere začeli označevati za negativne. Algoritem user-wALS zaradi same zasnove uporabniško usmerjene sheme obteževanja dosega nekoliko drugačne rezultate. Ta pri obteževalnem faktorju $wf = x$ dosega vrednosti iz intervala $[x, 1]$. Utež za negativne primere je pri tej shemi omejena navzdol, medtem ko je pri na stvareh temelječi omejena navzgor ter konstantna pri uniformni. Kot je vidno tudi na sliki 3.6, algoritem user-wALS kot posledica tega tudi pri nizkih faktorjih uteži doseže dobre rezultate. Algoritem se takrat uči predvsem na uporabnikih, ki so zaznamovali veliko stvari.



Slika 3.7: Mera MAP za različne faktorje uteži neznanim primerom, K=20

Rezultati, ki smo jih dobili pri opazovanju mere MAP v odvisnosti od faktorja uteži (za neznane primere), so pri vseh treh shemah algoritma wALS podobni. Rezultati so predstavljeni na sliki 3.7. Tudi tu vsaka točka na grafu predstavlja povprečje rezultatov vseh izvajanj notranjih prečnih preverjanj za določen faktor uteži, medtem ko je bilo število značilnosti nastavljeno na 20. Kot smo že omenili, je prednost mere MAP v upoštevanju števila napovedanih pozitivnih primerov. Največja povprečna vrednost mere MAP je pri item-wALS ter uniform-wALS dosežena, ko je faktor enak 0,6. User-wALS doseže vrh nekoliko prej in sicer pri vrednosti 0,4. Item-wALS in user-wALS v povprečju dosegata nekoliko boljše rezultate v primerjavi z uniform-wALS.

Rezultati zunanjega prečnega preverjanja (glej sliko 3.5) so zapisani v tabeli 3.2. V prvem stolpcu so navedena imena algoritmov, v drugem in tretjem pa vrednosti spremenljivk števila značilnosti in faktorjev uteži. Faktor uteži za

posamezen algoritem je zapisan kot povprečje najboljših faktorjev vsakega izmed izvajanj zunanjega prečnega preverjanja ter največje odstopanje od njega. V zadnjih dveh stolpcih so podane povprečne vrednosti mer PPV in MAP vseh izvajanj zunanjega prečnega preverjanja.

Tabela 3.2: Tabela rezultatov algoritmov wALS pri preverjanju odvisnosti od faktorja uteži wf.

Algoritem	k	wf	PPV	MAP
item wALS	20	0.6±0.55	0.2135	0.3449
user wALS	20	0.44±0.34	0.222	0.2447
uniform wALS	20	0.61±0.56	0.2252	0.3922

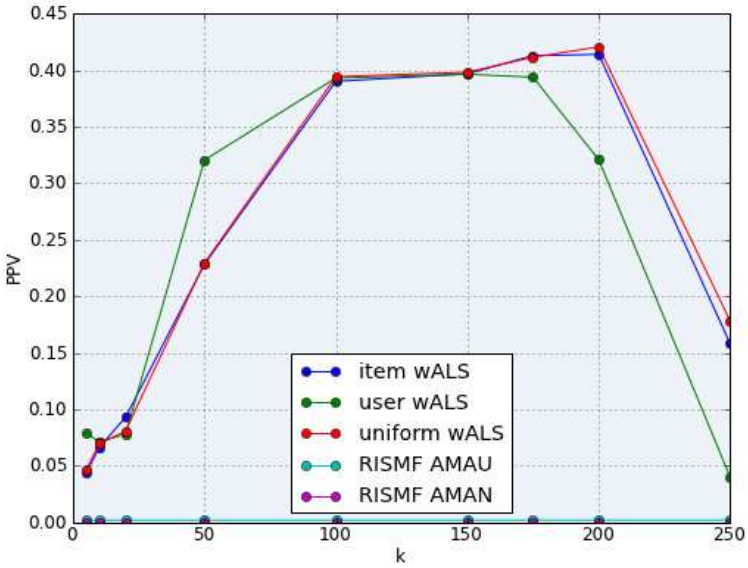
Izmed treh preizkušenih algoritmov wALS z različnimi shemami obteževanja najboljše rezultate mere MAP pri 20 značilnostih doseže algoritem z uniformno shemo obteževanja, najslabše pa uporabniško usmerjena obteževalna shema. Prilagodljivi shemi obteževanja torej nista izboljšali rezultatov. Natančnosti algoritmov (mera PPV) se med seboj razlikujejo minimalno.

3.4.2 Odvisnosti algoritmov wALS in SVD od števila značilnosti

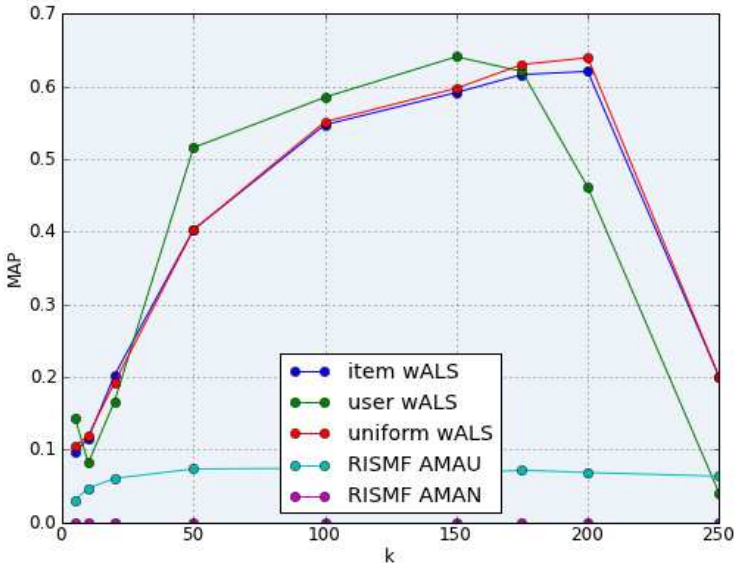
Sliki 3.8 in 3.9 prikazujeta rezultate mer PPV in MAP vseh treh shem algoritma wALS ter algoritmov RISMF s tehniko AMAU in AMAN. Točke na grafu predstavljajo povprečje rezultatov vseh izvajanj notranjih prečnih preverjanj za določeno število značilnosti. Za testiranje algoritmov wALS je bil uporabljen faktor uteži $wf = 0{,}6$.

Kot smo omenili v poglavju 2.1, tudi testi potrjujejo, da se algoritem RISMF, kadar uporablja tehniko AMAU (brez obravnavanja manjkajočih primerov), izkaže za neuporabnega pri reševanju problema z enorazrednimi podatki. Kot je značilno za algoritme, ki za učenje uporabljajo le pozitivne primere, algoritem hitro najde trivialno rešitev označitve vseh primerov testne množice za pozitivne. To potrjujeta tudi grafa 3.8 in 3.9, pokaže se le rahla anomalija: algoritem RISMF z AMAU pri nižjem številu značilnosti, nekaterih primerov ne označi za pozitivne, kar bi lahko bila posledica premajhnega števila iteracij.

Algoritem RISMF s tehniko AMAN na naši množici podatkov prav tako ne doseže dobrih rezultatov. Izkaže se, da zaradi svoje zasnove - enakovrednega obravnavanja pozitivnih in negativnih primerov - ter mnogo večjega števila negativnih v primerjavi s pozitivnim primeri v učni množici vse primere testne



Slika 3.8: Mera PPV v odvisnosti od števila značilnosti za algoritme wALS in RISMF.



Slika 3.9: Mera MAP v odvisnosti od števila značilnosti za algoritme wALS in RISMF

množice označi za negativne pri vseh vrednostih števila uteži. Mera MAP in PPV tako dosegata vrednosti 0.

Če med seboj primerjamo rezultate različnih shem algoritma wALS, opazimo, da item-wALS ter uniform-wALS dosegata zelo podobne rezultate, ki pa so vseeno malenkost v prid uniform-wALS. S pomočjo slike 3.8 smo ugotovili, da algoritem wALS z uporabniško usmerjeno shemo obteževanja doseže svoj najboljši rezultat pri manjšem številu značilnosti, hkrati pa nekoliko slabšo natančnost napovedanih pozitivnih primerov (mera PPV) v primerjavi z ostalima shemama. Algoritem user-wALS je manj senzitiven, saj povprečno označuje nekoliko več primerov za pozitivne kot uniform-wALS oziroma item-wALS. To se kaže tudi na meri MAP (slika 3.9), ki kot smo že omenili, upošteva tudi število napovedanih pozitivnih primerov. Za mero MAP tako v povprečju najboljši rezultat doseže algoritem user-wALS.

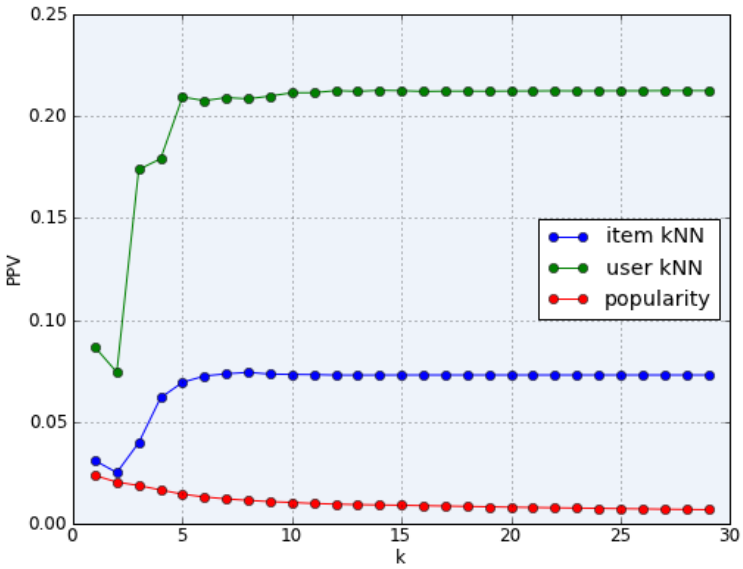
3.4.3 Odvisnost mer od velikosti množic sosedov za algoritme najbližjih sosedov

Sliki 3.10 in 3.11 prikazujeta, kako se meri PPV in MAP spreminjata, ko se spreminja velikost soseščine k pri algoritmih najbližjih sosedov. Vsaka točka na grafih predstavlja povprečje rezultatov vseh izvajanj notranjih prečnih preverjanj za določeno število sosedov. Testirali smo sledeče algoritme: item-kNN, user-kNN ter algoritem popularnosti.

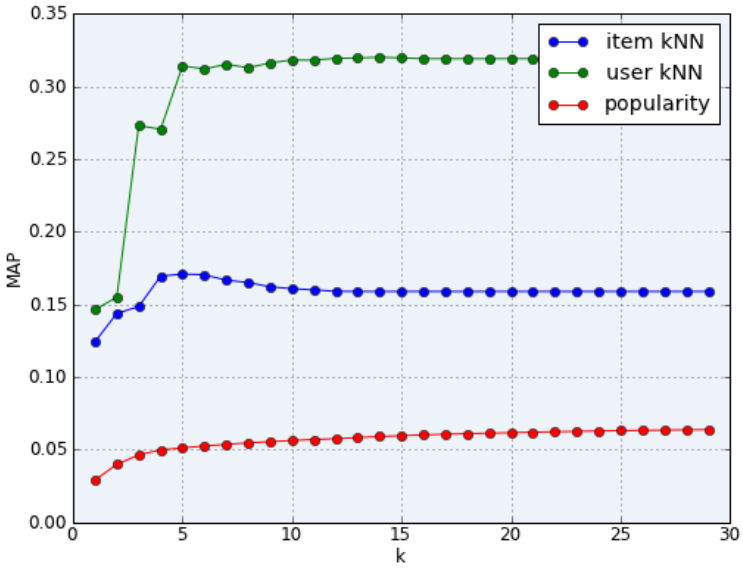
Algoritem popularnosti predstavlja najosnovnejšo napovedovalno tehniko. Deluje tako, da med vsemi stvarmi iz učne množice poišče podano število k največkrat zaznamovanih stvari, katere nato priporoči uporabnikom. Ponavadi se jo uporablja le za primerjanje uspešnosti drugih algoritmov. Uporabna je predvsem za nove uporabnike, ki še niso zaznamovali nobene stvari. Za ta algoritem je značilno, da vsem uporabnikov priporoča iste stvari.

Kot smo omenili že v prejšnjih poglavjih, predstavlja k pri na uporabnikih temelječi metodi najbližjih sosedov število najbolj podobnih uporabnikov aktivnemu uporabniku, za katerega algoritem napoveduje oceno za določno stvar. Podobno pri na stvareh temelječi metodi najbližjih sosedov (algoritem item-kNN) k predstavlja število najbolj podobnih stvari. Pri algoritmu popularnosti je k število najbolj popularnih stvari, katere se vse priporočijo aktivnemu uporabniku.

Dobljeni rezultati za meri MAP kot PPZ kažejo, da na uporabnikih temelječa metoda k najbližjih sosedov na naših podatkih bolje napoveduje od na stvareh temelječe metodi najbližjih sosedov. Pri obeh algoritmih opazimo, da z naraščanjem števila sosedov naraščata tudi obe ocenjevalni meri. Vrednost



Slika 3.10: Mera PPV v odvisnosti od števila najbližjih sosedov k



Slika 3.11: Mera MAP v odvisnosti od števila najbližjih sosedov k

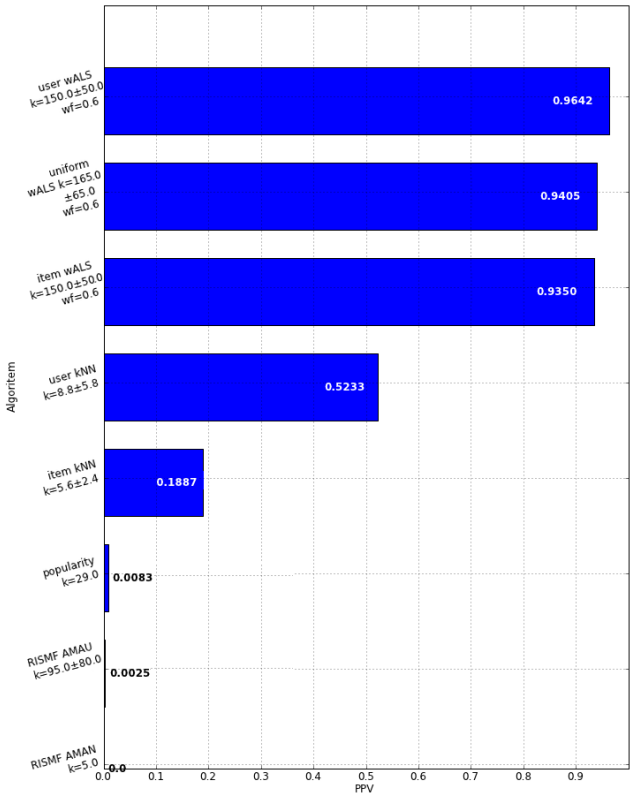
MAP skonvergira k 0,32 za user-kNN oziroma k 0,16 za item-kNN, vrednost PPV pa k 0,25 oziroma 0,06. Enačbi (2.17) in (2.19) poskrbita, da se kakovost napovedi navkljub vedno večji množici sosedov ne slabša. Vedno večja množica sosedov namreč pomeni učenje na vedno manj podobnih uporabnikih oziroma stvareh. Posledica upoštevanja podobnosti v števcih teh dveh enačb je, da se manj podobne primere manj upošteva pri končnih napovedih, zato se ne zgodi, da bi meri PPZ oziroma MAP začeli močno padati, ko se velikost množice sosedov veča proti številu vseh uporabnikov pri user-kNN in proti številu vseh stvari pri item-kNN. Algoritem popularnosti se je pričakovano izkazal najslabše, saj gre za najenostavnejši algoritem med vsemi testiranimi.

3.4.4 Skupni rezultati

Na sliki 3.13 ter 3.12 so grafično prikazani najboljši rezultati vsakega od algoritmov, testiranih v prejšnjih poglavjih. Kot opisuje slika 3.5, algoritmi na vsaki učni množici zunanjega prečnega izvajanja poiščejo število značilnosti oziroma število najbližjih sosedov, pri katerem naj bi na testni množici dosegli najboljše rezultate. Ob imenih algoritmov je zapisano povprečje najboljših spremenljivk vsakega izmed izvajanj zunanjega prečnega preverjanja ter največje odstopanje od povprečja. Rezultati so povprečne vrednosti mer PPV in MAP vseh izvajanj zunanjega prečnega preverjanja.

Od vseh algoritmov so najboljše rezultate dosegli algoritmi wALS, ki so se izkazali presenetljivo dobro za reševanje enorazrednega problema. Med tremi opisanimi shemami obteževanja je algoritem user-wALS dosegel najvišjo natančnost napovedovanja pozitivnih primerov z 96%, medtem ko sta item-wALS in uniform-wALS dosegla rezultat okoli 94%. Pri meri MAP, ki upošteva tudi število napovedanih primerov, se je najbolje izkazal algoritem z uniformno shemo obteževanja, kar pomeni da dinamični shemi obteževanja (uporabniško ter stvarno usmerjena shema) nista prinesli izboljšanja napovedovanja. Od vseh treh obteževalnih shem se je algoritem s stvarno usmerjeno shemo obteževanja obnesel najslabše.

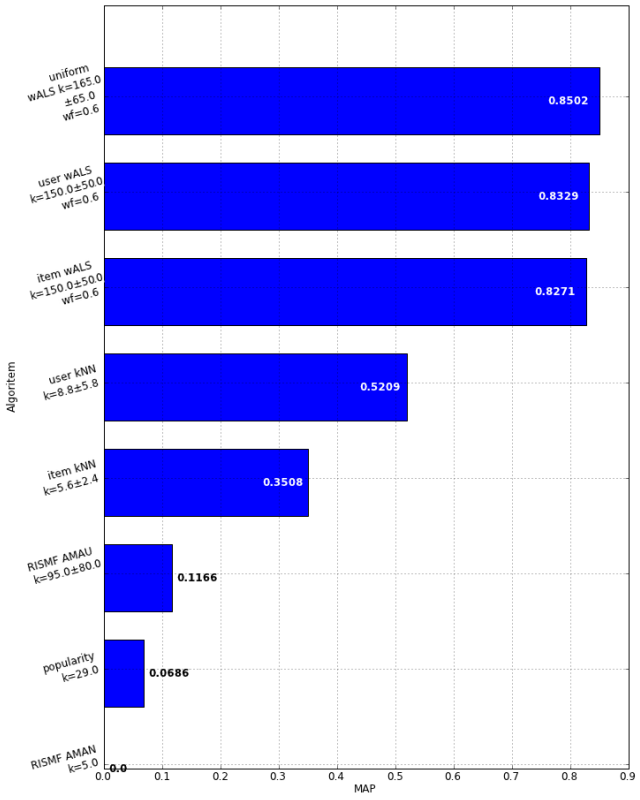
Po natančnosti ocenjevanja sledita algoritma k najbližjih sosedov. Kot smo že omenili je na uporabnikih temelječ algoritem k najbližjih sosedov dosegel boljše rezultate kot na stvareh temelječ algoritem najbližjih sosedov. Prednost slednjega je, da se novi uporabniki dodajajo večkrat kot nove stvari, zato je potrebno manjkrat osveževati matriko podobnosti. To matriko se pri tem algoritmu, kot smo že omenili, lahko izračuna vnaprej. Mera MAP za obe metodi najbližjih sosedov dosega višje vrednosti kot v drugih primerljivih virih [9], kar je posledica odstranitve vseh primerov z malo oziroma veliko zaznamki iz



Slika 3.12: Najboljši rezultati vseh testiranih algortimov za mero MAP

množice primerov.

Algoritma RISMF AMAN in AMAU sta na testu dosegla najslabša rezultata. Prvi ni nobenemu uporabniku priporočil ničesar, drugi pa je vse primere označil za pozitivne. Pri meri MAP je dosegel boljše rezultate le od algoritma popularnosti, ki je dosegel najboljši rezultat, ko je vsem uporabnikom priporočil 29 največkrat zaznamovanih stvari. Povzetek najboljših rezultatov vseh testiranih algoritmov za obe meri je zapisan tudi v tabeli 3.3.



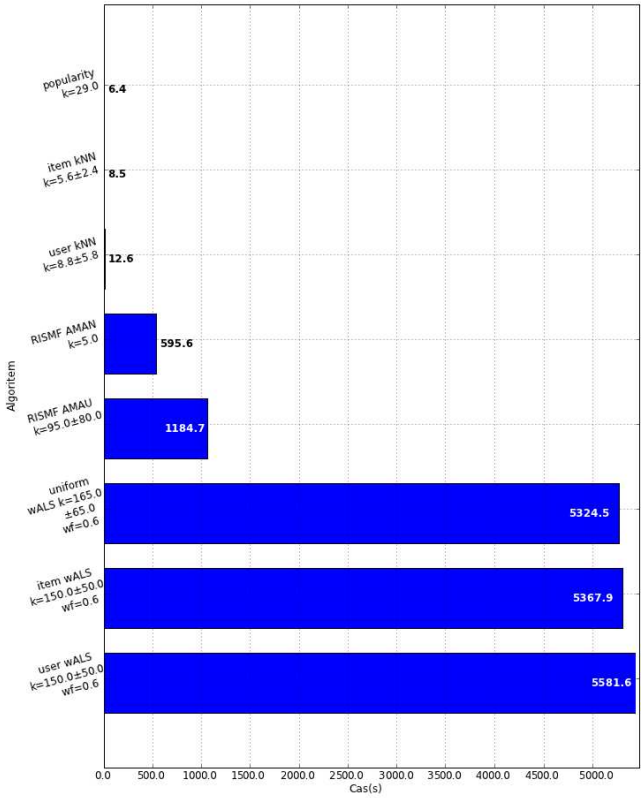
Slika 3.13: Najboljši rezultati vseh testiranih algortimov za mero MAP

3.4.5 Testiranje časovne kompleksnosti

Za konec smo preverili še časovno porabo posameznih algoritmov. Slika 3.14 prikazuje koliko časa se izvajajo posamezni algoritmi. Prikazani so časi algoritmov ob parametrih, s katerimi so dosegli najvišjo vrednost MAP mere.

Najslabše rezultate so dosegli wALS algoritmi. Ti imajo zaradi svoje zasnove in obravnavanja vseh primerov (tudi manjkajočih) najvišjo časovno zahtevnost. Najhitrejši iz družine wALS algoritmov (uniform-wALS) se izvede kar 420-krat počasneje od na uporabnikih temelječega algoritma najbližjih sosedov. Algoritmi wALS so primerni le, ko nimamo časovnih omejitev in je v ospredju kar najboljše doseganje rezultatov. Algoritma RISMF imata sicer

podobno časovno zahtevnost, vendar namesto metode izmenjujočih se najmanjših kvadratov uporabljata tehniko inkrementalnega gradientnega spusta, ki je nekoliko hitrejša [4].



Slika 3.14: Časovna poraba algoritmov, zmerjena pri najboljšem MAP rezultatu

Med metodami najbližjih sosedov, se časovno najbolje izkaže na stvareh temelječa metoda k najbližjih sosedov. Kot smo omenili, se pri tem algoritmu vnaprej izračunajo podobnosti med stvarmi ter za vsako k najbližjih sosedov, zato mu tega ni potrebno računati vsakem koraku napovedovanja. Izkaže se, da je le malo počasnejši od algoritma popularnosti, ki za vsakega uporabnika poišče le k najbolj popularnih stvari, katerih ta uporabnik še ni zaznamoval.

Vse dobljene meritve so povzete v tabeli 3.3.

Tabela 3.3: Tabela najboljših rezultatov implementiranih algoritmov.

Algoritem	k	*wf*	PPV	MAP	Čas trajanja
item kNN	5.6±2.4	-	0.1887	0.3508	8.5
user kNN	8.8±5.8	-	0.5233	0.5209	12.6
popularity	29.0	-	0.0083	0.0686	6.4
item wALS	150.0±50.0	0.6	0.9350	0.8271	5367.9
user wALS	150.0±50.0	0.6	0.9642	0.8329	5581.6
uniform wALS	165.0±65.0	0.6	0.9405	0.8502	5324.5
RISMF AMAU	95.0±80.0	-	0.0025	0.1166	1184.7
RISMF AMAN	5.0	-	0.0000	0.0000	595.6

Poglavje 4

Sklepne ugotovitve

Pogostokrat imamo na voljo enorazredne podatke, pri katerih so podani le pozitivni primeri. Zanje se uporablja enorazredna strategija filtriranja medsebojnih povezanosti. Za probleme z enorazrednimi podatki je značilno, da ne moremo ločiti med primeri, ko je nek uporabnik neko stvar označil za negativno ali pa je enostavno ni videl. V diplomski nalogi smo opisali, implementirali ter med seboj na enorazrednih podatkih primerjali 7 različnih algoritmov.

Za priporočilnih sisteme se največkrat uporablja strategija, ki temelji na filtriranju medsebojnih povezanosti. Algoritmi, ki se pri tem uporabljajo, s pomočjo matrične faktorizacije identificirajo medsebojna razmerja med uporabniki in stvarmi, katera nato uporabijo pri napovedovanju. Algoritmi, ki na enorazrednih podatkih za učenje uporabljajo le pozitivne primere, najdejo trivialno rešitev - označitev vseh primerov za pozitivne, kar potrjuje tudi naš test algoritma RISMF AMAU. Če namesto učenja le na pozitivnih primerih, vse manjkajoče primere označimo za negativne, zaradi prevelike obtežitve in velikega števila manjkajočih vrednosti, na naših podatkih algoritem RISMF AMAN vse primere označi za negativne. Algoritem wALS uporablja manjšo obtežitev za neznane primerov kot za pozitivne. Preverili smo tri različne načine obtežitev za ta algoritem. Rezultati za vse sheme kažejo, da ob primernem faktorju uteži in številu značilnosti z njimi lahko dosegamo zelo dobre rezultate. Zaradi svoje časovne potratnosti so ti algoritmi uporabni le, ko nimamo časovnih omejitev.

Algoritmi k najbližji sosedov se v praksi pogosteje uporabljajo. Imajo nizko časovno zahtevnost, so kompaktni ter dosegajo relativno dobre rezultate. Na uporabnikih temelječa metoda k najbližjih sosedov dosega le nekoliko slabše rezultate v primerjavi z wALS algoritmi in to ob v 420-krat hitrejšem času (na naši bazi podatkov). Na stvareh temelječa metoda najbližjih sosedov je

popularna predvsem, ker omogoča, da lahko vnaprej izračunamo sosede vseh stvari in je zato mogoče realno časovno napovedovati, ali bo uporabnik določeno stvar zaznamoval ali ne. Napoved je odvisna le od števila stvari, ki jih je uporabnik že zaznamoval.

Algoritme, ki smo jih predstavili, se da uporabiti na vseh področjih, kjer je potrebno priporočanje na množici enorazrednih podatkov, torej neodvisno od domene. Ponavadi so to predvsem spletne strani, saj je od uporabnikov pogosto nerodno zahtevati, da ocenjujejo veliko stvari. Zato mnogo sistemov temelji na implicitno pridobljenih podatkih kot so npr. klik na povezavo, nakup predmeta, zaznamek strani ...

V bodoče bi bilo zanimivo raziskati odvisnost rezultatov od različnih normalizacijskih faktorjev ter konvergenco v lokalne minimume za različne začetne vrednosti matrik P in Q pri RISMF in wALS algoritmih ter implementirati še kakšno obteževalno shemo algoritma wALS. Pri algoritmih wALS bi bilo možno preveriti tudi odvisnost mer od števila značilnosti ter obteževalnega faktorja hkrati. Podobno kot pri algoritmih z matrično faktorizacijo bi za metode k najbližji sosedov lahko testirali, če se z manjšo obtežitvijo negativnih primerov da izboljšati dosežene rezultate. Zanimiva bi bila tudi primerjava z enorazredno različico metode podpornih vektorje (*angl. Support Vector machine, SVM*), katera se tudi uporablja za reševanje problemov z enorazrednimi podatki [13].

Dodatek A

Izvorna koda algoritmov matrične faktorizacije

```
'''
Created on 27.4.2011

@author: Matic Perov"sek
'''

import numpy
import sys
from collections import defaultdict

def wALS(no_of_users,no_of_items,R, K=20, niters=100,w_factor
    =0.005, w_scheme="uniform",regularization_factor=0.002):
    """ wALS algorithm, returns factorization matrices P and Q of
        matrix R

    Keyword arguments:
    R -- ratings matrix
    K -- number of features (default 20)
    niters -- number of iterations (default 100)

    """

    sys.stdout.flush()
    #initialize P and Q with small random numbers
    P = numpy.random.uniform(0.04,0.06,(no_of_users,K)) # NxK
    Q = numpy.random.uniform(0.04,0.06,(no_of_items,K)) # MxK

    wf_normalizer=1.
    if w_scheme=="item_oriented":
```

```
        item_count=defaultdict(int)
        for user in R:
            for item in R[user]:
                item_count[item]+=1
        wf_normalizer=max([no_of_users-item_count[item] for item
            in range(no_of_items)])
    elif w_scheme=="user_oriented":
        wf_normalizer=max([len(R[i]) for i in range(no_of_users)])


    iters=0
    cond=True
    while iters<niters and cond:
        iters+=1

        def update_P_row(i):
            """ Updates i-th row in matrix P """

            #set weights for negative examples
            user_row=numpy.zeros(no_of_items)
            W_hat_i=w_factor*numpy.eye(no_of_items)

            for item in range(no_of_items):
                if w_scheme=="user_oriented":
                    W_hat_i[item][item]*=len(R[i])*1./
                        wf_normalizer if i in R and len(R[i])>0
                        else 0.000000001
                elif w_scheme=="item_oriented":
                    W_hat_i[item][item]*=(no_of_users-item_count[
                        item])*1./wf_normalizer

            #set weights to 1 for positive examples
            for item in R[i]:
                user_row[item]=1
                W_hat_i[item][item]=1

            #calculate new row
            W_i_sum=sum([W_hat_i[abc][abc] for abc in range(
                no_of_items)])
            W_hat_dot_Q=numpy.dot(W_hat_i,Q)

            inner_k_x_k=numpy.linalg.inv(numpy.dot(Q.T,W_hat_dot_Q
                )+regularization_factor*W_i_sum*numpy.eye(K))
            P[i,:]=numpy.dot(numpy.dot(user_row,W_hat_dot_Q),
                inner_k_x_k)

        def update_Q_row(j):
```

```
            """ Updates j-th row in matrix Q """

            #set weights for negative examples
            item_row=numpy.zeros(no_of_users)
            W_hat_j=w_factor*numpy.eye(no_of_users)
            #positive=0

            for user in range(no_of_users):
                if w_scheme=="user_oriented":
                    W_hat_j[user][user]*=len(R[user])*1./
                        wf_normalizer if user in R and len(R[user])
                        >0 else 0.000000001
                elif w_scheme=="item_oriented":
                    W_hat_j[user][user]*=(no_of_users-item_count[j
                        ])*1./wf_normalizer

            #set weights to 1 for positive examples
            for user in R:
                if j in R[user]:
                    item_row[user]=1
                    W_hat_j[user][user]=1

            #calculating new row
            W_j_sum=sum([W_hat_j[abc][abc] for abc in range(
                no_of_users)])
            W_hat_dot_P=numpy.dot(W_hat_j,P)

            inner_k_x_k=numpy.linalg.inv(numpy.dot(P.T,W_hat_dot_P
                )+regularization_factor*W_j_sum*numpy.eye(K))
            Q[j,:]=numpy.dot(numpy.dot(item_row,W_hat_dot_P),
                inner_k_x_k)


        #update rows in P and Q for every user and item
        map(update_P_row, xrange(no_of_users))
        map(update_Q_row, xrange(no_of_items))

        sys.stdout.flush()
    return P, Q



def RISMF(no_of_users,no_of_items,R, K=2, niters=150,learning_rate
    =0.2, regularization_factor=0.02):
    """ RISMF algorithm, returns factorization matrices P and Q of
        matrix R
```

```
    Keyword arguments:
    R -- ratings matrix
    K -- number of features (default 20)
    niters -- number of iterations (default 100)
    learning_rate -- learning weight for every iteration (default
        0.2)
    regularization_factor -- the wight of requlariza tion (default
        0.02)

    """

    print "RISMF K="+str(K)
    sys.stdout.flush()

    #initialize P and Q with small random numbers
    P = numpy.random.uniform(0.04,0.06,(no_of_users,K)) # NxK
    Q = numpy.random.uniform(0.04,0.06,(no_of_items,K)).T #(MxK).T


    previous_SSE_apos=999999999999 #initial big value
    iters=0 #iteration counter
    cond=True
    while iters<niters and cond:
        iters+=1

        #update P and Q for every example
        for u in R:
            for i in R[u]:
                #the error of rating u on i
                e_ui = 1. - numpy.dot(P[u,:],Q[:,i])
                #stohastic gradient descent
                for k in range(K):
                    gradient_p_uk= -e_ui * Q[k][i] +
                        regularization_factor * P[u][k]
                    gradient_q_ki= -e_ui * P[u][k] +
                        regularization_factor * Q[k][i]
                    P[u][k] = P[u][k] + learning_rate * (-
                        gradient_p_uk)
                    Q[k][i] = Q[k][i] + learning_rate * (-
                        gradient_q_ki)

        #calculate RMSE error
        SSE_apos = 0
        for u in R:
            for i in R[u]:
                #SSE_apos=sum of all e'_ui
```

```
                    #calculate e'_ui
                    SSE_apos = SSE_apos + pow(1.0 - numpy.dot(P[u,:],Q
                        [:,i]), 2)
                    for k in range(K):
                        SSE_apos = SSE_apos + (regularization_factor
                            /2) * (pow(P[u][k],2) + pow(Q[k][i],2))

            #break if difference smaller than epsilon
            if previous_SSE_apos-SSE_apos < 0.000001:
                cond=False
                break
            previous_SSE_apos=SSE_apos
        return P, Q.T


def RISMF_AMAN(no_of_users,no_of_items,R, K=2, niters=150,
    learning_rate=0.2, regularization_factor=0.02):
        print "RISMF K="+str(K)#,iters,SSE_apos
        sys.stdout.flush()
        #initialize P and Q with small random numbers
        P = numpy.random.uniform(0.04,0.06,(no_of_users,K)) # NxK
        Q = numpy.random.uniform(0.04,0.06,(no_of_items,K)).T #(MxK).T

        #print P,Q

        previous_SSE_apos=999999999999
        iters=0
        cond=True
        while iters<niters and cond:
            iters+=1
            for u in range(no_of_users):
                for i in range(no_of_items):
                    in_training_set=u in R and i in R[u]
                    r_ui=1. if in_training_set else 0.
                    #the error of rating u on i
                    e_ui = r_ui - numpy.dot(P[u,:],Q[:,i])
                    #stohastic gradient descent
                    for k in range(K):
                        gradient_p_uk= -e_ui * Q[k][i] +
                            regularization_factor * P[u][k]
                        gradient_q_ki= -e_ui * P[u][k] +
                            regularization_factor * Q[k][i]
                        P[u][k] = P[u][k] + learning_rate * (-
                            gradient_p_uk)
                        Q[k][i] = Q[k][i] + learning_rate * (-
                            gradient_q_ki)
```

```
        #print len(R)
        eR = numpy.dot(P,Q)
        SSE_apos = 0
        for u in range(no_of_users):
            for i in range(no_of_items):
                in_training_set=u in R and i in R[u]
                r_ui=1. if in_training_set else 0.
                #SSE_apos=sum of all e'_ui

                #calculate e'_ui
                SSE_apos = SSE_apos + pow(r_ui - numpy.dot(P[u,:],
                    Q[:,i]), 2)
                for k in range(K):
                    SSE_apos = SSE_apos + (regularization_factor
                        /2) * (pow(P[u][k],2) + pow(Q[k][i],2))

        if previous_SSE_apos-SSE_apos < 0.0001:
            cond=False
            break
        previous_SSE_apos=SSE_apos
    return P, Q.T
```

mf.py

Dodatek B

Izvorna koda na uporabnikih temelječe metode *k* najbližjih sosedov

```
import numpy as np
from collections import defaultdict


def find_user_neighbours(active_user,user_rating_dict,k=5):
    """ finds and returns k nearest users for active_user

    Keyword arguments:
    active_user -- active user
    item_rating_dict -- dictionary of users and items they
        bookmarked
    k -- number of neighbours

    """

    scores=[]
    ratings_user=user_rating_dict[active_user]

    for other_user in user_rating_dict.keys():
        if active_user!=other_user:
            s=jaccard_similarity(ratings_user, user_rating_dict[
                other_user])
            scores.append((s,other_user))

    scores.sort(reverse=True)
    return scores[0:k]
```

```
def kNN_get_recommendations(no_of_users, no_of_items, training, user,
    k=5):
    """ returns recommendation ratings of active user for every
        item

    Keyword arguments:
    active_user -- active user
    training -- training set
    k -- number of neighbours

    """


    totals={}
    simSums={}

    neigbours_with_similarity=find_user_neighbours(user, training,
        k)

    #calculate ratings
    for (similarity,neighbour) in neigbours_with_similarity:
        if similarity!=0:
            for item in range(no_of_items):
                totals.setdefault(item,0)
                totals[item]+=1.0*similarity if neighbour in
                    training and item in training[neighbour] else
                    0.
                simSums.setdefault(item,0)
                simSums[item]+=similarity


    rankings=defaultdict(float)

    #divide by sum of similarities
    for item,total in totals.items():
        rankings[item]=total*1./simSums[item]

    return rankings


def jaccard_similarity(A_items,B_items):
    """ returns Jaccards's similarity between two users """

    a=A_items
    b=B_items
```

```
    intersection=a.intersection(b)
69  return 0. if len(intersection)==0 else 1.*len(intersection)/
        len(a.union(b))
```

kNN_user.py

Dodatek C

Izvorna koda na stvareh temelječe metode k najbližjih sosedov

```
import numpy as np
from collections import defaultdict


def find_item_neighbours(active_item,no_of_items,item_rating_dict,
    k,similarity):
    """ finds and returns k nearest items for active_item

    Keyword arguments:
    item_rating_dict -- dictionary of items with users who
        bookmarked
    k -- number of neighbours
    similarity -- similarity measure

    """

    scores=[]

    for other_item in range(no_of_items):
        if active_item!=other_item:
            #print user_rating_dict
            s=similarity(active_item, other_item, item_rating_dict
                )
            scores.append((s,other_item))

    scores.sort(reverse=True)
```

```
    return scores[0:k]

def conditional_similarity(item,other_item,item_rating_dict):
    """ returns conditional probability-based similarity between
        two items"""

    freq_i=len(item_rating_dict[item])
    freq_i_oi = len(item_rating_dict[item].intersection(
        item_rating_dict[other_item]))
    return 0. if freq_i_oi==0 else freq_i_oi*1./freq_i

def find_neighbours_for_every_item(no_of_items,training,k=5,
    item_similarity=conditional_similarity):
    """ returns k nearest items for every item

    Keyword arguments:
    item_rating_dict -- dictionary of items with users who
        bookmarked
    k -- number of neighbours
    similarity -- similarity measure

    """

    item_neigbours_dict={}
    for item in range(no_of_items):
        item_neigbours_dict[item]=find_item_neighbours(item,
            no_of_items,training,k,item_similarity)
    return item_neigbours_dict

def kNN_item_rating(training,item_neighbours,user):
    """ returns score by user for active item

    Keyword arguments:
    training -- training set
    item_neigbours -- active item's nearest neighbours
    user -- active user

    """

    total=0
    simSum=0

    for (similarity,neighbour) in item_neighbours:
        total+=1.0*similarity if user in training and neighbour in
            training[user] else 0.
        simSum+=similarity
```

```
    return 0. if simSum==0 else total/simSum

def popularity_ranking(training,user,k=5):
    """ popolarity algorithm, returns dictionary with k most
        popular items not voted by user with their scores

    Keyword arguments:
    training -- training set
    user -- active user
    k -- number of items recommended

    """

    item_count=defaultdict(int)
    max=0
    #count item bookmarks
    for _,items in training.items():
        for item in items:
            if not item in training[user]:
                item_count[item]+=1
                if item_count[item]>max:
                    max=item_count[item]
    #set scores
    rankings=[(0.5+count*1./(2*max),item) for item,count in
        item_count.items()]
    rankings.sort(reverse=True)
    rezultz=defaultdict(int)

    #list to dicitonary
    for score,item in rankings[0:k]:
        rezultz[item]=score
    return rezultz
```

kNN_item.py

Slike

Tabele

Literatura

[1] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems," *The Journal of Machine Learning Research*, vol. 10, str. 623–656, 2009.

[2] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," v *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on.* IEEE, 2008, str. 502–511.

[3] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, str. 734–749, 2005.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, str. 30–37, 2009.

[5] T. Segaran, *Programming collective intelligence: building smart web 2.0 applications.* O'Reilly Media, 2007.

[6] R. Bell and Y. Koren, "Improved neighborhood-based collaborative filtering," v *KDD-Cup and Workshop.* Citeseer, 2007.

[7] M. Hahsler, "Developing and testing top-n recommendation algorithms for 0-1 data using recommenderlab," 2011.

[8] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, str. 5–53, 2004.

[9] T. Bogers and A. Van Den Bosch, "Recommending scientific articles using citeulike," v *Proceedings of the 2008 ACM conference on Recommender systems.* ACM, 2008, str. 287–290.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," v *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, str. 285–295.

[11] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, str. 143–177, 2004.

[12] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, str. 76–80, 2003.

[13] Y. Yajima and T. Kuo, "Efficient formulations for 1-svm and their application to recommendation tasks," *Journal of Computers*, vol. 1, no. 3, str. 27–34, 2006.