Bayesian and Neural Networks for Motion Picture Recommendation

by Robert Russo

Boston College Honors Thesis
Advisor: Professor Sergio A. Alvarez
May 12, 2006

Abstract

This thesis applies machine learning techniques to a dataset of movies described by collaborative (social) and content attributes in order to create a mixed recommender system for movies. Bayesian networks, two versions of neural networks, decision trees, and simple rule classifiers are compared. It is determined that Bayesian and neural networks outperform the remaining techniques. Both techniques achieved remarkably high top-10 precision, an important metric of recommendation quality. An attempt to contrast recommendation quality for content-only and collaborative-only datasets as compared with a dataset described by both content and collaborative attributes yields inconclusive results. Insufficient content information in the current datasets may be the reason for this.

Introduction

Definition: *A **recommender system** is a system that takes data about a user's past history in a certain industry, such as products they have purchased, movies they have seen, or websites they have visited, and predicts what the user may prefer to purchase or see in the future.*

Some recommender systems are *collaborative* systems, in which other users' past histories are used in trying to link the particular user to a group of users with similar interests or purchases. This group will then influence what the recommender system will output based on what the group likes or dislikes. Other systems are *content-based* systems, in which details of the product, movie, website, or other item are compared against those of similar items that the user has been in contact with. These similar items are helped to gauge whether this user will like or dislike the item. Some recommender systems use a mixture of collaborative and content-based approaches.

Recommender systems have become a popular subject starting in the late 1990's (Resnick & Varian, 1997). Many online retailers, such as Amazon.com, use recommender systems in order to recommend new products to customers in order to try to maximize profits. For example, suppose a customer buys a science-fiction book. The next time the customer visits the site, it might recommend books by the same author, or may suggest other science-fiction books that other customers have bought in addition to the one the customer purchased the last time he or she visited. By having this type of system in place, retailers are able to allow customers to more easily find products that they may be interested in than their retail store counterparts.

There are some recommender systems which allow users to listen to different types of music and set preferences based on the songs they listen to. These types of recommender systems, such as Yahoo.com's LAUNCHcast Radio, allow a user to get instant updates in recommendations due to being able to rate songs as they are playing. As such, users are able to listen to more songs that they could possibly like on a personalized radio station.

Many websites also use recommender systems to personalize their interface with users in order to maintain visits to their site. For example, at a news site, if a person enters their zip code, the site may contain local news, weather, and sports that the user may prefer to read over national or global content. By allowing this customization users may better enjoy the site and more easily find articles that they are interested in.

My intention is to find a better algorithm which combines content-based data about products and collaborative data from other consumers in order to determine whether a target use would like an item. I focused on predicting whether a user will like or dislike a film based on the details of the film (content-based details), as well as what similar users have rated the film (collaborative details). By ranking the movies based on their probability of being liked, an effective movie recommender system can be set up. There are some movie recommender systems already on the market such as Reel.com's Movie Matches which attempt to provide recommendations. The dataset I shall be using in order to test algorithms is the MovieLens dataset, which is comprised of movies with some content details, as well as ratings from various users about each movie (Harper, Li, et. al., 2005).

Machine Learning Concepts

Machine learning is a rapidly growing field within computer science.

Definition: "*A computer program is said to* ***learn*** *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997).*

Definition: *A* ***dataset*** *is a group of data. It is basically an n-by-m matrix with n rows and m columns. The rows are called* ***instances****. Instances are basically different occurrences of a situation. The columns are called* ***attributes****. Attributes are certain details that were recorded during every instance.*

Figure 1 is what a generic dataset may look like.

Figure 1: **Sample Dataset**

Student	Graduation Year	Major	GPA
Bob	2005	Computer Science	4.0
Joe	2006	Math	3.5
Kim	2004	Drama	2.6

The dataset is then put into a classifier.

Definition: *A* ***classifier*** *is an algorithm that maps the instances of a dataset to a discrete set of bins (****classes****).*

The output of the classifier is normally one of the attributes listed in the dataset, normally defined as the target attribute. The dataset is normally broken into a training dataset, with which the classifier builds itself in order to make hypotheses about the data and how to predict values, and a test dataset which the classifier uses to classify new instances that it has never seen before.

Simplistic Rule Classifier

One of the simplest classifiers is the rule classifier. In particular, there are simplistic rule classifiers which apply rules based on different values of attributes. In this way, different rules are applied to different attributes, and based on these rules an output is chosen. The simplest of these rule classifiers is the majority class classifier, called 0R. The 0R classifier takes a look at the target attribute and will always output the value that is most commonly found in that column. The next common one is one in which there is only one rule applied to the dataset, called 1R. In this case that classifier will select one attribute and find the best classification rule based on it. Using the value of this attribute for a given instance, the rule will then predict the value of the target attribute.

Decision Trees

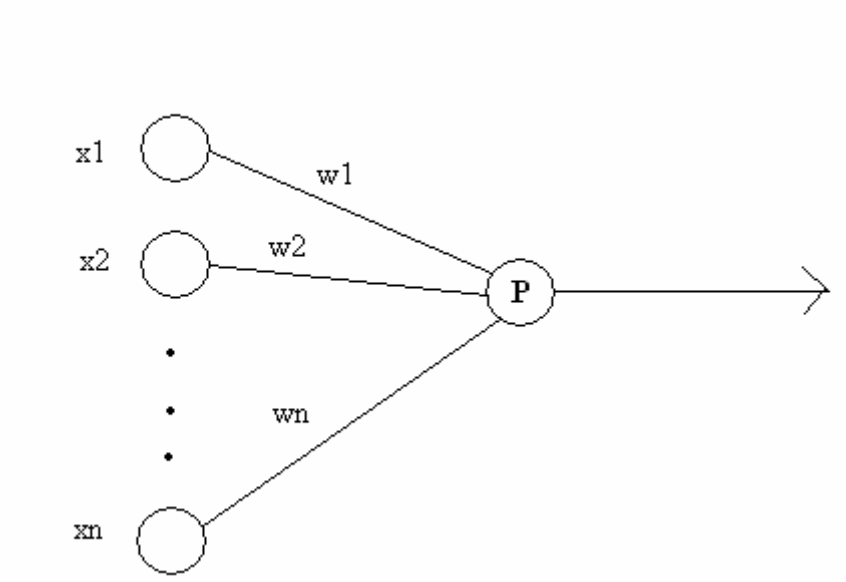
Another classifier which builds upon rule classifiers is one with decision trees. Decision trees basically take a look at multiple attributes and break them into separate pairings, predicting different values for the target attribute based on the pairing of other variables. New branches are made for new pairings and as a result an entire tree is made, which the classifier branching at each point depending on what the value of a particular attribute is. Many algorithms have been made using decision trees, the most basic among them being the ID3 algorithm (Quinlan, 1993). However, decision trees initially had a problem in overfitting data, in that random occurrences would create new branches to trees and would therefore lead to a more complex tree, one that could predict false outcomes due to some instances whose target attribute value would be different than other instances with similar values for its attributes. As a result, new algorithms were made using pruning. Pruning would remove branches of the tree if it improved the overall accuracy of the tree. Examples of decision tree algorithms include C4.5 (Quinlan, 1993) which built upon ID3 by adding pruning, as well as allowing numeric attribute values.

Neural Networks

Artificial neural networks are another type of machine learning technique. It was inspired by the neurons inside a human brain which connect to each other and generate outputs based on stimuli from other neurons. In artificial neural networks, perceptrons are used instead of neurons. A perceptron takes in many inputs and assigns some constant called to each one, called a weight. The weight represents the importance of the input to the perceptron. In the machine learning case the inputs are the different values of the attributes of an instance. The perceptron then adds all of the inputs multiplied by their weights. It then applies a threshold to this value. If it is above 0, the perceptron will output a 1. If the value is not above 0, then the perceptron will output a -1. For example, Figure 2 is a diagram of a perceptron.

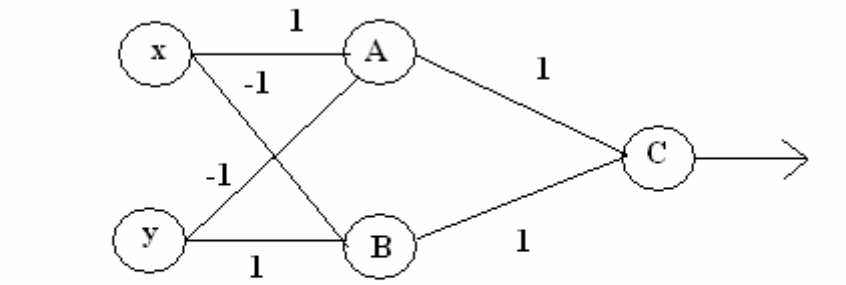
Formula: Given inputs $x_1, x_2, \ldots, x_n$, a perceptron will assign weights $w_1, w_2, \ldots, w_n$ to each of them. It will then calculate $x_1w_1 + x_2w_2 + \ldots + x_nw_n$) and then output a single value based on whether or not that value is greater than 0.

Figure 2: Example of a Perceptron



Perceptrons can be hooked up with one another like neurons, in order to create networks consisting of multiple layers of perceptrons. For example, such a network can be used to compute the Boolean logic function XOR. Figure 3 is a diagram of how doing x XOR y would work.

Figure 3: A Network of Perceptrons that Computes the XOR Boolean Function



In this case, A, B, and C are perceptrons. A would only output a 1 if x is 1 and y is 0. B would only output a 1 if x is 0 and y is 1. C would only output a 1 if A or B is outputting a 1. Therefore, this network of perceptrons models the XOR function. In general, perceptrons are referred to as nodes, and perceptrons that only output values that go into more perceptrons are called hidden nodes.

Network Training

The way a neural network learns is by training its weight values. Each time an iteration of passing the inputs through the network is done, the output values are compared against the target attribute's values. Then a calculation is performed to create the delta that must be added to the original weight. This continues until the weights do not change, or until the amount of pre-determined iterations is reached. There are several ways to update weights for perceptrons. One way is to assign random weights to each input and then iterating the perceptron using the following rule for updating each weight w_i for input x_i:

Formula: $w_i \leftarrow w_i + \Delta w_i$ where $\Delta w_i = \eta(t - o)x_i$

In the above formula, η is the learning rate used to determine to what extent the weight will be changed, t is the target attribute's value, o is the output value given by the perceptron, and x_i is the input value fed into the perceptron. This method guarantees convergence if the classes are linearly separable (Minsky & Papert, 1965). However, most datasets do not have linearly separable classes.

A solution to this problem was using error backpropagation. In this way, the outputs are calculated, and then the error is sent backwards through the network to update the weights. Each output perceptron k calculates its error term δ_k using the following formula:

Formula: $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$

Here, o_k is the output value for the perceptron, and t_k is the target attribute value. These values are used in calculating the error term for each hidden node h using the following formula:

Formula: $\delta_h \leftarrow o_k(1 - o_k)\Sigma\, w_{kh}\delta_k$

Note that w_{kh} denotes the weight from node h to node k. The term $\Sigma\, w_{kh}\delta_k$ for hidden node h is therefore the sum of the weights times the error values of all the output nodes k that are connected from h.

The weights are then updated in the following way:

Formula: $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where $\Delta w_{ji} = \eta\delta_j x_{ji}$

This update rule reduces the mean square error at the output layer. Connecting many perceptrons together with many hidden layers using error backpropagation can help improve accuracy but leads to a sharp increase in training time. Varying the threshold, the number of layers, and the amount of time taken to train the network can all help improve accuracy. Tests have been done using these different variations. (Rumelhart, Widrow, & Lehr, 1994).

Using error backpropagation in the way above uses a threshold in order to converge to one value. However, there is a way to modify error backpropagation where instead of using the threshold, a probability is instead computed, and that is used in determining the output attribute value. In this way, the value for the error term for each output node k would then be calculated in the following way:

Formula: $\delta_k \leftarrow o_k(t_k - o_k)$

The value for the error term for each hidden node h would then be calculated in the following way:

Formula: $\delta_h \leftarrow o_k \Sigma w_{kh}\delta_k$

In this way, the threshold would be removed and be replaced with a probability. Various work has been done using this modified form as well (Bishop, 1996)

Bayesian Techniques

Another type of classifier uses Bayesian reasoning. Bayesian techniques are based on probability distributions and that using these probabilities on observed data can improve performance. It tries to produce the best hypothesis from some space of hypotheses H given some training data D. Most of Bayesian learning relies on Bayes' theorem. Bayes' theorem assumes that for each hypothesis h, there is a prior probability already calculated, called P(h). P(D) is the prior probability that the training data D will be observed. P(D|h) is the probability that the training data D will be observed given that the hypothesis h holds. And P(h|D) is the probability that the hypothesis h will hold given the training data D. To determine P(h|D), Bayes' theorem provides the following formula:

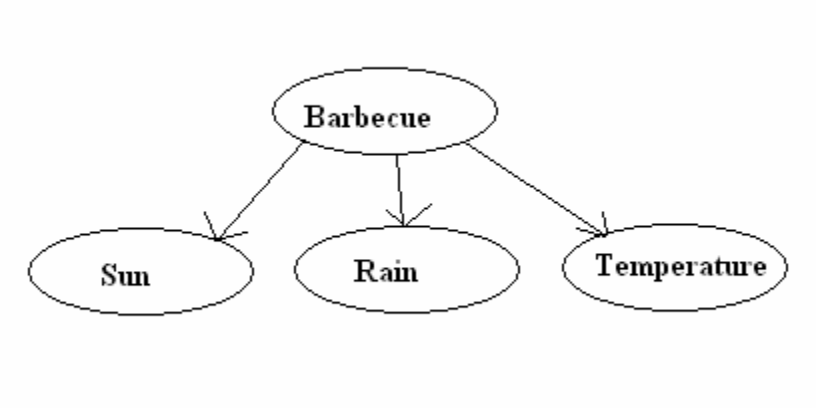
Formula: $P(h|D) = P(D|h)P(h) / P(D)$

Bayes' theorem allows classification by selecting the maximum a posteriori (MAP) hypothesis. For every h in H, P(D|h)P(h) is calculated. The denominator P(D) in Bayes' formula is removed because the training data D never changes for each change of h in H. The hypothesis h with the highest value of P(D|h)P(h) is then used to classify each instance.

Naïve Bayes

There are two standard methods in which Bayesian learning is done. One is a technique called Naïve Bayes. In Naïve Bayes, the algorithm creates a set of all possible target attributes. It then calculates the probability terms P(h) and P(D|h) as stated above. However, it breaks each different attribute value d of the training data D and calculates each one of them separately. It then takes the maximum values of P(h) multiplied by the product of all the probabilities of P(D|h). For example, consider Figure 4.

Figure 4: Example of a Naïve Bayes model



Barbecue is the target attribute, and has the values yes or no. The classifier would first build up the different instances of the barbecue data D and calculate each probability given either yes or no. It would also calculate the overall probability of barbecuing. For example, to classify the instance (Sun = out, Rain = none, Temperature = hot), the classifier would calculate P(yes) * P(out|yes) * P(none|yes) * P(hot|yes) and P(no) * P(out|no) * P(none|no) * P(hot|no) The end result would be the classifier selecting the maximum value of the two and choose either yes or no. Naïve Bayes has been used for tasks such as sorting out news articles (Joachims, 1996).

Bayesian Networks

Naïve Bayes assumes that all attribute values are conditionally independent given a target attribute value. Thus, there needs to be a way to classify some attributes as conditionally independent, but not others. The solution is a Bayesian network.

Definition: *Attributes are **conditionally independent** of one another if given the value of one or more attributes $Y_1 \ldots Y_m$ determines the value of attributes $X_1 \ldots X_m$ independent of values of attributes $Z_1 \ldots Z_m$* (Mitchell, 1997).

In this way, a network is created by connecting attributes that are not conditionally independent of one another and calculating their conditional probability. Therefore a system of nodes is created similar to the system created by neural networks. In this situation, you can infer target values for attributes by calculating the prior probability of each value given the parents of that node. In this way you can connect nodes together and have a network of nodes having many parents. An example of a Bayesian network is shown in Figures 5a and 5b.

Figure 5a: Example of a Bayesian Network

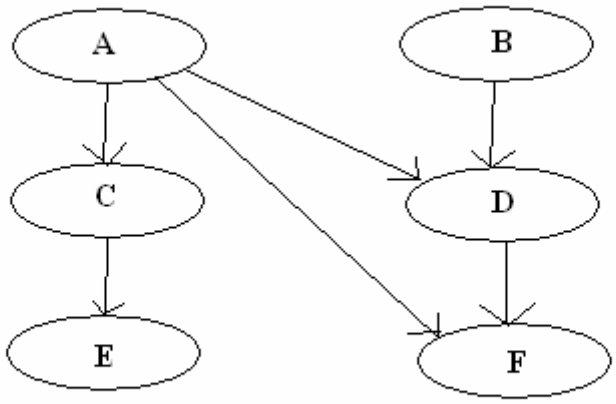


Figure 5a: Example of a Conditional Probability Table

	a,d	a, ¬d	¬a,d	¬a, ¬d
f	0.4	0.1	0.8	0.2
¬f	0.6	0.9	0.2	0.8

In this example C has parent D, E has parent C, D has parents A and B, and F, the target attribute, has parents A and D. It is not joined in layers as in neural networks, as the parents of nodes can also be the parents of their children. Therefore calculating the probability of F being f, given the data that A is a and D is d would be, for example,

would be P(F=f|A=a,D=d). Using the conditional probability table above, this value would therefore be 0.4.

Unlike neural networks, in which errors want to be minimized, Bayesian networks can be trained to maximize the probability of the observed data given the network parameters. Weights (conditional probabilities in this case) are updated using the following formula:

Formula: $w_{ijk} \leftarrow w_{ijk} + \eta \Sigma(P_h(y_{ij},u_{ik}|d)/w_{ijk})$

In this formula, w_{ijk} is the conditional probability calculated, y_{ij} is the value of the attribute, u_{ik} is the values of the parents, and d is the current piece of data from the data D. Σ sums up all these values within the dataset D, and η is the constant learning rate as before. The weights must then be normalized to be between 0 and 1 and so that the sum of the probabilities for each weight given the values y_{ij} and u_{ik} equals 1, as 1 is the probability that one of the attribute values occurs.

However, creating a Bayesian network is complex and could take a long amount of time. Some algorithms have been tried in order to speed up the process while trying to retain the Bayesian network structure. One such algorithm is the K2 algorithm, which focuses on accuracy over training data instead of the complexity of the Bayesian network (Cooper & Herskovits, 1992).

I shall be using these various machine learning concepts and applying them to the dataset. In applying the techniques described above I hope to achieve maximum accuracy over the dataset. In particular I shall be focusing on neural networks and Bayesian networks.

Machine Learning Work with Recommender Systems

Many collaborative filtering techniques have been used before. For example, Ringo was a social information filter used to try and make music recommendations to listeners (Shardanand & Maes, 1995). In such, recommendations are passed through "word of mouth", rather than content, so music recommendations may be drastically different than those that a user has listened to before. There have been many other tests including testing with various predictive collaborative-based algorithms in order to improve accuracy (Breese, Heckerman, & Kadie, 1998). The MovieLens dataset was used in helping to determine different ways of collaborative filtering using user ratings about movies (Harper, Li, et. al., 2005). I will be using information in this dataset in order to evaluate various machine learning algorithms.

Many machine learning algorithms have been applied to recommender systems as well (Billsus & Pazzani, 1998). Such approaches include neural networks and decision trees. I shall be using neural and Bayesian networks in this thesis. There have been experiments with the combination of both collaborative filtering and content-based filtering. The Fab system was created as a hybrid in order to incorporate both methods to increase personalization for users (Balabanovic & Shoham, 1997). Probabilistic models that combine both filters have been tested as well (Popescul & Ungar, 2001). There is also an upcoming paper involve a "mixture of experts" neural networks that statistically improves recommendation results more than just pure collaborative filtering (Alvarez, Ruiz, et. al., 2006).

The Dataset

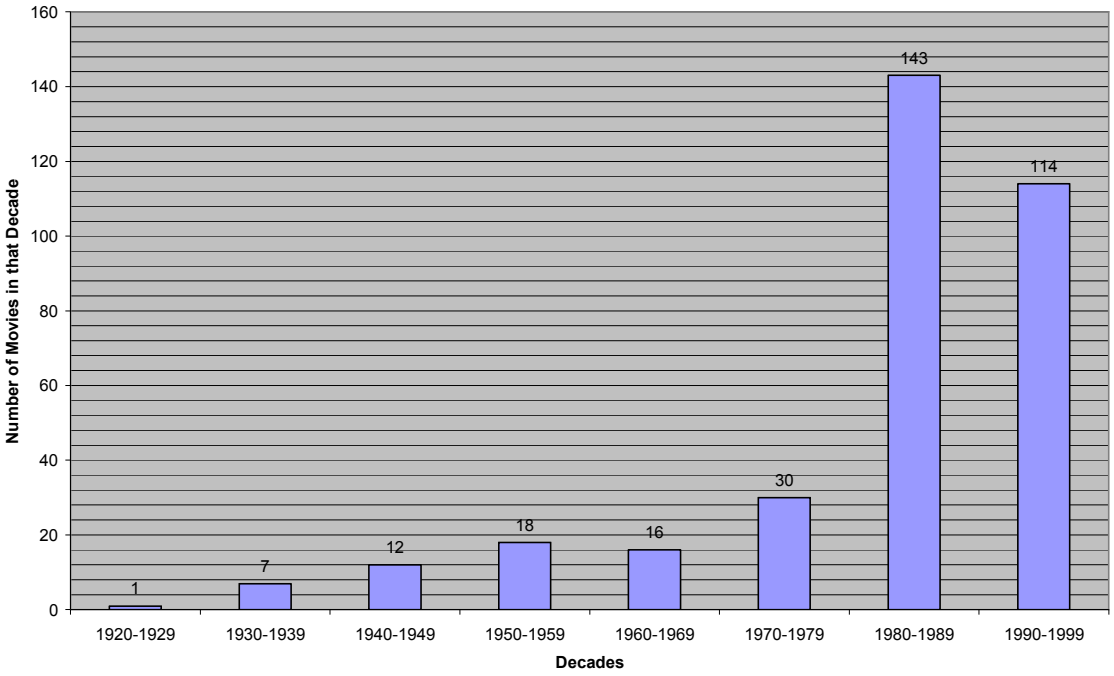
I have created a dataset using data from the MovieLens dataset (Harper, Li, et. al., 2005). The MovieLens dataset is composed of a set of movies with content information about the movie and ratings from many users for each movie. I have created a sample dataset containing 337 instances of movies, each of which has been seen by at least 100 users and all of which have been seen by the target user.

Each movie has the following information broken up into 1042 attributes:

Content-Based Information

- Year
  - Year in which the movie was released
  - Ranges from 1926 to 1999

Figure 6: Year Distribution



- Genre
  - Genres the movie was considered a part of
  - 3 genre attributes
  - Possible values: (Drama, Animation, Musical, Action, Comedy, Adventure, Romance, Children's, Thriller, Crime, Western, Documentary, Mystery, Horror, Sci-Fi, Film-Noir, War, and/or Fantasy)
  - 160 (47%) missing values in Genre2, 277 (82%) missing values in Genre3

Figure 7a: Genre1 Distribution

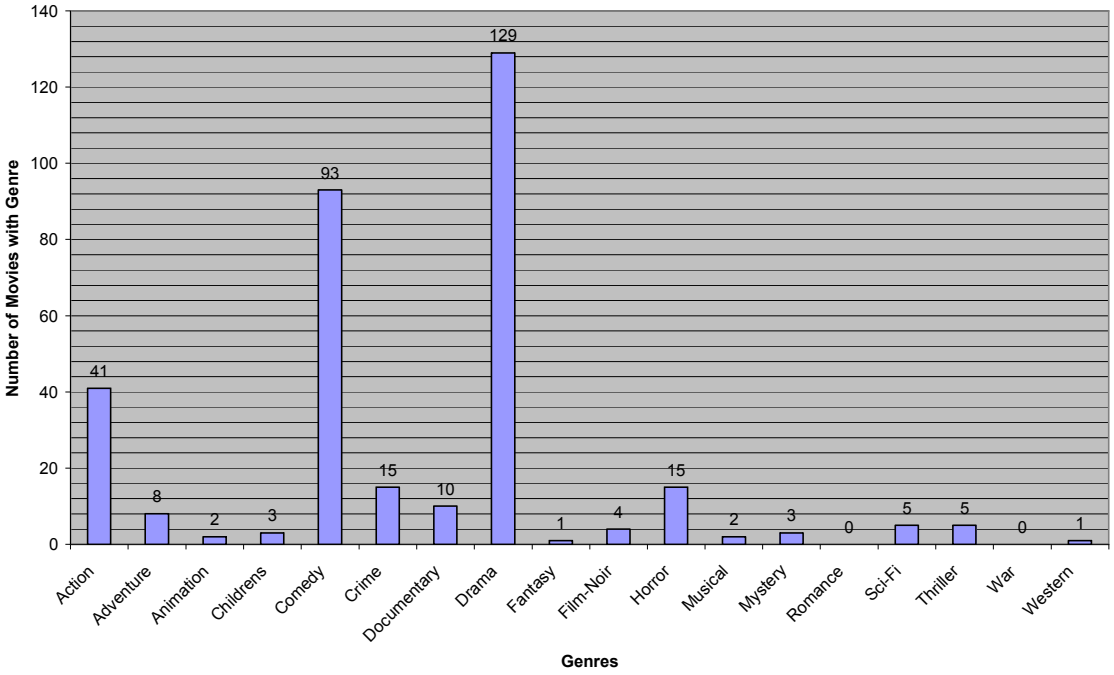


Figure 7b: Genre2 Distribution

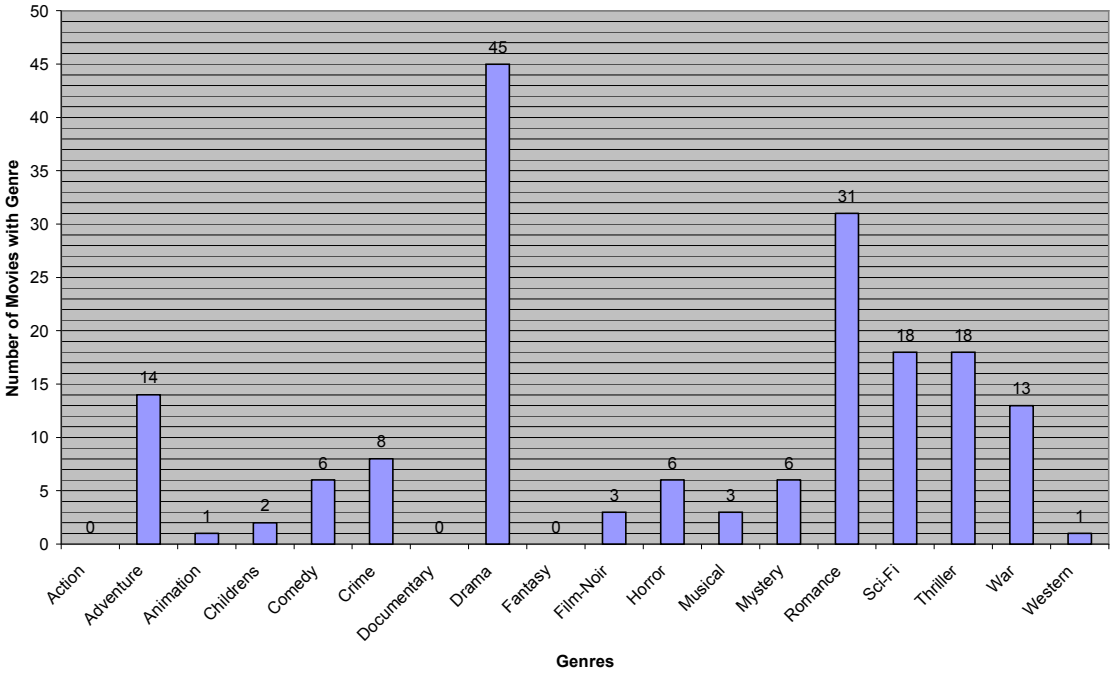
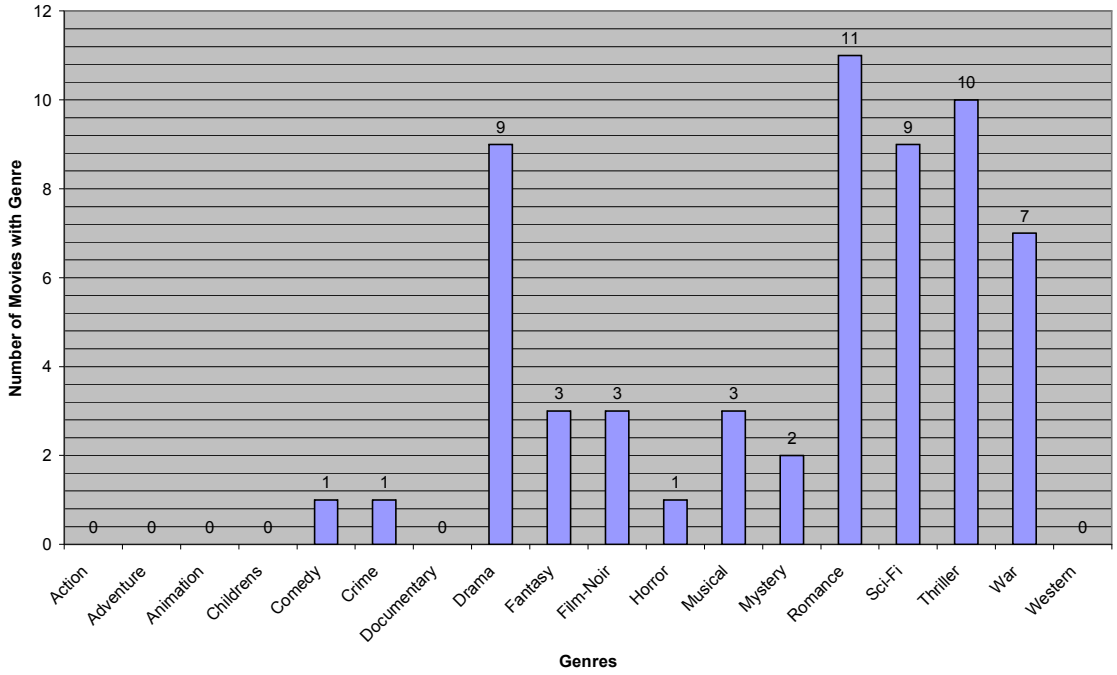


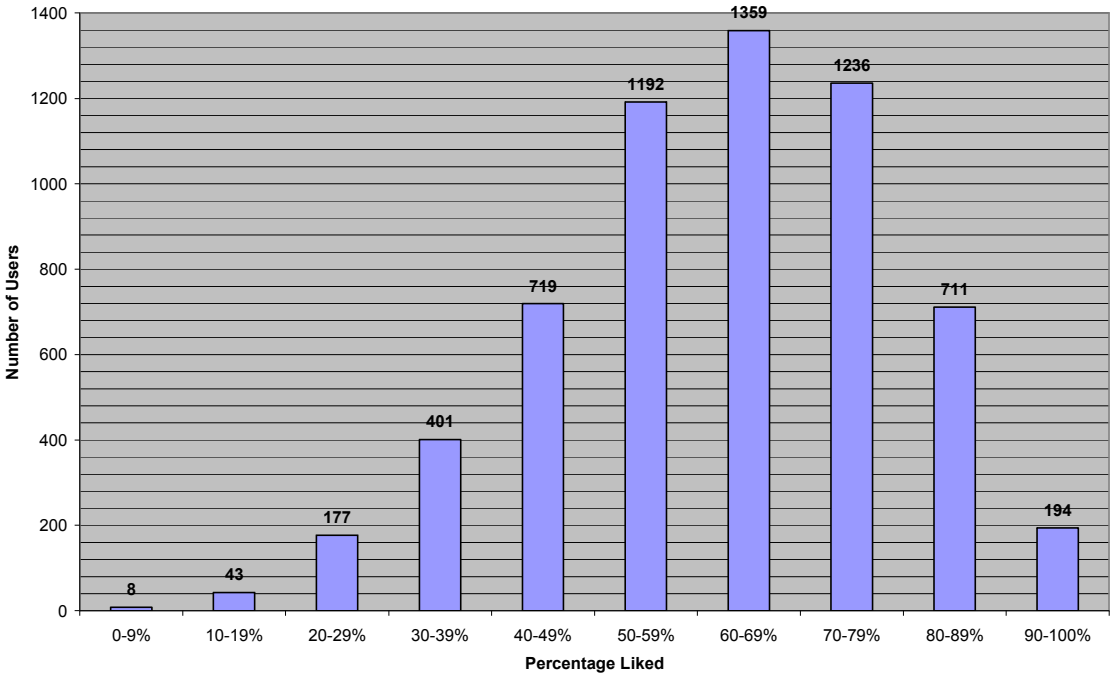
Figure 7c: Genre3 Distribution



Collaborative Information

- Like/Dislike
  - 698 binary like/dislike ratings for each user
  - Like = 1 and Dislike = 0 if user gave the movie a rating of 3, 4 or 5
  - Like = 0 and Dislike = 1 if user gave the movie a rating of 1 or 2
  - Like = 0 and Dislike = 0 if user has not seen the movie

Figure 7d: Number of Movies Liked By User Per Number of Movies Seen



Target Attribute

- Like
  - 1 binary like rating for target user
  - Like = 1 if user liked movie
  - Like = 0 if user disliked movie

Datasets Used in Thesis

I decided to separate the data into content-only, collaborative-only, and a mixture of both collaborative and content information. The content-only dataset contains the year and 3 genre attributes of a film and the target user's like. The collaborative-only dataset contains the binary like/dislike ratings for each user and the target user's like. The combined version of the dataset incorporates both the content-only attributes and the collaborative-only attributes. I used the combined version for all testing except for evaluating the combined version against the content-only and collaborative-only versions.

Performance Metrics and Evaluation Protocol

I decided to evaluate the results of tests of the machine learning algorithms on the dataset by using accuracy, precision, and top-N precision.

Definition: ***Accuracy*** *is the percentage of instances that are correctly classified by the system.*

Definition: ***Precision*** *is the percentage of like predictions that agree with the user's taste.*

Certain machine learning algorithms, such as Bayesian and neural networks, output numerical values that may roughly be interpreted as probabilities of each of the possible values of the target attribute. Instances may then be ranked by their *like* probability

Definition: ***Top-N Precision*** *is the percentage of top N like predictions that agree with the user's taste. This can only be done for classifiers that can rank instances by their like probability.*

The simple majority class classifier 0R can be used as the baseline for the data for which to test results. The accuracy and precision of 0R over the datasets used in this thesis are 62.02 %. I used top-10 precision as the baseline for top-N precision, as most users do not want to look through more than 10 movies when choosing a movie.

Performance was evaluated using 10 repetitions of 10-fold cross-validation. In 10-fold cross-validation, the instances are broken up into 10 non-overlapping groups. For each group, the classifier uses the other 9 groups to train on and uses that one group for testing. Performance measures are averaged over the 10 folds. The entire process is then repeated 10 times, for a total of 100 folds, each involving a different split of the dataset into training and testing portions.

Attribute Selection

Because the dataset was large and took a long amount of time to process, the attributes were filtered using attribute selection techniques, resulting in a smaller set of attributes that contain only important information about the initial dataset. Two attribute selection techniques were used: principal components, which combines information about attributes, and BestFirst, which selects a subset of the original set of attributes. By doing so, the number of attributes was reduced from 1402 to 251 in principal components and 65 in BestFirst.

Software

In order to perform these experiments, I used a program called Weka. Weka is a Java program developed at the University of Waikato in New Zealand that contains machine learning algorithms and can evaluate the accuracy and precision of these

algorithms on datasets (Witten & Frank, 2005). It also contains methods for doing attribute selection.

Because Weka does not have a built-in function for top-N precision, I had to create Java code that implements the machine learning algorithms given in Weka. Because neural networks and Bayesian networks can output the probability of the target user liking a movie, I was able to rank each instance based on its like probability. I could then take the top N of them and evaluate them using Weka's built-in methods. The code used for top-N precision is shown below in the Appendix.

Experimental Parameters

There were many parameters I experimented with. For neural networks, I experimented with two different variants of error backpropagation, and varied the value of the learning rate and the number of hidden nodes used. For Bayesian networks, I changed the maximum number of parents used. For attribute selection, I used both principal components and BestFirst. And for the dataset, I used content-only, collaborative-only, and combined datasets.

Results

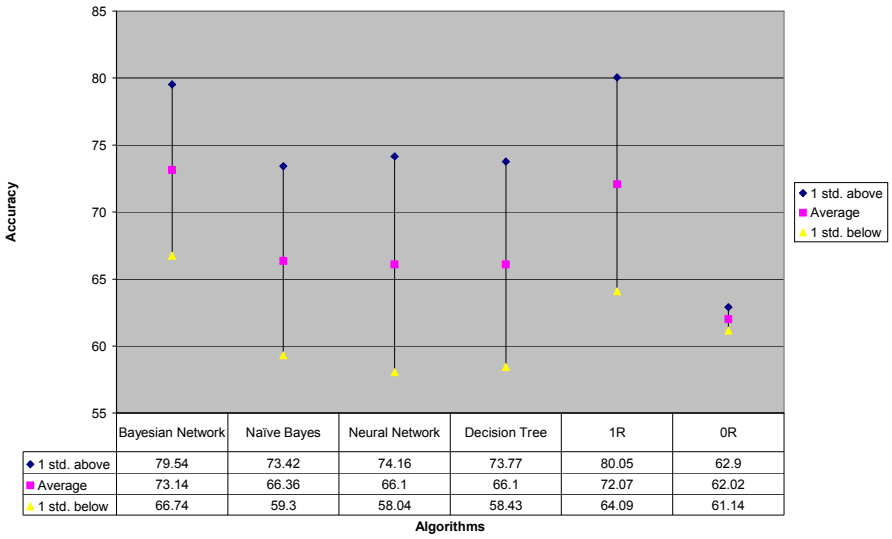
Attribute Selection

I tested both principal components and BestFirst to select the attributes to be fed as inputs to the various machine learning algorithms.

Principal Components

Figures 8a, 8b, and 8c show the performance of the different machine learning algorithms over data that was preprocessed using principal components attribute selection.

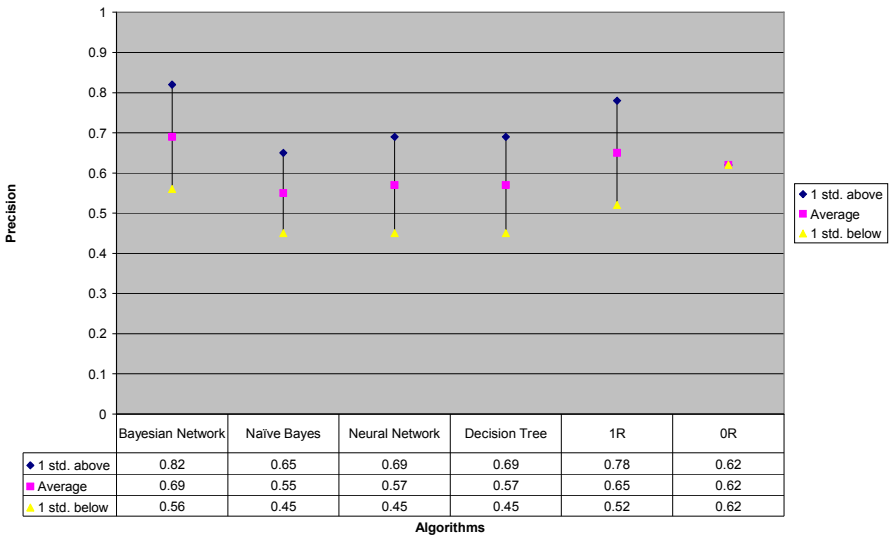
Figure 8a: Accuracy Using Principal Components Attribute Selection



	Bayesian Network	Naïve Bayes	Neural Network	Decision Tree	1R	0R
1 std. above	79.54	73.42	74.16	73.77	80.05	62.9
Average	73.14	66.36	66.1	66.1	72.07	62.02
1 std. below	66.74	59.3	58.04	58.43	64.09	61.14

Algorithms

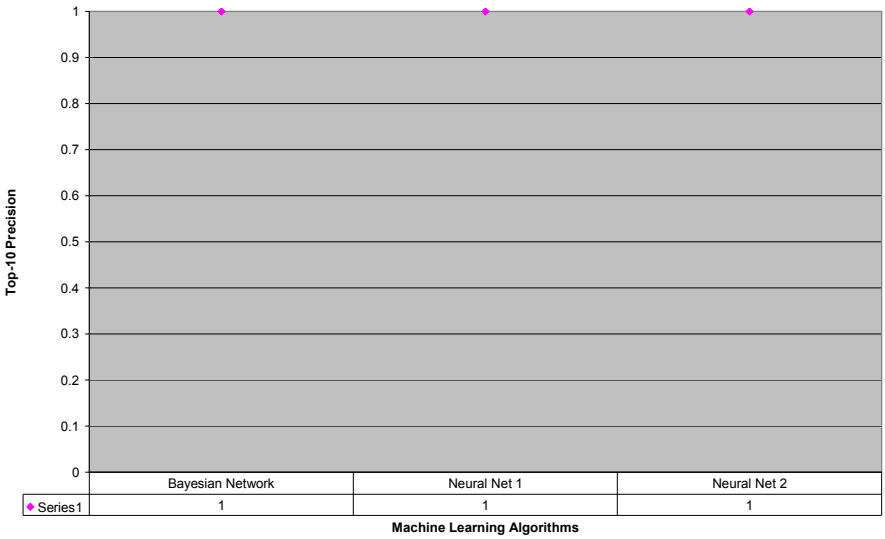
Figure 8b: Precision Using Principal Components Attribute Selection



	Bayesian Network	Naïve Bayes	Neural Network	Decision Tree	1R	0R
1 std. above	0.82	0.65	0.69	0.69	0.78	0.62
Average	0.69	0.55	0.57	0.57	0.65	0.62
1 std. below	0.56	0.45	0.45	0.45	0.52	0.62

Algorithms

Figure 8c: Top-10 Precision Using Principal Components Attribute Selection

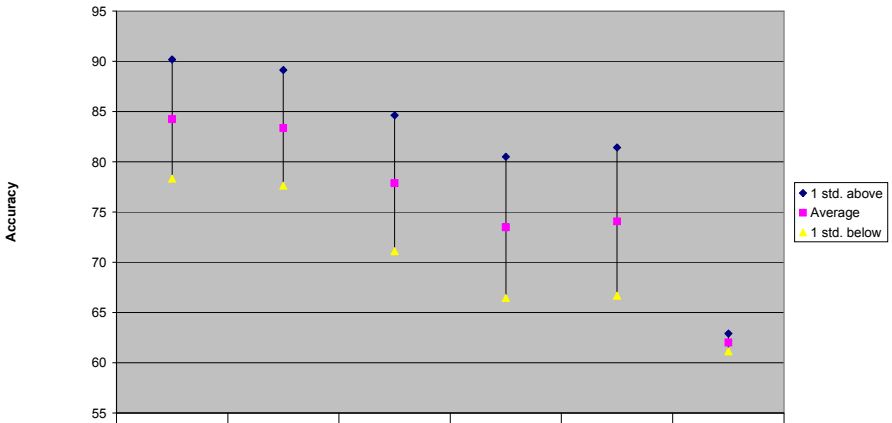


Looking at the graphs, both Bayesian networks and neural networks had a remarkable 100% top-10 precision. Bayesian networks are somewhat superior in terms of accuracy and precision as compared to its naïve counterpart, as well as to neural networks, decision trees and 0R. However, 1R works almost as well as Bayesian networks. Conducting both algorithms on the dataset prior to principal components, it appears that the Bayesian network makes the target attribute value the parent of all other attributes and uses that for its network structure. 1R classifier separates its predictions into years. However, because the dataset is small, and because the year distribution is mainly in more recent year as shown above, the 1R only breaks predictions into years more distinctly in years in the 1980's and 1990's. Also, Bayesian networks achieve slightly better precision than 1R.

BestFirst

I then evaluated the performance obtained using BestFirst attribute selection. The results are shown in Figures 9a through 9c.

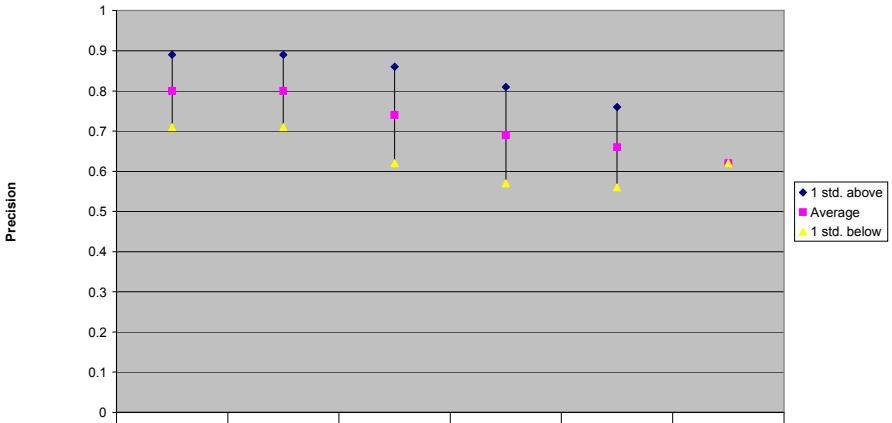
Figure 9a: Accuracy Using BestFirst Attribute Selection



	Bayesian Network	Naïve Bayes	Neural Network	Decision Tree	1R	0R
1 std. above	90.18	89.14	84.64	80.51	81.44	62.9
Average	84.25	83.38	77.88	73.48	74.06	62.02
1 std. below	78.32	77.62	71.12	66.45	66.68	61.14

Algorithms

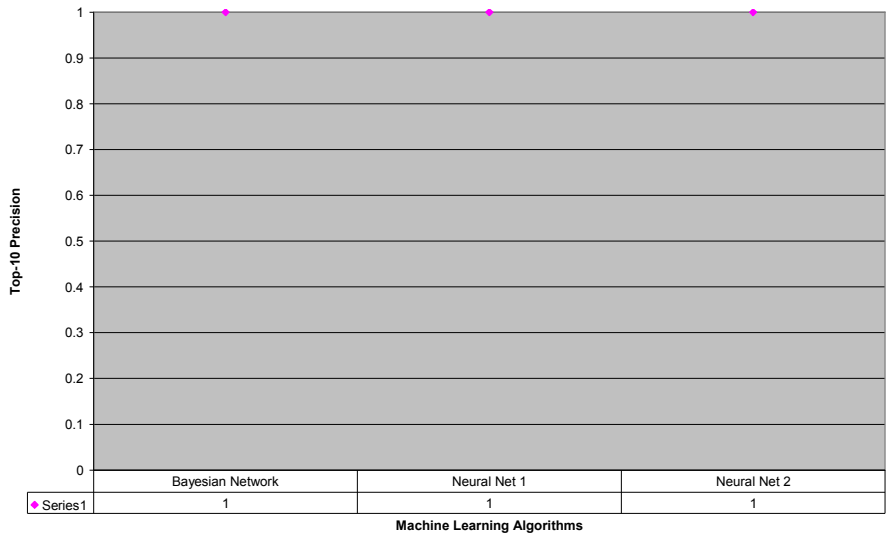
Figure 9b: Precision Using BestFirst Attribute Selection



	Bayesian Network	Naïve Bayes	Neural Network	Decision Tree	1R	0R
1 std. above	0.89	0.89	0.86	0.81	0.76	0.62
Average	0.8	0.8	0.74	0.69	0.66	0.62
1 std. below	0.71	0.71	0.62	0.57	0.56	0.62

Algorithms

Figure 9c: Top-10 Precision Using BestFirst Attribute Selection



From these results, BestFirst appears to outperform principal components for attribute selection, as it leads to higher overall accuracy and precision. Bayesian learning techniques are best suited for this type of performance, with Bayesian networks slightly outperforming its naïve counterpart. Because BestFirst performs better for accuracy and precision, I used BestFirst as the method of attribute selection for all future tests.

Top-N precision as a function of N (the number of recommendations)

Both attribute selection techniques performed extremely well in top-10 precision, so I decided to vary the N in top-N precision. The results appear in Figures 10a and 10b.

Figure 10a: Top-N Precision as a Function of N with Principal Components Attribute Selection

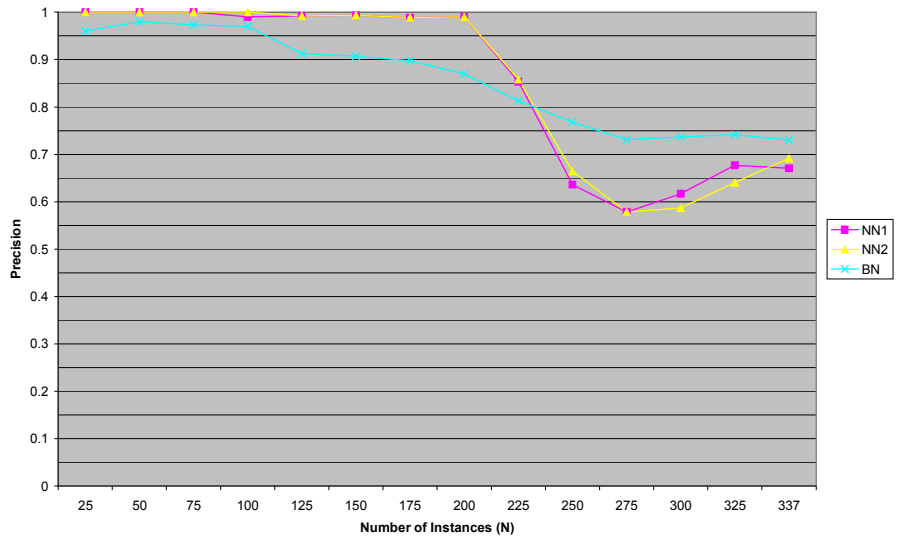
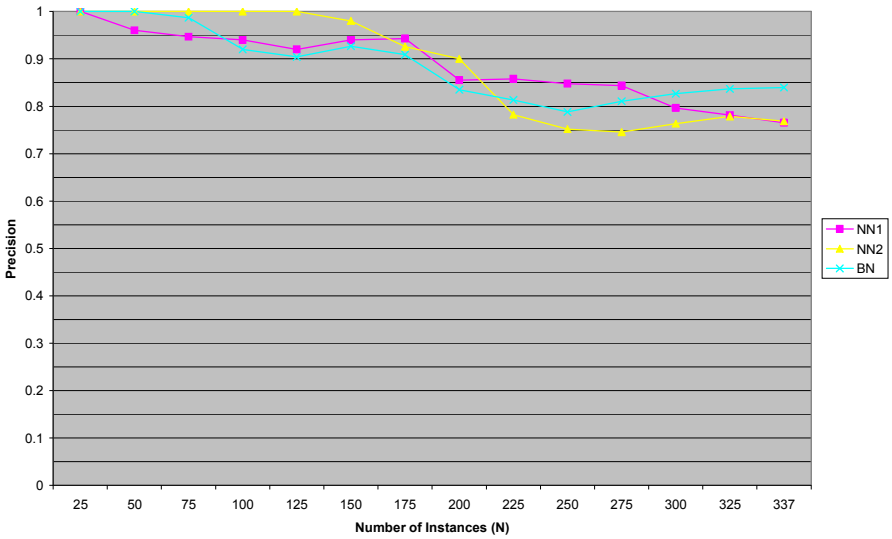


Figure 10b: Top-N Precision as a Function of N with BestFirst Attribute Selection

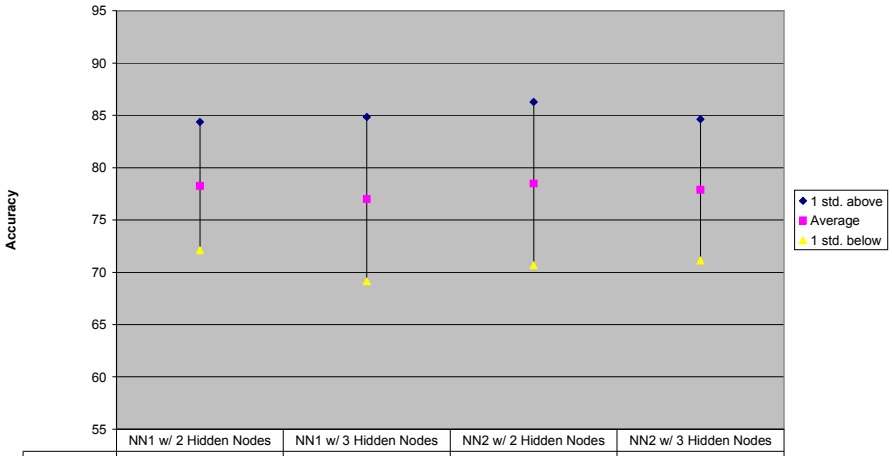


Neural networks under principal components attribute selection worked the best when N was the smallest. However, as N gets larger, Bayesian networks and BestFirst attribute selection perform better in top-N precision. Therefore neural networks and principal components should be used for a smaller N, while Bayesian networks and BestFirst should be used for a larger N. Top-N precision performing well for values of N up to 200 in neural networks using principal components is very significant in being able to set up an effective movie recommender system.

Neural Networks

I tested the neural networks with the standard version of error backpropagation (NN1) against neural networks with the modified version of error backpropagation (NN2) as defined in the Network Training sections. I tested each with having both 2 and 3 hidden nodes. The results are shown in Figures 11a, 11b and 11c.

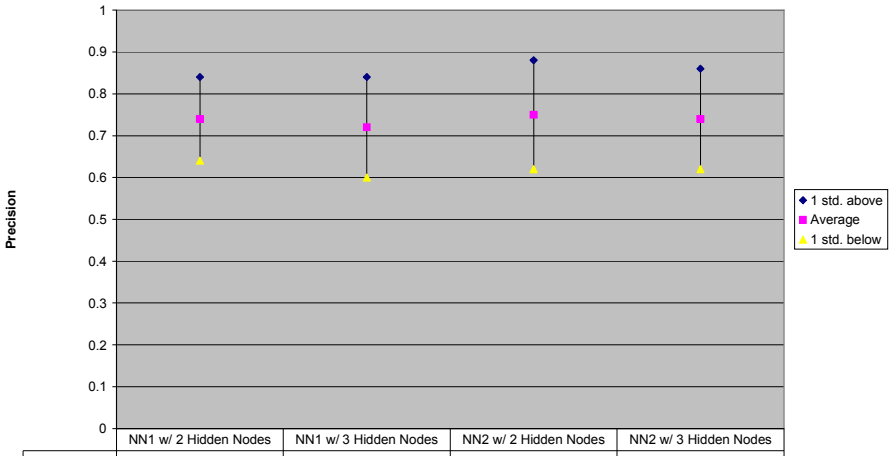
Figure 11a: Modifying Error Backpropagation in Neural Networks



	NN1 w/ 2 Hidden Nodes	NN1 w/ 3 Hidden Nodes	NN2 w/ 2 Hidden Nodes	NN2 w/ 3 Hidden Nodes
1 std. above	84.37	84.84	86.29	84.64
Average	78.24	76.99	78.48	77.88
1 std. below	72.11	69.14	70.67	71.12

Algorithms

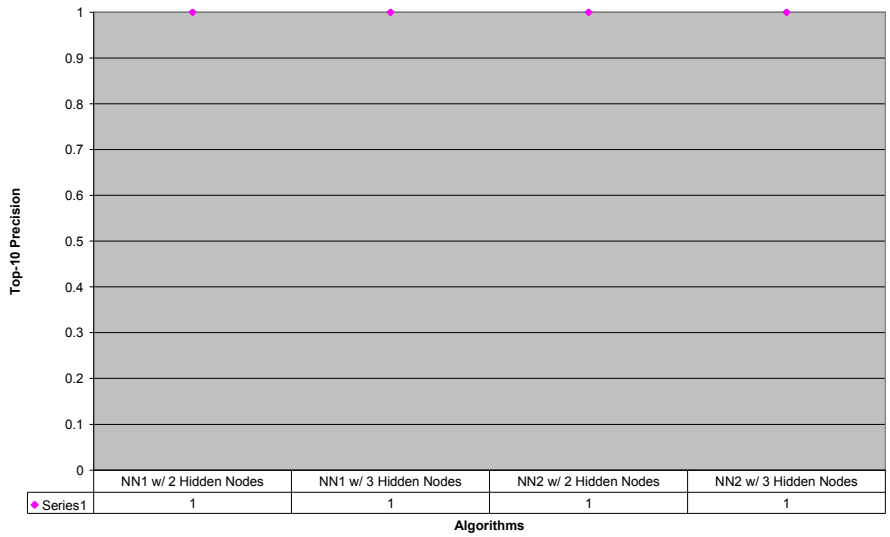
Figure 11b: Modifying Error Backpropagation in Neural Networks



	NN1 w/ 2 Hidden Nodes	NN1 w/ 3 Hidden Nodes	NN2 w/ 2 Hidden Nodes	NN2 w/ 3 Hidden Nodes
1 std. above	0.84	0.84	0.88	0.86
Average	0.74	0.72	0.75	0.74
1 std. below	0.64	0.6	0.62	0.62

Algorithms

Figure 11c: Modifying Error Backpropagation in Neural Networks



All of the tests produced 100% top-10 precision. NN2 with 2 hidden nodes seemed to have a slight advantage in terms of accuracy. The change in precision is minimal as well. Because neural networks take a long amount of time to train, I used the modified version in future testing.

Number of Hidden Nodes

I then decided to test more hidden nodes with NN2. The performance values appear in figures 12a, 12b and 12c.

Figure 12a: Modifying Hidden Nodes in Neural Networks

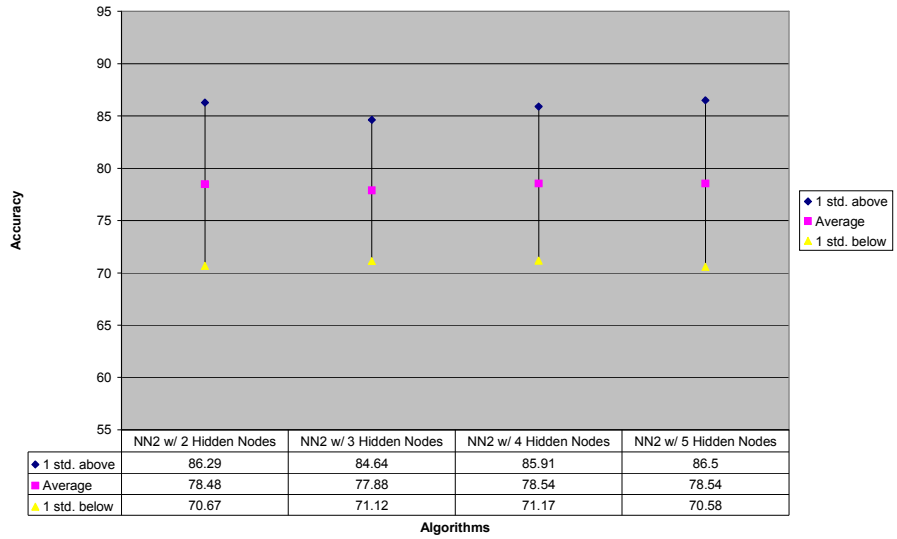
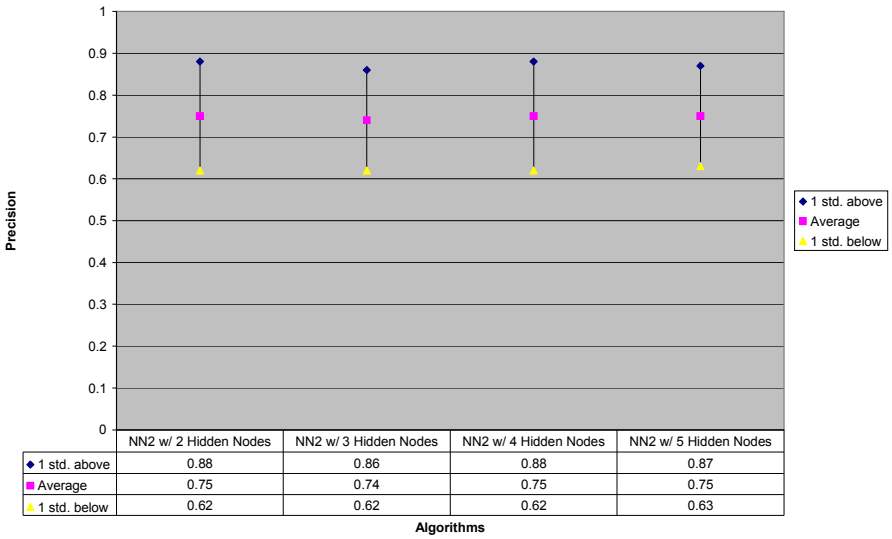
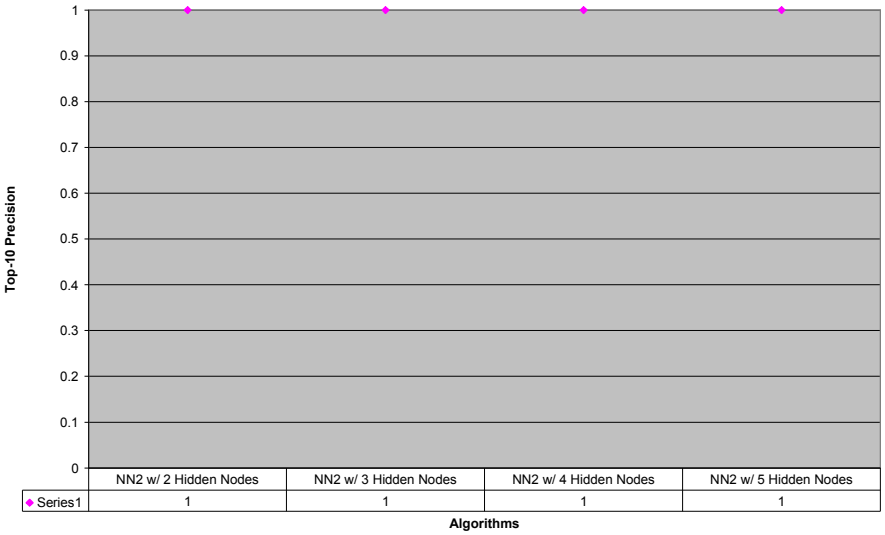


Figure 12b: Modifying Hidden Nodes in Neural Networks



	NN2 w/ 2 Hidden Nodes	NN2 w/ 3 Hidden Nodes	NN2 w/ 4 Hidden Nodes	NN2 w/ 5 Hidden Nodes
1 std. above	0.88	0.86	0.88	0.87
Average	0.75	0.74	0.75	0.75
1 std. below	0.62	0.62	0.62	0.63

Figure 12c: Modifying Hidden Nodes in Neural Networks



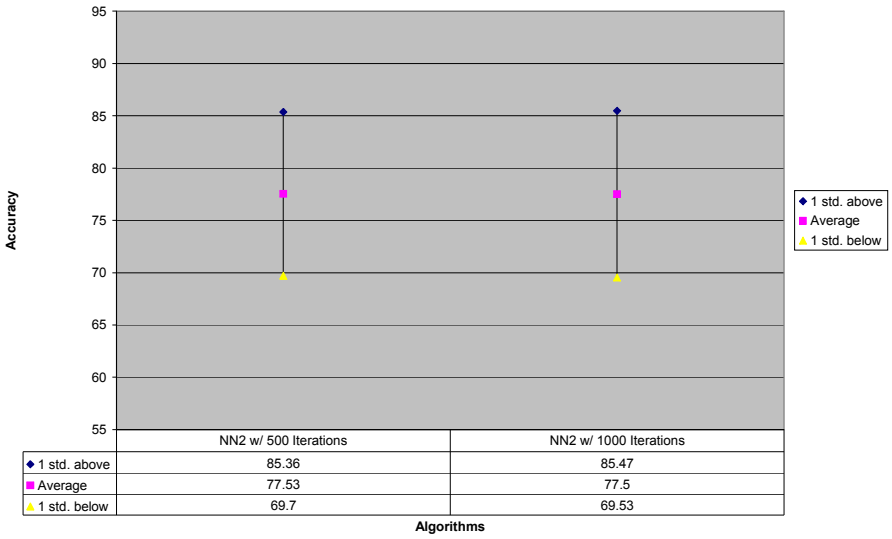
	NN2 w/ 2 Hidden Nodes	NN2 w/ 3 Hidden Nodes	NN2 w/ 4 Hidden Nodes	NN2 w/ 5 Hidden Nodes
Series1	1	1	1	1

All tests produced 100% top-10 precision. The accuracy decreased at 3 hidden nodes and went back up for 4 and 5. The precision changed at the same pace. Therefore changing the amount of hidden nodes had no real effect.

Number of Training Epochs

I then decided to increase the standard training time on the data from 500 iterations to 1000 iterations. The results are shown in Figures 13a, 13b and 13c.

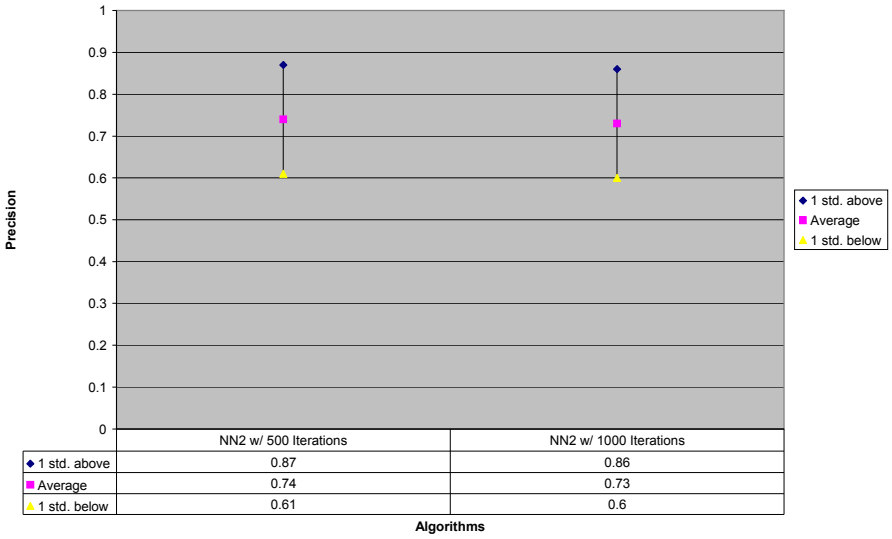
Figure 13a: Modifying Training Time in Neural Networks



	NN2 w/ 500 Iterations	NN2 w/ 1000 Iterations
1 std. above	85.36	85.47
Average	77.53	77.5
1 std. below	69.7	69.53

Algorithms

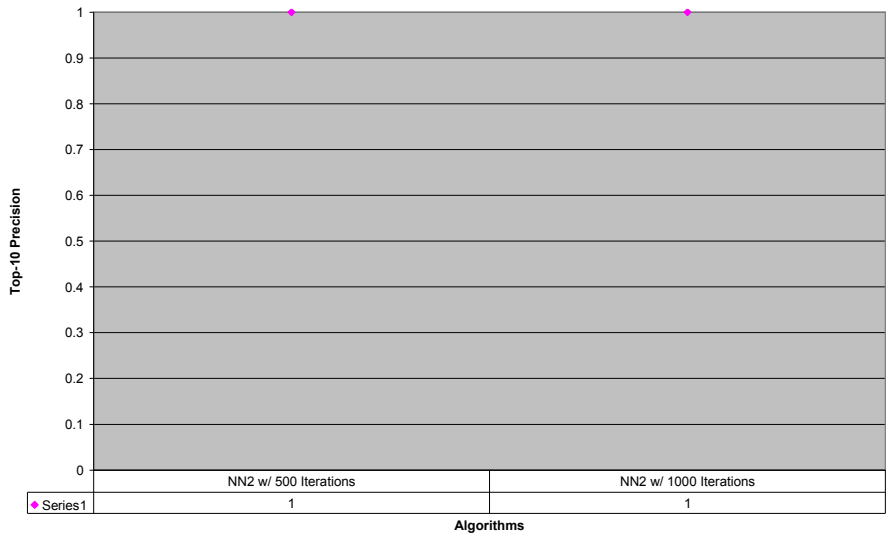
Figure 13b: Modifying Training Time in Neural Networks



	NN2 w/ 500 Iterations	NN2 w/ 1000 Iterations
1 std. above	0.87	0.86
Average	0.74	0.73
1 std. below	0.61	0.6

Algorithms

Figure 13c: Modifying Training Time in Neural Networks

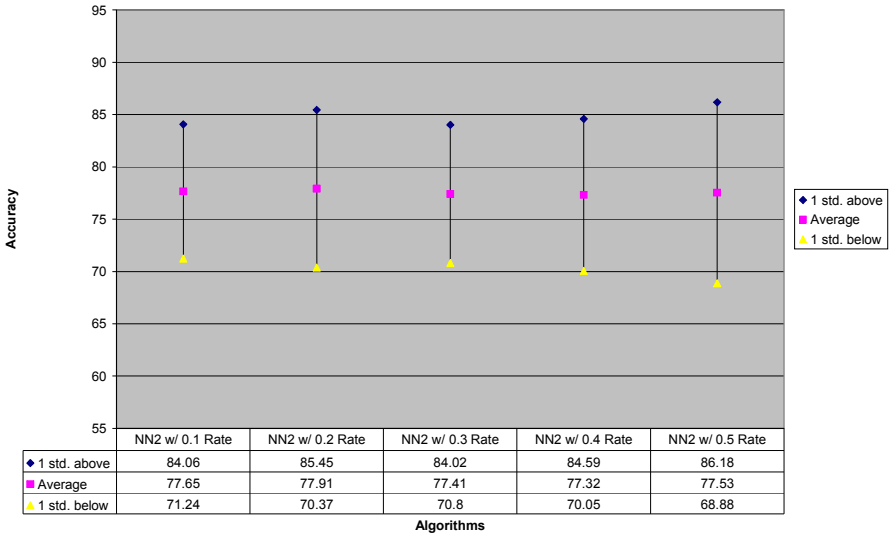


Looking at the results, top-10 precision was the same despite increasing the training time. Accuracy slightly decreased, and the standard deviation slightly increased. Precision also slightly decreased. Therefore, despite increasing the training time, the accuracy and standard deviation did not have a significant change. Therefore, I did not bother increasing the training time for future experiments.

Learning Rate

I also considered modifying the learning rate within neural networks to determine whether convergence to a better minimum could be achieved. The results are displayed in Figures 14a, 14b and 14c.

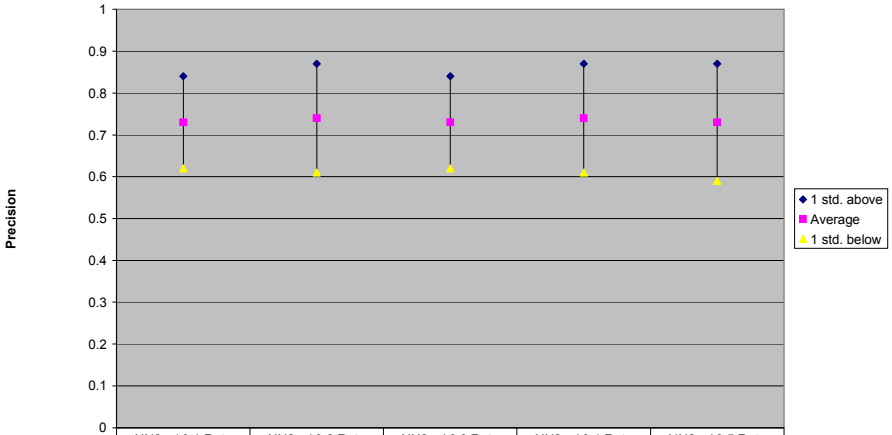
Figure 14a: Modifying the Learning Rate in Neural Networks



	NN2 w/ 0.1 Rate	NN2 w/ 0.2 Rate	NN2 w/ 0.3 Rate	NN2 w/ 0.4 Rate	NN2 w/ 0.5 Rate
1 std. above	84.06	85.45	84.02	84.59	86.18
Average	77.65	77.91	77.41	77.32	77.53
1 std. below	71.24	70.37	70.8	70.05	68.88

Algorithms

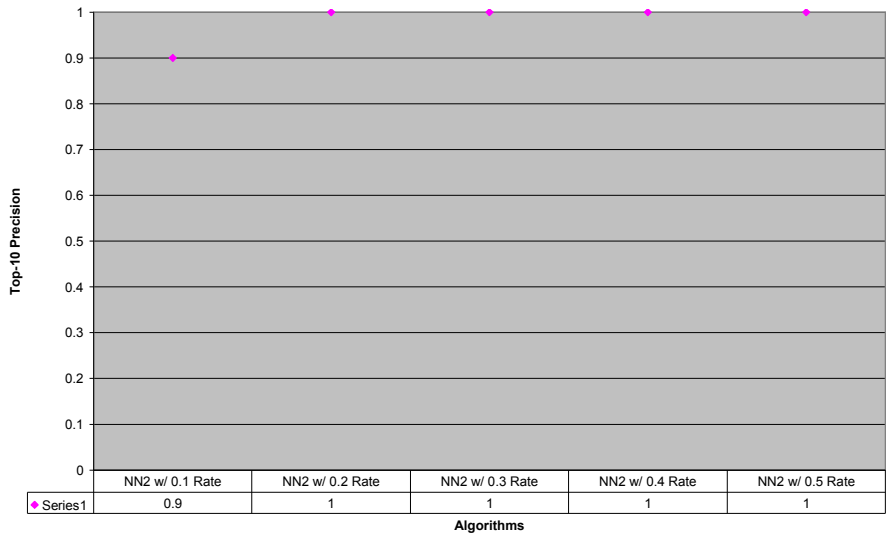
Figure 14b: Modifying the Learning Rate in Neural Networks



	NN2 w/ 0.1 Rate	NN2 w/ 0.2 Rate	NN2 w/ 0.3 Rate	NN2 w/ 0.4 Rate	NN2 w/ 0.5 Rate
1 std. above	0.84	0.87	0.84	0.87	0.87
Average	0.73	0.74	0.73	0.74	0.73
1 std. below	0.62	0.61	0.62	0.61	0.59

Algorithms

Figure 14c: Modifying the Learning Rate in Neural Networks



The results seemed to fluctuate as the learning rate changed. 0.1 has a lower top-N precision but the rest performed at 100%. 0.2 had a slightly higher accuracy but it is not a noticeable change. The precision also does not change significantly when modifying the learning rate.

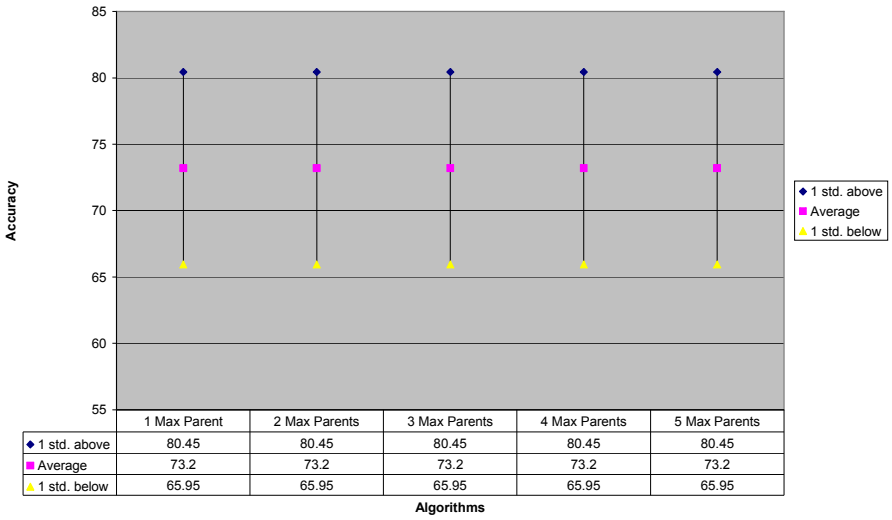
Bayesian Networks

I decided to test the maximum number of parents each node can have on the content-only, collaborative-only, and combined datasets.

Content Data Only

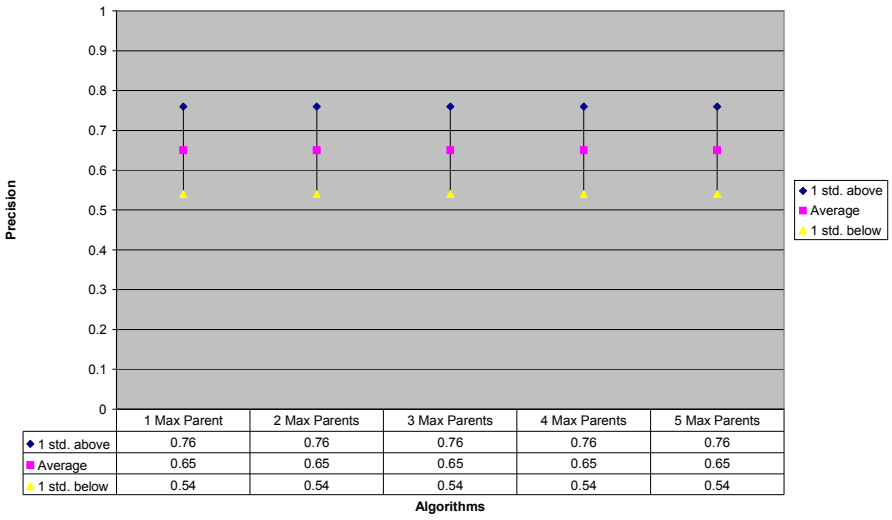
The results for the content-only version are shown in Figures 15a, 15b and 15c.

Figure 15a: Varying the Number of Parents in Bayesian Networks (Content-Only)



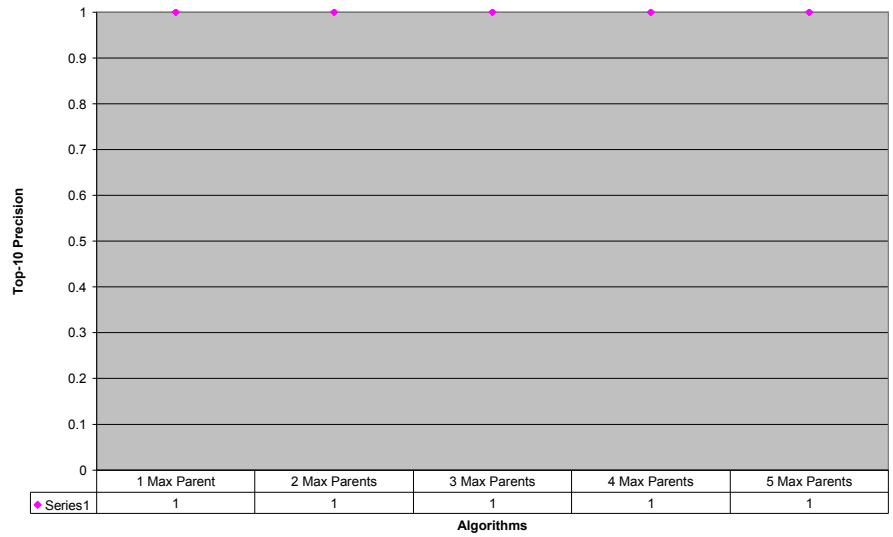
	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	80.45	80.45	80.45	80.45	80.45
Average	73.2	73.2	73.2	73.2	73.2
1 std. below	65.95	65.95	65.95	65.95	65.95

Figure 15b: Varying the Number of Parents in Bayesian Networks (Content-Only)



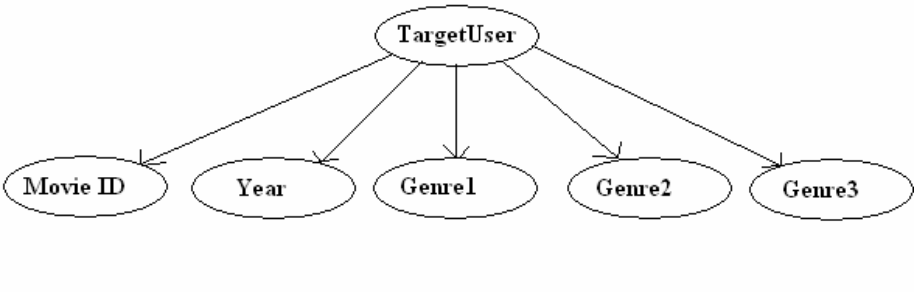
	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	0.76	0.76	0.76	0.76	0.76
Average	0.65	0.65	0.65	0.65	0.65
1 std. below	0.54	0.54	0.54	0.54	0.54

Figure 15c: Varying the Number of Parents in Bayesian Networks (Content-Only)



As shown in the graphs, changing the maximum number of parents does not change the top-N precision, accuracy, precision or the standard deviation. Therefore, it appears it is only using one parent for each node. Going back and checking the data, the network is connecting all of the content attributes to the target attribute as the parent for all of them, similar to Naïve Bayes. Modifying the maximum number of parents does not change, so the result is a network similar to that shown in Figure 15d.

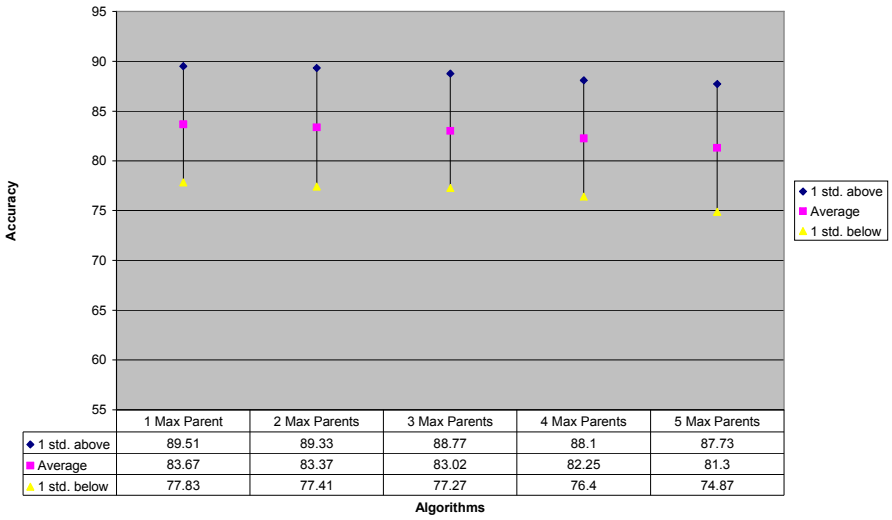
Figure 15d: Bayesian Network (Content-Only)



Collaborative Data Only

Figures 16a, 16b and 16c show the graphs for collaborative-only Bayesian networks.

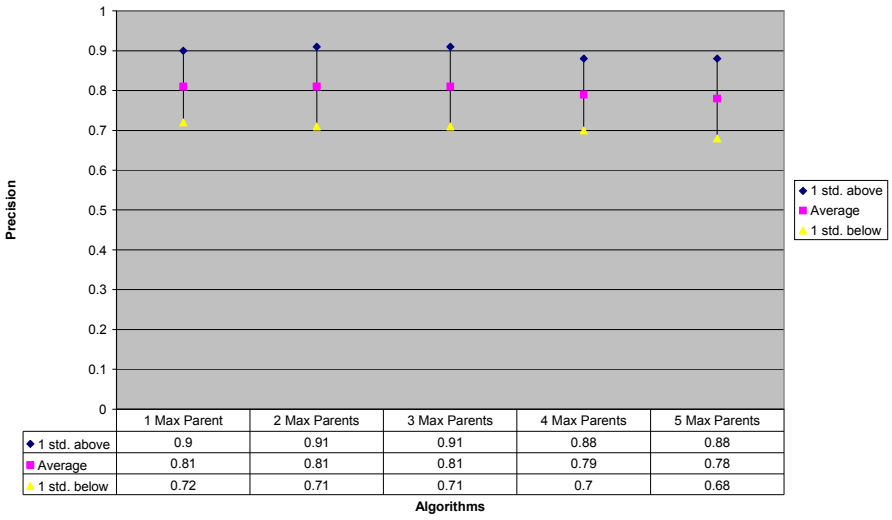
Figure 16a: Varying the Number of Parents in Bayesian Networks (Collaborative-Only)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	89.51	89.33	88.77	88.1	87.73
Average	83.67	83.37	83.02	82.25	81.3
1 std. below	77.83	77.41	77.27	76.4	74.87

Algorithms

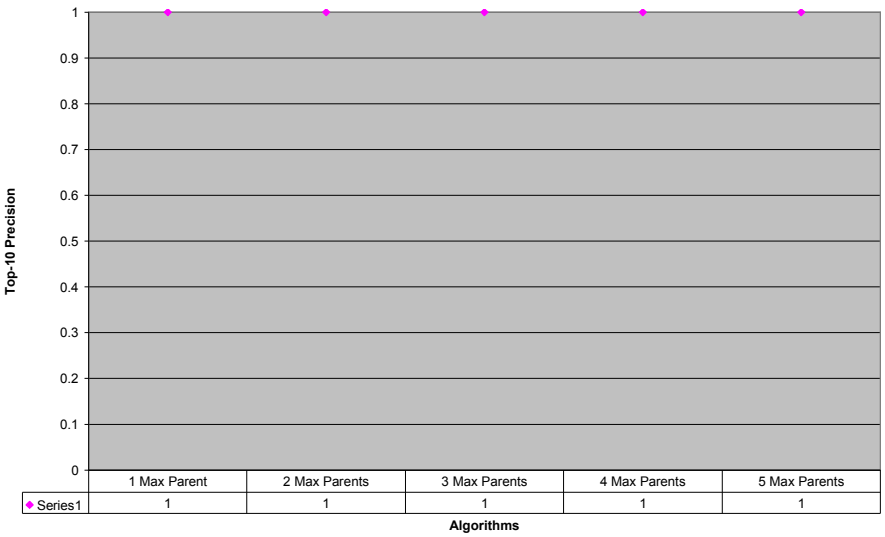
Figure 16b: Varying the Number of Parents in Bayesian Networks (Collaborative-Only)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	0.9	0.91	0.91	0.88	0.88
Average	0.81	0.81	0.81	0.79	0.78
1 std. below	0.72	0.71	0.71	0.7	0.68

Algorithms

Figure 16c: Varying the Number of Parents in Bayesian Networks (Collaborative-Only)



Some of the results for collaborative data are a little different than those obtained over content data. For collaborative data, top-N precision stays constant as the number of parents increases. However, the accuracy and precision steadily decrease. When comparing this against the content-only data, the top-N precision is the same, while the accuracy and precision are significantly higher. The standard deviation does not vary drastically with any maximum number of parents.

Collaborative and Content Data Combined

I then decided to use the regular dataset which includes both content and collaborative data. Figures 17a, 17b and 17c show the results.

Figure 17a: Varying the Number of Parents in Bayesian Networks (Collaborative/Content Combined)

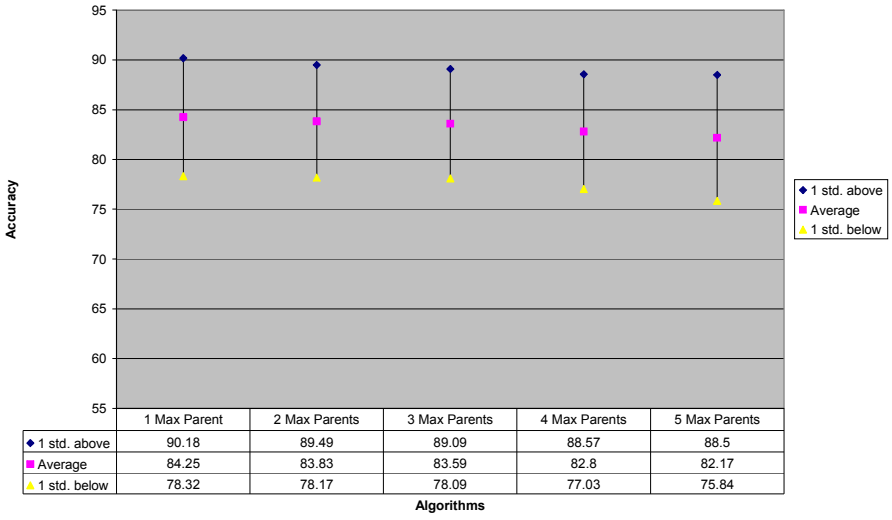
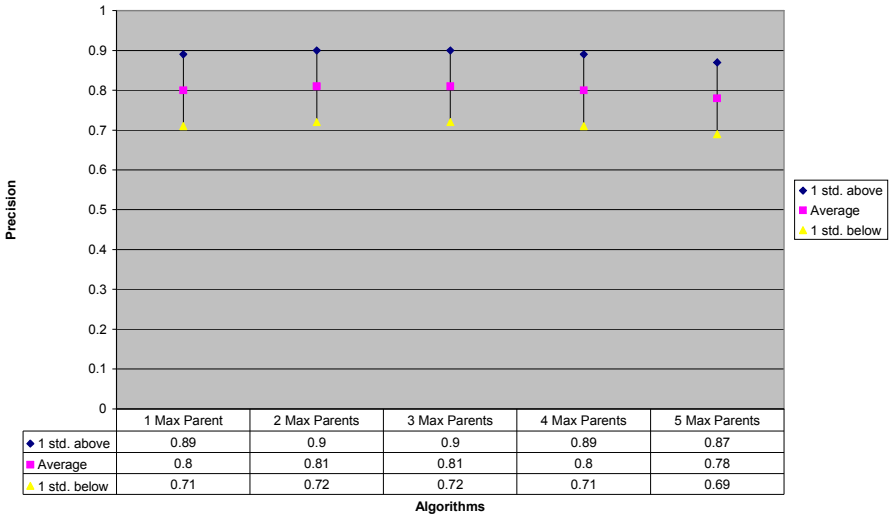
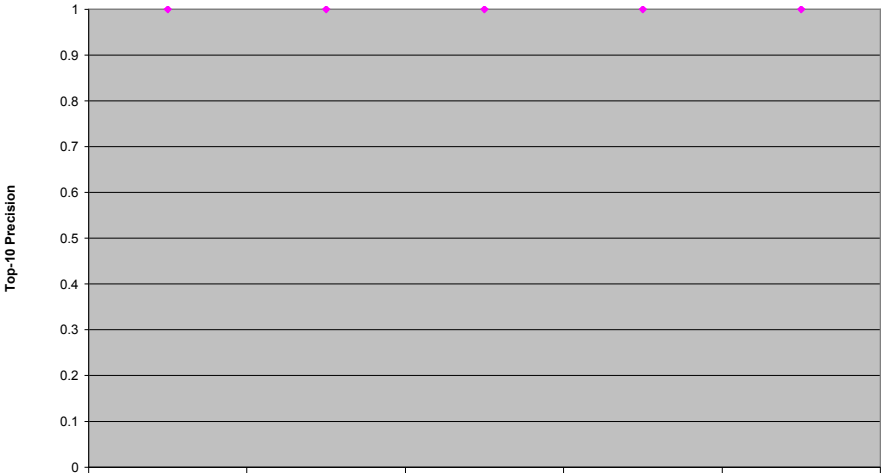


Figure 17b: Varying the Number of Parents in Bayesian Networks (Collaborative/Content Combined)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	0.89	0.9	0.9	0.89	0.87
Average	0.8	0.81	0.81	0.8	0.78
1 std. below	0.71	0.72	0.72	0.71	0.69

Figure 17c: Varying the Number of Parents in Bayesian Networks (Collaborative/Content Combined)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
Series1	1	1	1	1	1

Using a combination of both collaborative and content data, the top-N precision remains the same, while the accuracy and precision decrease again as the maximum number of parents goes up. Comparing with the results obtained over collaborative data alone, the addition of content data does not lead to a significant difference in top-N precision, accuracy or precision.

Enhanced Content Data

Due to the content-only performance being poor in terms of accuracy and precision and not having much effect when added to the collaborative data, I thought that more content data was needed to make a decision. Therefore, I added the title, director, language, and rating of each film into the dataset as well. The breakdowns of language and rating are shown in Figures 18a and 18b.

Figure 18a: Language Distribution

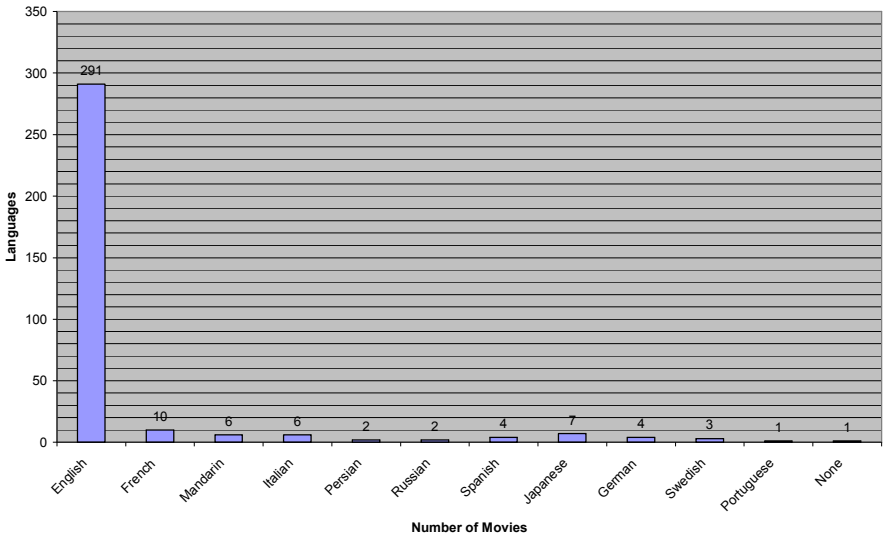
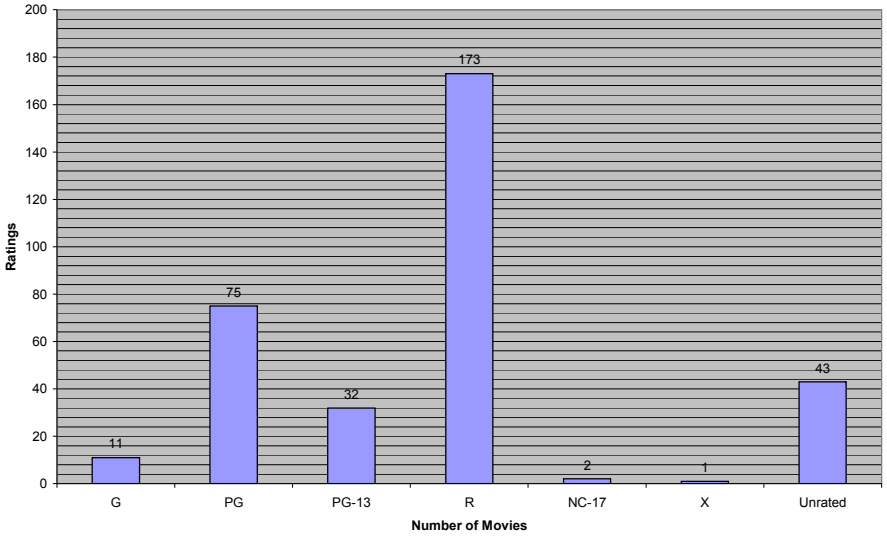
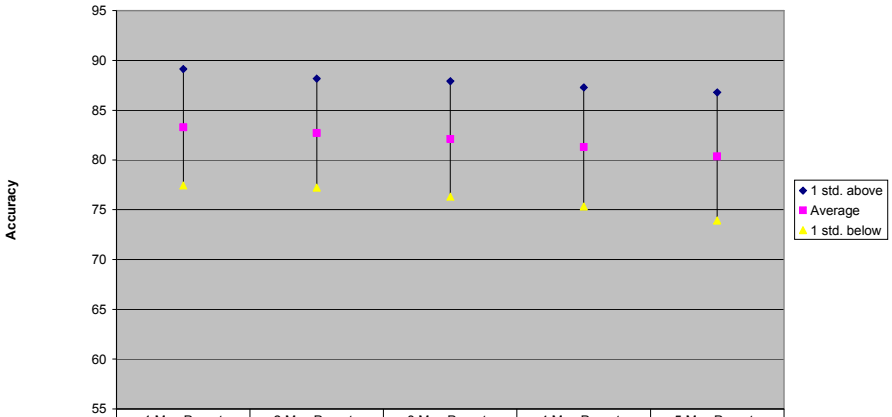


Figure 18b: Rating Distribution



Performance results for the enhanced-content dataset appear in Figures 18c, 18d, 18e.

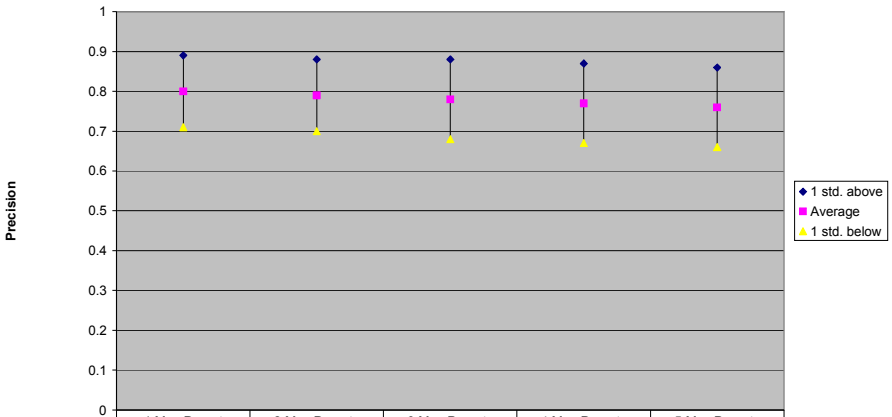
Figure 18c: Varying the Number of Parents in Bayesian Networks (More Content)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	89.15	88.18	87.92	87.3	86.8
Average	83.29	82.7	82.11	81.31	80.36
1 std. below	77.43	77.22	76.3	75.32	73.92

Algorithms

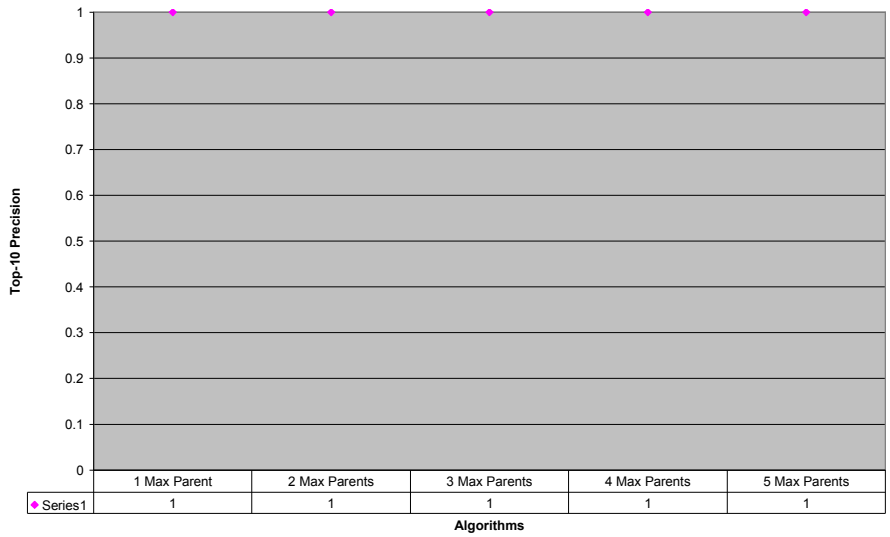
Figure 18d: Varying the Number of Parents in Bayesian Networks (More Content)



	1 Max Parent	2 Max Parents	3 Max Parents	4 Max Parents	5 Max Parents
1 std. above	0.89	0.88	0.88	0.87	0.86
Average	0.8	0.79	0.78	0.77	0.76
1 std. below	0.71	0.7	0.68	0.67	0.66

Algorithms

Figure 18e: Varying the Number of Parents in Bayesian Networks (More Content)



Top-N precision remains at 100%. Accuracy and precision are once again at a steady decline as the maximum number of parents is increased. Comparing against the other graphs, top-N precision, accuracy, standard deviation and precision do not change significantly when the new content is added. Therefore new content data was not helpful in improving performance.

Conclusions and Future Work

The purpose of this thesis was to analyze machine learning techniques and apply them to recommender systems in order to make recommendations for movie patrons. By applying different strategies to both the dataset and the different algorithms, the hope was to find an optimized way of predicting whether users would like certain movies based on content information about the movie and collaborative information from other moviegoers' ratings. From the work done, it can be concluded that certain machine learning algorithms can indeed perform well in making movie recommendations.

In terms of top-N precision, neural networks and principal components perform extremely well for a low value of N (number of movies recommended). This is very important for movie recommendations, as most users would only want to sort through 10 or fewer movies in order to pick one. The fact that top-N precision remains high up to N=200 is also a good source of other sort of recommendations, such as recommendations for people who buy or rent DVDs or watch movies on television. Those people might need a bigger list as they want a list of movies to watch rather than just a select few.

Bayesian networks take much less training time than neural networks while having the same top-10 precision, so it could be more useful for re-training when users add their likes and dislikes into a system. Also, in terms of accuracy, Bayesian networks display a significant improvement over neural networks, with both the standard version and modified version of error propagation, as well as decision trees, Naïve Bayes, 1R and 0R classifiers. Bayesian networks perform well with both principal components and the BestFirst attribute selection, something that none of the other algorithms tested do. It is close with 1R in principal components but this could be due to the size of the dataset. A dataset with move movies for each year could possibly lower 1R's performance.

BestFirst appears to be an effective technique for attribute selection. It reduces the number of attributes considerably, gives 100% top-10 precision and increases the maximum accuracy from the 0R baseline. BestFirst is also faster than principal components in that it selects features rather than applying weights to all of them, so it is better in terms of accuracy, precision and time, but not in overall top-N precision.

There does not appear to be a big change in top-10 precision, accuracy or precision when comparing neural networks with the standard version of error backpropagation and the modified version. More testing would need to be done over longer amounts of time to see if one is significantly better than the other.

More testing could be done on the parameter settings of neural networks in order to determine what parameter values optimize accuracy, precision and top-10 precision. It appears that adjusting the training time, learning rate and other variables does not significantly change the accuracy and the precision of the recommendations.

The results of the optimal algorithms exceed the benchmarks set by 0R significantly. The best performing algorithms have 100% top-10 precision, which is a very important statistic. They also increased the accuracy and precision from 62% to 83% and 62% to 80%, respectively.

It appears that using content-only, collaborative-only, or a combined dataset does not change the top-10 precision, as all three dataset versions gave 100% top-10 precision. 100% top-10 precision in content-only data is especially good for a new movie for which no collaborative data is available. Collaborative data only proved more useful in

increasing overall accuracy. However, it would be interesting to see what would happen if even more content data was added, such as actors, as some moviegoers have favorite actors and are more likely to see films by those actors.

Finally, Bayesian and neural network techniques should be tested on other datasets that contain more data. The dataset chosen was rather small due to the time complexities of a larger dataset. It should be determined whether the conclusion taken from this set of experiments applies to larger datasets. Additional target users should be considered as well.

References

Alvarez, S. A., Ruiz, C., Kawato, T., & Kogel, W. (2006). Neural Expert Networks for Faster Combined Collaborative and Content-Based Recommendation. In *Journal of Computational Methods in Sciences and Engineering,* to appear.

Balabanovic, M., & Shoham, Y. (1997). Combining content-Based and collaborative recommendation. In *Communications of the ACM*, 40(3), pp. 66-72.

Billsus, D., & Pazzani, M.J. (1998). Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 46-54.

Bishop, C. M. (1996). *Neural networks for pattern recognition*. Oxford, England: Oxford University Press.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pp. 43–52.

Cooper, G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. In *Machine Learning*, 9, pp. 309-347.

Harper, F., Li, X., Chen,Y. & Konstan, J. (2005). An Economic Model of User Rating in an Online Recommender System. In *Proceedings of the 10th International Conference on User Modeling*.

Joachims, T. (1996). *A probabilistic analysis of the Rocchio algorithm with TFIDF for text Categorization*, (Computer Science Technical Report CMU-CS-96-118). Carnegie Mellon University.

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIE Press.

Mitchell, T. M. *Machine Learning*. (1997). Boston, MA: The McGraw-Hill Companies, Inc.

Quinlan, J. R. (1986). Induction of decision trees. In *Machine Learning*, 1(1), pp. 81-106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Resnick, P., & Varian, H. R., Eds. (1997). CACM Special Issue on Recommender Systems. In *Communications of the ACM*, 40(3), pp. 56-89.

Rumelhart, D. E., Widrow, B., & Lehr, M. A. (1994). Neural Networks: Applications in Industry, Business and Science. In *Communications of the ACM*, 37(3), pp. 93-105.

Shardanand, U., & Maes, P. (1995). Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of ACM Computer Human Interaction Conference on Human Factors in Computing Systems*, pp. 210–217.

Popescul, A. & Ungar, L. H. (2001). Probabilistic Models for Unified Collaborative and Content-Based Recommendation in Sparse-Data Environments. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pp. 437–444.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition. San Francisco, CA: Morgan Kaufmann.

Appendix

Top-N Precision w/ NN1 Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.Evaluation;


public class TopNPrecision {

	public static void main(String[] args){

		try{
			Instances data = new Instances(
					new BufferedReader(
						new FileReader("combBF.arff")));
			//setting class attribute
			data.setClassIndex(data.numAttributes() - 1);


			// load unlabeled data

			MultilayerPerceptron mlp = new MultilayerPerceptron();
			mlp.setHiddenLayers("3");		// set the hidden nodes
			mlp.setLearningRate(0.2);  // set the learning rate
			mlp.setTrainingTime(500);		// set the training time
			mlp.buildClassifier(data);   // build classifier

			Instances unlabeled = new Instances(
									new BufferedReader(
									  new
FileReader("combBF.arff")));
			// set class attribute
			unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
			// create copy
			Instances labeled = new Instances(unlabeled);
			// label instances
			Instance[] instances = new Instance[10];
			double[] values = new double[10];

			for(int i = 0; i < 10; i++){
				instances[i] = null;
				values[i] = 0;
			}

			//rank instances - only keep top N
			for (int i = 0; i < unlabeled.numInstances(); i++) {
			  double []clsLabel = mlp.distributionForInstance(unlabeled.instance(i));
```

```
                        if(values[0] <= clsLabel[1]) { //like probability
                                for(int j = 1; j < 10; j++){
                                        if(values[j] <= clsLabel[1] && j != 9){
                                                values[j-1] = values[j];
                                                instances[j-1] = instances[j];
                                        }
                                        else if (j == 9){
                                                values[j-1] = values[j];
                                                instances[j-1] = instances[j];
                                                values[j] = clsLabel[1];
                                                instances[j] = unlabeled.instance(i);
                                        }
                                        else{
                                                values[j-1] = clsLabel[1];
                                                instances[j-1] = unlabeled.instance(i);
                                                break;
                                        }
                                }
                         }
                        }

                        labeled.delete();
                        for(int i = 0; i < 10; i++)
                                if(instances[i] != null)
                                        labeled.add(instances[i]);

                        //evaluate the instances
                        Evaluation eval = new Evaluation(labeled);
                        eval.crossValidateModel(
                             mlp, labeled, 10, labeled.getRandomNumberGenerator(1));
                        System.out.println(eval.toSummaryString(true));

                        //write instances (to double check no doubles)
                        BufferedWriter writer = new BufferedWriter(
                                        new FileWriter("nn1.arff"));
                        writer.write(labeled.toString());
                        writer.newLine();
                        writer.flush();
                        writer.close();

                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

Top-N Precision w/ NN2 Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.functions.MultilayerPerceptronNew;
import weka.classifiers.Evaluation;


public class TopNPrecision2 {

	public static void main(String[] args){

		try{
			Instances data = new Instances(
					new BufferedReader(
						new FileReader("combBF.arff")));
			//setting class attribute
			data.setClassIndex(data.numAttributes() - 1);


			// load unlabeled data

			MultilayerPerceptronNew mlp = new MultilayerPerceptronNew();
			mlp.setHiddenLayers("3");		// set the hidden nodes
			mlp.setLearningRate(0.2);  // set the learning rate
			mlp.setTrainingTime(500);		// set the training time
			mlp.buildClassifier(data);   // build classifier

			Instances unlabeled = new Instances(
									new BufferedReader(
									  new
FileReader("combBF.arff")));
			// set class attribute
			unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
			// create copy
			Instances labeled = new Instances(unlabeled);
			// label instances
			Instance[] instances = new Instance[10];
			double[] values = new double[10];

			for(int i = 0; i < 10; i++){
				instances[i] = null;
				values[i] = 0;
			}

			//rank instances - only keep top N
			for (int i = 0; i < unlabeled.numInstances(); i++) {
			  double []clsLabel = mlp.distributionForInstance(unlabeled.instance(i));
			  if(values[0] <= clsLabel[1]) { //like probability
				for(int j = 1; j < 10; j++){
					if(values[j] <= clsLabel[1] && j != 9){
```

```
                                        values[j-1] = values[j];
                                        instances[j-1] = instances[j];
                                }
                                else if (j == 9){
                                        values[j-1] = values[j];
                                        instances[j-1] = instances[j];
                                        values[j] = clsLabel[1];
                                        instances[j] = unlabeled.instance(i);
                                }
                                else{
                                        values[j-1] = clsLabel[1];
                                        instances[j-1] = unlabeled.instance(i);
                                        break;
                                }
                        }
                 }
                }

                labeled.delete();
                for(int i = 0; i < 10; i++)
                        labeled.add(instances[i]);

                //evaluate the instances
                Evaluation eval = new Evaluation(labeled);
                eval.crossValidateModel(
                     mlp, labeled, 10, labeled.getRandomNumberGenerator(1));
                System.out.println(eval.toSummaryString(true));

                //write instances (to double check no doubles)
                BufferedWriter writer = new BufferedWriter(
                                new FileWriter("nn2.arff"));
                writer.write(labeled.toString());
                writer.newLine();
                writer.flush();
                writer.close();


        }catch(Exception e){
                e.printStackTrace();
        }
    }
}
```

Top-N Precision w/ BN Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.bayes.net.search.global.*;
import weka.classifiers.Evaluation;


public class TopNPrecision3 {

	public static void main(String[] args){

		try{
			Instances data = new Instances(
					new BufferedReader(
						new FileReader("combBF.arff")));
			//setting class attribute
			data.setClassIndex(data.numAttributes() - 1);


			// load unlabeled data

			BayesNet mlp = new BayesNet();
			K2 k2 = new K2();
			k2.setMaxNrOfParents(1);// set max # of parents
			mlp.setSearchAlgorithm(k2); // set search algorithm

			mlp.buildClassifier(data);   // build classifier

			Instances unlabeled = new Instances(
									new BufferedReader(
									  new
FileReader("combBF.arff")));
			// set class attribute
			unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
			// create copy
			Instances labeled = new Instances(unlabeled);
			// label instances
			Instance[] instances = new Instance[10];
			double[] values = new double[10];

			for(int i = 0; i < 10; i++){
				instances[i] = null;
				values[i] = 0;
			}

			//rank instances - only keep top N
			for (int i = 0; i < unlabeled.numInstances(); i++) {
			  double []clsLabel = mlp.distributionForInstance(unlabeled.instance(i));
```

```
                        if(values[0] <= clsLabel[1]) { //like probability
                                for(int j = 1; j < 10; j++){
                                        if(values[j] <= clsLabel[1] && j != 9){
                                                values[j-1] = values[j];
                                                instances[j-1] = instances[j];
                                        }
                                        else if (j == 9){
                                                values[j-1] = values[j];
                                                instances[j-1] = instances[j];
                                                values[j] = clsLabel[1];
                                                instances[j] = unlabeled.instance(i);
                                        }
                                        else{
                                                values[j-1] = clsLabel[1];
                                                instances[j-1] = unlabeled.instance(i);
                                                break;
                                        }
                                }
                         }
                        }

                        labeled.delete();
                        for(int i = 0; i < 10; i++)
                                labeled.add(instances[i]);

                        //evaluate the instances
                        Evaluation eval = new Evaluation(labeled);
                        eval.crossValidateModel(
                             mlp, labeled, 10, labeled.getRandomNumberGenerator(1));
                        System.out.println(eval.toSummaryString(true));

                        //write instances (to double check no doubles)
                        BufferedWriter writer = new BufferedWriter(
                                        new FileWriter("bn.arff"));
                        writer.write(labeled.toString());
                        writer.newLine();
                        writer.flush();
                        writer.close();

                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```