MSc Computer Science
Department of Computer Science and Information Systems
Birkbeck, University of London

Project Proposal

A Recommendation Engine

Amy Peters

This proposal is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service.

Abstract

I propose to implement a recommendation system, using collaborative filtering techniques, for the interactive online community *Deep Underground Poetry*. Using data which has been collected from members, I intend to compute a list of suggested reading which is personalised to each user. The engine will need to respond quickly to changes in data in order to give users up to date recommendations. The overall aim of the system is to promote interaction between members by increasing the number of views, ratings and comments each poem receives, by guiding people towards poems which they should enjoy.

Supervisor: Keith Mannock

April 2011

Introduction

Theory

Development

Introduction

1.1 Giving the People What They Want

In August 2009 I re-launched an online creative writing community called *Deep Underground Poetry* (*DU Poetry*) [1], which I originally started in 1999. For ten years, the website had been offline and the internet had changed enormously during this time. However, through re-establishing *DU Poetry* over the last few years, it became apparent that one of the key challenges still remained; finding ways to encourage people to read and comment on other peoples' submissions. Giving members the feedback and attention they crave is a key part of creating a thriving community and retaining an active membership. Discussions related to this issue frequently crop up on the *DU Poetry* "suggestions" forum. In addition, a homepage poll showed that feedback from others was important to more than half of respondents [2].

Fig.1 Screen grab from *DU Poetry*, take on 8th April 2011.

There are a number of ways in which online poetry communities have tried to increase commenting levels. Some websites force their members to make comments, so as to artificially increase the number of comments made. For example, *GS Poetry* [2] requires members to gain credits by making comments before they are allowed to "showcase" a poem. *GotPoetry* [3] implements a "karma" system, to stop less active members from rating poems without leaving a comment. *DU Poetry* offers incentives, like extra exposure for their poems, to "top-critiquers" (members who make lots of useful comments). Despite these efforts, the average number of comments that each poem receives still remains relatively low; currently less than 1.5 comments per poem on *DU Poetry*. In addition, the risk of enforcing measures is that the quality, usefulness and comprehensiveness of the feedback is diminished. Keen to explore more organic ways to increase the number of comments; I created another *DU Poetry* poll asking whether people were more likely to comment on poems they liked, or poems they disliked [4]. To date, 100% of respondents have answered that they more often comment on poems that they like. My challenge is to find a way for people to more easily and quickly discover poems that they might actually want to read and comment on. I also hope to address the issue of maintaining exposure for older submissions, which no longer appear on the first few pages of listings, and can therefore go undiscovered by new members.

Developments in technology (hardware, software and connection speeds) have made it possible to collect vast amounts of data from users and with this has come the increased use of collaborative filtering (CF). CF is a way of analysing information with the filtering process shared among a large group of people. Unlike conventional media like newspapers and television, where there are a few editors controlling output, the CF model can have infinitely many editors and improves as the number of participants increases [5]. If I can manipulate data about which poems my members respond to positively, then I can then use this to provide them with a list of personalised recommendations (poems they haven't read yet, but should theoretically enjoy).

1.2 Existing Recommendation Systems

Many successful websites use collaborative filtering techniques to implement some form of recommendation engine. This could be something which looks relatively simple on the surface, such as keeping track of positive votes (upvotes or “thumbs up”) and negative votes (downvotes or “thumbs down”). For example, *Reddit* [6], *Delicious* [7] and *Digg*[8] gauge the popularity of user posted links, based on analysing positive and negative votes. However, the algorithms involved in this process may be far more complex then they appear at first glance. Although these websites keep the specifics private (to avoid abuse of the system), much hypothesising takes place on forums and blogs as to what’s involved in the underlying algorithms. Suggestions include; the domain of the link, the profile of the submitter, profiles of the voters, similarity to other links, the source of the votes and the number of views and comments the link receives [9].

When Greg Linden, the creator of *Amazon*’s recommendation engine, left *Amazon* in 2002 he reported that over 20% of sales came from personalized recommendations, the figure is estimated to be even higher today [10]. *Amazon*’s recommender system suggests products to customers based on their past purchases and the items they currently have in their shopping basket. The idea is similar to that of placing impulse buy items near a supermarket checkout, but *Amazon*’s suggestions are targeted to each customer. An item-to-item collaborative filtering algorithm was developed in order to handle the massive data set involved and achieve the scalability and performance they required. The key to its success was that the computationally expensive task of creating the “similar items” table takes place offline. The online part (looking up similar items for a particular user) scales independently of the size of the overall data set, so the system remains fast despite the huge amount of data involved [11]. The algorithm also performs well with limited user data; it is able to produce recommendations based on just a few items.

Netflix is an American company that offers on-demand video streaming and online requests for video rentals. In 2006, *Netflix* deemed their film recommendation feature so critical to their business that they launched the “Netflix Prize” competition. They offered one million dollars to the winning team, if they could beat the accuracy of their “Cinematch” recommendation system by more than 10%. The competition ended in 2009 and the winning team *BellKor's Pragmatic Chaos* published a paper outlining the techniques they used. This included identifying a number of temporal effects which influenced the ratings a user gave; movie bias (film popularity fluctuations), user bias (fluctuations in how harshly the user rated things) and user preferences (for example, a fan of one genre may later become a fan of another, related genre) [12]. The “Netflix Prize” sparked a surge of interest and development in the area of recommendation systems [13]. Many of the papers cited in this proposal were published as a direct result of the progress made by those who took part.

1.3 Aims and Objectives

I intend to provide members with a personalised list of recommended reading and also display a selection of “poems like this” on each poem's page. My key objectives are:

- Tune an algorithm to give quantitatively “accurate” recommendations (see section 2.3).
- Look for patterns in users' tastes in relation to characteristics like their age and location.
- Build an efficient and scalable system which reacts quickly to new information.
- Develop the system in response to implicit and explicit feedback from users.

Theory

2.1 An Introduction to Collaborative Filtering

Recommendation engines use collaborative filtering (CF) as a method of making automatic predictions about which items will suit a user (filtering), by collecting information about the past actions of many users (collaboration). CF techniques don't rely on specific domain knowledge and can be applied to sparse data sets, thereby avoiding the need for comprehensive data collection (it should not be necessary for a user to evaluate a vast number of items). Yehuda Koren (one of the members of the winning *Netflix Prize* team) outlines two main approaches to CF; the neighbourhood approach and latent factor models [14], these can also be described as memory-based and model-based [15].

2.2.1 The Neighbourhood Approach (Memory-Based)

Neighbourhood based models are currently the most common approach to CF [14]. They use some form of the *k*-nearest neighbours algorithm (*k*-NN) to compute the relationship between items, or between users. *k*-NN is a simple machine learning algorithm where an item is classified by the average of its *k* nearest neighbours, where *k* is a positive integer, typically small (for example, if *k* = 1 then the unknown rating would get the value given by the most similar user who has rated that item). The best value for *k* is application specific and can only be found by experimenting with the data set. It is typical for a weighted average to be used, with the closest (most similar) neighbours contributing more. To work out which are the closest neighbours, a distance metric is used. A simple example is Euclidean Distance which measures a straight line distance between two points (X and Y) using the following algorithm:

$$\mathrm{d} = \sqrt{\sum_{i=1}^{k} (X_i - Y_i)^2}$$

Applying this to real data involves comparing two items (as if they were axes on a graph), using the ratings given by users who have ranked both items. I have used sample data from *DU Poetry* with ratings computed based on user actions (member and poem IDs have been concealed for data protection purposes).

	User 1	User 2	User 3	User 4
Poem A	1	3	2	3
Poem B	2	3	0	1

Fig. 2 Sample data from *DU Poetry*

Poem A has coordinate (1, 3, 2, 3) and poem B has coordinate (2, 3, 0, 1) so the Euclidean Distance between poem A and poem B is:

$$d = \sqrt{(1 - 2)^2 + (3 - 3)^2 + (2 - 0)^2 + (3 - 1)^2}$$

$$= \sqrt{1 + 0 + 4 + 4} = 2.236 \text{ (rounded)}$$

An even simpler alternative to Euclidean Distance is Manhattan Distance, where the differences between the co-ordinates are absolute values (not squared or square rooted). The Manhattan Distance between poem A and poem B is 3. Both of these distance metrics can be generalised as Minkowski Distance, where *r* in the following algorithm can be any number. The greater the value of *r*, the more that a difference in one dimension will influence the total difference [16]:

$$\mathrm{d} = \left(\sum_{i=1}^{k}(X_i - Y_i)^r\right)^{\frac{1}{r}}$$

When we are working with a data set where there are unknown values, it is necessary to compensate for the fact that the distance is likely to be shorter where users have lots of ratings in common; creating a false sense that they are more similar then they actually are. This can be remedied by dividing the total distance by the number of dimensions (items which both users have rated).

An issue with Minkowski Distance is that it doesn't allow for differences in the way users rate items, for example, one user may rate things more generously than another (referred to as "grade inflation"). The Pearson correlation coefficient is a more sophisticated way of determining similarity, which can correct this problem [17]. It is a measure of the correlation between two variables ranging from -1 to 1 inclusive; 1 indicates perfect agreement (all data points lying on a line for which *Y* increases as *X* increases) and -1 indicates perfect disagreement (all data points lie on a line for which *Y* decreases as *X* increases). A value of 0 implies that there is no linear correlation between the variables.

A number of other distance metrics exist including; cosine similarity which measures the similarity between two vectors by finding the cosine of the angle between them and Jaccard index which measures the similarity between sets. Finding the most effective distance metric to use will be a key part of developing the algorithms for my recommendation engine.

K-NN suits collaborative filtering well because it's an "online technique", meaning that new data can be added without having to retrain the data or estimate new parameters [17]. This means the system is able to provide updated recommendations immediately after a user has entered new feedback. However, for new items the system cannot respond immediately, as it needs to learn the new parameters [14]. This fits well with my intentions as the engine will need to be able to provide recommendations quickly to active new members, but won't need to promote the most recent poem submissions (as they will start out featuring prominently in listings). Another advantage of neighbourhood models is that they are explainable; it's possible to identify which past user actions have been the most influential in predicting which items to recommend [14, 17]. This can enrich the experience for the user in a number of ways; giving them information about how and why the recommendations were selected, building confidence in the system and allowing them to tell the system when it's wrong [18].

However, *k*-NN suffers from a major weakness; its relatively high computational cost. This is because every user must be compared to every other user in order to determine which are its closest neighbours [17], this limits the scalability of the neighbourhood approach.

2.2.2 Latent Factor Models (Model-Based)

Model-based approaches include neural networks, Bayesian networks and models using the linear algebra technique of singular value decomposition (SVD), in order to factorise a user-item ratings matrix. Recently, SVD models have become popular for collaborative filtering due to their accuracy and scalability [14, 19]. Rather than directly setting out to make predictions, instead they use machine learning to characterise the data and uncover the underlying causes (factors) that work in combination to describe the data set [17].

The idea is to decrease the amount of computation required by reducing the size of the matrix. Instead of comparing every object (item or user) to every other object, instead you define each object by a number of factors. For example, when the items are poems, factors which might emerge may be related to how funny or sad a poem is. Rather than having to manually define these characteristics, the algorithm finds patterns in the existing data and discovers the generalisations (also called eigenvectors) for itself. Imagine a data set of 1000 users and 15000 items; this would involve 1000 x 15000 = 15000000 computations. However, if each object had just 40 factors, that could be used to compare it to other objects, then it would only require 40 x (1000 + 15000) = 640000 computations [20]. Given a m x *n* matrix *A* with rank *r* (the number of factors which we want to define items by), the singular value decomposition *SVD(A)* is defined as:

$$SVD(A) = U \times S \times V^T$$

Matrix *S* is a diagonal matrix called the *singular* having only *r* nonzero entries (r x r), which means the effective dimensions of the orthogonal matrices *U* and V are *m x r and r x n* respectively. The first *r* columns of *U* and *V* represent the eigenvectors associated with the *r* factors of AA^T and A^TA. For the matrix *A* it can be said that the columns corresponding to the *r* values span the column space in *U* and the row space in *V*. The following illustration represents this on a very small scale:

$$\begin{array}{c} \\ \text{Poem 1} \\ \text{Poem 2} \\ \text{Poem 3} \end{array} \begin{array}{c} \begin{array}{cc} \textit{factor 1 (sad)} & \textit{factor 2 (funny)} \end{array} \\ \begin{pmatrix} 0 & 3 \\ 3 & 2 \\ 4 & 1 \end{pmatrix} \end{array} \times \begin{array}{c} \\ f1 \\ f2 \end{array} \begin{array}{c} \begin{array}{cc} \textit{User 1} & \textit{User 2} \end{array} \\ \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \end{array} = \begin{pmatrix} 0x2 + 3x1 & 0x0 + 3x3 \\ 3x2 + 2x1 & 3x0 + 2x3 \\ 4x2 + 1x1 & 4x0 + 1x3 \end{pmatrix} = \begin{array}{c} \\ \text{Poem 1} \\ \text{Poem 2} \\ \text{Poem 3} \end{array} \begin{array}{c} \begin{array}{cc} \textit{User 1} & \textit{User 2} \end{array} \\ \begin{pmatrix} 3 & 9 \\ 8 & 6 \\ 9 & 3 \end{pmatrix} \end{array}$$

For this example (using an *r* value of 2) the algorithm has identified the two most prevalent characteristics by which to describe users and poems (sad and funny). Each poem is given a value for each characteristic and every user has a value representing how much they like that characteristic. By multiplying the two matrices together we get a representation of the original matrix (which is a “marks out of ten” rating made by each user for each poem). We can see that User 2 has given Poem 1 a high mark and this was factorised into both being given a high value for the factor 2 characteristic “funny”.

One possible advantage of the *low-rank* approximation resulting from SVD is that the resulting data may in fact be better than the original full matrix, as the process may actually filter out some unwanted “noise” from the data [21]. However, the opposite may also be true, especially when applying SVD to collaborative filtering, due to the high proportion of missing data. Addressing only the known values is prone to overfitting (where there aren’t enough values to consider, so a random error or “noise” value may be given undue influence over results [22]). Variations of SVD have been developed to try and address this problem [23]. One notable example is that of incremental SVD, devised by Simon Funk during the “Netflix Prize” competition [20]. He incorporates a technique for regularising the model in order to identify abnormal values and avoid overfitting.

2.2.3 Hybrid Approach

When the *Bellkor* team won their progress award during the “Netflix Prize” competition in 2007, their solution consisted of blending the results of 107 algorithms [24]. They used the approaches I have described above, along with Restricted Boltzmann Machines (a type of neural network [25]) and regression models. Their research indicates that predictive accuracy can be greatly improved by blending different approaches, instead of refining a single technique.

2.3 Accuracy Testing

If a number of known values are removed from a data set and then computations are performed on the remaining data, the resulting values can be compared with the actual values in order to access the accuracy of the algorithm. In statistics, the mean squared error (MSE) is one way of quantifying the difference between the estimated values and the true values. It works by squaring the difference between each of the actual values and the estimations, then adding them together and diving by the total number of predictions. For example, if an algorithm predicted two item ratings as 2.5 and 3, and the actual values were 4 and 2 then the MSE would be:

$$(1.5 \times 1.5) + (1 \times 1) = 2.25 + 1 = 3.25 / 2 = 1.625$$

Root mean square error (RMSE) is slight variation on MSE and is useful when communicating results to others because the RMSE is measured in the same units as the data, rather than in squared units, so it’s representative of the size of an actual "typical" error. The RMSE of the values above is simply the square root of the result, so 1.275. The “Netflix Prize” used RMSE as the accuracy metric for judging the entries [13].

An alternative to this is mean absolute error (MAE) which is less sensitive to large prediction errors. It is simply the average of the absolute errors, so for the values above:

$$(1.5 + 1) / 2 = 1.25$$

By dividing the known *DU Poetry* data into segments, and using these estimators on each part independently, I will be able to assess the quantitative accuracy of my algorithms.

Development

3.1 The Existing System and Data Set

Like any website starting out on a new domain name (without a budget for promotion) it has taken *DU Poetry* time to gain popularity and establish reasonable search engine rankings. The amount of new members signing up continues to increase month on month, with 199, 252 and 279 new members joining in January, February and March of 2011 respectively. The number of active members; those logging in within the last two months, is approximately 1000 and the total number of poems published exceeds 15000.

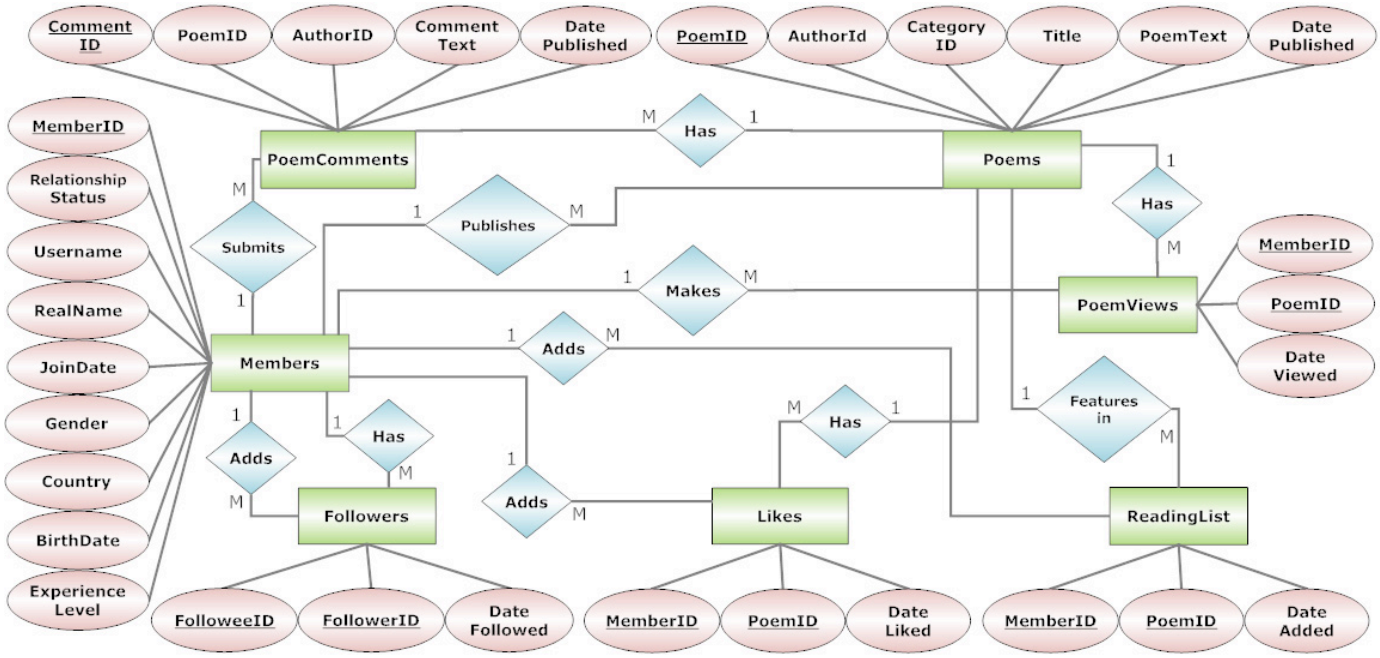

Fig. 3 An entity-relationship model using Chen notation, showing relevant tables and attributes in the existing *DU Poetry MySQL* database.

There are a number of features which I have implemented in order to gauge a member's reaction to a poem. These include an anonymous "like this?" option and a personal reading list, to which a member can add their favourite poems. In addition, a member can "follow" another member poet, this means they receive a message in their updates feed each time that member makes a new submission. Poem views are also logged, in order to keep track of which members have viewed which poems. Below is a summary of each of these features, along with the current usage statistics (from the website's *MySQL* database, queried on 11th April 2011):

Feature	**Launch Date**	**Total Entries**	**Entries Last Month** **(March '11)**
Reading List	30th December 2009	3749	490
Poem Views	11th July 2010	63976*	12691*
Followers	11th July 2011	2514	425
Likes	25th February 2011	3855**	2328**

* I have only included poem views made by members (counting only the most recent for any given poem), anonymous visitors are not included.

** "Likes" which also have an associated reading list entry are not included; a reading list entry takes precedence over a "like".

During the last month, since the "like" feature has been launched, there has been a significant increase in the amount of data collected. Over 22% of the time when a member views a poem it results in a positive action; a "like" or reading list entry.

I have always been keen to avoid a “marks out of five” style ratings system for poems, because it doesn’t fit well with the creative nature of the content. There is also strong evidence that five-star ratings systems don’t work. *YouTube* has abandoned its five star ratings system, switching instead to a simpler "Like / Don't Like" model, because almost all videos were receiving the maximum (five stars) rating from users [26]. For the *DU Poetry* recommendation engine, I intend to use the data which I currently collect in order to “simulate” poem ratings. For example, a followed author's poems may count as a rating of 1, a “like” count as a 2 and a reading list entry count as a 3. Although poem views may not have a major impact on the algorithm, they are useful in determining which poems a user has already looked at, so as not to recommend them poems which they have already read.

3.2 Implementation and Operational Issues

Although *DU Poetry* is coded primarily in Perl, I have decided to write my project in Scala because I would like to learn a new programming language. Scala is widely considered a reliable and scalable language, which has grown in popularity very quickly since its release in 2003, and has been adopted by major websites like *Twitter* and *LinkedIn* [27]. I am aware that it may take longer and also be more challenging to write code in an unfamiliar language.

Integrating my recommendation engine into the current *DU Poetry* website may present some complications. One consideration is how best to fit the Scala code into the current Perl HTML template system. Furthermore, the website is currently hosted on a shared web server and there are limits in place restricting the maximum number of simultaneous database connections. I also have very little control over the database settings. If this presents a problem I may need to implement a separate database structure for the recommendation engine, or move the website onto a semi-dedicated (virtual private) hosting package.

The user testing aspect of development may also present challenges, as I will be relying on a core membership of dedicated members to trial the system and provide me with timely feedback. Depending on how complete and user friendly the system is during the development process, I may have to limit the number of members involved in testing. In which case, along with managing the expectations of volunteers, I will also have to deal with any social issues which emerge, as a result of choosing the participants.

The sparse nature of the data set (outlined in 3.1) may restrict my choice of suitable algorithms. However, if growth continues at the current rate, then the amount of data collected should more than double over the next three months, with the number of “likes” set to more than triple. This issue may also be helped by the fact that a “followed” poet can contribute multiple ratings because every poem by the “followed” author is included. The average number of poems by each member, currently being followed by at least one other member, is 24.

3.3 Results and Analysis

It is recognised that accuracy metrics (outlined in section 2.3) can only partially evaluate a system; user satisfaction (for example, the diversity of recommendations given and the user’s trust in the system) are increasingly seen as important [15]. Throughout development I will be making full use of the fact that I have real data and real people to interact with, by collecting data both implicitly and explicitly. Direct feedback will be requested from users to help ascertain how useful each recommendation was and at the end of the process I will produce a questionnaire to assess the general response of participating members.

I also intend to produce statistics from tracking the actions taken (views, “likes”, comments and reading list entries) on poems which have been recommended. Benchmarks for assessing the results will be calculated from existing response levels, for example, to date there are 62550 recorded poem visits by members, and 13441 (or 21.5%) of these resulted in a comment being made (not including members viewing or adding comments to their own poems). Another consideration is the submission dates of the poems being recommended. I intend to compare the ratio of old and new with that of the current levels, to see if more exposure for older poems has been achieved. A sizeable 37% of poem views on *DU Poetry* during the last week relate to poems submitted during this time, despite them making up only 3% of the total poems.

3.4 Time Line

WEEK	KEY TASKS
1	Learn Scala basics
2	Code up first iteration of test algorithms using data snapshot.
3	Create front end for testing and collecting feedback from users.
4	Implement and collect results from first iteration code, address any front end usability issues raised by users.
5	Write code for second iteration.
6	Implement and collect results from second iteration, work out solution for efficient access to live data (for the system and for users).
7	Analyse what part user characteristics (age, location etc.) play in ratings, and see if the algorithms can be improved using this information, produce third iteration code.
8	Implement and collect results from third iteration, implement data access (and real time updates) solution.
9	Study results and feedback so far and produce a final iteration code.
10	Implement and collect results from final iteration, produce questionnaire.
11	Look at scalability issues to future proof the recommendation engine, such as data mining optimisations and cloud computing.
12	Evaluate the project; collect questionnaires and produce graphs and statistics of all results so far.
13	Write up project.

References

[1] Deep Underground Poetry, http://deepundergroundpoetry.com

[2] GS Poetry, http://www.gspoetry.com

[3] GotPoetry, http://www.gotpoetry.com

[4] Deep Underground Poetry, *Deep Underground Poetry Poll: Which do you most often comment on?,* http://deepundergroundpoetry.com/polls/12

[5] Read Write Web, *Collaborative Filtering: Lifeblood of the Social Web*, http://www.readwriteweb.com/archives/collaborative_filtering_social_web.php

[6] Reddit, http://www.reddit.com

[7] Delicious, http://www.delicious.com

[8] Digg, http://digg.com

[9] Rand Fishkin , *Everything in the Digg, Reddit & Netscape Algorithms*, http://www.seomoz.org/blog/everything-in-the-digg-reddit-netscape-algorithms, 2006

[10] Geeking with Greg, 35% of sales from recommendations, http://glinden.blogspot.com/2006/12/35-of-sales-from-recommendations.html, 2006

[11] G. Linden, B. Smith, J. York, *Amazon.com recommendations: Item-to-item collaborative filtering*, IEEE Internet Computing, 2003.

[12] Robert M. Bell, Yehuda Koren and Chris Volinsky, *The BellKor 2008 Solution to the Netflix Prize,* AT&T Labs – Research, 2008

[13] James Bennett, Stan Lanning, *The Netflix Prize*, Netflix, 2007.

[14] Yehuda Koren, *Factor in the Neighbors: Scalable and Accurate Collaborative Filtering*, Yahoo! Research, 2009

[15] Wikipedia, *Collaborative filtering*, http://en.wikipedia.org/wiki/Collaborative_filtering

[16] Ron Zacharski, *A Programmer's Guide to Data Mining*, http://guidetodatamining.com

[17] Toby Segaran, *Programming Collective Intelligence: Building Smart Web 2.0 Applications*, O'Reilly, 2007.

[18] Nava Tintarev, Judith Masthoff, *A Survey of Explanations in Recommender Systems*, University of Aberdeen, 2007.

[19] Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl, *Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems*, University of Minnesota, 2002

[20] Simon Funk, *Netflix Update: Try This At Home*, http://sifter.org/˜simon/journal/20061211.html, 2006.

[21] M. W. Berry, S. T. Dumais, G. W O'Brian, *Using Linear Algebra for Intelligent Information Retrieval*, *SIAM Review*, 1994

[22] Wikipedia, *Overfitting*, http://en.wikipedia.org/wiki/Overfitting

[23] Arkadiusz Paterek, *Improving regularized singular value decomposition for collaborative filtering*, Warsaw University, 2007

[24] Robert Bell, Yehuda Koren, Chris Volinsky, *The BellKor solution to the Netflix Prize*, AT&T Labs – Research, 2007

[25] Wikipedia, *Boltzmann machine*, http://en.wikipedia.org/wiki/Boltzmann_machine

[26] The Official YouTube Blog, *Five Stars Dominate Ratings*, http://youtube-global.blogspot.com/2009/09/five-stars-dominate-ratings.html, 2009

[27] The Scala Development Team, *Scala in the Enterprise*, http://www.scala-lang.org/node/1658