Predicting Movie Preferences

Ian Baker

1 Introduction

Accurately predicting customer preferences is a highly sought after goal, for it allows businesses to market a product to individuals in a way that is most likely to generate a sale. This can be particularly important in the movie industry, where individual preferences can vary greatly. Recognizing this, the online movie rental business Netflix has sponsored a contest to find better ways of making these kind of predictions.

Specifically, the problem is as follows. For a given customer and a given movie, predict how that customer would rate the movie on a 1-5 scale, using information on how that customer has previously rated other movies, and other customer ratings. More formally we wish to estimate

$$R_{cm} = \mathbf{E}[R|C = c, M = m]$$

where R, C, M are random variables representing the rating, customer and movie, respectively. Equivalently we estimate the probability $P(R|c, m)$ and use this to compute the expectation.

One possible approach to this estimation is to assume that while individuals' preferences may vary greatly from person to person, each person belongs to one of a small number of classes Z, and members of a given class tend to rate all movies in approximately the same way. Probabilistically we write

$$P(R|c, m) = \sum_z P(R, Z = z|c, m) = \sum_z P(R|z, c, m)P(z|c, m)$$

We assume that the class z captures all the information about how an individual will rate a given movie, i.e. $P(R|z, c, m) = P(R|z, m)$, and that the class of a customer does not depend on the movie rated, i.e. $P(z|c, m) = P(z|c)$. Assuming each member of a class rates a given movie about the same, we model $P(R|z, m) = \mathcal{N}(\mu_{zm}, \sigma_{zm})$ and $P(z|c) = \phi_{zc}$, for fixed parameters ϕ_{zc}.

Then given a set of training data – customer ratings $\{(r^i, c^i, m^i)\}$ –, we want to find a set of parameters that maximizes the likelihood

$$\begin{aligned} l(\mu, \sigma, \phi) &= \sum_i \log P(r^i|c^i, m^i; \mu, \sigma, \phi) \\ &= \sum_i \log \sum_z P(r^i|z, m^i; \mu_{zm^i}, \sigma_{zm^i})P(z|c^i; \phi_{zc^i}) \end{aligned}$$

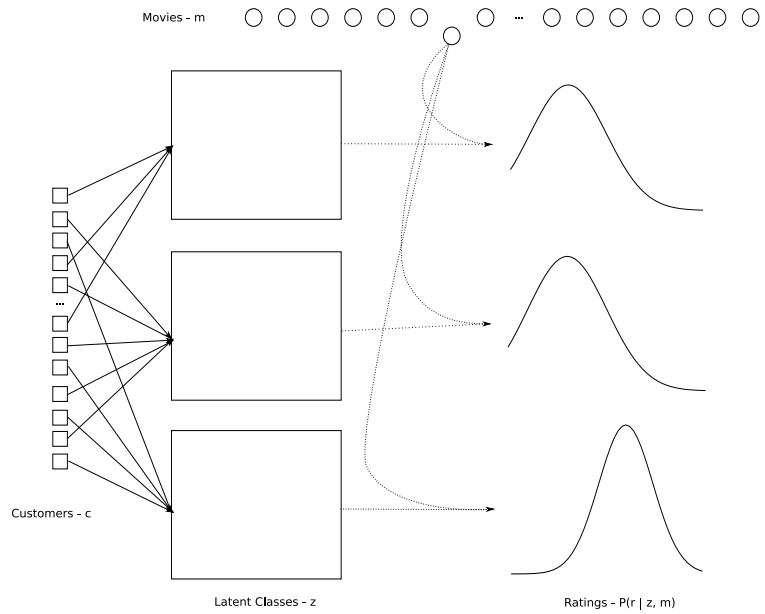

Figure 1: The probablistic model of ratings: Each customer c is given a class z with probability ϕ_{zc}, then the rating for a given movie m is generated from a normal distribution $\mathcal{N}(\mu_{zm}, \sigma_{zm})$.

This optimization can then be performed by using the Expectation Maximization algorithm. For the expectation step we compute

$$w_j^i = P(Z = j | r^i, c^i, m^i)$$

The optimal values of the parameters are then given by

$$\phi_{zc} = \frac{\sum_i w_z^i I\{c^i = c\}}{\sum_i I\{c^i = c\}}$$

$$\mu_{zm} = \frac{\sum_i w_z^i r^i I\{m^i = m\}}{\sum_i w_z^i I\{m^i = m\}}$$

$$\frac{1}{\sigma_{zm}^2} = \frac{\sum_i w_z^i (r^i - \mu_{zm})^2 I\{m^i = m\}}{\sum_i w_z^i I\{m^i = m\}}$$

in the maximization step. Once we have optimized these parameters by iterative updates, we can then make a prediction for a customer, movie pair (c, m) by

$$\begin{aligned}
\mathbf{E}[R|c,m] &= \sum_r r P(r|m,c) \\
&= \sum_r r \sum_z P(r|z,m) P(z|c) = \sum_z \left[P(z|c) \sum_r r P(r|z,m) \right] \\
&= \sum_z \phi_{zc} \mu_{zm}
\end{aligned}$$

2 Results

We implemented the algorithm described above, using training data consisting of approximately 100 million training examples, which Netflix has provided from their database as part of the contest. The error metric we use is the root mean squared error $RMSE = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(\rho^i - r^i)^2}$, where ρ^i and r^i are the predicted and actual values respectively.

A priori the number of classes, Z, is unknown, so it needs to be determined from the data. A straightforward method of choosing a good value is to train the algorithm for increasing values of Z and use hold out cross validation to choose the Z that minimizes the error on some held out test set.

Initially we performed this using a relatively small training set (about 8 million examples out of the 100 million available). We found $Z = 2$ had the smallest test error, though the error remained nearly constant as Z increased. However, the training error steadily decreased as Z increased, e.g. test error was 0.965 for $Z = 2$, 0.972 for $Z = 10$, and training error was 0.940 for $Z = 2$ and 0.85 for $Z = 10$.

The discrepancy between the training and test error seems to indicate significant variance in the method, and given the abundance of training data, it seemed likely that the larger Z values would perform better for larger training sets. We therefore trained the method for increasing Z and increasing numbers of training examples, the results of which are summarized in table 1. As is evident from the data in that table, the larger Z values do outperform $Z = 2$ as the training set size increases.

Given this dependency on training set size, validating on the largest training set possible seems to be an important goal. However, since the runtime requirements of the algorithm increase linearly in both time and space with the number of training examples, using tens to hundreds of millions of examples quickly becomes infeasible on a standard workstation without signficant performance tuning. Significant parts of the algorithm are parallelizable, and we implemented a parallel version which outperforms the serial version by as much as 50% per iteration.

Yet, there is still the issue of the memory consumption. Even using data structures designed to minimize the size of the data in memory, if all of the needed data is kept in main memory, the algorithm consumes more memory than a typical workstation has available when more than a few tens of millions of examples are used. In this case most of the time is spent waiting for the operating system to page data to and from the hard disk, and the time needed becomes impractical. Within these constraints we were able to train the algorithm for $Z = 4$ with 25 million examples and $Z = 2$ with 40 million examples.

After training on these subsets we obtained a test error of 0.942 for $Z = 2$ and 0.935 for $Z = 4$ on other randomly chosen subsets of the data. On a "probe" subset of the data, specified by Netflix, we obtained a test error of 0.988 for $Z = 2$, training set = 40 million, and 0.991 for $Z = 4$ training set = 25 million . Finally on the "quiz set" we obtained an error of 1.00 for $Z = 2$. (This is a set of data not included in

Training Examples (mil)	Z	15	25	40
Training				
	2	.934	.936	
	3	.913	.922	
	4	.899	.907	
Test				
	2	.957	.949	942
	3	.949	.943	
	4	.945	.935	
Probe				
	2	1.02	.998	.988
	3	1.01	.993	
	4	1.01	.991	

Table 1: RMS error on the training, test, and probe sets for varying values of Z and increasing numbers of training examples.

the training set, but for which Netflix will calculate the RMS error, given a set of predictions). The probe and quiz sets are not randomly chosen, but likely are chosen because they represent more "difficult" sets of data to predict. As a reference, the most naive method (always predicting the average rating) achieves an error of 1.05 on the quiz set, and approximately 1.02 on the entire training set.

3 Discussion

The method presented shows clear performance gains over the simplest of methods; however, there is also clearly still significant room improvement. One of the main problems with the method seems to be high variance, particularly as Z increases. Indeed, as Z increased the training errors were consistently and significantly reduced, yet the test errors only showed improvement for larger training sets. Given the abundance of training data (over 100 million examples available, plus Netflix total database contains billions), this is not necessarily a major problem with the method. However, the computational resources required is a more significant hurdle. In order to really test the potential of the method, it is necessary to train it with more than a few tens of millions of examples, yet to do this, one either needs more powerful hardware than a typical workstation, or a highly specialized implementation designed to parallelize the data across multiple machines or efficiently cache data to the disk during the run. Unfortunately neither of these is really an option for a one quarter student project.

In addition to improving the implementation, it seems likely the method can be

improved in other ways as well. One possibility is classifying not only the customers, but also the movies. We can then model the problem by assuming not only are classes of customers likely to rate the same movie similarly, but that generally movies of the same class are likely to be rated similarly by customers of a given class. Classes of movies could then be determined not only through internal consistency with the ratings from the training data, but also through external features, such as actors, directors, genre, etc.

4 Conclusions

We have presented a method for predicting how an individual will rate a movie, given information about their previous ratings and other customers ratings. The method uses a probablistic model that postulates various "classes" that customers belong to and which determine how customers will rate a given movie. Then given example data we can find a maximum likelihood estimate to the model using the expectation maximization algorithm.

Even for relatively small samples, this algorithm shows improvement over the simplest methods. However, in order to determine the true potential of the method, further work is required in order to reasonably train on significantly larger training sets.