What's the Score?
Building Text Processing Agents in the Domain of Movies

Nick Frederick
Department of Computer Science
University of Northern Iowa
nickfre@uni.edu

Kevin O'Kane
Department of Computer Science
University of Northern Iowa
okane@cs.uni.edu

Ben Schafer
Department of Computer Science
University of Northern Iowa
schafer@cs.uni.edu

Abstract

This paper will present the development of intelligent agents built to test the hypothesis that agents can be built which convert textual reviews of movies into a numerical rating, based solely on the text within the review. To test this hypothesis, we began with a base set of reviews from a single critic. These reviews consisted of both the textual review and a known numerical rating. Using this, each agent built a knowledge base for the specific writing style of the critic. This knowledge is then applied to the text of additional reviews in an effort to generate proper and accurate numerical predictions. This paper will compare three different agents which convert text to a numerical score using single term, double term and triple term matrices respectively. Discussed results include agent accuracy, under what situations the agent is incorrect, and possible modifications for future generations of agents.

Introduction

In a world where the number of choices can be overwhelming, recommender systems help users find and evaluate items of interest. They do so by connecting users with information regarding the content of recommended items or the opinions of other individuals. The more personalized forms of recommender systems often use a technology known as collaborative filtering (CF) [, ,]. CF systems generate a personalized "neighborhood" of like-minded individuals for *each* user of the system. For example, MovieLens [], a CF system in the domain of movies, might find the N people who most often agree with each of the members of the site. By using the opinions of these N people, MovieLens can make predictions about movies that a user has not yet viewed.

Unfortunately, CF systems suffer from several problems [,]. One of these is the first-rater problem. In order to be effective, CF systems have to have ratings for items in the system. If everyone waits until someone else has offered an opinion, the system stagnates and becomes unusable. Thus, CF systems need users who are willing to try the items being recommended even if they turn out to be lousy items. In a domain such as movies, these people readily exist. They are called critics. Since the dawn of the video age there have been critics writing reviews of movies. Unfortunately, they have not always used a numerical rating system as would be needed for a CF system. Even among the critics that do include numerical scores, they vary greatly in the scale that they choose to use (thumbs up/thumbs down, the 5-stars rating, the letter grade rating, etc).

In order to improve CF based recommender systems, our goal was to create one or more intelligent systems that would be able to read a written review of a critic, and determine what rating that the author of the review would have given the review. This is accomplished by creating a tree-like structure of the critic's reviews and the words that they contain.

The Data Set

The data set for this project was obtained from the Rotten Tomatoes movie review site []. Although all the reviews did not contain textual reviews of the movies and some of the textual reviews were not long enough to establish a 'good base'; we determined that there was enough complete data within the data set to give reliable results for this project.

Then the problem of how to use the data set came up. Since the language we were going to use to write the intelligent agents had an 80 character limit for the line length that could be read, we needed to devise a way to change that data set so that Mumps [], the programming language we would be using, could parse through the data. So we wrote a C program that parsed through the entire dataset and separated all of the lines accordingly.

Organizing the Data

Once we determined that we had a few authors in the data set with more than 150 full reviews each we decided the data set would be usable. We chose to use 100 reviews to train the intelligent agent and then use the remaining reviews to test how smart that the agents had become. To train the agent, we put the training data set in to a B-Tree using Mumps separating each review by author and then further separating by each movie each author had reviewed. A B-Tree is similar to a binary tree in that they both have some of the same rules. You can traverse a B-Tree just like a binary tree but the main difference is the number of nodes you can have at each level. In a B-Tree you can have as many nodes as you want at each level. The top level of a B-Tree may have 100 or more nodes as a binary tree can only have one. Then from this tree we picked one author who had around 150 full reviews to use as a test subject. This author was then separated in to their own B-Tree where all of the review ratings were normalized to an n-of-10 score to make the creation of the intelligent agent easier and to have more concise results. Also any word that was less than three characters was eliminated from the dataset as the words usually have little meaning.

The Agents

The agents that we created for this project used a single term matrix, a double term matrix, and a triple term matrix. A single term matrix is basically an one by n array of all the words that appear in the reviews. And each of these words has a value assigned to it that is its 'worth'. A double term matrix is a matrix that is a n by n matrix containing all the possible pairs of words that appear next to each other in the reviews. Also each pair of words has a value assigned to it that is its 'worth'. A triple term matrix is a matrix that is n by n by n matrix of all possible triplets of words that appear next to each other. Also each triplet has a value assigned to it that is its 'worth'. The initial 'worth' of a word, pair of words, or triplet of words is referring to the number of times that the agent counted them as being in the document. This initial 'worth' is edited later in the process to come up with a more meaningful value and is explained later in this paper.

Single Term Matrix

The top level of the single term matrix for the intelligent agent consisted of ten nodes one for each possible value of that a review can have on a one through ten scale. Then the level of the B-Tree below each node is a list of all possible words that appear for each review score leaving out words less than three characters. Each word node has a count assigned to it representing how many times that word appeared in all the reviews of that specific rating. While creating this B-Tree of words for each review we also created a B-Tree of all of the words and kept a count of how many times each word appears in the test set over all. Then after both trees were created we went back through the B-Tree that was separated by each review score and then went through each possible word in each of those nodes and divided the word count that was held there by the total word count for that word overall in all of the documents (Term Frequency Inverse Document Frequency). By doing this we had now established how much each word was 'worth' to each review score.

The next step was to test the intelligent agent that we had created. We did this by going through the remaining reviews that that author had left one at a time. For each review we created a B-Tree of all the words in the review and for each word kept a count of how many times each word had occurred in that specific review. Then we went had the intelligent agent go through the B-Tree we had created of rated reviews and establish a score for the new review by adding up the product of the word count times the 'worth' of that word for a specific score. The agent did this for all ten possible score levels. Then the score with the highest number at the end was declared the winner and the new was assigned that score. We then had the agent compare the score it gave the review with the actual score that the author had given it and then kept track of an average amount off (Mean Absolute Error) and the largest off that the agent had been (Maximum Error).

Double Term Matrix

The top layer for the double term matrix is the same as it was for the single term, one node for each possible review value. The next layer of this B-Tree is slightly different; each node is now represented by two words not one. So when parsing though each individual review to create the test tree, the agent captured two words at a time and then shifted one word to the right and then captured those two words, while skipping over words that are less than three characters. A word pair count was kept this time keeping track of how many times each pair of words appeared next to each other for each possible review. Also there was a word pair count kept for all possible reviews kept for making the same kind of calculations that were made for the single term matrix. After creation of both B-Trees the Intelligent then made the calculations for each word pair to give their 'worth'.

The next part was to test the double term matrix that we had created. This was done much the same way as we did with the single term matrix except we doubled up the words in each node. Then we had the intelligent agent go through and compare each of the ten possible scores looking for the highest total score for each specific review. We then had the agent compare the actual score with the score it had assigned it and keep track of the average amount off and the largest amount off.

Triple Term Matrix

The top layer of the triple term matrix is the same as the double and single term, one node for each possible review value. The next layer of this B-Tree is slightly different from the two-term matrix by instead of having two words in a node now there are three words per node. This agent also skips over words that are shorter than three characters. So now when the intelligent agent is parsing though all the base reviews it absorbs three words then shifts to the left one word and then absorbs three more words at a time (two of these words are the same as the previous time just in different positions). Also there is a 'triple word' count kept for each group of words and for each group of words for all the documents as a whole. These two counts are then divided just like they were in the single and double term matrices to give each set of terms their 'worth'.

The triple term matrix was then tested just like the single and double term ones were. The test reviews were now put in to triple term nodes and matched up and scored just like before. Then the agent took the highest score out of all of the possible ten scores and compared it to the actual score. The average amount off and the most ever wrong on all of the agent's predicted scores was recorded just like with the other two tests.

Results

The single term matrix had a maximum error of six points on a ten-point scale. This means that the most the intelligent agent was ever off on one of the guesses for what the reviewer would have given the review was six. But the average amount that the agent was off demonstrates that this was not a common occurrence. The average amount that the agent for the single term matrix was off was 0.545454 in either direction. This means that on average the agent was either half a point high or low on the prediction.

Agent	**MAE**	**MAX**
Single	0.55	6
Double	0.0	0
Triple	0.55	5

The double term matrix turned out to yield out best results. The maximum error for the triple term matrix was zero. This means that the agent never guessed wrong. And as one might guess the average amount wrong was also zero because of this.

The triple term matrix ended up having slightly better results than the single term matrix but not as good of results as the double term matrix. The maximum wrong that the agent for the triple term matrix ever had was five. Although the triple term matrix had a lower maximum wrong, which was desirable, it had a slightly higher average amount that the agent was wrong with a score of 0.5511363636. This means that it was wrong a little more often than the single term matrix.

These results obviously point to the fact that the double term matrix is the superior way to decide how to have the agent evaluate what score to give a movie review. Assuming that the agent is working on a matrix that was created by using reviews written by the same author as the review that the agent is making the prediction for.

Future Work

For the future we would like to try and evaluate all three methods on the full movie review database and to see if we get similar results. Also it would be nice to see if these methods are portable to other domains other than just movie reviews. The current belief that we have would be that the triple term matrix would end up being the superior way of evaluating reviews but more time and processing power is needed for a full evaluation of the dataset.

References

1. Good, N., et al. (1999). Combining Collaborative Filtering with Personal Agents for Better Recommendations. Proceedings of AAAI-99 pp.439-446.

2. Maes, P. (1994). Agents that Reduce Work and Information Overload. CACM 37(7) pp.31-40.

3. MovieLens website. http://movielens.umn.edu.

4. O'Kane, Kevin C. (1999), "An M Compiler for Internet server applications", M Computing, 7(1):11-17.

5. Rotten Tomatoes website. http://www.rottentomatoes.com.

6. Schafer, J.B., Konstan, J.A., and Riedl, J. (2001). E-Commerce Recommendation Applications. Data Mining and Knowledge Discovery 5(1/2) pp.115-153.

7. Shardanand, U. and Maes, P. (1995). Social Information Filtering: Algorithms

for Automating Word of Mouth. Proceedings of CHI-95 pp.210-217.