Three complementary approaches to context aware movie recommendation

Hossein Rahmani
Leiden Institute of Advanced Computer Science, Universiteit Leiden,
hrahmani@liacs.nl

Beau Piccart
Department of Computer Science, Katholieke Universiteit Leuven,
beau.piccart@cs.kuleuven.be

Daan Fierens
Department of Computer Science, Katholieke Universiteit Leuven,
daan.fierens@cs.kuleuven.be

Hendrik Blockeel
Department of Computer Science, Katholieke Universiteit Leuven,
blockeel@liacs.nl

ABSTRACT

We describe three different approaches to the Context Aware Movie Recommendation (CAMRa) challenge; each approach is based on different machine learning techniques. The results obtained with the three techniques are compared.

1. INTRODUCTION

The CAMRa challenge (Context Aware Movie Recommendation) [5] is about building models for recommending movies to users, taking certain background information ("context") into account. For more information on the datasets and tasks, see [5]. In this paper we address the "social" track. We formulate the task as a machine learning problem as follows. **Given:** a set of ratings that users have given to movies, plus information about which users are friends, which actors / directors are involved in which movies, which users have commented on / reviewed movies or actors, and more such background information; **Learn:** a function f that, from a user-movie pair (u, m) and some context (background information about the user and movie) C, predicts a rating $r(u, m)$. After predicting for a particular user his/her rating for all movies, the highest scoring movies are recommended.

We next describe three machine learning approaches, which differ w.r.t. the methods used and information exploited.

2. MACHINE LEARNING APPROACHES

2.1 Approach 1: a k-NN based approach

Our first approach uses a k-nearest neighbor classifier [1]. For each user u_i, first, we find the k most similar users to u_i ($MSU(u_i)$), then we rate each movie m_j with formula 1:

$$r(u_i, m_j) = \sum_{\forall u_k \in MSU(u_i)} UMIV(u_k, m_j) \quad (1)$$

$UMIV(u_k, m_j)$ (User-Movie-Interest-Value) indicates how much user u_k is interested in the movie m_j. It is not identical to the observed rating (if any); see Section 2.1.1 for its definition. The set of k most similar users, $MSU(u_i)$, obviously depends on the notion of similarity. Different user similarity metrics will be provided in the next sections.

2.1.1 User Movie Interest Value

Let U be the set of all users, and M the set of all movies, in the dataset. We want to estimate how much user $u_i \in U$ is interested in movie $m_j \in M$. To this aim, we integrate the information available in the entities: "Favorites", "Movie Comments", "Reviews" and "Review Ratings" into a matrix $UMIV$. Each cell $UMIV(u_i, m_j)$ shows how much user u_i is interested in the movie m_j and is calculated based on formula 2:

$$UMIV(u_i, m_j) = F(u_i, m_j) + C(u_i, m_j) + R(u_i, m_j) + Ra(u_i, m_j) \quad (2)$$

$F(u_i, m_j)$ represents the "Favorites" entity; it returns $+1$ if the user u_i likes the movie m_j, -1 if u_i dislikes m_j, and 0 if no information was given. $C(u_i, m_j)$ represents the "Movie Comments" entity. The Movie Comments file only shows whether a user commented on a movie, not whether the comment was positive or negative. Therefore, we assume that if u_i likes (dislikes) m_j, the comment was positive (negative). Thus, if user u_i made a comment on movie m_j, then $C(u_i, m_j) = F(u_i, m_j)$, and otherwise $C(u_i, m_j) = 0$. $R(u_i, m_j)$ represents the "Reviews" entity, which indicates whether or not a user reviewed a movie. Again, we guess the evaluation from $F(u_i, m_j)$: if the u_i made a review on m_j, then $R(u_i, m_j) = F(u_i, m_j)$, otherwise $R(u_i, m_j) = 0$. $Ra(u_i, m_j)$ shows how user u_i rated movie m_j, on a scale from 1 to 5.

$UMIV$ can be seen as an improved version of Ra; its computation reflects an assumption that users who, besides giving

a rating, explicitly comment on a movie, have a stronger or more informed opinion.

2.1.2 Distances

To express (dis)similarity among users, we consider the following distance measures.

Movie Interest Distance: This is the Manhattan distance between the UMIV values of two users; it simply reflects that users are similar when they have similar movie preferences.

$$d_M(u_i, u_j) = \sum_{\forall m \in M} |UMIV(u_i, m) - UMIV(u_j, m)| \quad (3)$$

Genre Interest Distance: Each movie m might belong to one or more genres g. If we represent the relationship between movies and genres in a zero-one matrix MG then we find the interest value of user u_i in genre g_j by simply multiplying $UMIV$ and MG (Formula 4).

$$UGIV = UMIV \times MG \quad (4)$$

With G the set of all genres in the dataset, we can define a distance between users that is the Manhattan distance between their genre interests:

$$d_G(u_i, u_j) = \sum_{\forall g \in G} |UGIV(u_i, g) - UGIV(u_j, g)| \quad (5)$$

Person Distance: Two users might be said to have similar interests if they like the same actors, directors, ... The "People in the Movie" entity indicates which people are involved in which movies; we represent it as a binary matrix MP. Multiplying $UMIV$ and MP gives a matrix UP that shows how much user u_i is interested in person p_j:

$$UP = UMIV \times MP \quad (6)$$

Further, the entity "personComments" describes whether the user u_i has made any comments on person p_j or not; we represent it as a matrix PC where $PC(u_i, p_j) = 1$ if u_i has commented on p_j and 0 otherwise. We use formula 7 to integrate the information in PC and UP (implicitly assuming that comments on people are positive):

$$UPIV = PC + UP \quad (7)$$

With P the set of all people in the dataset, we then define

$$d_P(u_i, u_j) = \sum_{\forall p \in P} |UPIV(u_i, p) - UPIV(u_j, p)| \quad (8)$$

Friend Distance: Here, we simply choose all the friends of the user u_i as its most similar users: $d_F(u_i, u_j) = 1$ if u_i and u_j are friends, and $d_F(u_i, u_j) = \infty$ otherwise. The number of most similar users may differ among users.

2.1.3 Integrated Framework

We could use each of the defined distance metrics to rank movies with Formula 1. Alternatively, we can combine them. Let $\vec{R}_M$ be a vector containing the rank of each movie when d_M is used as distance measure, and similarly for $\vec{R}_G$, $\vec{R}_P$ and $\vec{R}_F$. We can define an integrated rank vector $\vec{R}_I$:

$$\vec{R}_I = \alpha_M * \vec{R}_M + \alpha_G * \vec{R}_G + \alpha_P * \vec{R}_P + \alpha_F * \vec{R}_F \quad (9)$$

Setting Num	k	α_1	α_2	α_3	α_4
1	5	217	112.97	114.56	87.52
2	10	248.63	121.26	110.04	74.41
3	20	334.84	167.2	140.73	90.97
4	30	433.88	177.21	136.27	81.19
5	40	474.06	175.42	121.96	56.28
6	50	582.86	195.83	132.65	62.59
7	100	765.98	245.49	130.68	56.76
8	200	869.3	323.25	195.55	82.65
9	500	775.77	223.21	388.21	43.39

Table 1: Tuning different parameters of integrated framework.

with the α's to be determined. We will recommend movies with high $\vec{R}_I$ to the selected user.

The integrated framework has several parameters that we need to set or optimize: the number of nearest neighbors k, the α's, and the number of movies to recommend (or a threshold above which a movie should be recommended). We optimize these parameters on the training set.

The α's are initialized to zero, and then iteratively increased, with the increase depending on how well the corresponding distance performs. More specifically, we increase each coefficient based on the Jaccard similarity of the movies it recommends ($Pr_X(u_i)$, where X is M, G, P or F) and the movies actually seen by the user ($Re(u_i)$), where the number of predicted movies is chosen to be $|Re(u_i)|$:

$$\forall u_i \in trainingset : \alpha_X += \frac{|Pr_X(u_i) \cap Re(u_i)|}{|Pr_X(u_i) \cup Re(u_i)|} \quad (10)$$

We examined the values 5, 10, 20, 30, 40, 50, 100, 200 and 500 for k. Table 1 shows the α values learned for these k. The values show a clear tendency for the movie similarity itself to get a high weight, while the friend similarity tends to get a low weight, suggesting that the friendship relation is not very important. The weight of the genre and people involved fluctuates significantly.

2.2 Approach 2: regression-based k-NN

Our second k-NN based approach differs from the first mainly on the similarity measures used. The method combines two similarity measures which each result in a predicted rating; these are then averaged to get a final prediction for each user/movie-pair.

2.2.1 Linear Regression based k-NN

Our first similarity consists of the fit of a linear regression between the ratings given by two users. This is very similar to using correlation as a similarity measure, but is more robust to differences in calibration. Suppose user u_1 rated some movies as $(2, 3, 1, 4, 4, ?)$ (with ? the rating which we want to predict), and u_2 rated those movies as $(3, 4, 2, 5, 5, 3)$. The known ratings of u_1 and u_2 correlate perfectly. Standard 1NN would use u_2's rating of 3 as prediction for the unknown rating of u_1. This does not make much sense, as u_2 consistently rates movies higher than u_1. Performing a linear regression solves this problem: it results in the equation $y = x - 1$, and filling in u_2's rating of 3 would give

	Second	Third	Fourth
p@5	0.107823	0.147738	0.135386
p@10	0.103589	0.123460	0.096503
MAP	0.077488	0.096026	0.075459
AUC	0.980484	0.982339	0.980253

Table 2: Evaluation metrics for predictions based on degree 2, 3, and 4 social neighborhoods.

the more likely result: a predicted rating of 2. The linear regression based prediction is computed using

$$LR{-}Prediction(u_i, m_j) = \frac{\sum_{\forall u_k \in MSU(u_i,m_j)} LR(u_i, u_k, m_j)}{k} \tag{11}$$

where $MSU(u_i, m_j)$ is the set of most similar users, according to the fit of the linear regression, *among those who rated* m_j. $LR(u_i, u_k, m_j)$ is the prediction for pair u_i, m_j based on the linear regression between u_1 and u_2 in which the rating for m_j given by u_2 is filled in.

2.2.2 Social Neighborhood predictions

The second component of our second approach uses the social neighborhood of a user to predict ratings. A prediction is made by averaging the ratings given to the movie of interest by the user's friends:

$$FriendsPred(u_i, m_j) = \frac{\sum_{\forall u_k \in Friends(u_i,m_j)} Ra(u_k, m_j)}{k} \tag{12}$$

where $Friends(u_i, m_j)$ are u_1's friends who rated m_j, and $Ra(u_k, m_j)$ those ratings. This approach can be extended to include friends-of-friends. More generally, we define the i-th-degree social neighborhood of u_k as the set of all people within a friend distance of at most i from u_k. This gives the following formula to make predictions based on a user's social neighborhood.

$$SocNeighPred(u_i, m_j, n) = \frac{\sum_{\forall u_k \in SocNeigh(u_i,m_j,n)} Ra(u_k, m_j)}{k} \tag{13}$$

with n the degree of the social neighborhood. With a too low degree, we may not have enough people, and thus ratings, in the neighborhood; with too high a degree, we will overgeneralize.

The predictions from the linear regression method and the social neighborhood method are averaged to give the final prediction by our second approach.

2.2.3 Training

We tested social neighborhoods with degrees between 1 and 5. The performance increased up to degree 3 and went back down thereafter. Table 2 gives the results for degrees 2 to 4.

2.2.4 Evaluation procedure

Our results for the second method are obtained by evaluating the algorithm on all possible movie/user-pairs. To make this computationally feasible, one extra rule was added to the system: if a movie is not rated by at least 250 users, the prediction for that movie, for any user, is the lowest rating possible, i.e. it will not be recommended. As it turns out, this rule does not only make prediction more efficient, it also increases predictive performance by a large margin. This caused us to evaluate a trivially simple prediction method: recommend a movie, irregardless of the user, if and only if it is rated by over 100 users. This gave us a remarkably high AUC of 0.943. Thus, the simple rule "recommend a movie if many people have seen it" seems to work well, and might be useful as a reference point.

2.3 Approach 3: ILP

Our third approach is the use of inductive logic programming (ILP) [4]. This approach is motivated by the fact that ILP systems are good at handling background knowledge, as long as it can be expressed in a first order logic format.

We use the ILP system Tilde [2], in its regression setting [3]. This system uses a Prolog-like syntax. It takes a set of examples of the form $p(x_1, x_2, ..., x_n, y)$ together with background information (a Prolog program), and tries to construct a decision tree that predicts y from the other available information. The latter does not only include the values of the x_i; the x_i may serve as foreign keys that link the information on this example to other information, as in a relational database. For instance, we could have the logical facts

```
rating(user1,movie1,5)).
rating(user2,movie1,4)).
rating(user3,movie1,5)).
friends(user1,user3).
```

stating that $user1$ rates $movie1$ as 5, and similar for $user2$ and $user3$, and that $user1$ and $user3$ are friends. Then we could learn a rule stating that users who are friends tend to score movies alike.

```
rating(U,M,R) :- friends(U,U2), rating(U2,M,R).
```

Any kind of information that can be represented in a relational database can be represented in this logic-based format, and used by Tilde. Note that Tilde does not learn such rules directly; rather, it learns a decision tree, which can however be expressed as a rule set. Further information on Tilde is available at `http://dtai.cs.kuleuven.be/~ACE/`.

As Tilde was not able to handle the entire dataset, we used a random subset of 100,000 ratings from the training dataset. We also used the information about users (age, location), people in movies (actors, directors, writers), genres, comments, lists, friends and similar movies. This information was used to define relationships among and between users and movies. These definitions are available to Tilde as so-called background knowledge. Figure 1 shows a small fragment of this background knowledge. The rules define a relationship between users who differ at most 5 years in age (first rule), movies with the same director (second rule), etc.

Given this information, Tilde can learn a regression tree that predicts the rating. The internal nodes of the regression tree are boolean tests that can make use of the background knowledge. Figure 2 shows a fragment of the tree learned by

```
same_age_5(User,User2) :-
  users(User,Age,_,_,_), users(User2,Age2,_,_,_),
  not(Age=null), not(Age2=null), not(User=User2),
  Diff=abs(Age-Age2), Diff=<5.

same_director(Movie,Movie2) :-
  people_in_movies(Movie,Person,director),
  people_in_movies(Movie2,Person,director),
  not(Movie=Movie2).

same_genre(Movie,Movie2) :-
  genres(Movie,Genre),
  genres(Movie2,Genre), not(Movie=Movie2).
```

Figure 1: Fragment of the background knowledge for Tilde.

```
same_person(M,M2),rating2(U,M2,1) ?
+yes: 2.7728241350413
+no:  same_director(M,M3),rating2(U,M3,5) ?
    +yes: 3.89465124683773
    +no:  same_age_1(U,U2),rating2(U2,M,5) ?
          +yes: 3.57332009268454
          +no:  same_age_5(U,U3),rating2(U3,M,1) ?
                +yes: 2.83493166201012
                +no:  same_person(M,M4),rating2(U,M4,2) ?
                      +yes: 3.09792345276873
                      +no:  ...
```

Figure 2: Fragment of a regression tree for predicting the rating of a user U for a movie M. Nodes with question marks are internal nodes, other nodes are leaves. Free logical variables in internal nodes are (implicitly) existentially quantified.

Tilde. This tree can be used to predict the rating of any user for any movie, given the relevant background knowledge.

3. RESULTS

The results of the three approaches are summarized in Table 3. For the first approach, results for $k = 100$ and $k = 3000$ are reported. Between these values, all evaluation criteria increase monotonically with k. LR performs worse in terms of precision, but has a higher AUC. The social neighborhood based method performs much better on all metrics, and combining it with LR gives a small further improvement (results shown for $n = 3$). The ILP approach performs somewhat similar to the first k-NN approaches, with higher AUC but comparable or lower precisions.

These results are somewhat approximative. First, the P@5, P@10, MAP and AUC values reported here are obtained based on a ranking over all user/movie pairs; this gives a micro-average of the corresponding values per user. MAP is usually defined as a macro-average of average precision. Second, P@5 and P@10 are actually approximated by P@R, where R is the smallest recall value at least equal to 0.05 or 0.10, respectively. For micro-averaged values this gives a good approximation. Generally, the results are such that we expect LR+Soc, with $n = 3$, to make the best predictions on the evaluation set.

The different methods each have their own strengths; the first incorporates background knowledge into the distance and rating metrics; the second handles better differences in calibration between users; the third constructs relevant features from background knowledge automatically. One can imagine that combining the strenghts of the three will give better results. This was not investigated here for lack of time.

	100-NN	3000-NN	LR	Soc	LR+Soc	ILP
P@5	0.057	0.092	0.023	0.148	0.148	0.062
P@10	0.056	0.086	0.024	0.123	0.123	0.018
MAP	0.009	0.019	0.019	0.096	0.097	0.013
AUC	0.576	0.612	0.960	0.982	0.988	0.652

Table 3: Final results for the three approaches. LR+Soc evaluated on subset only.

Acknowledgements

Work supported by NWO (H.R., Vidi 639.022.605), K.U.Leuven (D.F., GOA/08/008), and FWO-Vlaanderen (B.P., project G.0255.08).

4. REFERENCES

[1] David W. Aha, Dennis F. Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[2] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

[3] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. In Jude W. Shavlik, editor, *ICML*, pages 55–63. Morgan Kaufmann, 1998.

[4] Luc De Raedt. *Logical and Relational Learning*. Springer, 2008.

[5] Alan Said, Shlomo Berkovsky, and Ernesto W. De Luca. Putting things in context: Challenge on context-aware movie recommendation. In *CAMRa2010: Proceedings of the RecSys '10 Challenge on Context-aware Movie Recommendation*, New York, NY, USA, 2010. ACM.