CS 289 Final Project Report: A Software Agent for Recommending Movies

Kevin P. Murphy

14 December 1995

1 Introduction

There is tremendous interest in systems which can make personalized recommendations of things, such as movies to watch, CDs to listen to, and books to read. For example, I know of two commercial systems on the Web for recommending movies[1], one for recommending CDs[2], and one generic one for recommending movies, CDs, newsgroups, etc.[3]. All of these work on the same basic principle, as follows: Each user rates movies they have seen in the past, and the system identifies users who have similar tastes; when you ask the system to recommend a new movie, it computes a weighted average of the ratings of all the other users who have seen this movie, where each weights is proportional to the degree of similarity between you and the other user; it then returns all the movies which have a high predicted rating. This method has been called "social information filtering" [SM95, Mae94], and is in contrast to content-based filtering, which uses information about the movie itself, rather than just the ratings of other users. One source for such information is the huge Internet Movie Database (IMDB)[4], which contains data on over 60,000 movies. This data is basically of two kinds, "objective" attributes, such as director, genre, cast, etc., and "subjective" reviews and ratings submitted by users of the `rec.arts.movies.reviews` newsgroup.

The purpose of this project is to compare the effectiveness of social-based and content-based filtering methods for recommending movies. I use two main approaches to content-based filtering. In the first, the reviews for an unseen movie are compared, using the vector space model [SM83], with reviews of previously rated movies, and a weighted average of these ratings is computed. In the second, I apply a variety of standard machine learning techniques, including decision trees [Qui86] and Bayes classifiers [DH73], to try and learn a mapping from attribute values to ratings.

The use of the word "agent" in the title is a little misleading. My original intention was to write a program which would be capable of "crawling the web" to find out about new movies, similar to "softbots" [EW94]. However, I never had time to do this. The main difficulty is that the information is stored in many different locations, and in many different formats. As far as I know, all of the programs mentioned above require information about new movies to be entered by hand. I started developing a web interface to allow users to do this easily, but did not receive permission from the system administrator to install my CGI (Common Gateway Interface) scripts.[5] Instead, I downloaded a copy of the Internet Movie Database. Unfortunately,

[1] `http://www.moviecritic.com` and `http://www.labs.bt.com/innovate/multimed/morse/morse.htm`.

[2] `http://www.agents-inc.com`. This system was first called Ringo, then HOMR, and now Firefly. It has been described in [SM95].

[3] `http://www.surf.com/rare/home.html`

[4] `http://rte66.com/Movies/` This is the US mirror of the original site in Cardiff, UK.

[5] Part of the problem was that I was using Lincoln Stein's Perl CGI library, which may contain security loopholes. However, it is tremendously useful. See `http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html`

the moderator of the database would not release information on how individual users had voted (their voting "profiles"), so I had to create my own. I generated a list of about 1000 movies[6], and asked various people to rate them on a scale of 1 to 10.[7] I received 26 replies, which formed my raw data. The rest of this report describes the results of various experiments on this dataset.

2 Social information filtering

The key thing in the social information filtering algorithm is to specify how to compute similarity between two users' profiles. Shardananda and Maes [SM95] suggest using the (Pearson) correlation coefficient, defined as follows. Let X, Y be two random variables, and let X^*, Y^* be their normalized counterparts, i.e., $X^* = \frac{X-\mu_x}{\sigma_x}$, where μ_x is the mean of X and σ_x is its standard deviation, and similarly for Y^*. Then the correlation coefficient of X and Y is defined as

$$\rho(X,Y) \stackrel{\text{def}}{=} Cov(X^*,Y^*) = \frac{Cov(X,Y)}{\sigma_x \sigma_y}$$

where $Cov(X, Y)$ is the covariance of X, Y,

$$Cov(X,Y) \stackrel{\text{def}}{=} \mathbf{E}\left[(X-\mu_x)(Y-\mu_y)\right]$$

and $\sigma_x^2 = Var(X)$ is the variance of X,

$$Var(X) \stackrel{\text{def}}{=} \mathbf{E}\left[(X-\mu_x)^2\right]$$

When we have n values (observations) each of X and Y, this reduces to the following familiar definition (the n's cancel, and we replace the means with the empirical means, $\overline{X}, \overline{Y}$):

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \overline{X})(y_i - \overline{Y})}{\sqrt{\sum_i (x_i - \overline{X})^2 \times \sum_i (x_i - \overline{X})^2}}$$

The correlation coefficient varies between -1 and 1, with -1 indicating negative correlation, 0 indicating no correlation, and 1 indicating complete correlation. Basically it tells us how far the set of points $\{(x_1, y_1), (x_2, y_2), \ldots\}$ lie from a straight line. The closer the coefficient is to 1 (in absolute value), the closer they lie to a straight line (i.e., $\rho = 1$ iff $Y = aX + c$ for some constants a and c).

Shardanand and Maes suggest computing $\rho(R^{(x)}, R^{(y)})$, where $R^{(x)}$ is user x's profile and $R^{(y)}$ is user y's profile, and the calculation is performed over movies in the training set which both users have rated. The predicted rating that user x will give to movie i is then computed as

$$r_i^{(x)} = \sum_{y \in U} \rho(x,y) \cdot R_i^{(y)}, \quad U = \{y' : y' \neq x,\ R_i^{(y)} \neq ?,\ \rho(x,y) > \theta\}$$

[6] These were the 1022 movies which had received over 100 votes by past users of IMDB. In hindsight, I realise that I should have produced a much shorter list, so that more people would respond to my survey.

[7] This scale was chosen to be compatible with IMDB, even though studies have shown that one does not gain much information in going beyond a seven point scale [SM95].

where U is the set of users whose similarity to x is greater than θ, a threshold, and whose rating of movie i is not ?, meaning unknown. The idea of the threshold is to ignore data from users who are not sufficiently correlated. They suggest setting $\theta \geq 0.35$. There are many problems with this method, however.

- Negative correlations, such as $\rho = -1$, can convey a lot information (roughly, you should vote the opposite way from this person), and yet are ignored. If the thresold is set at $\theta = -1$, it is possible to predict a negative rating using the above equation. Clearly a scale centered around zero, or around the mean, should be used for this.
- If the correlations are all very low (as I found them to be: see Table 1), setting θ too high will ignore all the data. If we have a low θ, and use the coefficients "as is", the predicted rating will be near zero. Clearly we need to normalize the coefficients in each row of the similarity matrix to 1 (although this requires that we ignore the negative coefficients).
- A correlation of zero does *not* mean the ratings are not related! (see e.g., [Fel57]). Nevertheless, it is not clear how to use the other user's ratings in such a case. The best we can do in this case is to predict the mean value.
- The calculation is only reliable if the profiles of the users have many movies in common. In my experience of using the Firefly system for recommending CDs [SM95], I found it would recommend music on the basis of a single artist that I had rated similarly to another user, which is not a very reliable indicator.

Nevertheless, Shardanand and Maes apparently get better results than the baseline (naive) strategy of simply predicting each user's mean rating. Their measure of performance was defined as follows. Let the error in rating an unseen movie i be $\epsilon_i = p_i - r_i$, where p_i is the predicted rating and r_i is the actual rating (for some user x). Then the mean absolute error is

$$|\overline{E}| = \frac{\sum_{i \in S_{\text{test}}} |\epsilon_i|}{|S_{\text{test}}|}.$$

They found that, for test sets containing movies with extreme-valued ratings (very low or very high), $|\overline{E}|=$ 1.8 for the naive strategy and $|\overline{E}|$=1.3 for the correlation strategy. I found $|\overline{E}|$=1.83 for the naive strategy and $|\overline{E}|$=5.17 for the correlation strategy. This is because most of my coefficients were near zero, and hence the predicted ratings were near zero; I did not have time to try using the predicted score as an offset. Presumably they did not run into this problem, since they had 1000 users, and hence there were enough large coefficients (i.e., similar users). (Currently, Firefly has 26,000 users.)

If I had more data, it would be interesting to see if users could be clustered into groups based on their profiles. Rather than using hierarchical clustering (see e.g., [Seb84]), something like Autoclass [CS95] might be more appropriate. Rather than forcing each point to belong to only one cluster, Autoclass can specify the probability that a point belongs to each cluster; this is a kind of "fuzzy membership" function. Background knowledge about a person, such as their age or sex, could be used to compute the prior probability that they like certain kinds of movies, and hence belong to certain clusters.

Clustering would improve the efficiency of the algorithm. Since it is unlikely to be feasible to explicitly store the $O(n^2)$-sized correlation matrix (where n is the number of users), the correlation coefficient will have to be computed on the fly, which takes $O(T)$ time (where T is the number of titles in the database). Rather than performing this computation n times, by comparing a user will all n other users, we could compare him with a representative of each cluster.

user	adam	brend	arno	lars	fink	dean	hauge	hoag	harald	erik	alison	ingrid	huma	geoff	nir	brant	galassi	julien	edw	aric	jeff	steph	friso	tozier	paul	josh	mean	num
adam	1.00	-0.00	-0.00	0.07	0.13	-0.01	0.11	0.14	-0.03	0.04	0.08	0.04	-0.00	0.10	0.09	0.10	0.04	-0.02	0.09	-0.02	0.07	0.12	0.01	0.06	0.13	0.03	5.11	444
brendan	-0.00	1.00	0.09	0.05	-0.02	-0.01	0.05	-0.01	0.04	0.11	0.09	0.08	0.02	0.10	0.04	-0.06	0.01	0.13	-0.01	0.10	0.06	-0.04	0.09	0.04	0.11	-0.03	6.62	99
arno	-0.00	0.09	1.00	0.12	-0.00	-0.02	-0.10	0.11	0.11	0.12	0.01	0.07	0.01	-0.06	0.01	-0.04	-0.04	-0.00	-0.04	0.05	-0.02	-0.04	0.18	-0.01	-0.00	0.03	6.14	88
lars	0.07	0.05	0.12	1.00	0.14	-0.02	0.04	0.08	0.01	0.22	0.07	0.15	0.06	0.04	0.05	-0.03	0.14	0.12	0.00	0.09	0.08	0.04	0.09	0.20	0.11	0.05	5.54	188
fink	0.13	-0.02	-0.00	0.14	1.00	0.02	0.18	0.11	0.03	0.17	0.02	0.06	0.03	-0.02	0.07	0.11	0.10	0.07	0.02	0.13	0.14	0.08	0.04	0.18	0.09	0.06	6.86	463
dean	-0.01	-0.01	-0.02	-0.02	0.02	1.00	0.05	0.05	-0.05	0.05	-0.04	-0.02	0.04	0.02	-0.01	0.09	0.02	0.05	0.03	-0.01	-0.05	0.29	0.00	0.04	-0.01	0.07	6.69	429
hauge	0.11	0.05	-0.10	0.04	0.18	0.05	1.00	0.09	0.11	0.05	0.00	0.05	0.02	0.14	0.10	0.13	0.08	0.11	0.14	0.16	0.22	0.15	-0.01	0.19	0.09	0.03	5.48	338
hoagland	0.14	-0.01	0.11	0.08	0.11	0.05	0.09	1.00	-0.02	0.05	-0.03	-0.01	-0.02	0.09	0.01	0.04	-0.06	0.03	0.09	0.02	-0.04	0.05	0.03	0.04	0.01	0.00	6.89	157
harald	-0.03	0.04	0.11	0.01	0.03	-0.05	0.11	-0.02	1.00	0.11	-0.02	0.04	0.07	-0.05	0.06	-0.02	0.04	0.14	0.01	0.13	0.06	-0.00	0.09	0.02	0.05	0.08	6.00	53
erik	0.04	0.11	0.12	0.22	0.17	0.05	0.05	0.05	0.11	1.00	0.07	0.17	0.15	0.10	0.07	-0.03	0.20	0.30	-0.02	0.18	0.12	0.07	0.18	0.20	0.16	0.05	5.30	375
alison	0.08	0.09	0.01	0.07	0.02	-0.04	0.00	-0.03	-0.02	0.07	1.00	0.12	0.10	-0.00	0.03	0.02	0.06	0.00	0.01	-0.02	0.00	0.05	-0.04	0.07	0.06	0.01	5.41	198
ingrid	0.04	0.08	0.07	0.15	0.06	-0.02	0.05	-0.01	0.04	0.17	0.12	1.00	0.08	0.04	0.05	-0.04	0.13	0.15	0.01	0.06	0.03	0.04	0.11	0.05	0.11	-0.01	6.03	75
huma	-0.00	0.02	0.01	0.06	0.03	0.04	0.02	-0.02	0.07	0.15	0.10	0.08	1.00	0.03	0.02	0.03	0.10	0.16	-0.02	0.06	0.11	0.07	0.00	0.08	0.12	0.00	6.81	77
geoff	0.10	0.10	-0.06	0.04	-0.02	0.02	0.14	0.09	-0.05	0.10	-0.00	0.04	0.03	1.00	0.05	0.03	0.10	0.17	0.09	0.05	0.08	0.11	-0.05	0.09	0.11	0.09	6.00	51
nir	0.09	0.04	0.01	0.05	0.07	-0.01	0.10	0.01	0.06	0.07	0.03	0.05	0.02	0.05	1.00	0.06	0.11	0.02	0.04	0.06	0.09	0.08	0.02	0.10	0.12	0.06	8.31	435
brant	0.10	-0.06	-0.04	-0.03	0.11	0.09	0.13	0.04	-0.02	-0.03	0.02	-0.04	0.03	0.03	0.06	1.00	0.05	0.11	0.09	0.04	-0.12	0.20	-0.03	0.03	0.02	0.11	6.82	239
galassi	0.04	0.01	-0.04	0.14	0.10	0.02	0.08	-0.06	0.04	0.20	0.06	0.13	0.10	0.10	0.11	0.05	1.00	0.13	0.03	0.12	0.14	0.05	0.14	0.17	0.10	0.07	6.94	353
julien	-0.02	0.13	-0.00	0.12	0.07	0.05	0.11	0.03	0.14	0.30	0.00	0.15	0.16	0.17	0.02	0.11	0.13	1.00	-0.02	0.24	0.10	0.07	0.06	0.13	0.20	0.12	7.32	152
edwards	0.09	-0.01	-0.04	0.00	0.02	0.03	0.14	0.09	0.01	-0.02	0.01	0.01	-0.02	0.09	0.04	0.09	0.03	-0.02	1.00	-0.01	0.07	0.10	-0.08	0.04	-0.01	0.04	6.59	230
aric	-0.02	0.10	0.05	0.09	0.13	-0.01	0.16	0.02	0.13	0.18	-0.02	0.06	0.06	0.05	0.06	0.04	0.12	0.24	-0.01	1.00	0.11	0.08	0.10	0.18	0.16	0.04	5.39	230
jeff	0.07	0.06	-0.02	0.08	0.14	-0.05	0.22	-0.04	0.06	0.12	0.00	0.03	0.11	0.08	0.09	-0.12	0.14	0.10	0.07	0.11	1.00	0.04	0.04	0.16	0.07	0.00	6.45	192
stephanie	0.12	-0.04	-0.04	0.04	0.08	0.29	0.15	0.05	-0.00	0.07	0.05	0.04	0.07	0.11	0.08	0.20	0.05	0.07	0.10	0.08	0.04	1.00	-0.04	0.12	0.11	0.20	5.68	241
friso	0.01	0.09	0.18	0.09	0.04	0.00	-0.01	0.03	0.09	0.18	-0.04	0.11	0.00	-0.05	0.02	-0.03	0.14	0.06	-0.08	0.10	0.04	-0.04	1.00	0.07	0.01	0.05	6.78	265
tozier	0.06	0.04	-0.01	0.20	0.18	0.04	0.19	0.04	0.02	0.20	0.07	0.05	0.08	0.09	0.10	0.03	0.17	0.13	0.04	0.18	0.16	0.12	0.07	1.00	0.16	0.04	6.36	630
paul	0.13	0.11	-0.00	0.11	0.09	-0.01	0.09	0.01	0.05	0.16	0.06	0.11	0.12	0.11	0.12	0.02	0.10	0.20	-0.01	0.16	0.07	0.11	0.01	0.16	1.00	0.08	6.55	350
josh	0.03	-0.03	0.03	0.05	0.06	0.07	0.03	0.00	0.08	0.05	0.01	-0.01	0.00	0.09	0.06	0.11	0.07	0.12	0.04	0.04	0.00	0.20	0.05	0.04	0.08	1.00	4.73	222

Table 1: The similarity between users, as measured by the correlation coefficients. “Mean” is the mean rating. “Num” is the number of movies rated.

Finally, if I had more time, I would try to use a first-order Bayes classifier, which seems like a much more well-principled approach. In this method, we compute the probability that the movie will receive rating r given a set of attributes $\mathbf{a}$ (the other users' ratings) as follows:

$$\Pr(R = r | \mathbf{A} = \mathbf{a}) = \alpha \Pr(R = r) \Pr(\mathbf{A} = \mathbf{a} | R = r) = \alpha \Pr(R = r) \prod_i \Pr(A_i = a_i | R = r)$$

where $\alpha = 1/\Pr(\mathbf{a})$ is a normalizing constant.[8] This derivation is valid providing the attributes (other users' ratings) are conditionally independent given our rating, which seems reasonable. In any case, in practice Bayes classifiers are very robust to violations of this independence assumption. The advantages of this approach are

- We can return a distribution over the predicted ratings, thus reflecting uncertainty. If we only want a single number, we can computed the expected rating, or the maximum-likelihood rating.
- The "weights" have a clear interpretation as conditional probabilities.
- We can use prior information in a simple way. For example, very few users give low scores, probably because they don't go to see movies which they know will be bad. So we can use a non-uniform prior on the classes.

I shall discuss the results of using a Bayes classifer with a different set of attributes later.

3 Applying the vector space model to reviews

In the vector space model [SM83, She94], a text document is represented as a vector of weights, $< w_{ij} >$, where w_{ij} is the weight of term j in text T_i. A term is usually a word, or possibly a word-stem. Usually, high frequency words such as "the" and "and" are excluded. The weight of a term is a product of two factors, the term frequency and the inverse document frequency. The term frequency is the number of times the term appears in a document; the idea is that words which occur frequently are likely to be important. Of course, if a word occurs frequently in *every* document, it is less likely to be important; this is reflected in the inverse document frequency. Formally we have

$$w_{ij} = tf_{ij} \times idf_j$$

where tf_{ij} is the number of occurrences of term t_j in document i, and idf_j is the inverse document frequency of the term t_j in the whole collection of documents. A commonly used measure for inverse document frequency is

$$idf_j = \log\left(\frac{N}{n_j}\right)$$

where N is the total number of documents in the collection, of which n_j contain term t_j.

[8] This constant is not important, since we are only interested in the relative probabilities of the different ratings. In the binary case, where we either like or don't like a movie, we can compute the likelihood ratio, $\frac{\Pr(1|\mathbf{a})}{\Pr(0|\mathbf{a})}$.

	SW	BR				WH		Four				
ID	0060	1135	1524	1525	1820	0580	0589	2475	2482	2499	2510	2634
0060	1.000	0.106	**0.122**	**0.100**	0.092	0.047	**0.121**	0.076	0.073	0.066	0.059	0.045
1135	**0.106**	1.000	0.081	**0.126**	0.073	0.024	0.040	0.054	0.042	0.032	0.035	0.032
1524	**0.122**	0.081	1.000	0.071	0.065	0.023	0.058	0.037	0.051	0.051	0.051	0.032
1525	**0.100**	**0.126**	0.071	1.000	**0.149**	0.051	0.038	0.027	0.046	0.030	0.036	0.024
1820	0.092	0.073	0.065	**0.149**	1.000	0.028	0.042	0.050	0.034	0.055	0.064	0.011
0580	0.047	0.024	0.023	0.051	0.028	1.000	**0.100**	0.033	0.064	0.052	0.052	0.034
0589	**0.121**	0.040	0.058	0.038	0.042	**0.100**	1.000	0.043	0.068	0.070	0.047	0.028
2475	0.076	0.054	0.037	0.027	0.050	0.033	0.043	1.000	0.079	0.052	0.068	0.049
2482	0.073	0.042	0.051	0.046	0.034	0.064	0.068	0.079	1.000	0.089	0.082	0.042
2499	0.066	0.032	0.051	0.030	0.055	0.052	0.070	0.052	0.089	1.000	0.074	0.030
2510	0.059	0.035	0.051	0.036	0.064	0.052	0.047	0.068	0.082	0.074	1.000	0.029
2634	0.045	0.032	0.032	0.024	0.011	0.034	0.028	0.049	0.042	0.030	0.029	1.000

Table 2: The similarity between review texts, as measured by the vector space model. Scores over 0.1 (excluding the main diagonal) are in bold face. "SW" denotes "Star Wars", "BR" denotes "Blade Runner", "WH" denotes "When Harry Met Sally", and "Four" denotes "Four Weddings and a Funeral". Each movie may have several reviews, distinguished by theur four digit identifiers, as used in the `rec.arts.movies.reviews` archive.

The similarity of two texts can be computed by taking the scalar product of their vectors:

$$S(T_i, T_j) = \sum_k w_{ik} \times w_{jk}$$

Since we often want to compare a text against many others, to find the most similar, the similarity scores must all be on the same scale. By normalizing the vectors so that they have unit magnitude, we can ensure that $-1 \leq S(T_i, T_j) \leq 1$. The similarity scores can then be used to compute a predicted rating for a movie in a similar manner to the social filtering algorithm.

To investigate the potential of this method, I downloaded some reviews from the `rec.arts.movies.reviews` newsgroup archive[9] for four different movies: "Star Wars", "Blade Runner", "When Harry Met Sally" and "Four Weddings and a Funeral". I then converted these to their vector representation, and computed their similarity; see Table 2. The initial results are suprisingly good, especially in view of the crudity of the preprocessing.[10] In particular, notice how the reviews for "Blade Runner" match the other reviews for "Blade Runner" more closely than the reviews for other movies, and similarly for "When Harry Met Sally".

However, on closer inspection, it seems that this might just be due to chance. For example, the review for "Star Wars" matches the review for "When Harry Met Sally" almost as closely as "Blade Runner", which is a little worrying. Table 3 illustrates some of the words which matched in each case. For example, they both contain the word "cute", although in one case it is used to describe aliens, and in the other to describe attractive actors!

Since these experiments suggest that the similarity scores will be very close to 0, if we use them directly as weights, as in social information filtering, we will encounter the same problem of consistently underestimating the rating. Clearly we need to normalize within each row as well. I did not have time to do this.

[9] `ftp.uu.net:/ ftp/usenet/rec.arts.movies.reviews`

[10] I only did the following processing to the text before computing the vectors: strip the punctuation, delete words which start with a capital letter (they might be proper nouns), convert to lower case, and remove words which occur in a short list of pronouns, prepositions, articles, etc. that I typed in by hand.

word	tf1	tf2	d	idf	word	tf1	tf2	d	idf
woman	1	2	3	0.85	fantasy	6	1	2	1.25
cute	2	3	3	0.85	achievement	1	1	2	1.25
really	2	2	3	0.85	intelligent	1	1	2	1.25
recartsmovies	1	1	3	0.85	original	2	2	5	0.34
people	4	8	6	0.15	people	4	2	6	0.15
film	51	7	7	0	film	51	12	7	0
films	13	1	5	0.34	satisfying	1	1	2	1.25

Table 3: An illustration of why some texts are considered similar. On the left is a comparison between a review for "Star Wars" (0060) and a review for "When Harry Met Sally" (0589). On the right is a comparison between the same review for "Star Wars" and a review for "Blade Runner" (1524). The tf columns indicate term frequency, the d columns indicates the total number of documents which have this term (out of $N = 7$), and the idf column is the inverse document frequency of this term. Terms which occur in every document, like "film", have zero idf. The total number of matching words was 118 for the first comparison and 132 for the second. The selection of words above is a representative sample from the set of matching words. The word "recartsmovies" is a result of stripping the punctuation from "rec.arts.movies", the name of a newsgroup. Notice how "film" has an idf of zero, but "films" (plural) has an idf of 0.34.

4 Learning a mapping from attributes to ratings

In this section we consider using structured information about each movie, rather than free-form text. The structured information available in the IMDB includes the director, the cast, a list of words describing the genre, etc. Collectively these are called attributes or features. The goal is to learn a mapping from the features to the rating. We shall view this as a classification problem, where the ratings are the different classes. The traditional performance metric in the classification community is the percentage of examples in the test set which we classify correctly. However, this ignores the fact that the ratings are not qualitatively different classes, but really just discretizations of a single, real-valued number. It makes more sense, therefore, to measure performance as the mean absolute error, as before. This can easily be computed from the confusion matrix (see e.g., Figure 1).

The first problem in learning a mapping from features to ratings is to choose a relevant subset of the features. It is obvious that we should not use the title as a feature: although we could learn a simple title-to-rating mapping, and achieve perfect performance on the training set, such a rule would fail completely on examples not in the training set. It is likely that using "director" as a feature is also a bad idea, for the same reason. For example, of the 176 movies that I rated, 121 were directed by different people. Of course, some individuals direct a lot (relatively speaking) of movies, but this may still be only a weakly relevant feature. For example, you might have hated all the movies by Steven Spielberg except for "Schindler's List". In [JKP94] they give an example where including a feature which predicts the classification 75% of the time can actually *lower* the accuracy of a decision tree.

In this project, I chose to only consider the genre feature, for simplicity. However, since this can take on a whole *set* of values, such as {"romantic", "comedy", "drama"}, I convert it to a vector of features, where each component represents one of the possible genres, and can take on values 0 or 1. This is called a *local encoding*. In my dataset, the number of possible genres is 120 (including such things as "based-on-true-story", "part-animated", "culture-clash", etc.)

By making use of an existing library of machine learning routines written in C++ [KJL+94], I had enough time to compare a variety of different classifiers. The results are summarized in Table 4. The majority classifier simply predicts the rating that occurs most often; in my dataset, this is 8. It forms a baseline for comparison purposes. The naive Bayes algorithm is a first-order Bayesian classifier, which assumes that the

```
Number of training instances: 126
Number of test instances: 44.  Unseen: 33,  seen 11.
Number correct: 14.  Number incorrect: 30
Generalization accuracy: 33.33%.  Memorization accuracy: 27.27%
Accuracy: 31.82% +- 7.10% [20.00% - 46.56%]

Displaying confusion matrix...
 (a)  (b)  (c)  (d)  (e)  (f)  (g)  (h)  (i)  (j)    <-- classified as
---- ---- ---- ---- ---- ---- ---- ---- ---- ----
   0    0    0    0    0    0    2    0    0    0    (a): class 1
   0    0    0    0    0    0    0    0    0    0    (b): class 2
   0    0    0    0    0    0    2    0    0    0    (c): class 3
   0    0    0    0    0    0    3    0    0    0    (d): class 4
   0    0    0    0    0    0    4    0    0    0    (e): class 5
   0    0    0    1    1    0    1    0    0    0    (f): class 6
   0    0    0    0    2    0   12    0    0    0    (g): class 7
   0    0    0    0    1    0    7    2    0    0    (h): class 8
   0    0    0    0    0    0    2    0    0    0    (i): class 9
   0    0    0    0    0    0    3    1    0    0    (j): class 10
```

Figure 1: An example of the output produced by ML-C++, to illustrate a confusion matrix. This was generated using ID3 with $M = 15$, and has a cross-validation accuracy of 22%; see the next section for details. From the confusion matrix we find that $|\overline{E}| = \frac{70}{44} = 1.59$, which is marginally better than the naive method, which gives $|\overline{E}| = 1.8$.

Algorithm	CV	TS	TS unseen	TS seen	MAE
Majority	14%	23%	24%	18%	1.83
Naive-Bayes	27%	20%	18%	27%	2.00
ID3	23%	32%	33%	27%	1.59

Table 4: Accuracy of different classification algorithms estimated in various ways. CV is the accuracy as estimated by performing cross-validation with five folds, repeated five times, on the training set (126 examples). TS is the overall accuracy on the test set (44 examples). The TS value has high variance (usually 6–7% compared with about 1% for CV). "TS unseen" is the (generalization) accuracy on the 33 examples in the test set which were not in the training set. "TS seen" is the (memorization) accuracy on the 11 examples in the test set which were also (by chance) in the training set. If the memorization is much greater than the generalization accuracy it suggests overfitting. MAE is the mean absolute error, as computed from the confusion matrix.

M.	nodes	leaves
1	179	90
5	29	15
10	13	7
15	11	6
20	5	3
25	5	3
30	5	3
35	5	3
40	5	3
45	3	2
50	1	1

Table 5: Decision tree complexity as a function of the minimum number of examples, M, allowed in each leaf. The columns indicate the total number of nodes in the tree, and the number of leaves.

attributes (genres) are independent; although this is not the case, it has the best overall accuracy (although the worst $|\overline{E}|$, and is also very simple to implement. ID3 is a simple method for inducing binary decision trees [Qui86], which we discuss in more detail below.

4.1 Using decision trees

Since there are so many features, it is important that we do not overfit the data by inducing a decision tree in which all the leaves are pure, i.e., only contain one example. Such a tree is overly complex; see for example Figure 4. By specifying the minimum number M of examples which must fall into each leaf we can achieve a form of top-down pruning. This produces smaller trees; see Table 5 and Figures 3 and 4. Furthemore, the smaller the tree, the better the accuracy, up to a point: see Figure 2. The tree which consists of a single leaf is in effect a majority classifier. The tree which consists of two leaves branches on the "action" attribute.

The documentation for ML-C++ recommends that serious users of decision trees use C4.5, a more robust version of ID3 which comes with Quinlan's book [Qui93]. Unfortunately, the book costs $45 and the disk costs an additional $35, and I was not able to locate a free copy. However, Geoff Zweig let me use his decision tree code, which does bottom-up χ^2 pruning. This achieved roughly the same performance as the best case of ID3 with top-down pruning, but resulted in a simpler, two-leaf tree. This is in agreement with the conclusions reached in [Hol93], where he finds that simple, depth one decision trees ("one-rules") often perform nearly as well as much more complex trees.

4.2 Discussion

The bottom line is that all of the classification techniques do very poorly in this domain, as judged by the low accuracy rates. There are several reasons for this. Firstly, the number of possible classes (ratings) is much larger than in the datasets which are normally studied in the machine learning literature, such as the UC Irvine repository, which are often just binary classification tasks (sometimes even linearly separable!). The current problem is inherently harder, as reflected in the much lower baseline accuracy. Secondly, there is intrinsic variability in how a person rates a movie,[11] so we cannot expect to predict their exact rating. If we look at the absolute mean error, the classification techniques seem to compare favorably with social

[11] We could estimate this by asking people to rate the same list of movies several times. The list and the time between trials must be sufficiently long that the person doesn't remember how he voted last time on marginal cases.

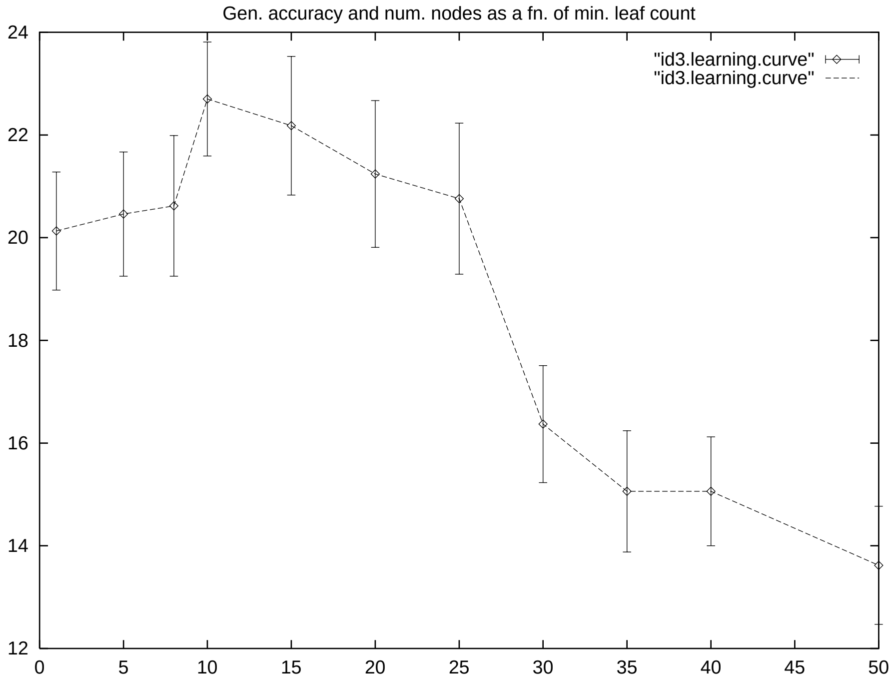

Figure 2: Percentage accuracy of the decision tree, as measured by five-fold cross validation, as a function of M, the minimum number of examples allowed in each leaf. As M increases, the tree gets smaller.

information filtering: in [SM95] they find that $|\overline{E}|$=1.3; using ID3, I find $|\overline{E}| \approx 1.6$; the naive method scores $|\overline{E}|$=1.8. Thirdly, and perhaps most importantly, a single attribute such as genre just does not contain enough information to be a good predictor; after all, there are good dramas and bad dramas. This issue is discussed more in the conclusions section.

5 Conclusions and further work

My results suggest that social information filtering is much worse than the naive strategy of simply predicting the most common rating, but this is probably due to the small size of my dataset — results in [SM95] with 1000 users suggest that social information filtering is much better. It would be interesting to verify this result on a large dataset, and also to carry out the experiments on clustering and user stereotyping mentioned earlier.

My results on the vector space model are inconclusive. I did not have time to give the technique a "run for it's money". For example, although the texts which matched well contain a lot of high frequency noise words (like "film" or "people"), as the number of documents go up, the idf's of these words will go down much faster than for less common words. A more sophisticated lexical analyser would also help (see e.g., [vR79]). It would be interesting to try some techniques from the field of natural language understanding as well. For example, Framer [Haa94] is a knowledge representation library which can draw analogies in natural language texts, and could be useful in this context.

Finally, the classifier results are also hard to interpret, partly because of the conflicting measures of performance. It certainly seems plausible that by allowing more features, such as director and cast, and using a feature subset selection algorithm (such as the one in [JKP94]), that the performance could equal that of

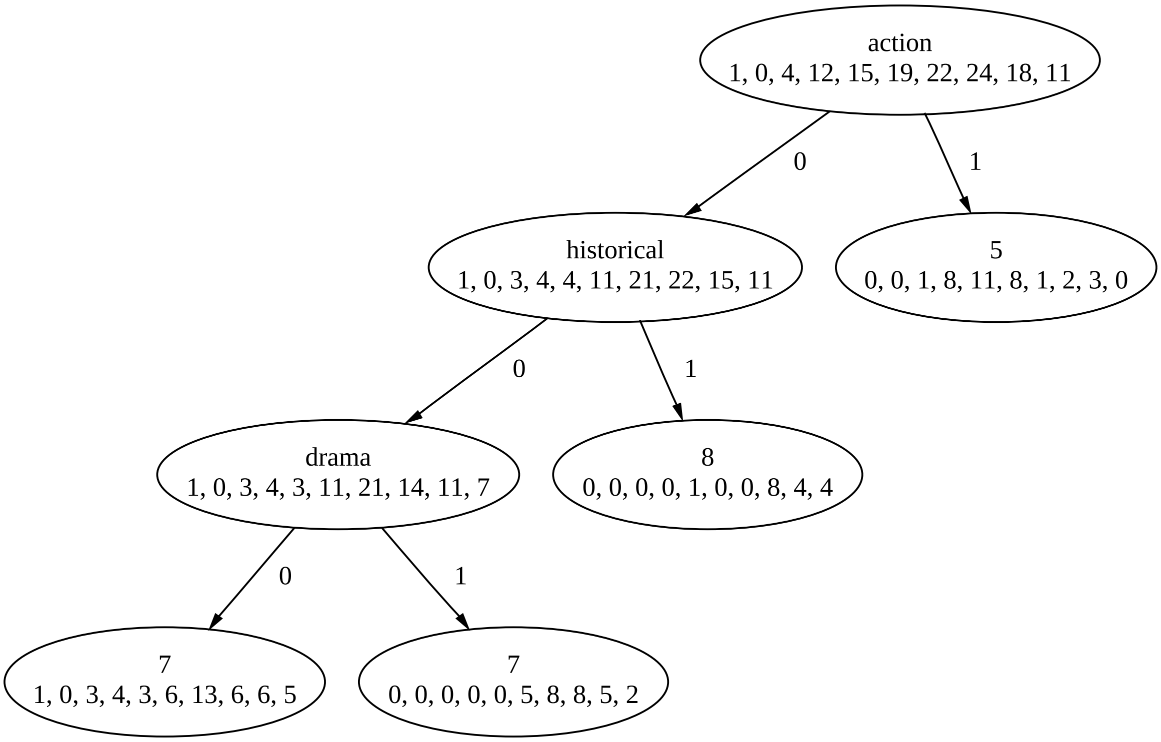

Figure 3: Decision tree when $M = 17$. The list of numbers in each node indicate the number of examples in each rating category (1 through 10) which reach this node. The word inside the internal nodes represents a feature value to test for; 0 means absent, 1 means present. The number inside each leaf indicates the rating given to all examples that reach this leaf. In the case of conflicts, the majority rating is taken.

Figure 4: Decision tree when $M = 1$.

social information filtering. However, there is a fundamental problem with these techniques, in that they should really being trying to compute a single, real-valued number from a vector of nominal attributes. The Bayes-classifer produces a probability distribution over ratings given attributes, which can be used to compute an expected rating. Similarly, we could take the mean rating of a leaf in the decision tree instead of the majority. However, "internally" both of these algorithms treat the classes (ratings) as qualitatively different. Neural networks might be more appropriate, since they work in a continuous space, but then there is the problem of mapping the nominal input features to real values.

To get the best performance, we will have to combine evidence from all three sources (data fusion). This is complicated by the mixture of nominal features such as genre, numerical features such as other users' ratings, and textual features such as reviews, each of which has a different measure of similarity. Even then it is not clear whether we have enough information; after all, people are often very poor at predicting whether they will like a movie — it often happens that you notice that a new movie is coming out by your favorite director, you read some favorable reviews, maybe even your friends recommend the movie, and then you go see it end up not enjoying it.

Ultimately, the true measure of success is not whether you correctly predict the user's rating, but whether they continue to use your system. This may require some basic concepts from utility theory. For example, recommending a bad movie which is free and on-campus is not as grave an error as recommending a bad movie which is expensive and far-away. Also, the system must be easy to use. One possibility would be to infer your movie preferences from your video rental patterns, which would avoid the need to type in any data. This already happens to some extent with credit card purchases. To the extent that what we spend our money on reflects what we value most in life, this is a Good Thing, since it helps us get more of what we want.

References

[CS95] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In Fayyad, Pratetsky-Shapiro, Smyth, and Uthurasamy, editors, *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1995.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley Interscience, 1973.

[EW94] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Comm. of the ACM*, 37(7), 1994.

[Fel57] W. Feller. *An Introduction to Probability Theory and Its Applications*. Wiley, 1957.

[Haa94] K. B. Haase. FRAMER: A persistent portable representation library. In *European Conf. on AI*, 1994. Submitted.

[Hol93] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[JKP94] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proc. of the 11th Intl. Conf*, 1994.

[KJL+94] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 740–743, 1994. Available by anonymous ftp from:
`starry.stanford.edu:pub/ronnyk/mlc/toolsmlc.ps`.

[Mae94] P. Maes. Agents that reduce work and information overload. *Comm. of the ACM*, 37(7), 1994.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. Reprinted in *Readings in Machine Learning*, Shavlik and Dietterich, eds.

[Qui93] J. R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kauffman, 1993.

[Seb84] G. A. F. Seber. *Multivariate Observations.* John Wiley, 1984.

[She94] B. D. Sheth. A learning approach to personalized information filtering. Master's thesis, MIT, 1994.

[SM83] Salton and McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[SM95] U. Shardanand and P. Maes. Social information filtering: algorithms for automating 'word of mouth'. In *Proc. CHI. Conf.*, Denver, CO., May 1995.

[vR79] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, 1979.