STATS 507 Project Preliminary Report:

`MovieLens` Datasets—Predicting and Analyzing User Ratings of Movies

TRONG DAT DO [1,*] and SIMON FONTAINE [1,**]

[1] *University of Michigan, Department of Statistics. West Hall, 1085 South University, Ann Arbor, MI, U.S.A., 48109. E-mail:* [*] `dodat@umich.edu`; [**] `simfont@umich.edu`

Abstract. The `MovieLens` datasets contain user ratings of movies as well as movie and user information. In this preliminary report, we consider four predictive models of the ratings based on the available information: K-nearest-neighbors, neural networks, matrix completion using singular value decomposition and restricted Bolztmann machine. We tune all models to finally compare them on their out-of-sample performance. All the code and results can be found at `https://github.com/fontaine618/507-Project/`.

Contents

1. The MovieLens Datasets

The MovieLens datasets (Harper and Konstan, 2015) contains user ratings of a variety of movies continuously collected starting from 1998. In addition to the user-movie-rating pairings, the datasets contains information about movie genres, word tagging of movies provided by users and user demographic information.

We will consider the MovieLens 100K Dataset[1], which is one of the multiple datasets provided by GroupLens[2]. We will be interested in this particular dataset because it contains additional demographic information about the users in the dataset. To include tagging information, we also consider the MovieLens Tag Genome Dataset[3]. Here is a summary of the contents of the datasets that will be used[4]:

MovieLens 100K Dataset The dataset was collected from the MovieLens website (movielens.umn.edu) between September 19th, 1997 through April 22nd, 1998. It has been pre-processed and cleaned to include only examples where the users have made at least 20 ratings during the collection period and where demographic information are complete. In the u.data file, there are 100,000 ratings on the scale of 1 to 5, taking only integer values. It contains the following entries: user id, item id, rating, timestamp. In the u.item file, there are 1681 movies with the following information: movie id, movie title, release date, IMDb URL and 19 columns indicating movie genre with 0-1 encoding where 1 denotes that the movie is of the corresponding genre. In the u.user file, there are 943 users with the following information: user id, age, gender, occupation (see u.occupation file for details) and zip code.

MovieLens Tag Genome Dataset This dataset contains tagging information of 9734 movies and 1128 tags. In particular, the tag_relevance file contains the relevance of all tags for all movies reported on a continuous scale from 0 to 1, where 1 indicates strong relevance.

2. Research Questions

2.1. Prediction Modeling

Our first research question is to construct a predictive model for the user ratings using the available information. In particular, we wish to produce a model that is able to

[1] Available at https://grouplens.org/datasets/movielens/100k/
[2] Organization website: https://grouplens.org/
[3] Available at https://grouplens.org/datasets/movielens/tag-genome/
[4] From the README.txt file attached to the datasets (http://files.grouplens.org/datasets/movielens/ml-100k-README.txt, http://files.grouplens.org/datasets/tag-genome/README.html)

accurately predict the movie rating (for some movie already in the dataset) by a given user (also in the dataset). This model could then be part of a *recommendation system* where the predicted rating could be used as input to produce the recommendations.

2.2. Exploratory Analysis

A secondary research question we are interested in is to analyze the effect of the available information on the user ratings. For example, we could look for genres and tags that are related to movies with better ratings. Then, we can perform more granular analyses using the demographic data: this could allow to extract correlations between population groups and movie interests. The insights recovered from such analyses could be relevant for decision-making such as identifying which movies to produce and which population groups to target with advertisement.

3. Methodology

3.1. Data Pre-processing

The `MovieLens 100K Dataset` and the `MovieLens Tag Genome Dataset` have both been extensively cleaned by `GroupLens`[5]. Thus, the main pre-processing we have to perform is the merging of the different datasets and the creation of the training and testing sets.

3.1.1. Merging the Datasets

First, the three datasets in `MovieLens 100K Dataset`, corresponding to user data, movie data and the ratings, have unique IDs which allows us to easily match them. There were no ratings which we were unable to match to users and/or movies in these datasets.

Second, the `MovieLens Tag Genome Dataset` also has unique IDs identifying movies, but they differ from those in the `MovieLens 100K Dataset`. Hence, we resort to matching the movie on the movie name. Direct matching of the strings allowed of to match a large proportion (∼80%) of the movies. There were also some movie that we were able to match using simple rules. For example, the movie names include the year of publication which were often mismatched by a year and prevented direct matching. Also, other movie names included the original name (in a foreign language) in either of the two datasets which prevented direct matching, but could still be detected. Finally, visual inspection (mostly by hand) of the remaining unmatched movies allowed to identify a few additional matches. In total, we were able to match 1558 out of the 1681 in the `MovieLens 100K Dataset`. We therefore dropped all ratings of the movies which we could not matched. Fortunately, only very marginal movies did not make the cut and only 588 ratings out of 100,000 had to be dropped: the resulting dataset therefore is composed of 99,412 ratings on 1558 movies by 943 users.

[5]See the two `README` files for details: http://files.grouplens.org/datasets/movielens/ml-100k-README.txt and http://files.grouplens.org/datasets/tag-genome/README.html

3.1.2. Data subsetting

Upon joining the different datasets, we further subset the data in order to insure adequate representation of all included movies. In particular, we identify that some movies have small frequency (some appearing only once, for example). To fix this problem, we omit all ratings of movies which appear less than 20 times in the dataset. The final dataset then contains 94,692 ratings on 935 movies; all 943 users remain in the dataset.

3.1.3. Data Splitting

In order to assess the performance of the various models we consider, we proceed to the usual training-testing splitting. The training set consists of 75% of the dataset (71,035 ratings) and the testing set of 25% of the data (23,657 ratings). For consistency of results, the splitting was kept constant throughout the analysis. To insure adequate representation of all movies in each sets, we proceeded to a stratified sampling of ratings within movies.

Furthermore, the training set was split into cross-validation sets in order to perform model tuning and model selection. In particular, the 71,035 ratings were each assigned to one of 5 folds randomly, yielding training sets of size 56,828, on average, and validations sets of size 14,207, on average. Again, this splitting was kept constant throughout the analysis.

3.2. Modeling

3.2.1. K-Nearest-Neighbors

The *K-nearest-neighbors* (K-NN) method is a non-parametric predictive model. Given a dataset of n examples $(\mathbf{x}_i, y_i)_{i=1}^n$, the prediction for a new $\mathbf{x}$ is obtained as follows: find the K indices $i_1, \ldots, i_K \in [n]$ such that the distances $d(\mathbf{x}, \mathbf{x}_{i_k})$ are minimized and aggregate the responses $y_{i_1}, \ldots, y_{i_K}$ into the prediction $\hat{y}$. To completely determine a K-NN model, we need to specify the number of neighbors K, the distance function d as well as the aggregation method.

Distance function. Our feature space consists predominantly of three types of information: movie genres (19 dimensions, encoded as 0/1), movie tags (1128 dimensions, in $[0, 1]$) and user information (24 dimensions, various). Thus, our feature space has 1171 dimensions.

We consider the Euclidean distance $d(\mathbf{x}, \mathbf{x}') = \| \mathbf{x} - \mathbf{x}' \|_2$ for our distance function, but we proceed to some transformation and parameterization to control the behavior of the distance.

First, we standardize the features to get a mean of 0 and a variance of 1 across all examples. This step is crucial for any K-NN model using a general distance such as the Euclidean distance since features with different scales will have a lot more weight in the total distance. For example, all of our features are between 0 and except for the user age

which varies from 7 to 73 and if we do not standardize this features, the K-NN will only match examples of very similar ages irrespectively of the other features.

Second, to account for the disproportional dimensions of the three groups of features as well as the relative importance of these groups, we introduce two parameters in the distance function. We fix the genre contribution to the distance to 1 and consider two multiplying factors $\alpha_{\text{tags}}, \alpha_{\text{user}} \geqslant 0$, respectively for tags and user information. Then, we transform the standardized features by scaling the correponding groups of features:

$$\mathbf{x} := (\mathbf{x}_{\text{genres}}, \mathbf{x}_{\text{tags}}, \mathbf{x}_{\text{user}}) \quad \mapsto \quad \tilde{\mathbf{x}} := (\mathbf{x}_{\text{genres}}, \alpha_{\text{tags}}\, \mathbf{x}_{\text{tags}}, \alpha_{\text{user}}\, \mathbf{x}_{\text{user}})$$

This rescaling has the effect of inflating or deflating the contribution of a group of features to the total distance; tuning α_{tags} and α_{user} can therefore improve the performance of the model by using a more adequate distance.

The features naturally encode the movie and the user. Indeed, two examples of ratings on the same movies will have a distance discount as 1147 features will be identical and do not contribute to the distance at all. Similarly ratings made by the same user will have a distance discount as 24 features will be identical. Then, for appropriate values of α_{tags} and α_{user}, we expect the set neighbors of some $\mathbf{x}$ to include ratings made by the same users as well as ratings on the same movie.

Aggregation. Since our responses—the ratings—take discrete numerical values, we consider two aggregation methods. First, the *regression* approach consists of simply averaging the K responses: $\hat{y} = \frac{1}{K}\sum_{k=1}^{K} y_{i_k}$. Second, the *classfication* approach is to set the prediction $\hat{y}$ using a majority vote: $\hat{y} = \arg\max_{y \in [5]} \sum_{k=1}^{K} \mathbb{1}(y = y_{i_k})$. It is worth noting that the classification approach loses the ordinal property of ratings: if the K neighbors have ratings equal to $(1, 1, 5, 5, 5)$, then the prediction would be 5 for classification while regression would predict 3.4.

Implementation details. We implement our K-NN models using the `sklearn.neighbors` module (Pedregosa et al., 2011): the classification predictions are made using `KNeighborsClassifier` and the regression predictions are made using `KNeighborsRegressor`. We use default settings except, obviously, for the number of neighbors. Our model has four tuning directions: aggregation method, number of neighbors and the two multipliers $\alpha_{\text{tags}}, \alpha_{\text{user}} \geqslant 0$.

3.2.2. Neural Networks

Our second predictive model uses a neural network (NN) approach. As input, we have the movie id, the movie genre and tags, the user id and information; as output, we have the rating. To fully define the network architecture, we need to specify how we treat the ids, the hidden layers structure, the final transformation and the loss function. A graphical representation of the NN can be found in Figure 2.

Movie and user embeddings. While genre, tags and user details contains *some* information about the movie and user, they do not capture the full description of these two object with respect to the ratings. In particular, without these ids, two movies with

identical features will be treated identically by the network even though one might be inherently better than the other (e.g., better ratings). Similarly, two users with identical features cannot be distinguished even tough they might have different rating habits or movie preferences. Also, simply encoding the ids in a *one-hot* fashion does not abstract the latent features of interest, which are most likely shared between some movies and some users. Thus, we augment the feature input with movie and user embeddings, which adds a fixed number of features, learned by the model, to each example.

Hidden layers. Given the augmented features, we include multiple fully-connected layers with linear nodes (with bias) linked through some activation function. We consider the ReLU and sigmoid activation functions as well as different numbers of layers and of hidden nodes.

Transformation and loss function. As was mentioned in the description of the K-NN model (Section 3.2.1), our discrete numerical response can be treated in multiple ways: for this NN approach we consider five such treatments.

First, a simple regression scheme can be employed. In this case, we treat the ratings as a continuous output. We connect the last hidden layer to the ratings using an identity link and use the squared error loss as our optimization criterion.

Second and third, we can utilize the fact that our numerical outcome as a fixed, finite range. Indeed, since ratings take values between 1 and 5, we can force the network to produce predictions within or close to that range. In practice, this has the effect of not penalizing as much predictions that are away from the $[1, 5]$ range: such predictions could be interpreted as that the model is really confident that the prediction should be at the boundary. To proceed as such we consider to simply clip the predictions to the range $[0, 6]$, which is commonly known as the ReLU6 transformation. Alternatively, we consider a fixed sigmoid transformed mapped to the interval $(-2, 8)$ which is fairly linear for the observed values but contracts predictions far from the true range closer to possible values. The three transformations are depicted in Figure 1.

Fourth, we can treat our responses as five classes where only one is observed per example. In that case, we encode the ratings in the *one-hot* fashion, that is, 4 gets encoded as $(0, 0, 0, 1, 0)$. Then, the final layer is transformed using the usual softmax function and the loss function used is the binary cross-entropy. Again, this encoding loses the ordinal nature of the ratings.

Fifth, we can choose a different encoding for the five classes that will reflect ordering of ratings. Indeed, we can encode a 4 as $(1, 1, 1, 1, 0)$ and model the responses in the multi-label classification way. Then, since all encoded responses have this ordinal structure, the NN will reproduce it in its predictions. In that case, we perform a sigmoid transformation on the five final nodes before comparing them to the encoded ratings; the loss function is again the binary cross-entropy.

Implementation details. Our NN models are implemented using the `PyTorch` module (Paszke et al., 2017) using stochastic gradient descent. We use a fixed learning rate of 0.01, batch size of 512 and number of epochs of 50; the activation functions are chosen to

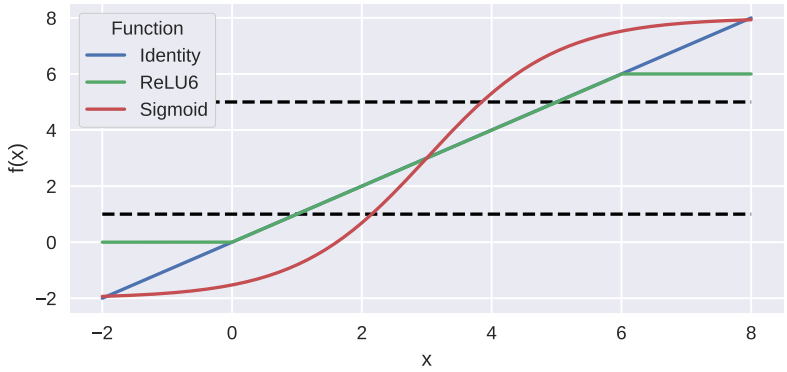

Figure 1. The three transformations considered at the output node for regression in a neural network. The dashed horizontal lines represent the boundary of observed ratings.

be the ReLU function in all cases to reduce the tuning space. For this model, we have five tuning directions: movie and user embedding sizes, hidden layers number and sizes and the final transformation. We also compare whether including the external information improves on simply using the two embeddings.

3.2.3. Matrix completion

Building a recommendation system by doing matrix completion is one of *collaborative filtering* techniques. This means that we will try to suggest movies to an user by inspecting the ratings of "similar" users. To be more specific, it is assumed that the taste, or the ratings, made by users follows some low-rank structure, and use that assumption to make prediction. The idea of using SVD for movie recommendation is simple but it turns out to be successful in application. The winner in the Netflix prize competition used SVD as the main technique in their algorithm (Bennett et al., 2007).

Model description. In this model, we use 2 hyper-parameters, which are the embedding dimension k and the number of iterations I. We set up and train the training data as follows.

1. Create a pivoted matrix A in which rows are users and columns are movies. The (i, j)-cell of the matrix is the rating of user i for movie j, and is `NA` if there is no information about that in the training set. We also create a masked matrix which indicate which cell in pivot matrix is `NA`. After that, we fill the `NA` cells in A with `0`.
2. Repeat I times:
   - Fill all the non-masked cell in A with the original visible rating.
   - Approximate A by its truncated SVD of rank k.

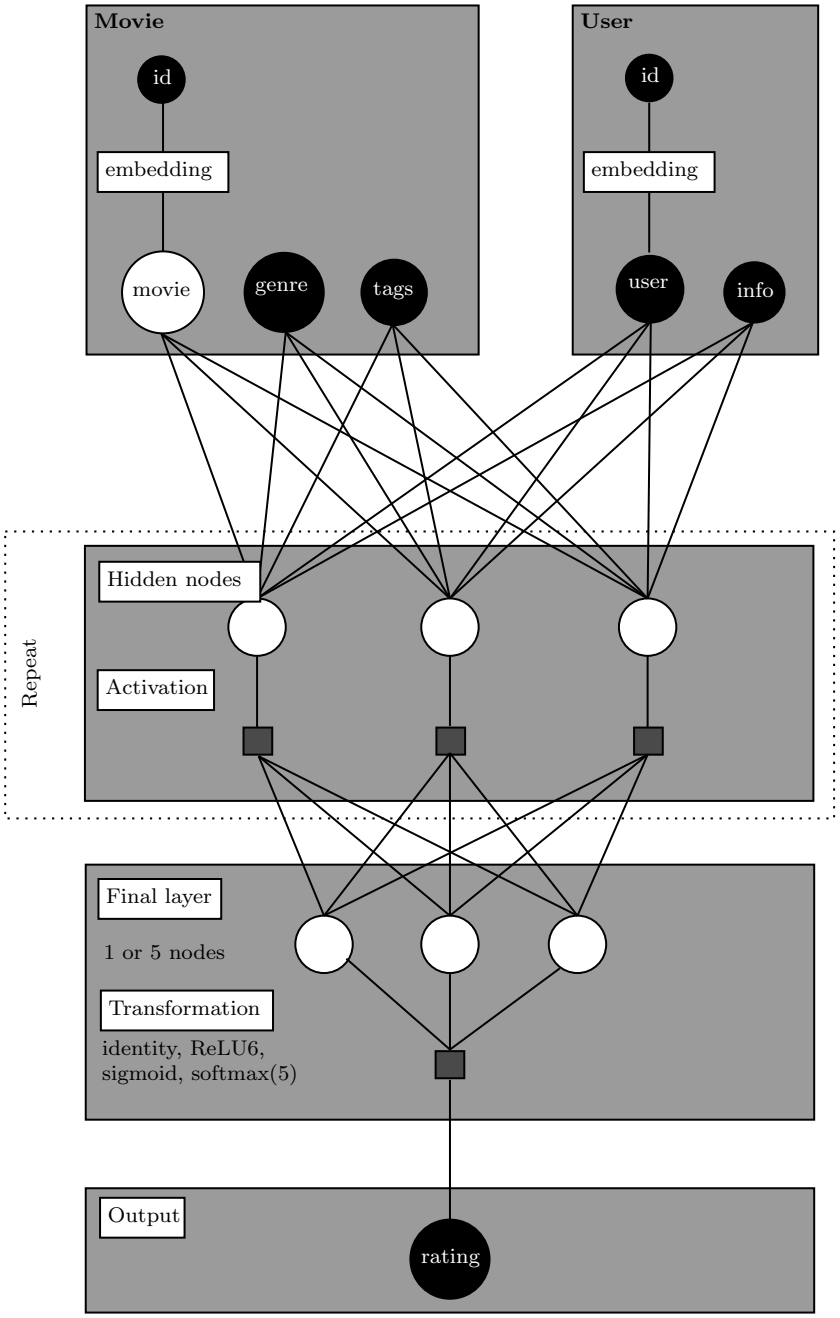

Figure 2. Neural network architecture described in Section 3.2.2. Black nodes represent observed quantities; white nodes, hidden variables; grey squares, functions.

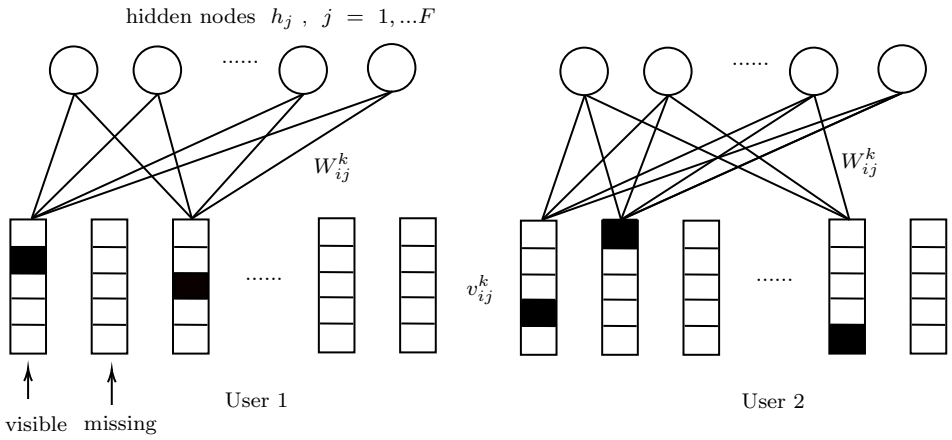

Figure 3. Restricted Boltzmann Machine in Section 3.2.4. Each user has a graphical model connecting hidden variables and visible ratings. Weights W_{ij}^k are shared among users.

By iterating two steps above, it allows us to reuse the visible information over and over, and extract the low rank structure of ratings.

Implementation details. We use the `svds` function from `scipy.sparse.linalg` module (Virtanen et al., 2020) to compute truncated SVD of pivoted matrix and repeat the two steps above to get the reconstruction of users-movies ratings matrix. After that, the information from the reconstructed matrix is used as predictions. This model has two turning parameters: the dimension k and number of iterations I.

3.2.4. Restricted Boltzmann Machine

Similar to matrix completion, Restricted Boltzmann Machine (RBM) is also a collaborative filtering method. It assumes ratings of users are affected by some latent binary variables. We construct a probabilistic model where there is F hidden nodes in total and a bipartite graph for each user connecting the hidden variables with the visible ratings.

Model description. Denote hidden variables by $\mathbf{h} = (h_1, h_2, \ldots, h_F)$ and suppose there are K rating scores in total (5 for our data set). For now, to reduce the burden of notation, we only consider the model for each user and will talk about the connection between them later. For a given user, suppose m movies were rated and write the visible ratings $\mathbf{V} = (v_i^k)_{i,k}$ where $v_i^k = 1$ if movie m was rated k and 0 otherwise. The generative model is defined by

$$p(\mathbf{V}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{V}, \mathbf{h})), \tag{3.1}$$

where $Z = \sum_{\mathbf{V}',\mathbf{h}'}$ is the normalization. The term $E(V,h)$ is so called *energy* and defined by

$$E(V,h) = -\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ij}^k h_j v_i^k - \sum_{i=1}^{m}\sum_{k=1}^{K} b_i^k v_i^k - \sum_{j=1}^{F} h_j b_j, \tag{3.2}$$

where W_{ij}^k is the weight connecting h_j to v_i^k, b_i^k is a bias term for v_i^k and b_j is the bias for h_j. Therefore, each user has a different bipartite graph model and visible ratings $\mathbf{V}$, but they all share the same weight (W_{ij}^k) and biases $(b_i^k), (b_j)$.

From this, we can calculate the conditional probability

$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k)}{\sum_{l=1}^{k} \exp(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l)}, \tag{3.3}$$

and

$$p(h_j = 1|\mathbf{V}) = \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k), \tag{3.4}$$

where σ is the sigmoid function. The marginal distribution for $\mathbf{V}$ is

$$p(\mathbf{V}) = \frac{1}{Z}\sum_{\mathbf{h'}} \exp(-E(\mathbf{V},\mathbf{h'})).$$

Our aim is to find $(W_{ij}^k, b_i^k, b_j)_{i,j,k}$ maximizing the marginal distribution for visible ratings $\mathbf{V}$, which corresponds to the *maximum likelihood* method. We will employ gradient descent to find the optimal weights.

Learning. We need to compute the gradient of $\log p(\mathbf{V})$ in order to perform gradient ascent. By the chain rule

$$\begin{aligned}
\frac{\partial \log p(\mathbf{V})}{\partial W_{ij}^k} &= \frac{1}{p(\mathbf{V})}\frac{\partial p(\mathbf{V})}{\partial W_{ij}^k} \\
&= \frac{1}{p(\mathbf{V})}\left(\frac{\sum_{\mathbf{h}} \mathbb{1}_{[h_j=1, v_i^k=1]} p(\mathbf{V},\mathbf{h})}{Z} - \frac{p(\mathbf{V})\sum_{\boldsymbol{\nu},\mathbf{h}} \mathbb{1}_{[h_j=1,\boldsymbol{\nu}_i^k=1]} p(\boldsymbol{\nu},\mathbf{h})}{Z}\right) \\
&= \mathbb{E}_{h|\mathbf{V}}(v_i^k h_j) - \mathbb{E}(\nu_i^k h_j) \\
&=: \left\langle v_i^k h_j \right\rangle_{data} - \left\langle \nu_i^k h_j \right\rangle_{model}
\end{aligned}$$

where $\langle\cdot\rangle_{subscript}$ denotes the conditional expectation with respect to the subscript. Similarly, we have

$$\frac{\partial p(\mathbf{V})}{\partial b_j} = \mathbb{E}_{h|\mathbf{V}}(h_j) - \mathbb{E}(h_j) = \left\langle h_j \right\rangle_{data} - \left\langle h_j \right\rangle_{model}, \tag{3.5}$$

and

$$\frac{\partial p(\mathbf{V})}{\partial b_i^k} = v_i^k - \mathbb{E}(\nu_i^k) = v_i^k - \left\langle \nu_i^k \right\rangle_{model}. \tag{3.6}$$

In each equation, it is customary to call the first term *positive statistics* and second term *negative statistics*. Hence, we have the analytical representation of gradient of weights and biased terms. In every equation, the first term is easy to compute thanks to (3.4). However, computing the terms $\langle \nu_i^k \rangle_{model}, \langle h_j \rangle_{model}$, and $\langle \nu_i^k \rangle_{model}$ requires to take the sum over all value of $\boldsymbol{\nu}, \mathbf{h}$, which takes exponential time and makes the algorithm inefficiently. We will instead use the *Contrastive Divergence* (CD) method (Hinton, 2012) which consists of approximating $\langle \nu_i^k h_j \rangle_{model}$ with $\langle \nu_i^k h_j \rangle_{recon}$, an approximated reconstruction of $\langle \nu_i^k h_j \rangle_{model}$ based on an idea similar to the Gibbs' sampler.

Computing gradient using Constrastive Divergence. The idea behind Contrastive Divergence (Hinton, 2012) is to approximate the gradient using the difference of two Kullback-Leibler divergences and ignore one trickly term. Although this is a crude approximation, it turns out to work really well in many applications. The Constrastive Divergence with n steps (CD_n) can be interpreted as follows. Given the visible ratings $\mathbf{V}$, first we sample binary value for hidden units $h_{data} = h$ by equation (3.4). Then we do n steps of Gibbs' sampling, each contains two intermediary steps

1. Sample $\boldsymbol{\nu} \leftarrow \boldsymbol{h}$ based on equation (3.3).
2. Sample $\boldsymbol{h} \leftarrow \boldsymbol{\nu}$ based on equation (3.4).

To derive the positive statistics, we use the value of data $\mathbf{V}$ and h_{data}. Although we can calculate it analytically using (3.4), using the sample values can reduce the noise when we take the difference with the negative statistics. Hence, we have

$$\langle v_i^k h_j \rangle_{data} \leftarrow v_i^k h_{data,j}, \quad \langle h_j \rangle_{data} \leftarrow h_{data,j}$$

When collect the negative statistics, it is advised in Hinton (2012) that in the last step, we should only collect $\boldsymbol{h}$ as the probability $p(\mathbf{h}|\boldsymbol{\nu})$ (but not sample from it) to get the updates

$$\langle \nu_i^k h_j \rangle_{recon} \leftarrow \nu_i^k h_j, \quad \langle h_j \rangle_{recon} \leftarrow h_{data,j}, \quad \langle \nu_i^k \rangle_{recon} \leftarrow \nu_i^k.$$

Making predictions. Given a set of visible unit $\mathbf{V}$, we can predict the rating for a new movie by using Bayes' rule and marginalization of $\mathbf{h}$:

$$\begin{aligned}
p(v_q^k = 1|\mathbf{V}) &\propto p(v_q^k = 1, \mathbf{V}) \\
&\propto \sum_{\mathbf{h}} p(v_q^k = 1, \mathbf{V}, \mathbf{h}) \\
&\propto \exp(b_k^q) \prod_{j=1}^{q} \sum_{h_j \in \{0,1\}} \exp(\sum_{il} v_i^l h_j W_{ij}^l + h_j W_{qj}^k + h_j b_j) \\
&= \exp(b_k^q) \prod_{j=1}^{q} \left(1 + \exp(\sum_{il} v_i^l W_{ij}^l + W_{qj}^k + b_j)\right)
\end{aligned}$$

Implementation details. We build and train a RBM model by ourselves using basic modules such as `numpy` (van der Walt, Colbert and Varoquaux, 2011). There are three turning parameters in the model: the number of hidden nodes, the number of Gibbs samplings in Contrastive Divergence, and the learning rate. Many details in the implementation is taken from Hinton (2012).

4. Results

We perform 5-fold cross-validation on many instances of our models against the training set and produce the *mean squared error* (MSE) and prediction accuracy averaged over the 5 folds. For models with numerical predictions, we assign the predicted rating to the nearest integer in the range $[1, 5]$. Models with class predictions simply use the class label as their numerical prediction.

It is worthwhile to mention that we are in a classification problem with 5 classes so a random assignment would produce 20% accuracy: we observe accuracies close to 40%, which is already a great improvement. MSEs less than 1 means that the model predicts, on average, ratings less than 1 point away from the recorded ratings.

We first detail the results for the four models we consider, noting some observation on their CV performance. Then, we proceed to a comparison of the four models in terms of CV and testing.

4.1. *K*-Nearest-Neighbors

Upon training and testing 35 instances of the K-NN model described in Section 3.2.1 with varying tuning parameters, we obtain the results depicted in Table 1 which contains a selection of the best models.

As could be expected, regression aggregation performs better in terms of MSE while classification aggregation performs better in terms of prediction accuracy. Also, the value of the tuning parameters α_{user} and α_{tags} for these best models seems to align with their purpose. Indeed, we observe that the best models require to inflate the importance of user information and deflate that of tags.

The K-NN models is particularly inefficient computationally: when K increases from 5 to 100, training time increases from minutes to hours.

4.2. Neural Network

We trained over 400 instances of the NN approach described in Section 3.2.2; the results for a selection of the best models can be found in Table 2. We make some observations with regards to the tuning parameters.

First, we observe that including the external features (genre, tags, user info) greatly improves the model as we see a substantial reduction in MSE and accuracy.

		α		CV		Test	
Aggregation	K	User	Tags	MSE	Acc.	MSE	Acc.
Regressor	50	200	0.5	0.9898	0.3741	0.9737	0.3767
		1000	1.0	0.9950	0.3733	0.9780	0.3779
		500	0.5	0.9966	0.3725	0.9802	0.3762
		200	0.2	1.0129	0.3703	0.9967	0.3742
		500	0.2	1.0129	0.3702	0.9967	0.3742
	75	200	0.5	1.0145	0.3680	0.9972	0.3714
	100	100	1.0	1.0166	0.3690	1.0045	0.3694
Classifier	100	100	1.0	1.2363	0.3900	1.2395	0.3915
	50	100	1.0	1.2482	0.3935	1.2547	0.3940
	100	1000	1.0	1.2979	0.3885	1.2759	0.3943
		100	0.1	1.3521	0.3832	1.3454	0.3832
		1000	0.1	1.3533	0.3833	1.3484	0.3831

Table 1. Cross-validation and testing metrics of some of the best K-nearest-neighbors models trained on the `MovieLens` dataset. The complete logs can be found alongside the project's code at https://github.com/fontaine618/507-Project/.

Second, we note that the ordinal transformation does not appear in the best results: inspecting the complete logs, we find that this method performs better than classification in terms of MSE but worse in terms of prediction and is therefore in the middle of the pack in both categories. The classification scheme yields good prediction performance, but we see accuracies similar to those from regression approaches. The three regression transformation—identity, ReLU6 and sigmoid—seem to perform rather similarly with the sigmoid transform being slightly worse generally.

Third, the best performing models in terms of CV metrics seem to require simpler hidden layers structures: while the experiments included networks with up to four hidden layers, the best models generally contain only two or three. This exhibits the benefit of using CV for model selection as it prevents overfitting. A more direct way to prevent overfitting is through weight decay: we find that the best CV metrics are obtained using non-zero weight decay.

In terms of computing time, the NN models we consider train for 50 epochs in around 1 minute on GPU.

4.3. Matrix completion

We trained around 100 instances of matrix completion models described in Section 3.2.3. We choose to present some models with its parameters. From the result, in Table 3, we can see some facts about turning parameters as below.

For the embedded dimension k, if it is chosen to be too small, the model will not capture enough the complexity of the data and lead to the underfitting phenomena. If k is large, the computational time will relatively increase but the prediction is not getting better, and there is even the overfitting phenomena.

For the iterative number I to use SVD, it can be seen that small I leads to underfitting model meanwhile large I leads to overfitting model. This can be interpreted that if we

	Embeddings				CV		Test	
Transform	User	Movie	Hidden layers	Decay	MSE	Acc.	MSE	Acc.
With features—Best MSE								
RegressorRelu6	128	64	1024, 128, 64	10^{-6}	0.9402	0.3985	0.9016	0.4165
			1024, 128	10^{-6}	0.9461	0.3810	0.9516	0.3810
				10^{-4}	0.9546	0.3758	0.9489	0.3841
Regressor	128	64	1024, 128	10^{-5}	0.9676	0.3870	0.9869	0.3631
RegressorRelu6	128	64	1024, 128	—	0.9693	0.3729	0.9418	0.3869
Regressor	128	64	1024, 128, 64	10^{-6}	0.9915	0.3874	1.0240	0.3570
RegressorRelu6	128	64	1024, 128, 64	—	0.9944	0.3749	1.1091	0.3330
With features—Best Accuracy								
RegressorRelu6	128	64	1024, 128, 64	10^{-6}	0.9402	0.3985	0.9016	0.4165
Regressor	128	64	1024, 128, 64	10^{-6}	0.9915	0.3874	1.0240	0.3570
			1024, 128	10^{-5}	0.9676	0.3870	0.9869	0.3631
			1024, 128, 64	10^{-5}	1.0155	0.3857	0.9780	0.3711
RegressorRelu6	128	64	1024, 128	10^{-6}	0.9461	0.3810	0.9516	0.3810
Classifier	128	64	1024, 128	10^{-6}	1.1216	0.3795	1.1070	0.3918
				10^{-5}	1.1219	0.3794	1.1071	0.3914
				—	1.1218	0.3794	1.1066	0.3919
				10^{-4}	1.1213	0.3793	1.1063	0.3915
Without features—Best MSE								
Regressor	128	64	1024, 128	10^{-6}	1.1661	0.3364	1.1633	0.3370
RegressorRelu6	128	64	1024, 128	10^{-5}	1.1670	0.3364	1.1802	0.3339
				10^{-6}	1.1685	0.3372	1.1606	0.3365
				—	1.1720	0.3363	1.1537	0.3410
Regressor	128	64	1024, 128	10^{-5}	1.1733	0.3358	1.1878	0.3318
Without features—Best Accuracy								
Classifier	128	64	1024, 128	10^{-5}	1.3126	0.3528	1.2825	0.3577
				10^{-6}	1.3127	0.3528	1.2823	0.3576
				—	1.3128	0.3528	1.2822	0.3575

Table 2. Cross-validation and testing metrics of some of the best neural network models trained on the `MovieLens` dataset. The complete logs can be found alongside the project's code at https://github.com/fontaine618/507-Project/.

		CV		Test	
Dimension	Nb. iterations	MSE	Acc.	MSE	Acc.
4	100	0.9533	0.4116	0.8810	0.4333
5	100	0.9774	0.4103	0.9038	0.4306
4	80	0.9804	0.4081	0.8954	0.4298
	70	1.0014	0.4046	0.9075	0.4271
5	80	1.0024	0.4049	0.9206	0.4257
	70	1.0220	0.4016	0.9338	0.4236

Table 3. Cross-validation and testing metrics of some of the best matrix completion (SVD) models trained on the `MovieLens` dataset. The complete logs can be found alongside the project's code at https://github.com/fontaine618/507-Project/.

reuse the training data too many times, the model will tend to prefer the training data and the test error will eventually increase. If k and I are large at the same time, the model will be surely overfitting.

It is amazing that this model only needs a few lines of code but the performance is comparable to what the complicated models like Neural network or Restricted Boltzmann Machine do. The idea behind this method is also innocent and it is computationally efficient (most of them only take less than a second to train in a 4 cores CPU machine).

4.4. Restricted Bolztmann Machine

We trained around 100 instances of RBM model as describes in Section 3.2.4. We choose to present some models with its parameters. From the result, in Table 4, we can see some facts about turning parameters as below.

For the number of hidden nodes, it turns out to be the best when set it around 15-20. If this number is too small, the model will be underfitting, but setting this number to large does not help the model to predict better.

For the number of Gibbs samplings to use in Contrastive Divergence, it is plausible to set it 1, because it does not help to improve the model significantly when it increases, but slow down the learning phase much (we need to do sampling for every (movie, user) couple.)

It is claimed in Salakhutdinov, Mnih and Hinton (2007) that RBM paper can outperform SVD models in a fine-tuned parameters scenario. However, we observe something contrary to that here. It may due to the fact that the research team at the University of Toronto spent several years to achieve the expertise in training RBM models. In the time to the final report, we will keep trying improve our RBM model and see how far can we go compared to what is presented in that paper.

4.5. Model comparison

Table 5 contains a comparison of the best version of each of the four models we consider in terms of both regression and classification metrics. Overall, matrix completion using

			CV		Test	
Hidden nodes	Gibbs iterations	Learning rate	MSE	Acc.	MSE	Acc.
15	5	1.0	1.0362	0.3609	1.0308	0.3682
	1	1.0	1.0375	0.3607	1.0330	0.3682
20	5	1.0	1.0375	0.3612	1.0320	0.3670
25	5	1.0	1.0394	0.3603	1.0344	0.3669

Table 4. Cross-validation and testing metrics of some of the best restricted Bolztmann machine models trained on the `MovieLens` dataset. The complete logs can be found alongside the project's code at https://github.com/fontaine618/507-Project/.

	CV		Test	
Model	MSE	Acc.	MSE	Acc.
Best regressor				
K-Nearest-Neighbors	0.9898	0.3741	0.9737	0.3767
Neural Network	0.9402	0.3985	0.9016	0.4165
Matrix completion	0.9533	0.4116	0.8810	0.4333
Restricted Boltzmann Machine	1.0362	0.3609	1.0308	0.3682
Best classifier				
K-Nearest-Neighbors	1.2482	0.3935	1.2547	0.3940
Neural Network	0.9402	0.3985	0.9016	0.4165
Matrix completion	1.0360	0.4116	0.9632	0.4333
Restricted Boltzmann Machine	1.1115	0.3609	1.1033	0.3682

Table 5. Cross-validation and testing metrics on the best version of the four models, respectively in terms of regression and classification, trained on the `MovieLens` dataset.

SVD systematically out-performs all other models in MSE and classification accuracy.

The K-NN models perform worse than NN and matrix completion and require much more computing time. This approach does not seem very promising as a predictive model for movie ratings.

The NN models yield interesting results and can be trained efficiently, but they do not achieve the performance level of matrix completion even though more information is included in the model. Indeed, matrix completion does not use movie genre and tags nor user details.

The SVD matrix completion model, it can be seen that this simple model can out-perform many other complicated models. Simon Funk, a participant in the Netflix prize competition (Bennett et al., 2007), used this simple model and finished in third place.[6] Its computational efficiency, compared to all three other models, is also a very appealing feature.

The RBM model, while a beautiful latent variable model, really needs time and expertise to be able to set it up correctly and implemenent efficiently.

[6] https://sifter.org/~simon/journal/20061211.html

5. Final report

For the final report, we will use results from the above model comparison in order to select a final model. Then, we will proceed to an exploratory data analysis using that model in order to extract insight about ratings with respect to user and movie information.

6. Contributions

The work was subdivided as follows: Simon worked on K-NN and NN models, Dat on matrix completion and RBM models. Each implemented, trained and analyzed the respective models; the respective part of the report were written accordingly. The remainder of the work was performed jointly.

References

Bennett, J., Lanning, S. et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop* **2007** 35. Citeseer.

Harper, F. M. and Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* **5** 1–19.

Hinton, G. E. (2012). A practical guide to training restricted Boltzmann machines. In *Neural networks: Tricks of the trade* 599–619. Springer.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. (2017). Automatic differentiation in PyTorch.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12** 2825–2830.

Salakhutdinov, R., Mnih, A. and Hinton, G. (2007). Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th International Conference on Machine Learning. ICML '07* 791–798. Association for Computing Machinery, New York, NY, USA.

van der Walt, S., Colbert, S. C. and Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering* **13** 22-30.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. and Contributors, S. . . (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17** 261–272.