T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

A NOVEL RECOMMENDER ENGINE

Master of Science Thesis

ALPER TÜFEK

İSTANBUL, 2014

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

A NOVEL RECOMMENDER ENGINE

Master of Science Thesis

ALPER TÜFEK

Supervisor: Assoc. Prof. Dr. Selim Necdet MİMAROĞLU

İSTANBUL, 2014

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Name of the thesis:
Name/Last Name of the Student:
Date of the Defense of Thesis:

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. Dr. Tunç BOZBURA
Graduate School Director
Signature

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Tarkan AYDIN
Program Coordinator
Signature

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Comittee Members	Signature
Thesis Supervisor Assoc. Prof. Dr. Selim Necdet MİMAROĞLU	-----------------------------------
Member Assoc. Prof. Dr. Mehmet Alper TUNGA	-----------------------------------
Member Asst. Prof. Dr. Devrim ÜNAY	-----------------------------------

ABSTRACT

A NOVEL RECOMMENDER ENGINE

Alper Tüfek

Computer Engineering

Thesis Supervisor: Assoc. Prof. Dr. Selim Necdet Mimaroğlu

April 2014, 38 Pages

Recommender systems are software tools and techniques that help users to find products/items which are of interest, from large catalogs. Available options extremely differ both in number and attributes depending on the domain, that is, the type of object/item needed to be selected.

Recommender systems can be classified broadly into three categories: *content-based*, *collaborative filtering* based and *hybrid* systems. Content-based systems generate recommendations based on descriptions or content of items. The user will be recommended items similar to the ones the user preferred in the past. The biggest limitation of content-based techniques is that extracting features associated with items to be recommended is usually a costly process. The content must either be in a form that can be parsed automatically (e.g., text) or the features should be assigned to items manually. *Collaborative filtering* is the most popular technique for recommender systems. Recommender systems of this group simulate taking recommendations from friends with similar tastes.

In this thesis, a novel recommender system based on collaborative filtering is designed which can be easily applied to many different domains. The main advantage of the new system is its ability to use both implicit and explicit information which considerably increases recommendation coverage. Also an asymmetric approach is proposed for similarity calculations during nearest neighbor selection procedure. Another objective that is aimed to observe is to be better at differentiating especially *liked* items from *disliked* ones. In this respect, a *penalization* scheme is incorporated to lower down the scores for items with low ratings whereas highlighting items with high ratings.

Keywords: Recommender Systems, Similarity, Collaborative Filtering

ÖZET

YENİ BİR TAVSİYE MOTORU

Alper Tüfek

Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Selim Necdet Mimaroğlu

Nisan 2014, 38 Sayfa

Tavsiye sistemleri; kullanıcıların, devasa kataloglardan beğenebilecekleri ürünleri bulmalarına yardımcı olacak yazılım araçlarıdır. Uygun seçenekler, sektöre bağlı olarak hem seçenek sayısı hem de nitelik bakımından oldukça çeşitlilik göstermektedir.

Tavsiye sistemleri genel olarak, *içerik-tabanlı*, *işbirlikçi filtreleme* tabanlı ve *melez* (*hibrit)* sistemler olmak üzere üç sınıfa ayrılırlar. *İçerik-tabanlı* sistemler, ürünlerin açıklamalarına veya içeriklerine dayalı olarak tavsiye üretirler. Kullanıcıya, geçmişte tercih ettiği ürünlere benzer nitelikte ürünler tavsiye edilir. İçerik-tabanlı sistemlerin en büyük dezavantajı, ürün açıklamaları veya niteliklerinin elde edilmesinin oldukça maliyetli bir işlem olmasıdır. İçeriğin, otomatik olarak okunup ayrıştırılabilen bir formatta (metin vb.) olması ya da ürün niteliklerinin el yordamıyla ürünlere atanması gerekir. *İşbirlikçi filtreleme*, tavsiye sistemlerinde en çok tercih edilen tekniklerin başında gelmektedir. Bu kategorideki sistemler, benzer zevklere sahip arkadaş çevresinden tavsiye alma kavramını taklit ederler.

Bu tez çalışmasında, birçok sektöre kolayca uyarlanabilecek, *işbirlikçi filtreleme* temelli yeni bir tavsiye sistemi geliştirilmiştir. Yeni sistemin en önemli avantajı, hem doğrudan hem de dolaylı tercih verilerini aynı anda kullanabilme becerisidir ki bu da önerilebilecek ürün kapsamını önemli ölçüde artırmaktadır. Çalışma kapsamında ayrıca, en benzer komşuların seçimi sırasındaki benzerlik hesaplamalarında asimetrik bir yaklaşım yöntemi de önerilmiştir. Çalışmada hedeflenilen bir başka sonuç ise, özellikle *sevilen* ürünleri *sevilmeyen* ürünlerden ayırt edebilme konusunda ortalamadan daha başarılı bir performans sergileyebilmektir. Bu amaçla, yüksek puanlı ürünleri öne çıkarırken düşük puanlı ürünleri mümkün mertebe aşağı çekecek bir *cezalandırma* düzeni de önerilmiştir.

Anahtar Kelimeler: Tavsiye Sistemleri, Benzerlik, İşbirlikçi Filtreleme

TABLE OF CONTENTS

TABLES

FIGURES

1. INTRODUCTION

In everyday life, people often face situations in which they need to make choices among different options. These options extremely differ both in number and attribute depending on the domain, that is, the type of object/item needed to be selected. Deciding what movie to watch, what book/article/news to read, in what restaurant to eat, what music to listen to, are only a few examples that come to mind at the first moment. In such cases, asking others' opininons about specific alternatives, or, requesting direct recommendations, without prior knowledge about options, from trusted friends, are the most prefered approaches to follow. All these scenarios and lots more are faced also in digital world today as we can do almost everything with a PC having an Internet connection. Today, we can order food, watch movies and TV programs, listen to music, read news/articles and do many more online.

As digital content is the case, the available options are usually much more than their corresponding physical counterparts, e.g. an online book store can contain millions of books, while a typical bricks and mortar store can offer maybe a little more than a hundred of thousands of books. A standard catalog of an online video on demand service such as Netflix[1] can have more than 100,000 titles. URLs of web sites are counted in tens of billions today. In other words, we are living in an era of huge catalogs and databases where one person cannot have an overview of what is available and what might be of interest to him/her.

In general, two types of systems have been developed to deal with the information overload: *search engines* and *automatic recommender systems (RS)*. Search engines are useful for people who know what exactly they want and who will perform a search query. Automatic recommender systems are often used as a support system for discovery and navigation or as a support system for decision making (Meyer 2012, p. 32). More specifically, recommender systems are software tools and techniques that

[1] Netflix.com is an online DVD rental and video-on-demand service.

help users to find products/items which are of interest, from large catalogs. Items can be any object that can be consumed, bought, read, viewed, etc.

In their paper, Rao and Talwar (2008) identified 96 recommendation systems in various domains, including both research oriented and industrial applications.

1.1 CLASSIFICATION OF RECOMMENDER SYSTEMS

Recommender systems can be classified broadly into three categories depending on how recommendations are made: *Content-based*, *collaborative filtering* based and *hybrid* recommenders (Adomavicius and Tuzhilin 2005):

1.1.1 Content-Based Recommenders

Recommendation is based on descriptions or content of items rather than other user's preferences. A target user will be recommended items similar to the ones the user preferred in the past. System analyzes a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user (Lops et al. 2012). In order to build a model or profile, item descriptions are processed by a content analyzer that extracts features (keywords, n-grams, etc.) from unstructured text to produce a structured item representation.

The biggest limitation of content-based techniques is that extracting features associated with items to be recommended is usually a costly process. The content must either be in a form that can be parsed automatically (e.g., text) or the features should be assigned to items manually (e.g., annotation). On the other hand, some domains have an inherent problem with automatic feature extraction. For example, automatic feature extraction methods are much harder to apply to multimedia data, e.g., graphical images, audio and video streams. Moreover, it is often not practical to assign attributes manually due to limitations of resources (Adomavicius and Tuzhilin 2005).

1.1.2 Collaborative Filtering Based Recommenders

The term *collaborative filtering (CF)* was first coined by the researchers at Xerox Palo Alto Research Center (PARC) in 1992 (Goldberg et al. 1992). Recommender systems of this group that simulate getting recommendations from friends with similar tastes, are very popular today, especially in e-commerce. The system will recommend items that have received high ratings by other users with similar tastes or interests. Collaborative filtering systems are not based on the content of items. Therefore, the system does not need to analyze content (and, therefore, it is valid for any type of item including nonannotated multimedia content). Collaborative filtering techniques are explained in more detail in Section 1.2.

1.1.3 Hybrid Systems

These recommender systems are based on the combination of collaborative filtering and content-based methods. A hybrid system combining techniques from both categories tries to use the advantages of one category to overcome the disadvantages of the other (Burke 2007). For instance, CF methods suffer from new-item problems, i.e., they cannot recommend items that have no ratings yet. This does not limit content-based approaches since the prediction for new items is based on their description (features).

1.2 COLLABORATIVE FILTERING TECHNIQUES

Unlike content-based approaches, collaborative filtering based systems rely on the ratings given by users for items. The user ratings are stored in a table known as the *rating matrix*. This table is processed to generate recommendations. CF systems need to relate two fundamentally different entities: items and users. There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: *neighborhood-based approach* and *model-based approach (latent factor models).*

1.2.1 Memory-Based (Neighborhood-Based) Approaches

User-item ratings are directly used to predict ratings for new items. Within this group, there are two approaches to calculate rating predictions: *user-based* or *item-based*. User-based techniques evaluate the interest of a user *u* for an item *i* using the ratings for this item by other users, called *neighbors*, that have similar rating patterns. Item-based techniques predict the rating of a user *u* for an item *i* based on the ratings of *u* for items similar to *i* (Desrosiers and Karypis 2011).

1.2.2 Model-based approaches

In these approaches, available ratings are used to construct a predictive model first. This phase is usually called *model training*. Then, the model is later used to predict ratings of users for new items. Techniques of this group are sometimes called *latent factor models*.

Latent factor models such as neural networks (Salakhutdinov et al. 2007), Latent Dirichlet Allocation (Blei et al. 2003), and models (also known as SVD-based models) that are induced by factorization of the user-item ratings matrix (Paterek 2007, Funk 2006), attempt to uncover latent features that explain observed ratings. Matrix factorization models, for instance, map both users and items to a joint latent factor space such that user-item interactions are modeled as inner products in that space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, latent factors might be genre, amount of action, orientation to children; less well defined dimensions such as depth of character development; or completely uninterpretable dimensions (Koren and Bell 2007).

1.3 SIMILARITY MEASURES

Memory-based algorithms, traditionally, use similarity measures to select users (or items) that are similar to the target user (or item). Then, the prediction is calculated from the ratings of these neighbors. This is why memory-based algorithms are also called *neighbor-based*. Depending on being *user-based* or *item-based*, neighbor selection is focused on finding *similar users* or *similar items*.

There are a number of similarity measures to choose from. After a preliminary evaluation of Euclidean Distance, Cosine Similarity, Pearson Correlation Coefficient and Tanimoto Coefficient measures, Tanimoto Coefficient has been decided to move on with. Before taking a look at these measures, a basic notation must be defined first.

Table 1.1: Basic notation

$\mathbb{U}$	set of all users in data set
$\mathbb{I}$	set of all items in data set
r_{ui}	the rate user u gave to item i
$\mathbb{U}_i$	subset of users who rated an item i
$\mathbb{I}_u$	subset of items rated by a user u
$\mathbb{I}_{uv}$ or $\mathbb{I}_u \cap \mathbb{I}_v$	set of items common rated by two users u and v
$\mathbb{U}_{ij}$	set of users who rated both items i and j
$\mathbb{I}_u \cup \mathbb{I}_v$	set of items rated by at least one of the two users u and v

1.3.1 Euclidean Distance

The Euclidean distance between users *u* and *v* is defined as (Amatriain et al. 2011):

$$E(u,v) = \sqrt{\sum_{k=1}^{n} (r_{uk} - r_{vk})^2} \tag{1.1}$$

where *n* is the number of items and r_{uk} and r_{vk} are the rates given to the kth item by users *u* and *v* respectively. If we consider each user a data point in an *n*-dimensional space, the Euclidean distance between two points is the length of the straight line between the two points. It ranges from 0 to +∞.

1.3.2 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors that measures the cosine of the angle between them. For this reason, it is also called Vector Space Similarity (VSS). Cosine similarity between users *u* and *v* is defined as (Amatriain et al. 2011):

$$Cosine(u, v) = \frac{u \cdot v}{||u||\,||v||} \tag{1.2}$$

where • indicates vector dot product and ||*u*|| and ||*v*|| are the norms of vectors *u* and *v* respectively. Equation (1.2) can be rewritten more expressly by using the notation in Table 1.1 as:

$$Cosine(u, v) = \frac{\sum_{k=1}^{n} r_{uk} r_{vk}}{\sqrt{\sum_{k=1}^{n} {r_{uk}}^2}\sqrt{\sum_{k=1}^{n} {r_{vk}}^2}} \tag{1.3}$$

The resulting similarity ranges from -1 meaning exactly opposite (180^0 angle), to 1 meaning exactly the same (0^0 angle), with 0 usually indicating independence (90^0 angle). If ratings cannot be negative, the cosine similarity between two user vectors can range from 0 to 1.

1.3.3 Pearson Correlation Coefficient (PCC)

It is a measure of the correlation (linear dependence) between two variables, giving a value between +1 (strong positive correlation) and -1 (strong negative correlation). Pearson Correlation Coefficient between users *u* and *v* is defined as follows (Amatriain et al. 2011):

$$Pearson(u, v) = \frac{\sum(u, v)}{\sigma_u \sigma_v} \tag{1.4}$$

where $\sum(u,v)$ is the covariance of data points u and v, whereas σ means standard deviation. Equation (1.4) can be rewritten more expressly by using the notation in Table 1.1 as:

$$Pearson(u,v) = \frac{\sum_{k=1}^{n}(r_{uk} - \overline{r_u})(r_{vk} - \overline{r_v})}{\sqrt{\sum_{k=1}^{n}(r_{uk} - \overline{r_u})^2}\sqrt{\sum_{k=1}^{n}(r_{vk} - \overline{r_v})^2}} \tag{1.5}$$

where $\overline{r_u}$ and $\overline{r_v}$ are average rates of users *u* and *v* respectively.

1.3.4 Tanimoto Coefficient (Extended Jaccard Coefficient)

If we consider each user a vector of ratings, Tanimoto Coefficient between user vectors *u* and *v* is defined as (Amatriain et al. 2011):

$$Tanimoto(u,v) = \frac{u \cdot v}{||u||^2 + ||v||^2 - u \cdot v} \tag{1.6}$$

where · indicates the vector dot product:

$$\mathrm{u} \cdot \mathrm{v} = \sum_{k=1}^{n} r_{uk} r_{vk} \tag{1.7}$$

$||u||$ and $||v||$ are the lengths of user vectors *u* and *v* respectively:

$$||u|| = \sqrt{\sum_{k=1}^{n} r_{uk}^2} = \sqrt{u \cdot u} \tag{1.8}$$

1.4 EXPLICIT vs IMPLICIT INFORMATION

Recommender systems rely on different types of input. Most convenient is the high quality explicit feedback, which includes explicit input by users regarding their interest in products. However, explicit feedback is not always available. In other words, explicit ratings are typically unknown for the vast majority of user-item pairs (called *sparsity problem*), hence applicable algorithms work with the relatively few known ratings while ignoring the missing ones. This fact usually has a negative influence on recommendation *accuracy* and item *coverage*.

The main advantage of implicit feedback is that it is much more abundant than explicit feedback. User preferences can be inferred, to some degree, from implicit feedback, which indirectly reflect opinion through observing user behavior. Types of implicit feedback include purchase history, browsing history, watching time, listening count, search patterns, or even mouse movements depending on application domain. With implicit feedback, it would be natural to assign values to more user-item pairs. If no action was observed, zero is set for that particular user-item pair, meaning zero watching time, or zero listening count, or zero purchases, etc.

The vast majority of the algorithms in literature is focused on processing explicit feedback. However, in many practical situations, recommender systems need to consider implicit feedback. This is because of the reluctance of users to rate products, or limitations of the system that is unable to collect explicit feedback.

For these purposes, in this study, we propose a novel approach which, in addition to explicit feedback, can also make use of implicit feedback if available. In our approach, a separate item scoring is performed on implicit feedback. At the final scoring phase, these scores are merged with the item scores calculated from explicit feedback. We utilize Borda count and a weighting scheme at merging step. Details are explained in sections 2.2.2 to 2.2.4.

2. A NOVEL RECOMMENDER ENGINE - REDCAP

In this chapter, different approaches to calculating similarities between users are explained first. Then, a novel approach to similarity calculation is proposed. Next, the details of our algorithm are given. Our novel system is called *REDCAP* as an abbreviation of Recommender Engine with Dual Capability and Asymmetric Approach.

2.1 APPROACHES TO SIMILARITY CALCULATION

In literature, there are three approaches for *non-rated* items (*null rates*) in similarity calculation: *Ignore*, *Replace with Zero*, and *Replace with Average*. Their descriptions and formal definitions are given below. All formal definitions are demonstrated with Euclidean Distance which was explained in Section 1.3.1.

i *Ignore:* If an item is rated by only one of the two users, it is ignored in calculation which means that only common rated items are used. Using the notation in Table 1.1, the formal definition would be:

$$E_{ignore}(u,v) = \sqrt{\sum_{k \in \mathbb{I}_{uv}} (r_{uk} - r_{vk})^2} \quad \textbf{(2.1)}$$

ii *Replace with zero (rwz):* In this approach, items that have been rated by at least one of the two users are used. 0 is assigned to the item for the user who did not rate the item. The formal definition can be given as:

$$E_{rwz}(u,v) = \sqrt{\sum_{k \in \mathbb{I}_u \cup \mathbb{I}_v} (f(r_{uk}) - f(r_{vk}))^2} \quad \textbf{(2.2)}$$

where $f(r_{ui})$ defined as:

$$f(r_{ui}) = \begin{cases} 0, if\ r_{ui}\ is\ null \\ r_{ui}, otherwise \end{cases} \tag{2.3}$$

iii Replace with average (rwa): As in replace with zero, items that have been rated by at least one of the two users are used in calculation. Instead of 0, the average rating of the user is assigned to the item for the user who did not rate the item. Formal definition would be;

$$E_{rwa}(u,v) = \sqrt{\sum_{k \in \mathbb{I}_u \cup \mathbb{I}_v} \left(g(r_{uk}) - g(r_{vk})\right)^2} \tag{2.4}$$

where $g(r_{ui})$ defined as;

$$g(r_{ui}) = \begin{cases} \overline{r_u}, & if\ r_{ui}\ is\ null \\ r_{ui}, & otherwise \end{cases} \tag{2.5}$$

and

$$\overline{r_u} = \frac{1}{|\mathbb{I}_u|} \sum_{k \in \mathbb{I}_u} r_{uk} \tag{2.6}$$

2.1.1 A Novel Similarity Approach: Target User Based Tanimoto Coefficient

When calculating the similarity between two users, traditionally, items that both users have rated are used. In this sense, similarity is symmetric which means sim(u,v)=sim(v,u). Similarities calculated according to three approaches explained above are symmetrical. But on the other hand, it may not be a good comparison for two pairs of users such that they have different numbers of common rated items in pairs.

We propose an asymmetric approach based on the *target user*. In our approach, we take the target user (the user for whom recommendations will be made) as the reference user and calculate the similarity with every other user according to the items that the target user has rated. For those items that the target user has rated but the other user has not, 0 (zero) is assigned. In this case, sim(u,v) may not necessarily be equal to sim(v,u).

The formal definition is given for Tanimoto Coefficient as:

$$Tanimoto_{TUB}\ (u,v) = \frac{\sum_{k\in\mathbb{I}_u} r_{uk} f(r_{vk})}{\sum_{k\in\mathbb{I}_u} r_{uk}^2 + \sum_{k\in\mathbb{I}_u} f(r_{vk})^2 - \sum_{k\in\mathbb{I}_u} r_{uk} f(r_{vk})} \tag{2.7}$$

where $f(r_{ui})$ is defined as;

$$f(r_{ui}) = \begin{cases} 0, if\ r_{ui}\ is\ null \\ r_{ui}, otherwise \end{cases} \tag{2.8}$$

For instance, for the pair of users in Table 2.1:

Table 2.1: Example rating vectors of users 1 and 2

User/Item	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	-	5	-	-	-	-	-	5	5	5	-	-
2	-	-	5	5	-	-	-	-	-	-	-	-	-

when calculating Sim(1,2), only those items that user 1 has rated are taken into account since user 1 is the target user. Among these items, 0 is assigned to the items which are not rated by user 2. Then the two user vectors will look like in Table 2.2 where the cells with gray background indicate the null rates replaced with 0. Striped columns, which indicate the items that the target user has not rated, are left out during similarity calculation.

Table 2.2: User 1 is the target user

User/Item	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	-	5	-	-	-	-	-	5	5	5	-	-
2	-	-	5	5	-	-	-	-	0	0	0	-	-

If $Tanimoto_{TUB}(1,2)$ is calculated according to these vectors by the formula in equation (2.7):

$$Tanimoto_{TUB}(1,2) = \frac{25}{(25+25+25+25)+(25)-(25)} = 0\,.25$$

On the other hand, when calculating Sim(2,1), only those items that user 2 has rated are taken into account since user 2 is the target user this time. Likewise, among these items, 0 is assigned to the items which are not rated by user 1. Then the two user vectors will look like in Table 2.3:

Table 2.3: User 2 is the target user

User/Item	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	-	5	0	-	-	-	-	5	5	5	-	-
2	-	-	5	5	-	-	-	-	-	-	-	-	-

In this case, $Tanimoto_{TUB}(2,1)$ would be:

$$Tanimoto_{TUB}(2,1) = \frac{25}{(25+25)+(25)-(25)} = 0.50$$

In this approach, Sim(1,2) indicates the similarity between user 1 and user 2 from user 1's point of view (user 1 is target user) whereas Sim(2,1) indicates the similarity between user 1 and user 2 from user 2's point of view (user 2 is target user).

2.2 SELECTION OF NEAREST NEIGHBORS

Neighbood-based systems, such as ours, try to simulate the common principle of word-of-mouth, where a person relies on the opinion of like-minded people or other trusted sources to evaluate the value of an item (movie, book, news, article, etc.) according to his/her own preferences (Desrosiers and Karypis 2011). For this purpose, there is a need for a way of finding users with similar tastes.

For a selected (target) user, the similarities with every other user are calculated first. Then, these similarities are sorted in descending order and the list is cut-off at some point according to a *similarity threshold* or a fixed number, *k* (taking *k* most similar neighbors into account, *kNN*).

2.2.1 Nearest Neighbors from Explicit Ratings

Explicit ratings are usually numerical values at different scales (1-5 stars, 1-10 points, etc.). In some applications, rating scale consists of ordinal values (e.g. strongly agree, agree, neutral, disagree, strongly disagree). Ordinal values must be mapped to numerical values first. Ratings are direct representations of the possible levels of appreciation of users for items. Therefore, ratings which are explicitly given by users for items are the most valuable source for recommendation systems.

Table 2.4 Pseudocode for "Explicit Nearest Neighbors Selection" procedure

```
Procedure: Explicit Nearest Neighbors Selection
  Input: $R_{expl}$: Explicit ratings matrix, $u$: user to recommend for, $k$: number of top-k neighbors
  Output: $NNS_{exp}$: explicit nearest neighbor set for user $u$
1 Initialize an empty queue $Q$;
2 foreach user $v$ other than $u$ do //$\forall v \in \mathbb{U} \backslash u$ | $\mathbb{U}$: set of all users in data set
3     S=$Tanimoto_{TUB}(u, v)$ //calculate the similarity between $u$ and $v$
4                              //with Target User Based Tanimoto Coefficient ($u$ is target user)
5     Add pair (v,S) to Q
6 Sort Q in decreasing order with respect to similarity S
7 Copy top $k$ pairs from Q to $NNS_{exp}$
```

We used Target User Based Tanimoto Coefficient (explained in Section 2.1.1) to select the most like-minded users for each target user. Pseudocode of this process is given in

Table 2.4. We performed some preliminary tests to determine an appropriate number k, to cut-off the nearest neighbors list and determined k as 30. More details about these tests and how the value of k effects the quality of recommendations, can be found in *Chapter 3 Discussion And Experimental Results* (see Section 3.2.3).

2.2.2 Nearest Neighbors from Implicit Preferences

Explicit ratings are crucial for recommendation systems but there is a major problem associated with explicit ratings. Usually, there exists ratings for only a small portion of the user-item matrix in practice (called sparcity problem) because users usually refrains from explicitly giving rates. This limits the coverage of items to be recommended or the total number of users for whom recommendation can be made. Most of the time, only 10 percent or less of the user-item matrix is filled with ratings, 90 percent or more being empty, i.e. containing nulls.

On the other hand, many domains contain some kind of implicit indicator of preference. For example, In an online food ordering web site, items are restaurants. How many orders a user made from a restaurant can be used as an implicit sign of satisfaction. If a User A has ordered 10 times from Restaurant R_1 while he/she has ordered only twice from Restaurant R_2, then we can say, to some degree, that User A prefers R_1 over R_2. Likewise, listening counts of each song can be used as an implicit indicator of preference in a music recommendation domain.

We performed tests on the data of a prominent online food ordering site in Turkey to see recommendation quality based on implicit preferences. Raw data obtained from this web site contains 9,996,961 orders given by 473,504 distinct users from 3,421 distinct restaurants within the period of one year. After preprocessing, we converted Orders table into a user-item (restaurant) table where each row represents a user while each column represents an item (restaurant) and each cell holds the total number of orders for the corresponding user-restaurant pair. Implicit rating (preference) values can take a value from 0 to theoretically ∞. For this reason, we decided to use Cosine similarity since it does not require us to explicitly normalize implicit ratings into a fixed range beforehand. Details about the procedure of selecting implicit nearest neighbors are given in Table 2.5:

Table 2.5 Pseudocode for "Implicit Nearest Neighbors Selection" procedure

```
  Procedure: Implicit Nearest Neighbors Selection
  Input: $R_{imp}$: Implicit ratings matrix, $u$: user to recommend for, $k$: number of top-k neighbors
  Output: $NNS_{imp}$: Implicit nearest neighbor set for user $u$
1 Initialize an empty queue $Q$;
2 foreach user $v$ other than $u$ do //$\forall v \in \mathbb{U} \backslash u$ | $\mathbb{U}$: set of all users in data set
3       S=Cosine(u,v)   //calculate the similarity between $u$ and $v$
4                       //with Cosine similarity (see section 1.3.2)
5       Add pair (v,S) to Q
6 Sort Q in decreasing order with respect to similarity S
7 Copy top k pairs from Q to $NNS_{imp}$
```

2.2.3 Scoring Items Based on Implicit and Explicit Ratings

After constructing the set of nearest neighbors, we calculate a score for each item which a selected user has not rated yet, by using weighted ratings of the nearest neighbors. The score for the pair of a particular user u and an item i is calculated as follows:

$$Score(u,i) = \frac{\sum_{v \in N} sim(u,v) r_{vi}}{\sum_{v \in N} sim(u,v)} \tag{2.9}$$

where N is the set of nearest neighbors, r_{vi} is the rate that user v gave to item i. Set of nearest neighbors N consists of either set of explicit nearest neighbors (NNS_{exp}) or set of implicit nearest neighbors (NNS_{imp}) depending on whether implicit or explicit information is being used.

2.2.3.1 Penalization in explicit scoring

When scoring items based on explicit ratings, we incorporate a penalization mechanism which maps each rating according to the following table assuming that rating scale is 1 (lowest) to 5 (highest):

Table 2.6: Rating mappings for penalization

Rating	Mapped Value
Not Rated	0
1	-2
2	-1
3	0
4	4
5	5

The purpose of penalization is to lower down the scores for items with low ratings whereas highlighting items with high ratings. Penalization is applied in Equation (2.9) for each r_{vi} value. In other words, if an explicit rating r_{vi} is 1, 2 or 3, it is replaced with -2, -1 or 0 respectively.

2.2.4 Calculating Final Scores by Merging Implicit and Explicit Scores

When merging two scored lists of items, we utilize Borda count method (Liu 2011). In this method, a borda point is assigned to each item depending on the item's index position in the list. The last item in the list (the item that has the lowest score) is assigned 1. Then the item with next higher score is assigned 2, and so on. Ties, which means items that have the same score, gets the same borda point.

In our approach, for each unseen item, we look at its index positions at both lists of predicted implicit and explicit scores. Items which are at upper positions in both lists must be at higher positions in the final recommendation list.

On the other hand, if a user has explicitly rated very few items, similarity calculations in finding his/her explicit nearest neighbors are less reliable as compared to a user who has given explicit ratings to much more items. For this reason, we also propose a weighting scheme to apply this confidence effect to calculation of final scores.

Let us suppose that the positions of an item i in explicit and implicit nearest neighbor sets are pos_{exp} and pos_{imp} respectively. Confidence weightings for implicit and explicit scores are calculated as follows:

$$w_{exp} = \frac{|explicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|} \quad \textbf{(2.10)}$$

$$w_{imp} = \frac{|only\ implicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|} \quad \textbf{(2.11)}$$

Final item score is then given by the following formula for a user-item pair (u,i):

$$FinalScore(u, i) = w_{exp} * pos_{exp} + w_{imp} * pos_{imp} \quad \textbf{(2.12)}$$

Items having implicit ratings represent *previously seen items*. Users usually do not explicitly rate every item that they have seen before. Therefore, the number of items having explicit ratings is usually much less than the number of items having implicit ratings. At the extreme point, these number are equal which means that the user explicitly has given ratings to every single item that he/she has seen before. In this case, the weighting for explicit score, w_{exp}, will be 1 while the weighting for implicit score, w_{imp}, will be 0 which means that only explicit score will be taken into account. This is logical since explicit ratings are direct indicators of preference so are more trusted than implicit preferences.

General flow of our algorithm is given in Table 2.7. Let us explain our algorithm in more detail with an example: Let us suppose that there are 5 users and 10 items in the system where each item is a restaurant. Let the sample input dataset be like in Table 2.8. Each cell contains a pair of values seperated by comma. Firt value is implicit rating which is assumed to be the total number of orders from that particular restaurant. Second value is explicit rating in 1-5 rating scale. If a user does not explicitly rate a restaurant, the corresponding cell contains a dash (-) character to the right of comma which means a *null* value.

Table 2.7 Pseudocode for REDCAP

```
Algorithm: REDCAP
Input: $\boldsymbol{R_{exp}}$: Exlpicit ratings matrix, $\boldsymbol{R_{imp}}$: Implicit prefrences matrix, $\boldsymbol{u}$: target user, n:number of items to be recommended
Output: I: Set of recommended items sorted from best to worst
1  Compose $NNS_{exp}$, explicit nearest neighbor set for user $u$ using $R_{exp}$ and Target User Based Tanimoto Coefficient
2  Compose $NNS_{imp}$, implicit nearest neighbor set for user $u$ using $R_{imp}$ and Cosine similarity
3  $I'_{exp}$ ← Calculate predicted scores for each one of previously unrated items according to $NNS_{exp}$
4  $I'_{imp}$ ← Calculate predicted scores for each one of previously unseen items according to $NNS_{imp}$
5  $w_{exp} = \frac{|explicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|}$ //confidence weighting for explicit score (2.10)
6  $w_{imp} = \frac{|only\ implicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|}$ //confidence weighting for implicit score (2.11)
7  Initialize an empty list $I'$
8  foreach unseen item $i$ do
9      $pos_{exp} = i's\ borda\ point\ in\ list\ I'_{exp}$ //see section 2.2.4
10     $pos_{imp} = i's\ borda\ point\ in\ list\ I'_{imp}$ //see section 2.2.4
11     $FinalScore(u,i) = w_{exp} * pos_{exp} + w_{imp} * pos_{imp}$ //Final score for the pair of user $u$ and item $i$
12     Add $(i; FinalScore(u,i))$ pair into $I'$
13 Sort $I'$ with respect to final scores in descending order
14 Copy top $n$ items from $I'$ into $I$
```

Table 2.8: Sample input data containing both implicit and explicit ratings

User ID / tem ID	1	2	3	4	5	6	7	8	9	10
1	(0,-)	(1,-)	(10,5)	(0,-)	(0,-)	(3,-)	(0,-)	(0,-)	(15,5)	(9,5)
2	(2,3)	(2,4)	(8,5)	(0,-)	(1,-)	(0,-)	(0,-)	(1,-)	(11,5)	(1,3)
3	(0,-)	(7,4)	(8,5)	(0,-)	(2,4)	(1,3)	(3,4)	(6,5)	(0,-)	(4,4)
4	(10,4)	(0,-)	(10,4)	(5,3)	(0,-)	(4,4)	(20,5)	(18,5)	(13,4)	(0,-)
5	(1,-)	(2,3)	(7,4)	(0,-)	(0,-)	(1,-)	(0,-)	(12,4)	(20,5)	(19,5)

In this case, separate implicit and explicit rating matrices will look like in Table 2.9 and Table 2.10 respectively.

Dash symbols in Table 2.9 mean that there is no explicit rating for the corresponding user-item pair (meaning *null*). On the other hand, in implicit rating matrix, there will be no *null* values. Implicit values can be order (purchasing) counts, listening counts, time spent on a web page/document depending on the domain. For now, we consider it as online food order data and each implicit rating value indicates the total number of orders from a particular restaurant. For instance, user 1 has given 10 orders from restaurant 3 while he/she has not given any orders from restaurants 1, 4, 5, 7, and 8 yet.

Table 2.9: Sample explicit ratings matrix

User ID / tem ID	1	2	3	4	5	6	7	8	9	10
1	-	-	5	-	-	-	-	-	5	5
2	3	4	5	-	-	-	-	-	5	3
3	-	4	5	-	4	3	4	5	-	4
4	4	-	4	3	-	4	5	5	4	-
5	-	3	4	-	-	-	-	4	5	5

Table 2.10: Sample implicit preference matrix

User ID / tem ID	1	2	3	4	5	6	7	8	9	10
1	0	1	10	0	0	3	0	0	15	9
2	2	2	8	0	1	0	0	1	11	1
3	0	7	8	0	2	1	3	6	0	4
4	10	0	10	5	0	4	20	18	13	0
5	1	2	7	0	0	1	0	12	20	19

Let us find the explicit nearest neighbors of user 2. We must first calculate the similarities between user 2 and every other users according to Target User Based Tanimoto Coefficient (Section 2.1.1) bearing in mind that user 2 is the active user.

Table 2.11: Explicit user vectors of user 1 and 2 (user 2 is the active user)

User ID / tem ID	1	2	3	4	5	6	7	8	9	10
1	0	0	5	-	-	-	-	-	5	5
2	3	4	5	-	-	-	-	-	5	3

$$Tanimoto_{TUB}(2{,}1) = \frac{3*0+4*0+5*5+5*5+3*5}{(3^2+4^2+5^2+5^2+3^2)+(0^2+0^2+5^2+5^2+5^2)-(3*0+4*0+5*5+5*5+3*5)}$$

$$= \frac{25+25+15}{(9+16+25+25+9)+(25+25+25)-(25+25+15)}$$

$$Tanimoto_{TUB}(2{,}1) = 0.691$$

After calculating $Tanimoto_{TUB}(2{,}3)$, $Tanimoto_{TUB}(2{,}4)$ and $Tanimoto_{TUB}(2{,}5)$ likewise, resulting similarities will be:

$$Tanimoto_{TUB}(2{,}1) = 0.691$$

$$Tanimoto_{TUB}(2{,}3) = 0.602$$

$$Tanimoto_{TUB}(2{,}4) = 0.65$$

$$Tanimoto_{TUB}(2{,}5) = 0.828$$

Now we can write the explicit nearest neighbors of user 2 as below:
$NNS_{exp} = \{(5; 0.828), (1; 0.691), (4; 0.65), (3; 0.602)\}$
where each (u;s) pair represents a neighbor user *u* and its similarity *s* to the active user. The list is ordered by similarity in descending order. Since there can be only 4 neighbors at most in this example, we will use all the neighbors as much as possible. If the list had happened to be too long, we would take only some top *k* most similar users into account.

The next step is to calculate explicit scores for each item that the active user has not rated. Considering user 2 as the target user, for instance, explicit scores for previously unrated items 4, 5, 6, 7, and 8 must be predicted to decide what items might be of interest. Let us calculate user 2's explicit score for item 4 according to Equation 2.7.

$$Score(2,4) = \frac{\sum_{v \in NNS_{exp}} sim(2,v) r_{v4}}{\sum_{v \in NNS_{exp}} sim(2,v)}$$

$$= \frac{sim(2,5) * r_{54} + sim(2,1) * r_{14} + sim(2,4) * r_{44} + sim(2,3) * r_{34}}{sim(2,5) + sim(2,1) + sim(2,4) + sim(2,3)}$$

$$= \frac{0.828 * 0 + 0.691 * 0 + 0.65 * 3 + 0.602 * 0}{0.828 + 0.691 + 0.65 + 0.602} = \frac{1.95}{2.771}$$

$$Score(2,4) = 0.704$$

After calculating Score(2,5), Score(2,6), Score(2,7) and Score(2,8) likewise, resulting scores will be:

$$Score(2,4) = 0.704$$

$$Score(2,5) = 0.869$$

$$Score(2,6) = 1.590$$

$$Score(2,7) = 2.042$$

$$Score(2,8) = 3.454$$

Ordering items by their scores in descending order gives us a recommendation list I'_{exp} for user 2 sorted from highest predicted score to lowest:

$$I'_{exp} = \{(8; 3.454), (7; 2.042), (6; 1.590), (5; 0.869), (4; 0.704)\}$$

Items can be recommended from this list in the order beginning from the first item continuing towards the end of the list if there exists only explicit ratings. Recommandation can be terminated at any desired point depending on a number *n* (top-n recommendation) or on a threshold score.

In the second phase of the algorithm, the implicit nearest neighbors of each user are found in a similar way. To this end, the similarities between an active user and every other users are calculated according to Cosine Similarity. Assuming user 2 as the current user to recommend for, Cosine(2,1) is calculated as it follows:

Table 2.12 Implicit user vectors of user 1 and 2

User ID / tem ID	1	2	3	4	5	6	7	8	9	10
1	0	1	10	0	0	3	0	0	15	9
2	2	2	8	0	1	0	0	1	11	1

$$Cosine(2,1) = \frac{\sum_{k=1}^{10} r_{2k} r_{1k}}{\sqrt{\sum_{k=1}^{10} {r_{2k}}^2}\sqrt{\sum_{k=1}^{10} {r_{1k}}^2}}$$

$$= \frac{2*0+2*1+8*10+0*0+1*0+0*3+0*0+1*0+11*15+1*9}{\sqrt{(2^2+2^2+8^2+1^2+1^2+11^2+1^2)}\sqrt{(1^2+10^2+3^2+15^2+9^2)}}$$

$$= \frac{2+80+165+9}{\sqrt{(4+4+64+1+1+121+1)}\sqrt{(1+100+9+225+81)}}$$

$$= \frac{256}{\sqrt{196}\sqrt{416}}$$

$$Cosine(2,1) = 0.896$$

After calculating Cosine(2,3), Cosine(2,4) and Cosine(2,5) likewise, resulting similarities will be:

$$Cosine(2,1) = 0.896$$

$$Cosine(2,3) = 0.480$$

$Cosine(2,4) = 0.554$

$Cosine(2,5) = 0.721$

Now the list of implicit nearest neighbors of user 2 can be written as the following:

$NNS_{imp} = \{(1; 0.896), (5; 0.721), (4; 0.554), (3; 0.480)\}$

where each (u;s) pair represents a neighbor user *u* and its similarity *s* to the active user. The list is ordered by similarity in descending order. Implicit scores of each item that user 2 has not seen yet can be calculated now. In this case, items 4, 6, and 7 are previously unseen items. The calculation of user 2's implicit score for item 4 according to Equation 2.7 is below:

$$Score(2,4) = \frac{\sum_{v \in NNS_{imp}} sim(2,v) r_{v4}}{\sum_{v \in NNS_{imp}} sim(2,v)}$$

$$= \frac{Cosine(2,1) * r_{14} + Cosine(2,5) * r_{54} + Cosine(2,4) * r_{44} + Cosine(2,3) * r_{34}}{Cosine(2,1) + Cosine(2,5) + Cosine(2,4) + Cosine(2,3)}$$

$$= \frac{0.896 * 0 + 0.721 * 0 + 0.554 * 5 + 0.480 * 0}{0.896 + 0.721 + 0.554 + 0.480} = \frac{2.77}{2.651}$$

$Score(2,4) = 1.045$

After calculating Score(2,6) and Score(2,7) likewise, resulting scores will be:

$Score(2,4) = 1.045$

$Score(2,6) = 2.303$

$Score(2,7) = 4.723$

Since implicit ratings for items 5 and 8 are present, these implicit values are considered as their implicit scores as it is. Then, ordering items by their implicit scores in descending order gives us the list I'_P for user 2:

$$I'_{imp} = \{(7; 4.723), (6; 2.303), (4; 1.045), (5; 1), (8; 1)\}$$

In the last phase of the algorithm, final scores for previously unseen or unrated items are calculated. The choice of whether for previously unseen items or for unrated items that final scores will be calculated is optional. Let us assume that only those items which are not rated yet can be recommended. Thus, final scores for items 4, 5, 6, 7, and 8 must be calculated to determine in what order items will be recommended to user 2.

User 2 has explicitly rated 5 out of 10 items (items 1, 2, 3, 9, 10) whereas he/she has seen 7 out of 10 items (items 1, 2, 3, 5, 8, 9, 10). The number of only implicitly rated items, which means the number of items that user 2 has seen before but not rated yet, is 2 (items 5 and 8). According to these values, confidence weightings for explicit and implicit scores are calculated respectively as follow:

$$w_{exp} = \frac{|explicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|} = \frac{5}{7}$$
$$w_{imp} = \frac{|only\ implicitly\ rated\ items|}{|either\ implicitly\ or\ explicitly\ rated\ items|} = \frac{2}{7}$$

Let us recall the lists I'_{exp} and I'_{imp}:

$I'_{exp} = \{(8; 3.454), (7; 2.042), (6; 1.590), (5; 0.869), (4; 0.704)\}$

$I'_{imp} = \{(7; 4.723), (6; 2.303), (4; 1.045), (5; 1), (8; 1)\}$

Explicit and implicit Borda points for each unrated item can be seen in Table 2.13:

Table 2.13: Implicit and explicit borda points for unrated items

Item ID	Explicit Borda Point	Implicit Borda Point
4	1	2
5	2	1
6	3	3
7	4	4
8	5	1

Item 4 is assigned 1 as explicit borda point since it is the last item in list I'_{exp}. An important point is that if no score can be calculated for an item in either implicit or explicit scoring, zero is assigned as the corresponding borda point for that item. According to Table 2.13, let us calculate the final scores using (2.12):

$$FinalScore(u,i) = w_{exp} * pos_{exp} + w_{imp} * pos_{imp}$$

$$FinalScore(2,4) = \frac{5}{7} * 1 + \frac{2}{7} * 2 = 1.285$$

$$FinalScore(2,5) = \frac{5}{7} * 2 + \frac{2}{7} * 1 = 1.714$$

$$FinalScore(2,6) = \frac{5}{7} * 3 + \frac{2}{7} * 3 = 3.0$$

$$FinalScore(2,7) = \frac{5}{7} * 4 + \frac{2}{7} * 4 = 4.0$$

$$FinalScore(2,8) = \frac{5}{7} * 5 + \frac{2}{7} * 1 = 3.857$$

The final recommendation list is:

$$I = \{(7; 4.0), (8; 3.857), (6; 3.0), (5; 1.714), (4; 1.285)\}$$

Now, let us find the final recommendation list for the sample above by using classical collaborative filtering (CF) with cosine similarity. In this case, only explicit ratings table is used. First, the similarities between active user (user 2) and every other user are calculated.

$$Cosine(2,1) = 0.819$$

$$Cosine(2,3) = 0.521$$

$$Cosine(2,4) = 0.511$$

$$Cosine(2,5) = 0.823$$

Now we can write the explicit nearest neighbors of user 2 as below:

$$NNS_{exp} = \{(5; 0.823), (1; 0.819), (3; 0.521), (4; 0.511)\}$$

Next, scores for each item that user 2 has not rated are calculated.

$$Score(2,4) = 0.573$$

$$Score(2,5) = 0.779$$

$$Score(2,6) = 1.349$$

$$Score(2,7) = 1.735$$

$$Score(2,8) = 3.161$$

Finally, explicit recommendation list, I'_{exp} which is also the final recommendationd list in this case, for user 2:

$$I'_{exp} = I = \{(8; 3.161), (7; 1.735), (6; 1.349), (5; 0.779), (4; 0.573)\}$$

Let us recall REDCAP's final list:

$$I = \{(7; 4.0), (8; 3.857), (6; 3.0), (5; 1.714), (4; 1.285)\}$$

In this example, REDCAP produces a final list very similar to classical CF algorithm's. As the number of users and items increase, however, also the number of items that classical CF algorithms are unable to predict a score, will increase. As compared to classical CF algorithms, REDCAP can recommend more items since it can produce predicted scores for more items by using implicit preferences which is much denser than explicit ratings. Test results related to this aspect are explained in Section 3.2.2.

3. DISCUSSION AND EXPERIMENTAL RESULTS

In this chapter, we discuss the important features of our algorithm, REDCAP. We also provide our experimental results on both real and research data sets.

3.1 DISCUSSION OF REDCAP

The most distinctive characteristic of REDCAP is its ability to utilize both implicit and explicit data when both are available. If only implicit or only explicit data is available, REDCAP can still recommend items using only available type of data. In addition, REDCAP incorporates a penalization scheme at explicit item scoring phase to lower down the scores for items with low explicit ratings whereas highlighting items with high explicit ratings.

3.2 EXPERIMENTAL EVALUATIONS

This section includes experimental results of REDCAP on varying data sets from different domains and having different properties.

3.2.1 Test Methodology

Most of the existing collaborative filtering algorithms try to predict unknown ratings and their prediction accuracy is usually measured by the difference between the rating the algorithm predicts and the real rating (Herlocker et al. 2004). The most popular of this kind of metric is the mean absolute error (MAE). It is computed over all the ratings available in the evaluation subset, using the formula:

$$MAE = \frac{\sum_{i=1}^{N} |p_i - r_i|}{N} \qquad \textbf{(3.1)}$$

where p_i is the predicted rating for item i, r_i is the user 's true rating for item i.

Another related metric is root mean squared error (RMSE) which has become extremely popular in recent years, after being used in the Netflix Prize competition (Bennet and Lanning 2007). It is calculated using the formula:

$$RMSE = \frac{\sum_{i=1}^{N}(p_i - r_i)^2}{N} \qquad \textbf{(3.2)}$$

RMSE places greater emphasis on larger errors by squaring the error before summing it.

MAE or RMSE may be less appropriate for tasks such as *Find Good Items* where a ranked result is returned to the user, who then only views items at the top of the ranking. For these tasks, users may only care about errors in items that are ranked high, or that should be ranked high. It may be unimportant how accurate predictions are for items that the system correctly knows the user will have no interest in. Even if the recommender system's predicted ratings are incorrect, the system may be able to correctly rank a user's item recommendations.

REDCAP tries to recommend *Good Items* by assigning a score to each previously unseen item, instead of trying to predict every item's true rating. Because scoring values create an ordering across the items, predictive accuracy can be measured with the ability to correctly rank items with respect to user preference. In this approach, the aim is to determine the correct order of a set of items for each user and measure how close a system comes to this correct order. In order to evaluate a predicted ranking with respect to a reference ranking (a correct order), it is first necessary to obtain such a reference.

For each user u and for each couple of item (i, j) in the test set rated by u with $r_{ui} < r_{uj}$ or $r_{ui} > r_{uj}$ the preference given by u is compared with the predicted preference given by the recommender system, using the predicted ratings $\hat{r}_{ui}$ and $\hat{r}_{uj}$. In literature, most frequently used measure to evaluate a ranking with respect to a reference ranking is called the Normalized Distance-based Performance Measure (NDPM) which is given by:

$$NDPM = \frac{2C^{-} + C^{u0}}{2C^{u}} \quad \textbf{(3.3)}$$

where, C^{-} is the number of contradictory preference relations between the predicted ranking and the user ranking. A contradiction happens when the system says that item i will be preferred to item j whereas user ranking says the opposite. C^{+} is the number of pairs that the predicted ranking asserts the correct order. C^{u0} is the number of pairs where the reference ranking does not tie but the predicted ranking ties., i.e. user rates one item higher than the other, but the system ranks both items at the same level. C^{u} is the number of preferred relationships of the user: the number of pairs of rated items (i, j) for which the user gives a higher rating for one item than for the other. C^{u0} is calculated by:

$$C^{u0} = C^{u} - (C^{+} + C^{-})$$

Thus, the NDPM measure gives a perfect score of 0 to systems that correctly predicts every preference relation asserted by the reference. The worst score of 1 is assigned to systems that contradict every reference preference relation. Not predicting a reference preference relation is penalized only half as much as contradicting it. Predicting preferences that the reference does not order (i.e. when we do not know the user's true preference) is not penalized (Shani and Gunawardana 2011).

Another rank evaluation metric is just the percentage of compatible preferences (Meyer et al. 2012) which can be formulated as:

$$comp = \frac{C^{+}}{C^{u}} * 100 \quad \textbf{(3.4)}$$

This metric can be modified to evaluate the performance of correctly ranking only item pairs one of which is a *liked* (*good*) item whereas the other one is a *disliked* (*bad*) item. For instance, in a rating scale 1-to-5, items having ratings 4 or 5 can be regarded as "*liked*" whereas items having ratings 3, 2 or 1 can be regarded as "*disliked*" items.

3.2.2 Test Results of REDCAP

We have conducted experiments on a laptop computer having 2.2GHz processor with 8GB of main memory, running Windows 7 operating system. All test programs are implemented in Java language. On the other hand, LensKit Recommender Toolkit [LensKit, 2013] is an open source software package written in Java which contains implementations of the most well known and state-of-the-art algorithms in literature. We used Lenskit to test the performance of existing algorithms and to compare the results with REDCAP. The details about test methodology will be explained later in this section.

We performed tests on MovieLens [MovieLens, 2013] dataset to evaluate the performance of item recommendation using only explicit data. MovieLens is a free service provided by GroupLens Research Group at the University of Minnesota. The MovieLens dataset contains real data corresponding to movie ratings captured on the website of the MovieLens movie recommender (http://www.movielens.org). There are three versions of MovieLens data sets which contain approximately a hundred thousand (ml-100k), one million (ml-1m), and ten millions (ml-10m) of ratings respectively. Each version contains at least 20 ratings per user. The ratings are discrete and go from 1 (low rating) to 5 (high rating). More statistical details about each version can be found in the table below:

Table 3.1: MovieLens Datasets

Dataset Version	Statistics
ml-100k	100,000 ratings, from 943 users, on 1,682 movies
ml-1m	1,000,209 ratings, from 6,040 users, on 3,900 movies
ml-10m	10,000,054 ratings, from 71,567 users, on 10,681 movies

We also performed tests on food ordering data which contains 9,996,961 orders given by 473,504 distinct users from 3,421 distinct restaurants within the period of one year. The advantages of this dataset is that it contains both implicit and explicit preference information. In *Orders* table, there are 3,384,135 distinct user-restaurant pairs which also indicates the number of implicit preferences. There exist explicit ratings for only 670,834 among these distinct user-restaurant pairs . Our algoritm can work on a user-restaurant matrix, consisting of only ratings explicitly given by users for restaurants (explicit ratings), or consisting of only purchasing/order counts for each user-restaurant pair (implicit ratings), or both.

In order to be able to make tests, some preprocessing needed to be done to convert Orders table to a user-restaurant matrix. The first thing that must be handled is multiple explicit ratings to a particular restaurant given for different aspects: *the flavor of the food*, *the quality of the service*, and *the speed of delivery*. There are different approaches in literature to choose from, e.g. taking the average or the minimum of the three as the overall explicit rating. We preferred to choose taking the average of the three. Besides multiple-aspect ratings, a user can rate a restaurant more than once in different times. In these cases, we chose the last given rating as the overall rating since it represents the user's most up-to-date degree of satisfaction received from the specified restaurant. Indeed, if a user is exposed to a bad experience from his/her last order from a certain restaurant, he/she probably will never order from that restaurant again in the future.

At the first stage of the evaluation process, REDCAP is compared to other state-of-the art algorithms. Two metrics are used for comparison. The first one is the percentage of compatible preferences (3.4) and the second one is NDPM (3.3). For the percentage of compatible preferences, two kinds of approaches is selected. In the first approach, all pairs of items in test set are taken into account, whereas, in the second approach, only *good* vs *bad* item pairs only are taken into account.

In this process, each dataset is randomly divided into two subsets as training set and test set with the proportion of 80 percent over 20 percent. Test procedure is repeated five times for 5 fold cross validation where each experiment is perfomed with different training/test set pair and the average of the results are taken. The summarized results can be seen in Table 3.2.

Table 3.2: Comparisons of test results using explicit ratings only (80 percent/20 percent train/test split, 5-run average)

Algorithm	Dataset	Percentage of compatible preferences (Good-Bad pairs/All pairs)	NDPM (Lower is better)
REDCAP	MovieLens-100k	73,58/70,82	0,292
User-to-user CF (Cosine)	MovieLens-100k	74,23/71,47	0,286
User-to-user CF (PearsonCorrelation)	MovieLens-100k	74,09/71,35	0,286
User-to-user CF (TanimotoCoefficient)	MovieLens-100k	74,23/71,48	0,285
Funk-SVD (FeatureCount:14)	MovieLens-100k	73,51/70,85	0,291
Funk-SVD (FeatureCount:40)	MovieLens-100k	73,50/70,84	0,291
SlopeOne	MovieLens-100k	73,36/70,72	0,293
REDCAP	MovieLens-1m	75,61/73,14	0,268
User-to-user CF (Cosine)	MovieLens-1m	76,76/74,33	0,257
REDCAP	Food Orders	63,14/59,14	0,409
User-to-user CF (Cosine)	Food Orders	63,64/59,50	0,405

As can be seen from the table, the percentage of compatible preferences for good-vs-bad item pairs is bigger than the one for all item pairs as expected because it is relatively easier to differentiate a good item from a bad item. This metric is used to see the effect of the penalization scheme (Section 2) to recommendation quality, since, in penalization, bad items are specifically penalized to lower their scores down against good items.

At the second phase of the evaluation process, we tried to compare the performance of REDCAP to other algorithms' by considering both implicit and explicit data. In this phase, food ordering data is used since it contains both implicit and explicit information. Among the algorithms tested, REDCAP is the only one which can use both implicit and explicit ratings at the same time. For this reason, at the first place, REDCAP is run on the dataset and all user-item pairs for which prediction has been made are recorded into a file. Next, the other algorithms try to make predictions for the exact same user-item pairs that REDCAP made predictions for in previous step. The test procedure is explained step by step as follows:

a. Pick up a random user,
b. Hide 20 percent of explicit ratings for the selected user,
c. Two approaches are tried with respect to hiding implicit ratings. First, hide as many implicit items as the hidden explicit items. The second approach is to hide 20 percent of implicit ratings for the selected user. For example, if a user has 10 explicit ratings vs 20 implicit ratings, 2 out of 10 explicit ratings and 2 out of 20 implicit ratings are hidden in the first approach. On the other hand, 4 out of 20 implicit ratings are hidden whereas 2 out of 10 explicit ratings are hidden in the second approach.
d. Calculate the final scores for all held-out ratings according to REDCAP.
e. Compare predicted orderings for all hidden items to their known (reference) orderings pair by pair. If explicit ratings exist for both items of a pair, ordering by explicit rate values is considered as the reference ranking since explicit ratings are direct indicators of true preference and are more reliable than implicit preferences. Otherwise, ordering by implicit preference values is considered as the reference ranking.
f. Repeat the previous steps for a certain amount of time (e.g, an hour) and record all user-item pairs for which prediction has been made into a file. At the end, take the percentage of item pairs whose predicted rankings is compatible with reference rankings over all hidden item pairs as the final performance metric.
g. The other algorithms are tested with the exact user-item pairs saved in previous step for a healty comparison. In other words, users and items to be hidden are determined according to the saved file. Since the competing algorithms are run on explicit ratings only, number of items for which a score cannot be calculated will be more as compared to REDCAP. If a competing algorithm could not make a prediction for both items in an item pair whereas REDCAP could, the competing algorithm gets a minus for that item pair in overall performance calculation.

The summarized results can be seen in Table 3.3.

Table 3.3: Comparisons of test results using both implicit and explicit ratings

Algorithm	Percentage of compatible preferences	Rating hiding approach
REDCAP	%57	hide %20 of explicit ratings, hide as many implicit items as the hidden explicit items
User-to-user CF (Cosine)	%50	hide %20 of explicit ratings, hide as many implicit items as the hidden explicit items
REDCAP	%57	hide %20 of both explicit and implicit ratings
User-to-user CF (Cosine)	%45	hide %20 of both explicit and implicit ratings
REDCAP	%55	hide %30 of explicit ratings, hide as many implicit items as the hidden explicit items
User-to-user CF (Cosine)	%47	hide %30 of explicit ratings, hide as many implicit items as the hidden explicit items
REDCAP	%55	hide %30 of both explicit and implicit ratings
User-to-user CF (Cosine)	%42	hide %30 of both explicit and implicit ratings
REDCAP	%53	hide %40 of explicit ratings, hide as many implicit items as the hidden explicit items
User-to-user CF (Cosine)	%43	hide %40 of explicit ratings, hide as many implicit items as the hidden explicit items
REDCAP	%52	hide %40 of both explicit and implicit ratings
User-to-user CF (Cosine)	%39	hide %40 of both explicit and implicit ratings

We can see from the table that as the percentage of hidden items increases, the percentage of compatible preferences decreases as expected. On the other hand, even if the 40 percent of both explicit and implicit ratings are hidden, REDCAP does not fall under 50 percent hit rate. A classical collaborative filtering algorithm performs 50 percent and less as the percentage of hidden items increases. This is because of the fact that the hidden implicit ratings will increase as the percentage of hidden items increases and so will the number of items for which a score can not be calculated. In literature, the percentage of items that can be recommended among unseen items is called *coverage*. The higher the coverage of a recommender engine, the better the quality is. Coverage shows the degree to which recommendations cover the set of available items and the degree to which recommendations can be generated to all potential users. A recommender system with high coverage represents to the end user a more detailed and careful investigation of the product space, therefore an indicator of quality. If we use I to denote the set of available items and I_p to denote the set of items for which a prediction can be made, a basic measurement for prediction coverage can be given by:

$$coverage = \frac{|I_p|}{|I|} \tag{3.5}$$

Test results show that, REDCAP's coverage is always higher than other algorithms when both implicit and explicit ratings are used. Tests on explicit data only (MovieLens data) show that the coverage is nearly same for both REDCAP and other algorithms which is approximately 99.84 percent. Tests on food ordering data show that the coverage of the competing algorithms is 87.28 percent whereas REDCAP's coverage is 100 percent.

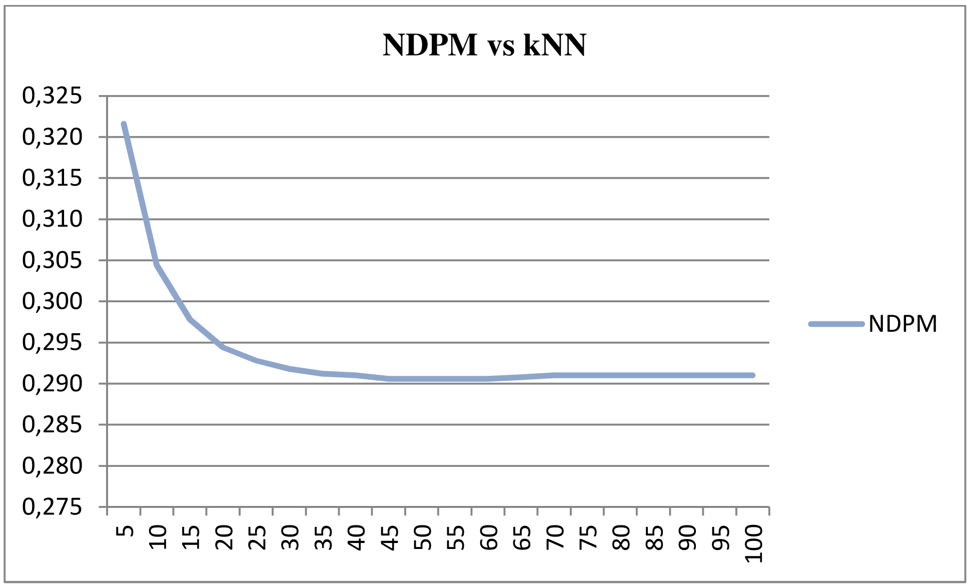
3.2.3 Determining Optimum Number of User Neighbors (kNN)

As explained in Section 2.2, neighbood-based recommenders like ours find users with similar tastes for a taret user. This set of users is usually called *nearest neighbors*. The size of the nearest neighbors set is an important parameter to decide. When it is selected too small, the accuracy can decrease dramatically. On the other hand, if it is selected too large, processing load increases which degrades the speed of the recommender. Besides, after some point, increasing the number of nearest neighbors does not affect the accuracy which means that there will be unnecessary processing after that point.

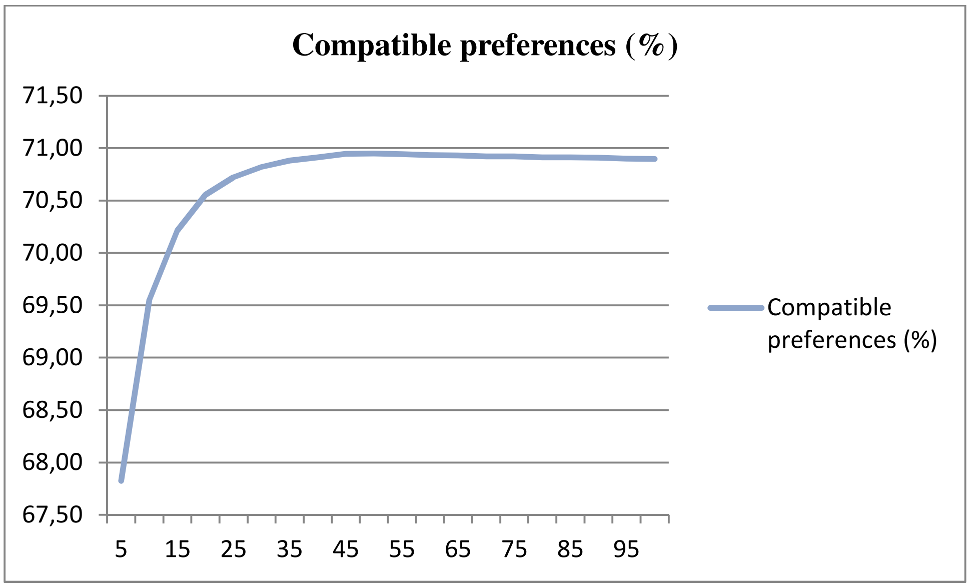
To determine an optimum number, we conducted some tests by increasing k by 5 at each step. As Figure 3-1 shows, there is a bending point (knee point) after which the accuracy does not increase much or at all. After this analysis we decided to select k as 30. All tests are conducted with 5-fold validation where each test is run five times on five different train/test set splits and the average is taken. More details about each test run can be found in Appendix A.

This optimum number can be different for various data sets from different domains. Selecting a fixed value for *k* apriori for every data set can affect the accuracy of the recommender badly. A knee finding algorithm can be employed beforehand to dynamically determine an appropriate *k* value for a particular dataset.

Figure 3-1 kNN's effect on accuracy (MovieLens dataset is used)



a) Accuracy metric is *NDPM*



b) Accuracy metric is *percent of compatible preferences*

3.2.4 Effectiveness of Implicit and Exlicit Confidence Weighting Scheme

In Section 2.2.4, we propose a confidence weighting scheme in calculation of final scores (see Equations (2.10) and (2.11)). In a simple approach, we can attach equal importance to both implicit and explicit scores in final merging step which means $w_{exp} = w_{imp} = 0.5$. To see how this scheme affects the accuracy, we performed tests on food ordering data which contains both implicit and explicit preferences. In simple approach, the percentage of compatible preference is 48,56 percent whereas in our weighting approach, it is 56,32 percent.

4. CONCLUSION

In this thesis, we introduced a novel method for recommending items, which can be easily applied to different domains where either implicit or explicit information exist. The accuracy of a recommender system is directly proportional to the amount of available data. In other words, the more ratings we have available for each item and for each user, the more specific and/or personalized can the generated recommendations be. In this respect, our tests prove that the accuracy (percentage of compatible preferences) is higher when incorporating implicit data. If only explicit ratings are used, accuracy performances are reasonably close for both REDCAP and the other algorithms. On the other hand, incorporating implicit data increases the coverage ratio. This means that REDCAP can recommend more items among the set of available items and to more users among all potential users.

On the other hand, there is a trade-off between accuracy and coverage. Using implicit data in addition to explicit ratings increases the accuracy but this also increases the complexity of the algorithm and the time that one recommendation cycle takes. The increase in algorithm complexity results from the additional procedure to predict item scores by using implicit ratings and merging these scores with explicit scores by borda count at the final stage. Besides, since implicit ratings are much more than explicit ratings in number, predicting item scores using implicit ratings takes relatively more time. To overcome this disadvantage, techniques such as precomputing implicit similarities beforehand can be incorporated.

REFERENCES

Books

Amatriain, X., Jaimes, A., Oliver, N., and Pujol, J.M., 2011. Data mining methods for recommender systems. *Recommender Systems Handbook*. US:Springer, pp. 39-71.

Burke, R., 2007. Hybrid web recommender systems. *The Adaptive Web*. Berlin:Springer pp. 377–408.

Desrosiers, C. and Karypis, G., 2011. A comprehensive survey of neighborhood-based recommendation methods. *Recommender Systems Handbook*. US:Springer, pp. 107-144.

Koren, Y., and Bell, R., 2001. Advances in collaborative filtering. *Recommender Systems Handbook*. US:Springer, pp. 145-186.

Liu, B., 2011. Information retrieval and web search. *Web Data Mining*. Berlin:Springer, pp. 211-268.

Lops, P., Gemmis, M., and Semeraro, G., 2012. Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook*. US:Springer, pp. 73-105.

Meyer, F., Fessant, F., Clérot, F., and Gaussier, E., 2012. Toward a new protocol to evaluate recommender systems. *Workshop on Recommendation Utility Evaluation: Beyond RMSE (RUE 2012)*. Dublin, p. 9.

Shani, G. and Gunawardana, A., 2011. Evaluating recommendation systems. *Recommender Systems Handbook*. US:Springer, pp. 257-297.

Periodical Publications

Adomavicius, G. and Tuzhilin, A., 2005. 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*. **17**(6), pp.734-749.

Bennett, J. and Lanning, S., 2007. The netflix prize. *Proceedings of KDD Cup and Workshop 2007*. p. 35.

Blei, D.M., Ng, A.Y., and Jordan, M.I., 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*. **3**(2003), pp. 993-1022.

Goldberg, D., Nichols, D., Oki, B.M., and Terry, D., 1992. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*. **35**(12), pp. 61-70.

Herlocker, J.L., Konstan, J.A., Terveen, L.G., and Riedl, J.T., 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*. **22**(1), pp. 5-53.

Paterek, A., 2007. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop 2007*, pp. 5-8.

Rao, K.N. and Talwar, V. G., 2008. Application domain and functional classification of recommender systems: A survey. *DESIDOC Journal of Library and Information Technology*. **28**(3), pp. 17-35.

Salakhutdinov, R., Mnih, A., and Hinton, G., 2007. Restricted boltzmann machines for collaborative filtering. *Proceedings of the 24th Annual International Conference on Machine Learning*, pp. 791-798.

Other Publications

Funk, S., Netflix update: Try this at home, 2006. [online] http://sifter.org/~simon/journal/20061211.html. [10 November 2013]

LensKit, LensKit Recommender Toolkit, 2013. [online] http://lenskit.grouplens.org. [2 November 2013]

Meyer, F., 2012. Recommender systems in industrial contexts. [online] http://arxiv.org/ftp/arxiv/papers/1203/1203.4487.pdf. [6 October 2013].

MovieLens, MovieLens Data Set, 2013. [online] http://grouplens.org/datasets/movielens. [6 October 2013]

APPENDICES

APPENDIX A: TABLES

Table A.1 5-run test results on MovieLens dataset showing the effect of the number of nearest neighbors to accuracy (Accuracy metric is NDPM)

NDPM value						
kNN	Run 1	Run 2	Run 3	Run 4	Run 5	Average
5	0,318	0,319	0,334	0,317	0,320	0,322
10	0,297	0,302	0,316	0,304	0,303	0,304
15	0,289	0,297	0,309	0,298	0,296	0,298
20	0,287	0,293	0,305	0,294	0,293	0,294
25	0,285	0,290	0,304	0,293	0,292	0,293
30	**0,285**	**0,288**	**0,302**	**0,292**	**0,292**	**0,292**
35	0,284	0,287	0,302	0,291	0,292	0,291
40	0,283	0,287	0,302	0,291	0,292	0,291
45	0,283	0,287	0,301	0,290	0,292	0,291
50	0,283	0,287	0,301	0,290	0,292	0,291
55	0,283	0,287	0,301	0,290	0,292	0,291
60	0,283	0,287	0,301	0,290	0,292	0,291
65	0,283	0,287	0,302	0,290	0,292	0,291
70	0,283	0,288	0,302	0,290	0,292	0,291
75	0,283	0,288	0,302	0,290	0,292	0,291
80	0,283	0,288	0,302	0,290	0,292	0,291
85	0,283	0,288	0,302	0,290	0,292	0,291
90	0,283	0,288	0,302	0,290	0,292	0,291
95	0,283	0,288	0,302	0,290	0,292	0,291
100	0,283	0,288	0,302	0,290	0,292	0,291

Table A.2 5-run test results on MovieLens dataset showing the effect of the number of nearest neighbors to accuracy (Accuracy metric is the percentage of compatible preferences considering only good-vs-bad item pairs)

Percentage of compatible preferences (Only *Good* vs. *Bad* item pairs are considered)						
kNN	Run 1	Run 2	Run 3	Run 4	Run 5	Average
5	70,87	70,48	68,86	70,83	70,84	70,38
10	73,12	72,44	70,85	72,23	72,54	72,23
15	73,94	72,94	71,39	72,99	73,13	72,88
20	74,16	73,39	71,78	73,40	73,53	73,25
25	74,39	73,74	71,98	73,53	73,71	73,47
30	**74,49**	**73,88**	**72,08**	**73,74**	**73,72**	**73,58**
35	74,59	73,99	72,09	73,77	73,72	73,63
40	74,66	73,95	72,07	73,82	73,74	73,65
45	74,69	73,97	72,21	73,89	73,68	73,69
50	74,68	74,02	72,21	73,84	73,67	73,68
55	74,70	74,02	72,18	73,91	73,58	73,68
60	74,66	73,96	72,17	73,92	73,65	73,67
65	74,69	73,94	72,14	73,95	73,63	73,67
70	74,68	73,94	72,15	73,91	73,62	73,66
75	74,71	73,96	72,17	73,86	73,61	73,66
80	74,70	73,94	72,15	73,85	73,61	73,65
85	74,71	73,92	72,18	73,85	73,61	73,65
90	74,67	73,94	72,16	73,86	73,63	73,65
95	74,68	73,89	72,13	73,86	73,64	73,64
100	74,66	73,89	72,15	73,89	73,60	73,64

Table A.3 5-run test results on MovieLens dataset showing the effect of the number of nearest neighbors to accuracy (Accuracy metric is the percentage of compatible preferences considering all item pairs)

Percentage of compatible preferences (All item pairs are considered)						
kNN	Run 1	Run 2	Run 3	Run 4	Run 5	Average
5	68,20	68,09	66,57	68,29	67,98	67,82
10	70,28	69,83	68,43	69,57	69,66	69,55
15	71,10	70,34	69,07	70,20	70,35	70,21
20	71,32	70,75	69,47	70,57	70,66	70,55
25	71,51	71,04	69,60	70,66	70,80	70,72
30	**71,54**	**71,17**	**69,76**	**70,83**	**70,79**	**70,82**
35	71,63	71,30	69,79	70,88	70,80	70,88
40	71,69	71,31	69,82	70,90	70,84	70,91
45	71,72	71,31	69,89	70,98	70,84	70,94
50	71,73	71,33	69,90	70,97	70,81	70,95
55	71,75	71,32	69,90	70,99	70,75	70,94
60	71,71	71,29	69,86	71,01	70,80	70,93
65	71,72	71,26	69,85	71,04	70,79	70,93
70	71,72	71,25	69,83	71,01	70,80	70,92
75	71,73	71,25	69,83	70,99	70,80	70,92
80	71,74	71,23	69,80	70,97	70,81	70,91
85	71,74	71,21	69,83	70,98	70,80	70,91
90	71,72	71,23	69,81	70,99	70,81	70,91
95	71,72	71,20	69,78	70,98	70,82	70,90
100	71,71	71,20	69,79	70,98	70,80	70,90