Comparing Different Learning Algorithms for Movie Recommendations

CS289 Graduate Project - Fall 2017

Léo Jules Laugier
Master of Engineering,
University of California,
Berkeley
leo_laugier@berkeley.edu

Daniel Avery Nisbet
Master of Engineering,
University of California,
Berkeley
danisbet@berkeley.edu

Evan Thompson
Master of Engineering,
University of California,
Berkeley
tthompe5@berkeley.edu

Alexandros Elio Vlissidis
Master of Engineering,
University of California,
Berkeley
alex_vlissidis@berkeley.ed

***Abstract* — Collaborative filtering is the method of making predictions of a user's preferences based on passively collected data. Data used to train machine learning models to perform collaborative filtering are inherently sparse and models must be adapted to handle the data sparsity. In this study, we implement four models to perform collaborative filtering to predict the movies a user would like to watch. The predictions were made using user's prediction of other movies as the only features. We analyze the models by the utility of its output for a recommendation system, the quality of the predictions, and the tractability of both training and prediction. Our results reinforce that the industry standard, neighborhood based collaborative filtering, is the best model to perform movie predictions**

Keywords—collaborative filtering, neural networks, support vector machines, ridge regression, recommendation system

I. Introduction

Recommendation systems are a special application of information filtering that allows the presentation of items that users may find interesting to consume. Recommendation systems enable one to predict users' preferences thanks to either items themselves (content-based approach) or the social environment (collaborative filtering). In this project we focus on collaborative filtering i.e. we aim to predict users ratings or preferences for movies based on their own known ratings and the known ratings of the other users. We build four approaches in order to compare results based on the MovieLens Dataset.

II. The MovieLens Dataset

The MovieLens dataset [1] describes a 5-star rating and free-text tagging activity from [MovieLens](http://movielens.org), a movie recommendation service. It contains 100004 ratings and 1296 tag applications across 9125 movies. These data were created by 671 users between January 09, 1995 and October 16, 2016. This dataset was generated on October 17, 2016.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

This and other GroupLens data sets are publicly available for download at http://grouplens.org/datasets/.

III. Data Pre-processing

The movie data are stored in comma separated values (CSV) files. The data are processed to form a $u \times m$ data matrix populated by ratings, where u is the number of users, and m is the number of movies. If a particular user, movie pair does not have a rating, the value at that point in the data matrix is left as 0. Mathematically,

$$A \in \mathbb{R}^{m,n} = \begin{cases} r \text{ if rating exists} \\ 0 \text{ otherwise} \end{cases}$$

Afterwards, the data needed to be separated into training and validation error. A k-cross validations algorithm was written such that one out of every k values from A were removed and added to a validation matrix. For every training model, k=10 was used, such that the validation matrix had 1/10th the values of the original data matrix A. Mathematically, where T is the training matrix, and V is the validation matrix,

$$T \in \mathbb{R}^{m,n} = \begin{cases} r \text{ if sample number mod } 10 \neq 0 \\ 0 \text{ otherwise} \end{cases}$$

$$V \in \mathbb{R}^{m,n} = \begin{cases} r \text{ if sample number mod } 10 = 0 \\ 0 \text{ otherwise} \end{cases}$$

This separation of training and validation data left 90,003 samples for training and 10,001 samples for validation

IV. Collaborative Filtering

Collaborative Filtering represents the set of methods that leads to a recommendation system based on group of people's ratings to predict an individual's future tastes. A naive approach to make recommendations would be to assign an average score of interest on each movie, based on the number of votes it received, and suggest the most "interesting" ones to all users. Collaborative filtering aims to improve upon this method by individualizing user interests. Suggestions are based on ratings from other users that have similar opinions to the user. This assumes that people that have similar opinions on some moves, are more likely to have similar opinions for an unwatched movie, than a random person.

In this report, we explore various learning algorithms, presented below, and assess which perform better, drawing some interesting conclusions about their general operation.

A. Neighborhood Based Collaborative Filtering

In neighborhood based CF, we try to predict the ratings of one user at a time; this is the active user. In order to do this, the similarity of the active user to all the users in the dataset is computed. Based on that we can perform a linear combination of their ratings to make a prediction about the ratings of the active user for movies they have not seen.

The whole idea of a neighborhood in this context is to identify the k most similar users to the active user. Only these users will have an effect on our prediction. Thus, the choice of hyperparameter k is important. The general steps of neighborhood based CF are the following:

1) Assign a weight to all users representing their similarity with the user of interest.
2) Select k users that have the highest similarity with the user of interest, i.e. the *neighborhood.*
3) Compute a prediction from a weighted combination of the selected neighbors' ratings.

It has been shown [2], that one the most effective and most applicable to 5-star rating systems, similarity measure is the Pearson Correlation coefficient. This is used to assign a weight that measures the similarity of the active user to all the other users in the dataset, resulting to a weight vector $w \in \mathbb{R}^{n \times m}$. The measure of similarity between the user u and the user of interest a is given by:

$$\rho(a,u) = \frac{\sum_{i \in I}(r_{a,i} - \bar{r}_a) \cdot (r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I}(r_{a,i} - \bar{r}_a)^2 \cdot \sum_{i \in I}(r_{u,i} - \bar{r}_u)^2}}$$

Where I is the set of items rated by both users, $r_{u,i}$ is user u's rating for movie i, $r_{a,i}$ is the active user's rating for movie i, $\bar{r_a}$ is the mean rating of the active user and $\bar{r}_u$ is the mean rating by user u.

In step 3, predictions are computed as the weighted average of deviations from the neighbor's mean, given by:

$$\rho_{a,i} = \bar{r}_a + \frac{\sum_{u \in K}(r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

Where K is the neighborhood or set of most similar users and $w_{a,u}$ is the correlation coefficient of user u with the active user a.

B. *Significance Weighting*

The method mentioned above do not address the issue of having a high user similarity with few samples. Namely, the user of interest can have multiple very high similarity users based on only 1 rating. This would distort our suggestions. In order to avoid this, we can use a significance weighting factor, which reduces the value of correlated users with few samples. In order to implement this, a threshold s is set, on the cardinality of I. Namely, if the number of movies that the active user and user u, have both rated is below h, then the weight $w_{a,u}$ is scaled by $|I|/s$. Mathematically:

$$c = \begin{cases} 1 \text{ if } |I| > s \\ |I|/s \text{ elsewhere} \end{cases}$$

V. Ridge regression model for Collaborative Filtering

A. Collaborative Filtering for Implicit Feedback Datasets

After the implementation of Neighborhood Collaborative Filtering, we implement a different approach which relies on the Ridge Regression version of Collaborative Filtering. This algorithm was inspired from the work performed by Hu *et al.* [3]. Their output was K recommendation TV programs predicted from the input matrix R where for a user u and a TV show i, $r_{u,i}$ was the time the user u spent to watch i. Their fundamental assumption is the longer a TV show is watched, the more prefered it is.

We adapt this algorithm to a movie recommendation system where we predict K movies that each user u is likely to prefer.

B. *Alternating weighted ridge regression*[1]

Let us formalize the iterative weighted ridge regression described by Hu *et al.* [3].
Define the preference of the user u to the movie m as $p_{u,m} = 1$ *if* $r_{u,m} > 0$ *and* $p_{u,m} = 0$ *if* $r_{u,m} = 0$ where $r_{u,m}$ is the rating of the user u to the movie m.
Define the "confidence" of the user u to the movie m as $c_{u,m} = 1 + \alpha * r_{u,m}$ where $r_{u,m}$ is the rating of the user u to the movie m and α is a hyperparameter set to 40 [3].
For n movies and k users,
let X be the kxf user matrix where $\forall\ u \in [1;k]$, *the* u^{th} *row* x_u *of* X represents the uth user in a f-dimensional space. The hyperparameter f is called the factor dimension. f = 200 is chosen by Hu *et al.* [3]. X represents the "user-factors";
let Y be the fxn movie matrix where $\forall\ m \in [1;n]$, *the* m^{th} *column* y_m *of* Y represents the mth movie in a f-dimensional space. Y represents the "item-factors".
We first initialize X and Y to matrices full of ones but random initialization produces better results.

Training

To produce the user-factors X and the item-factors Y, we would solve the following weighted ridge regression:

$$min_{x,y} \sum_{u,m} c_{u,m} * (p_{u,m} - x_u y_m)^2 + \lambda \left[\sum_u \|x_u\|^2 + \sum_m \|y_m\|^2 \right]$$

The value of λ is a hyperparameter set to ~100 in the work done by Yifan Hu *et al.* [3].

Actually, they showed that the Ridge Regression is equivalent to an efficient iterative least square regression.

We define:
$\forall\ u \in [1;k]$, $C^u = diag((c_{u,m})_{m \in [1;n]})$ and
$p(u) = [\,(p_{u,m})_{m \in [1;n]}\,]$

[1] This paragraph is a reference to the Graduate Project Proposal: Personalized Movie Recommendations (3.2.2 Ridge Regression)

$\forall\ m \in [1;n], C^m = diag((c_{u,m})_{u \in [1;k]})$ and
$p(m) = [\ (p_{u,m})_{u \in [1;k]}\ ]$

Our algorithm iterates over the two following steps until the stabilization of X and Y:
Step 1:
$\forall\ u \in [1;k], x_u = (Y\ C^u Y^T + \lambda I)^{-1} Y\ C^u p(u)$

Step 2:
$\forall\ m \in [1;n], y_m = (X^T C^m X + \lambda I)^{-1} X^T C^m p(m)$

Prediction

For a test user u_t, we compute the predicted preference of the user u for the movie m:
$\widehat{p_{u_t,m}} = x_{u_t} y_m$. The idea is to recommend the K movies with the highest user's preferences.

C. Interesting interpretation to build the preference matrix

We identify a relevant proposition for movie ratings from Hu *et al.* [3]. Back to the building of the preference matrix p, we interpret a rated movie as a movie the user is interested in compared to the set of all movies (if you see and rate a movie, you have to feel a minimum of interest in it). One can discuss this assumption by setting a threshold on $r_{u,i}$, for instance $r_{u,m} \geq 3 \implies \mathrm{p}_{u,m} = 1$ and $r_{u,m} < 3 \implies \mathrm{p}_{u,m} = 0$. From Figure 5, we can conclude that actually a 0 threshold gives the best results.

D. Evaluation method

The metric used to evaluate the model is the "expected percentile ranking". $\forall\ u \in [1;k]$, for K=n movies ordered by preference, define $\forall\ m \in [1;n]$, the "percentile-ranking" of movie m in the user u predicted preferences $rank_{u,m} = \dfrac{ordered\, position\ of\ m}{n}$.
The expected percentile ranking is defined by $\dfrac{\sum_{u,m} r_{u,m}^{test} rank_{u,m}}{\sum_{u,m} r_{u,m}^{test}}$.

The lower the expected percentile ranking, the better we predict preferences since the movies at the top of the recommendations are multiplied by high ratings. Finally, an expected percentile ranking higher than 50% does not produce a better prediction than random recommendation.

To conclude, we can adapt ridge regression model for Collaborative Filtering to movie rating in order to produce a recommender system. With regard to results in figure 2,3,4 and 5, this recommender system is interesting for users and further investigations using GPU designed models can be worth it.

VI. Neural Network

Neural Networks are explored as a method to predict what a user would rate a movie. Neural networks also need to overcome the disadvantages of working with sparse input data. For collaborative filtering with neural networks, two objective functions are commonly used: pointwise loss and pairwise lost [4]. Given a potential movie scoring by a user in the user a_{ij} in our feature matrix A, pointwise learning aims to minimize the squared loss between the estimated $\widehat{a_{ij}}$ and the true value a_{ij}, which is not always known. Pairwise learning maximizes the margin between a known, observed aij and unknown a_{ij} values.

A. Architecture of the Network

The neural network is implemented using Keras, a high level API for TensorFlow. The architecture of the network is inspired by the research performed by He *et al.* to implement neural nets in the sparse collaborative filtering problem [4]. Our network is a three layer network where the number of neurons per layer decreased geometrically. The input layer is a one-hot encoded input of the user and movie for a single rating. The output is a one-hot encoded output of the score, such that each rating level, incremented by .5, would correspond to an output neuron. Since there were 9066 movies and 671 users, the input layer has 9737 neurons. Since there are 10 possible ratings, the output has 10 neurons.

A ReLu is used as the activation function for every system since this was found to have higher performance than both the sigmoid and tanh activations by He *et al.* [4]. The adaptive moment activation method (ADAM) is used since the network converges faster using ADAM than stochastic gradient descent (SGD) [4]. Furthermore, the learning rate does not need to be tuned as much using ADAM [4]. The most significant difference between our neural network and the one implemented by He *et al.* is that our network predicts the rating of an item, rather than simply whether or not the item should be recommended. For this task, we both one hot encode the output and use the categorical cross entropy loss function.

The optimal position of several hyperparameters were not discussed by He *et al.* This includes the batch size, the decay rate of the ADAM learning and the initial learning rate. We set the batch size to 128, given that 90,000 training samples were available. The decay rate is set to .001. The initial training rate is set to .1.

The prediction of the neural net could have several neurons that are nonzero. To determine what the predicted score was, the neuron with the largest output was taken, the associated score was used. Therefore prediction values of the neural network could be on the range of [.5, 5] at discrete intervals of .5.

VII. Support Vector Machine

In addition to the models discussed earlier, Support Vector Machines can also implemented as a method of determining a user's rating of a movie. The algorithm discussed by Xia *et al.* is inspired by the success of SVMs in text categorization and the issues collaborative filtering experiencing with regard to highly sparse data [5].

A. Original Algorithm

The fundamental concept of the algorithm is based on the idea of replacing the pearson coefficient in neighborhood collaborative filtering with a Support Vector Machine. The data is binarized with 1 representing a score of [3.5-5] and 0 representing [0.5-3.5]. Prediction errors are penalized in the cost function [5]:

$$c(y, f(x)) = \begin{cases} 1 \text{ if } f(x) \neq y \\ 0 \text{ elsewhere} \end{cases}$$

This in turn makes the risk function [Xia *et al.* 2006]:

$$R(f) = \frac{1}{l}\sum_{i=1}^{l} c(y, f(x))$$

Normal SVM formulation is [Xia *et al*. 2006]:

$$min\frac{1}{2}||w||^2 + \frac{1}{l}\sum_{i=1}^{l}\xi_i^2$$
$$\text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Smoothing can be done using Smoothing SVM (SSVM). This allows for $(w, b) \in \mathbb{R}^{N+1}$. This changes the minimization function to be [5]:

$$min\frac{v}{2}||p(\gamma - (A_1 w - \gamma b), a)||^2 + \frac{1}{2}(w^T w + b^2)$$

The algorithm proposed by Xia *et al.* is an iterative algorithm designed to create an SVM for each entry of the matrix that does not have a true rating associated with it.

During implementation, the sheer number of SVMs can be seen as a significant bottleneck. In order to calculate the number of SVMs for a matrix of 671x9066 (our full datasize, users x ratings) we would need to fit and predict on roughly 5.6 million SVMs (considering a sparsity of *~0.94*). This was deemed to be highly impractical. Even on a small scale, 671x1000, test this would be around 630,000 SVMs. For this reason, we altered the algorithm to make the problem more feasible while maintaining some of the gains from iteration.

B. Our Approach

The algorithm was modified to create SVMs for each movie in the data instead of for every missing rating. This cuts down the number of required SVMs significantly. Computation is still time consuming but it is feasible. Sklearn's SVC is used to to be the SVM algorithm with an RBF kernel instead of the SSVM described by Xia *et al.* This is due to the difficulty of finding an implementation of the SSVM algorithm that would fit our needs. It is possible to implement our own but it would be highly inefficient and could be tricky given how intricate the SSVM algorithm is.

C. Our Pseudocode

We follow the general pattern laid out by Xia *et al.* However, there are some slight variations. What follows is a hybrid pseudocode (Xia *et al.* pseudocode combined with ours) for our implementation.

Overview:

1. k represents the iteration
2. T_c^k represents the number of correct classifications at iteration k
3. ε is the difference threshold from iterations

(Hybrid implementation from Xia *et al.* and ours)

```
Initialize k = 1, $T_c^0 = 0$, $T_c^1 = 2\varepsilon$
Initialize user-movie matrix $A$ by randomly filling 0 or 1
  for empty entries

while $(T_c^k - T_c^{k-1}) \leq \varepsilon$:
  for m = 1,...,M   // where M is the number of columns
        build a classifier $f_m$ with SVM (RBF kernel)
        predict values for column m
        re-assign column m with the predicted values
                (only assign to non-ground truth values)
  end
  k += 1
  compute $T_c^k$ on the validation data
end
```

VIII. Results

A. Neighborhood Based Collaborative Filtering Hyperparameter Tuning

The Neighborhood based collaborative filtering algorithm, has many hyperparameters to tune. For example, the size of neighborhood k, the threshold for significance weighting s, the similarity measure, etc. This study only performed grid search on s and k for ranges of $[1,100]$ in steps of 10 and $[1,200]$ in steps of 50, respectively.

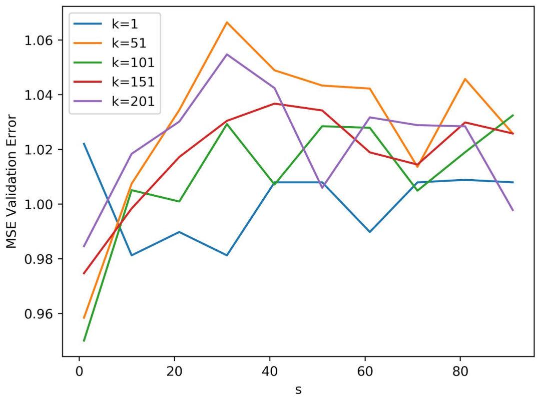

Fig. 1. Neighborhood Based Collaborative Filtering tuning curves, to optimally set hyperparameters s and k.

Fig1 illustrates the validation error of the algorithms, as s is swept across the x axis. The different curves correspond to different k values. The global minimum is observed at $k=100$ and $s=1$. The average MSE on our validation dataset was 0.957846516348, with these parameters.

B. Ridge Regression Hyperparameter Tuning

We tuned four hyperparameters independently and for 100 movies because the algorithm's spatial and time complexity are very high (see [3]) and a grid search approach taking into account all the 9125 movies would need a parallel computing implementation (using cuda for instance).

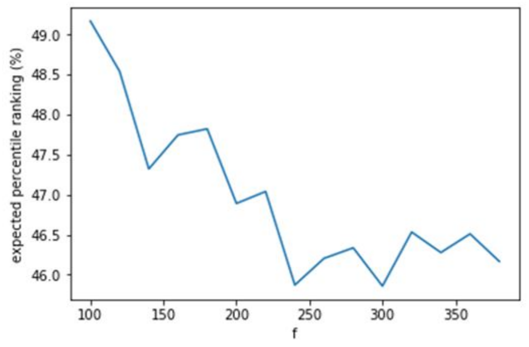

Fig. 2. Ridge Regression Collaborative Filtering tuning curve, to optimally set factor dimension. We see an optimal f in 240, near f = 200 used in the work from Yifan Hu *et al.*.

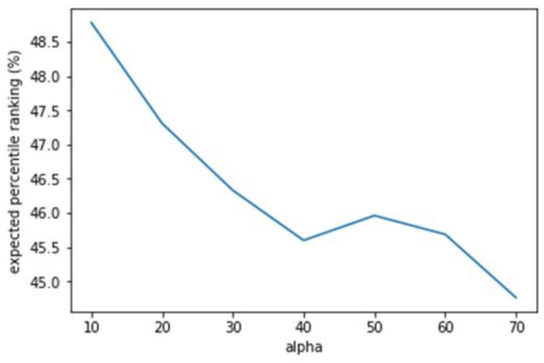

Fig. 3. Ridge Regression Collaborative Filtering tuning curve, to optimally set hyperparameter alpha. Notice that α= 70 gives better results than α= 40 proposed by Yifan Hu *et al.*.

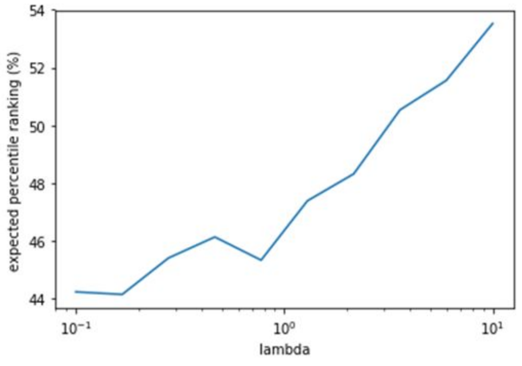

Fig. 4. Ridge Regression Collaborative Filtering tuning curve, to optimally set hyperparameter alpha. We choose λ~ 0.1.

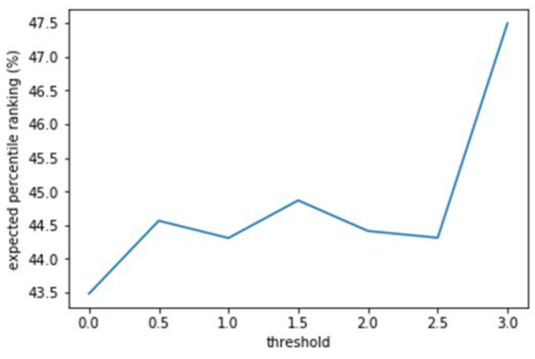

Fig. 5. Using different thresholds as proposed in V.C., we observe that the best results are obtained with a 0 threshold.

C. Mean Squared Prediction Error From Each Model

	Model Type			
	Neighborhood Collaborative Filtering	Ridge Regression	Neural Net	Support Vector Machine
Training MSE		-	3.1815	-
Validation MSE	0.9578	-	4.8034	-

Table 1. Mean squared error (MSE) for each of the four models. Since the ridge regression can only produce the k best movies to recommend, and not predict the scores for each movie in our validation set, there is no MSE to report. The support vector machine only decides whether a movie should or shouldn't be recommended, so does not have an MSE.

The neighborhood collaborative filtering has the lowest training and validation mean squared error (MSE). The neural network's validation MSE is approximately the same size as the range of potential values for the predictions.

The accuracy of the neural network is measured in addition to the MSE. The accuracy is measured as the rate at which the neural network correctly predicted the score. The score could be correctly predicted to full precision since the neural network architecture led it to classify the prediction as one of the ten possible values. Despite both the training and validation MSE being quite poor, the training accuracy for the neural network is 71.3%, and the validation accuracy is 33.3%.

The support vector machine's performance can only be measured by accuracy since ratings are not predicted. The accuracy of the support vector machine measures how often a movie is either recommended when the score = [3.5, 5] or is not recommended when the score = [.5, 3]. The training accuracy of the model was 65% and the validation accuracy are also 65%.

IX. Discussion

In this work, we explore four different algorithms for predicting movies for a user to watch. One model, ridge regression predicts a list of best movies to recommend. The support vector machine provides a binary prediction about whether a movie should be recommended or not. The other two of the four models directly predict a score that the user would rate a movie, to guide a recommendation system.

Both the neural network and the support vector machine have accuracy measurements. The accuracy of the neural network defines how often the neural network's prediction is the exact value rated by the user. Since the neural network classified each user-movie pair into a binned value, it could provide an errorless prediction. The accuracy of the support vector machine defines how often the support vector machine would recommend a movie that the user rated poorly.

A. Model Accuracy

The neighborhood collaborative filtering model has both the best prediction and validation error. The ridge regression and support vector machine models cannot be measured by MSE. However, the ridge regression had an expected percentile ranking around 50% and the Support Vector Machine had a validation accuracy of 65%. Both of these values are far too poor to use for a recommendation system.

The MSE of the neural net is far worse than the neighborhood collaborative filtering, even for the training error. However, the training accuracy is remarkably high. This suggests that when the neural net predicts a score incorrectly, its prediction is far off. This is reflected in the model architecture. Since the neural net attempts to classify which prediction value a user-movie pair should have but doesn't realize that there's an inherent relationship between classes, there's no mechanism to train the model to predict similar ratings when wrong. Training a model that can predict the score using a linear output, rather than several classes may yield better results. Alternatively, rather than just take the output neuron with the highest value, take a weighted average of the neural net's prediction where the weights are normalized confidence estimates provided by the network.

B. Model Tractability

Neighborhood collaborative filtering performs well given the dataset. The training algorithm has a $O(u^2)$ runtime where u is the number of users. Therefore even as the amount of data increases it can scale well.

The ridge regression model needs significant memory to store the confidence matrices, C^u and C^m, so a single 8GB laptop cannot train the model for 9125 movies. Even running on the 64GB server we used, the execution was slow because the implementation is not optimized and should invoke parallel computing methods.

The neural network has a significant training time, but it is possible to train on the full dataset. With the 90,003 sample training set, using TensorFlow without GPU optimization and the learning parameters as described earlier, each epoch takes approximately 30 minutes. Predicting the scores of 10,001 user, movie pairs takes approximately one minute. With an initial learning rate of .1, the loss converges after 13 epochs. Even if the batch size and learning rate were to decrease, on a GPU optimized computer, the training and prediction times would be very reasonable.

The Support Vector Machine model is much too slow for a large scale prediction algorithm. Even with our modifications, the time to train and predict are far too costly for this to be considered a feasible algorithm. This essentially boils down to extremely poor scaling.

Also, the predictions are, as a whole, somewhat poor. While it is a better than predicting all 1 or 0, this improvement came at the cost of needing lots of computation.

C. Model Utility

The first measure of how useful a model is for a recommendation system is what values the model predicts. The neighborhood collaborative filtering and neural network models are the best in this measure. Since they predict the user's ratings, it allows the recommendation system more flexibility in the movies recommended. The movies recommended can be limited to a fixed number of movies or ratings above a certain threshold. The ridge regression, on the other hand, could only predict the top movies to recommend. That the user would actually rate all those movies highly is not guaranteed, but the movies are ranked in an order of best-to-recommend to worst-to-recommend. Finally the support vector machine only predicts whether a movie should be recommended. The confidence of that recommendation is not measured whatsoever. The neighborhood collaborative filtering had the lowest runtime both in training and an prediction. It also had the lowest mean squared error. On all measures of utility, the neighborhood collaborative filtering model is the best for recommendation systems.

X. Appendix

A. Neighborhood Collaborative Filtering Sample Recommendations

Below is an example of recommendations for user_id = 2, based on the full MovieLens dataset:

5-star rated Movies	Top 5 Recommendations
Brady Bunch Movie, The (1995) | Comedy/Parody	Clerks (1994) | Comedy
Clueless (1995) | Comedy/Romance	Shallow Grave (1994) | Comedy/Drama/Thriller
Apollo 13 (1995) | Adventure/Drama/IMAX	Naked Gun 33 1/3: The Final Insult (1994) | Action/Comedy
Circle of Friends (1995) | Drama/Romance	Santa Clause, The (1994) | Comedy/Drama/Fantasy
Like Water for Chocolate (Como agua para chocolate) (1992) | Drama/Fantasy/Romance	Highlander III: The Sorcerer (a.k.a. Highlander: The Final Dimension) (1994) | Action/Fantasy
Legends of the Fall (1994) | Drama/Romance/War/Western	
Nightmare Before Christmas, The (1993) | Animation/Children/ FanTasy/Musical	
Sense and Sensibility (1995) | Drama/Romance	
Terminator 2: Judgment Day (1991) | Action/Sci-Fi	
Dances with Wolves (1990) | Epic Western/Drama	
Batman (1989) | Fantasy/Crime film	

Table 2. Neighborhood collaborative filtering recommended movies for user_id = 2

It can be seen from Table 1 that user_id 2 rated Comedies, Dramas, Fantasy and Action movies very highly. The top 5 movies suggested also fall in those genres. Since no information about the movie genre was embedded in the recommendation, this is an indication that indeed users that rate movies in a similar fashion, tend to like the same genres of movies.

B. Ridge Regression Sample Recommendations

We provide below an example of recommendations for user_id = 2, based on the full MovieLens dataset:

5-star rated Movies	Top 5 Recommendations
Brady Bunch Movie, The (1995) | Comedy/Parody	Vamp (1986) | Comedy/Horror
Clueless (1995) | Teen film/Romance	Tarzan, the Ape Man (1981) | Adventure
Apollo 13 (1995) | Drama/History	OH in Ohio, The (2006) | Comedy
Circle of Friends (1995) | Film adaptation/Indie film	Funny Lady (1975) | Comedy/Musical
Like Water for Chocolate (Como agua para chocolate) (1992) | Drama/Romance	Merry War, A (1997) | Comedy
Legends of the Fall (1994) Drama/Romance/War/ Western	
Nightmare Before Christmas, The (1993) | Animation/Children/ FanTasy/Musical	
Sense and Sensibility (1995) | Drama/Romance	
Terminator 2: Judgment Day (1991) | Action/Sci-Fi	
Dances with Wolves (1990) | Adventure/Drama/Western	
Batman (1989) | Action/Crime/Thriller	

Table 3. Ridge regression recommended movies for user_id = 2 (K=5, f = 200, λ=1, α=40)

References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=<http://dx.doi.org/10.1145/2827872>

[2] Asela G., Guy Shani. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. Journal of Machine Learning Research 10 (2009) 2935-2962.

[3] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. IEEE International Conference on Data Mining, 263–272.

[4] He, X., Liao, L., & Zhang, H. (2017). Neural Collaborative Filtering. Proceedings of the 26th International Conference on World Wide Web, 173-182. http://papers.www2017.com.au.s3-website-ap-southeast-2.amazonaws.com/proceedings/p173.pdf

[5] Xia, Z., Dong, Y., & Xing, G. (2006). Support Vector Machines for Collaborative Filtering. Proceedings of the 44th annual Southeast regional conference, 169-174. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.400&rep=rep1&type=p