PHASE 2

In phase 2, we develop a recommender system for Netflix using the algorithms we learned in the course.

K-Means and K-Nearest Neighbor

Question 1

In this question, we want you to make a system that predicts the user's preference of a movie on the basis of different parameters. Our approach:

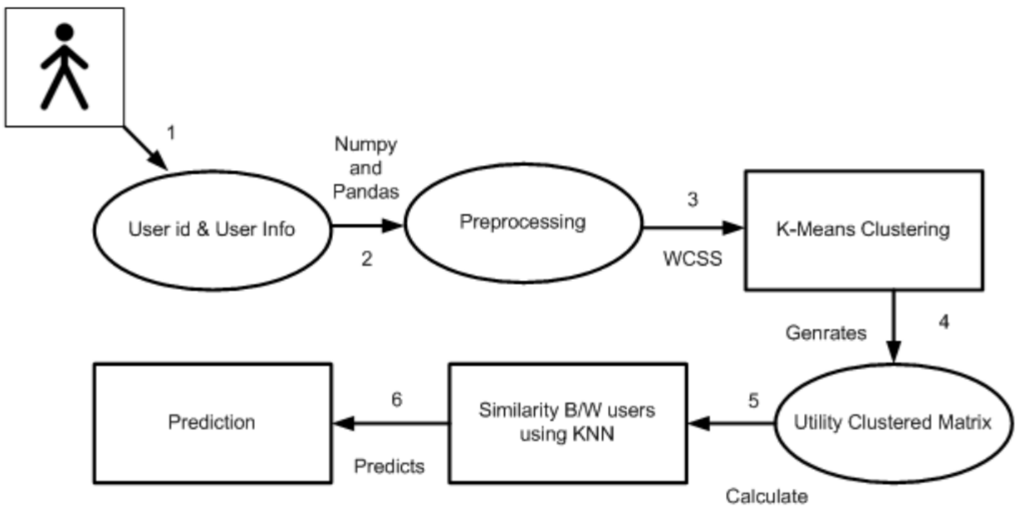


Figure 1: Process flow diagram

Input Module: In this module, the user is asked to give the details as input. In input, the user gives the detail about himself by providing details such as userId, age, gender, pin code. This information is further passed to the next module i.e. the processing module.
Processing Module: In this, the panda's module first separates the data from the raw files. It separates the information about the user and movie items into a separate data frame using the panda's library. After separating the data from the raw form, in a utility matrix module a utility matrix is built which defines which user rated which movie. This helps in figuring out how many times each movie is rated by the users. Then based on previous preprocessing of data, separate data frames for the training set and testing set is created. This is done to further evaluate the performance of the system. After getting the utility matrix, K-means clustering is used to build a separate data frame which shows which movie belongs to which genre. The Within-Cluster Sum of Squares (WCSS) is a measure of the variability of the observations within each cluster. In general, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares. In WCSS module the right no of clusters is chosen using the technique

Question 1 continued on next page...

Within Clustered Sum of Square. Now, for calculating the average rating given by each user given to each cluster, a utility clustered matrix is created. In utility clustered module the utility clustered matrix is used to calculate the similarity between the users. The PCS and normalization module calculates the correlation using the utility clustered matrix. Finally, in the KNN module and similarity module using the K-Nearest Neighbor predictions for movie rating is calculated with the help of the similarity matrix and utility clustered matrix.
Output Module: The output module describes the predicted movies that the input user might like. Further, in the output along with the movies, their predicted ratings are also defined which input user might give to the movies.

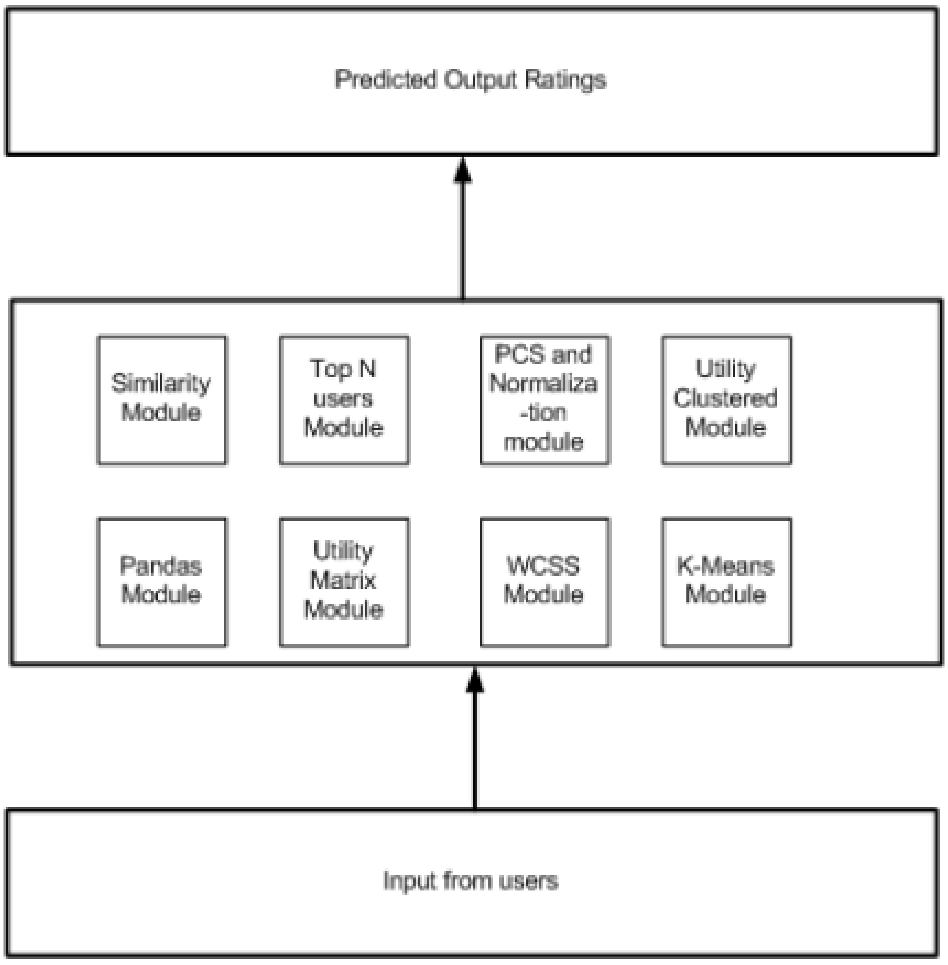


Figure 2: Architecture of the proposed system

Implement the proposed system in python programming language. Use MovieLens 100k data set. Note that if the proposed system uses inputs(or other parts of the proposed system) different from MovieLens 100k data set, you should change the system to work with this data(the same is true for other parts of the project)
For more information, read [Movie Recommender System Using K-Means Clustering AND K-Nearest Neighbor]. To use an optimized KMeans and KNN, you can find other related papers.

SVM

Question 2

Support Vector Machine (SVM) is a widely adopted machine learning method. Compared with other machine learning approaches, SVM has some advantages. It assures that once a solution is reached, it is the global optima. Since the forecasting accuracy of SVM is parameter sensitive, it would be important to choose the kernel function and tune the kernel parameters to achieve the desired performance. Many heuristic algorithms have thus been used for the parameter optimization of SVM, such as the Grid Search (GS), Genetic Algorithm (GA), and Particle Swarm Optimization (PSO). Comparing with other algorithms, PSO is recognized to have merits of strong global search capability and ease of implementation. But the standard PSO also has some demerits. It often pre-matures into the local optimum and has slow convergence speed.
Here, we want you to implement a novel improved PSO algorithm, and then implement a personalized movie recommendation system based on the proposed improved PSO and SVM. In particular, the major goals of this question include:
(1) To overcome the shortcomings of the conventional PSO, proposing an improved PSO with the contraction factor and dynamic adaptively inertia weight(CF-IWA PSO). CF-IWA PSO is embedded with a selfadaptively parameter adjustment mechanism to enhance both the global search ability and convergence speed. Then using the CF-IWA PSO to do the parameter optimization of SVM;
(2) Based on CF-IWA PSO and SVM, proposing a personalized movie recommendation system. Comparing with the traditional CF methods which only use the historical score data to calculate similarity, the proposed system not only utilizes the user's demographic information, but also their rating information. These two kinds of information can well reflect the user's preferences
Select 2000 users' score data from the MovieLens 1M data set as the experimental data set. For each user, randomly select 10 data as testing data, add them to the test data set, and the remaining data are used as the training set.
To establish the personalized movie recommendation model, the user's demographic information, user's behavioral information ("ratings"), and movie's content information are integrated to form a "user-movie" correlation matrix. The correlation matrix is then trained by a training model, and finally the movies are classified (or "recommend"). The proposed PRS performs the movie recommendation based on the classification method instead of the similarity calculation of the traditional CF methods. Before establishing a classification model, movies are divided into two categories: "like" (recommended) and "dislike" (not recommended), based on the users' ratings. Classified the "like" category as the movies with 4 or 5 stars, and the "dislike" category as the movies with 1, 2 or 3 stars. The procedures of building the proposed personalized recommendation model are as below.
"User-Movie" Correlation Feature Extraction: In our movie recommendation system, the relationship between the user and movie is essential for establishing the classification model. Here, based on the MovieLens dataset, use the user's demographic information, movie's information, and user's ratings information about movies to realize the correlation between the user's preference characteristics and movie's information, shown in Fig. 3. There are 3 files in the MovieLens data set: movie.data, ratings.data and users.data. As implied by the file names, these files store the information of movies, users, and users' ratings on movies, respectively. The primary and foreign keys of the 3 data tables provide the correlation relationships of above 3 categories of information. By analyzing the correlation relationships, we extract the users' behavior and their preference information about the movies, and the 'User-Movie' relationship feature vector can be formed.

Question 2 continued on next page...

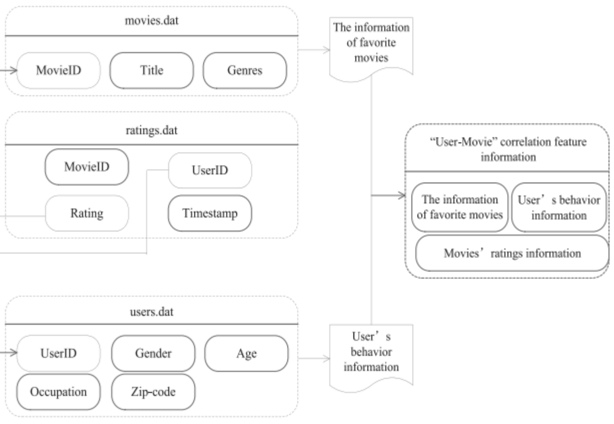

movies.dat
MovieID
Title
Genres
ratings.dat
MovieID
UserID
Rating
Timestamp
users.dat
UserID
Gender
Age
Occupation
Zip-code
The information of favorite movies
"User-Movie" correlation feature information
The information of favorite movies
User's behavior information
Movies' ratings information
User's behavior information

Figure 3: "User-movie" correlation feature extraction

Personalized Movie Recommendation System: As you know, the CF methods have some limitations. The user-based collaborative filtering (UserCF) method needs to calculate the similarity between two users based on the items' rating matrix; the item-based collaborative filtering (ItemCF) needs to calculate the similarity between two items based on the items' rating matrix. The computational complexity of the UserCF is related to the number of users, which is proportional to the square of the number of users. For the ItemCF, when the number of items is large, its computational cost is also very high, which is proportional to the product of the square of the number of items and the sparsity. Comparing with the conventional CF methods, the machine learning based approach can significantly reduce the computational complexity. Moreover, taking into account the user's demographic information can also alleviate the "cold start" problem to a large extent. The personalized movie recommendation model is shown in Fig. 4. First divide the 'User-Movie' relationship matrix (i.e. feature vector) into training data set and testing data set respectively, and perform the feature transformation on each of them. Then, apply the proposed CF-IWA PSO to optimize the parameters of SVM. After that, we train the personalized movie recommendation model based on the SVM classifier and make predictions on the relationships between users and movies. Based on the prediction results, the movie recommendation list can be formed.

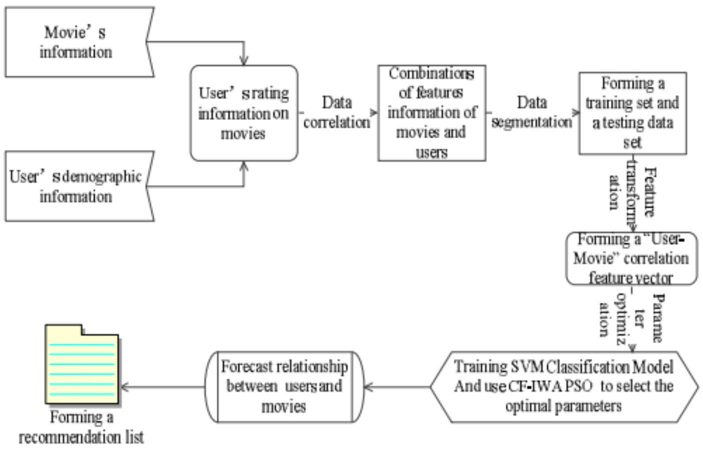

Movie's information
User's demographic information
User's rating information on movies
Data correlation
Combinations of features information of movies and users
Data segmentation
Forming a training set and a testing data set
Feature transformation
Forming a "User-Movie" correlation feature vector
Parameter optimization
Training SVM Classification Model And use CF-IWA PSO to select the optimal parameters
Forecast relationship between users and movies
Forming a recommendation list

Figure 4: Personalized movie recommendation process

Matrix Factorization

We define the problem of collaborative filtering (CF) in the following setting. The problem can be modeled by the random triplet (U,I,R), where:

- U taking values from $\{1, ..., N\}$ is the user identifier,
- I taking values from $\{1, ..., M\}$ is the item identifier,
- R taking values from $X \subset R$ is the rating value.

A realization of (U,I,R) denoted by (u,i,r) means that user u rated item i with value r. The goal is to estimate R from (U,I) such that the root mean squared error of the estimate,

$$RMSE = \sqrt{\mathbb{E}\{(\widehat{R} - R)^2\}}$$

is minimal, where $\widehat{R}$ is the estimate of R.
In practice, the distribution of (U, I, R) is not known, we are only given a finite sample, $T' = \{(u_1, i_1, r_1), (u_2, i_2, r_2), ..., (u_n, i_n, r_n)\}$ generated by it. The sample T' can be used for training predictors. We assume "sampling without replacement" in the sense that (user ID, item ID) pairs are unique in the sample, which means that users do not rate items more than once. Let us introduce the notation $T = \{(u, i) : \exists r : (u, i, r) \in T'\}$ for the set of (user ID, item ID) pairs. Note that $|T'| = |T|$, and typically $|T| \ll N \cdot M$, because most of the users, rate only a few items. Denote the set of items rated by the u-th user by $T_u = \{i : (u, i) \in T\}$. Denote the set of users who rated the i-th item by $T(i) = \{u : (u, i) \in T\}$. The sample can be represented as a partially specified matrix denoted by **R** $\in \mathbb{R}^{(N \times M)}$, where the matrix elements are known in positions $(u, i) \in T$, and unknown in positions $(u, i) \in T$. The value of the matrix **R** at position $(u, i) \in T$, denoted by r_{ui}, stores the rating of user u for item i. For clarity, we use the term (u,i)-th rating in general for r_{ui}, and (u,i)-th training example if $r_{ui} : (u, i) \in T$. When we predict a given rating r_{ui} by $\widehat{r_{ul}}$ we refer to the user u as active user, and to the item i as active item. The (u,i) pair of active user and active item is termed query.
Now that we have formulized our problem, it's time for us to introduce a solution for it.
Matrix Factorization
Matrix factorization (MF) is one of the most often applied techniques for CF problems. The idea behind MF techniques is very simple. Suppose we want to approximate the matrix **R** as the product of two matrices:

$$\mathbf{R} \approx \mathbf{PQ}$$

where **P** is an $N \times K$ and **Q** is a $K \times M$ matrix. This factorization gives a low dimensional numerical representation of both users and items. Note, that **Q** and **P** typically contain real numbers, even when **R** contains only integers. In the case of the given problem, the unknown ratings of **R** cannot be represented by zero. For this case, the approximation task can be defined as follows. Let $\mathbf{p}_{uk}$ denote the elements of **P** $\in \mathbb{R}^{(N \times K)}$, and $\mathbf{q}_{ki}$ the elements of **Q** $\in \mathbb{R}^{(K \times M)}$. Let p_u^T denote the transpose of the u-th row of **P**, and q_i the i-th column of **Q**. Then:

$$\widehat{r_{ul}} = \sum_{k=1}^{K} p_{uk} q_{ki} = \mathbf{p}_u^T \mathbf{q}_i$$

$$e_{ui} = r_{ui} - \widehat{r_{ul}} \ \ for \ (u, i) \in T$$

$$SSE = \sum_{(u,i)\in T} e_{ui}^2 \ , \quad RMSE = \sqrt{\frac{SSE}{|T|}}$$

$$(\mathbf{P}^*, \mathbf{Q}^*) = argmin_{\mathbf{P},\mathbf{Q}} SSE$$

Question 3

Using the above formulation, implement the code corresponding to finding the matrices **P** and **Q** for the given data set using gradient descent. (Note to partition your data into training and validation sets.) After finding the corresponding matrices, construct the matrix **R** and justify the recommendation of the algorithm for some users at random using your own explanation.

Question 4

Now, in order to avoid overfitting, consider the regularization term in the error function as below:

$$e'_{ui} = \frac{1}{2}(e_{ui}^2 + \lambda \mathbf{p}_u^T \mathbf{p}_u + \lambda \mathbf{q}_i^T \mathbf{q}_i)$$

$$SSE' = \sum_{(u,i)\in T} e'_{ui}$$

Find the corresponding formula for the updating the parameters for the new error function.

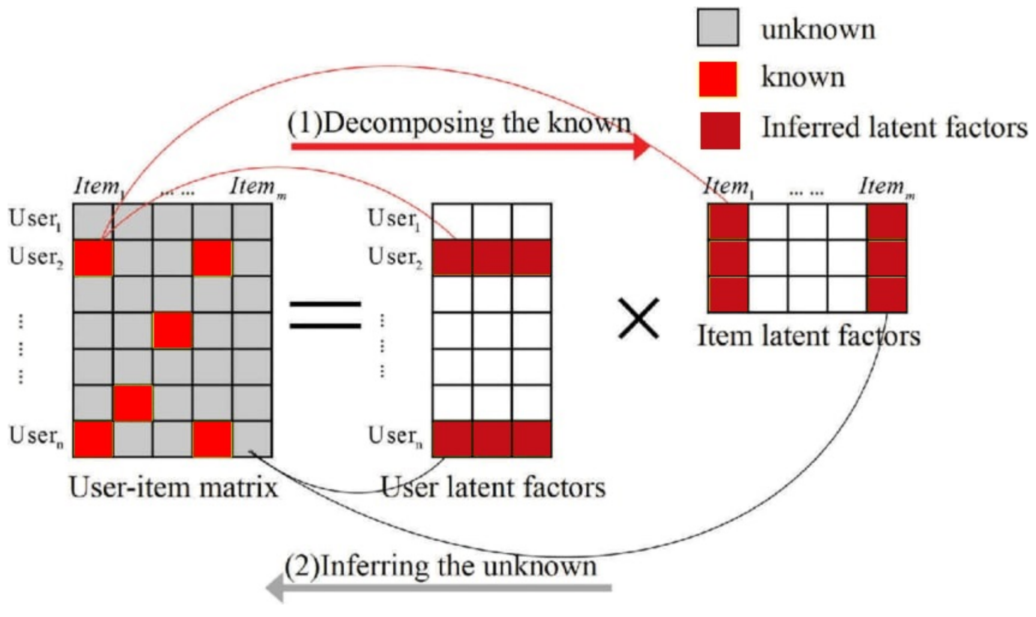


Figure 5: Matrix Factorization

Question 5

Implement the regularized matrix factorization algorithm using the derived formulas and test it on the given dataset. (Note to partition your data into training and validation sets, in order to find a good value for the hyper-parameter of learning rate.) After finding the corresponding matrices, construct the matrix **R** and justify the recommendation of the algorithm for some users at random using your own explanation. Compare the result with the previous algorithm.

Question 6

Now we would extend our algorithm for better performance. The presented MF (Matrix Factorization) has 4 important meta parameters: λ, η, and the initial value of **P** and **Q**, which we denote by $\mathbf{P}_0$ and $\mathbf{Q}_0$ resp. We can get the proper setting of these parameters via trial by error or a parameter optimization algorithm to maximize the performance of a recommender system.
If we introduce more parameters, then we are able to get better performance, but the task of setting these parameters will become harder. In the followings, we introduce many-many parameters to get a very general method, then we present some special cases that will have advantageous properties.
Let p_0^{uk} and q_0^{kj} denote the elements of $\mathbf{P}_0$ and $\mathbf{Q}_0$ resp. We replace λ (regularization factor) and η (learning rate) with λ_p (u,i,k), λ_q (u,i,k), η_p (u,i,k) and η_q (u,i,k) resp.
Find the corresponding formulation for the parameter updates for the above definition using gradient descent.

Question 7

One of the suggested ways to improve the previous algorithm, is to introduce a bias feature per user and per item for the algorithm. Here's the special setting for this improvement.

- Let $p_{.1}^0$. Let the rest of $\mathbf{P}_0$ be small random numbers.
- Let $q_{2.}^0$. Let the rest of $\mathbf{Q}_0$ be small random numbers.
- Let $\eta^p(u,i,1) = 0$. Let $\eta^p(u,i,2) = \eta^{pb}$
  Let $\eta^p(u,i,k) = \eta^p$ for $k > 2$
- Let $\lambda^p(u,i,1) = 0$. Let $\lambda^p(u,i,2) = \lambda^{pb}$
  Let $\lambda^p(u,i,k) = \lambda^p$ for $k > 2$
- Let $\eta^q(u,i,2) = 0$. Let $\eta^q(u,i,1) = \eta^{qb}$
  Let $\eta^q(u,i,k) = \eta^q$ for $k > 2$
- Let $\lambda^q(u,i,2) = 0$. Let $\lambda^q(u,i,1) = \lambda^{qb}$
  Let $\lambda^q(u,i,k) = \lambda^q$ for $k > 2$

These settings mean that we set the first feature of each user and the second feature of each item to 1, and never change these values. We have different learning rate and regularization for users and items, and for bias and non-bias features. In short, this algorithm has the following meta parameters:$\eta^p, \eta^{pb}, \lambda^p, \lambda^{pb}, \eta^q, \eta^{qb}, \lambda^q, \lambda^{qb}$, and two random variables to initialize $\mathbf{P}_0$ and $\mathbf{Q}_0$.
By implementing this setting, analyze the convergence rate and accuracy of it on the give dataset. After finding the corresponding matrixes, construct the matrix **R** and justify the recommendation of the algorithm for some users at random using your own explanation.

Question 8 continued on next page...

Question 8

Now, by introducing another extension of the previous algorithm in the form of a different regularization, we try to improve the accuracy and convergence rate of the algorithm.
In second part algorithm, the regularization of user and item feature vectors is done per rating. Thus, the weight penalization of a user's or item's feature vector is proportional to the number of ratings of that user or item resp.
However, a CF problem may require that the features of users or items with many ratings should be penalized less. To achieve this goal, we introduce an extension of the previous algorithm by specializing it in the following way: We replace regularization and learning rate parameters with functions with 7 parameters such that the value of those functions depends only on the number of ratings of active user and item. Let:

$$f(u, i, [p_1, p_2, p_3, p_4, p_5, p_6, p_7]) = p_1 + \frac{p_2}{log(|T_u|)} + \frac{p_3}{\sqrt{|T_u|}} + \frac{p_4}{|T_u|} + \frac{p_5}{log(|T^{(i)}|)} + \frac{p_6}{\sqrt{|T^{(i)}|}} + \frac{p_7}{|T^{(i)}|}$$

The corresponding setting for this extension would be:

- Let $p^0_{\cdot\cdot}$. Let the rest of $\mathbf{P}_0$ be small random numbers.
- Let $q^0_{2\cdot}$. Let the rest of $\mathbf{Q}_0$ be small random numbers.
- Let $\eta^p(u, i, 1) = 0$. Let $\eta^p(u, i, 2) = f(u, i, [\eta^{pb}_1, \eta^{pb}_2, \eta^{pb}_3, \eta^{pb}_4, \eta^{pb}_5, \eta^{pb}_6, \eta^{pb}_7])$
  Let $\eta^p(u, i, k) = f(u, i, [\eta^p_1, \eta^p_2, \eta^p_3, \eta^p_4, \eta^p_5, \eta^p_6, \eta^p_7])$ for $k > 2$
- Let $\lambda^p(u, i, 1) = 0$. Let $\lambda^p(u, i, 2) = f(u, i, [\lambda^{pb}_1, \lambda^{pb}_2, \lambda^{pb}_3, \lambda^{pb}_4, \lambda^{pb}_5, \lambda^{pb}_6, \lambda^{pb}_7])$
  Let $\lambda^p(u, i, k) = f(u, i, [\lambda^p_1, \lambda^p_2, \lambda^p_3, \lambda^p_4, \lambda^p_5, \lambda^p_6, \lambda^p_7])$ for $k > 2$
- Let $\eta^q(u, i, 2) = 0$. Let $\eta^q(u, i, 1) = f(u, i, [\eta^{qb}_1, \eta^{qb}_2, \eta^{qb}_3, \eta^{qb}_4, \eta^{qb}_5, \eta^{qb}_6, \eta^{qb}_7])$
  Let $\eta^q(u, i, k) = f(u, i, [\eta^q_1, \eta^q_2, \eta^q_3, \eta^q_4, \eta^q_5, \eta^q_6, \eta^q_7])$ for $k > 2$
- Let $\lambda^q(u, i, 2) = 0$. Let $\lambda^q(u, i, 1) = f(u, i, [\lambda^{qb}_1, \lambda^{qb}_2, \lambda^{qb}_3, \lambda^{qb}_4, \lambda^{qb}_5, \lambda^{qb}_6, \lambda^{qb}_7])$
  Let $\lambda^q(u, i, k) = f(u, i, [\lambda^q_1, \lambda^q_2, \lambda^q_3, \lambda^q_4, \lambda^q_5, \lambda^q_6, \lambda^q_7])$ for $k > 2$

Now we have 7 times as many parameters as previous algorithm. Implement this algorithm and check its performance in terms of convergence rate and accuracy. Like the previous parts, after finding the corresponding matrices, construct the matrix **R** and justify the recommendation of the algorithm for some users at random using your own explanation.

Neural Networks

In this approach, we change the formulation of collaborative filtering a little. So, here's the collaborative filtering problem.
Let M and N denote the number of users and items, respectively. We define the user–item interaction matrix $Y \in \mathbb{R}^{M\times N}$ from users' implicit feedback as,

$$y_{ui} = \begin{cases} 1 & if\ interaction\ (user\ u,\ item\ i)\ is\ observed \\ 0 & otherwise \end{cases} \tag{1}$$

Here a value of 1 for y_{ui} indicates that there is an interaction between user u and item i; however, it does not mean u actually likes i. Similarly, a value of 0 does not necessarily mean u does not like i, it can be that the user is not aware of the item. This poses challenges in learning from implicit data, since it provides only noisy signals about users' preference. While observed entries at least reflect users' interest on items, the unobserved entries can be just missing data and there is a natural scarcity of negative feedback.
The recommendation problem with implicit feedback is formulated as the problem of estimating the scores of unobserved entries in **Y**, which are used for ranking the items. Model-based approaches assume that data can be generated (or described) by an underlying model. Formally, they can be abstracted as learning $\widehat{y_{ul}} = f(u, i|\theta)$ where y_{ui} denotes the predicted score of interaction y_{ui}, θ denotes model parameters, and f denotes the function that maps model parameters to the predicted score (which we term as an interaction function). For example, in the previous part, we explained the matrix factorization (MF) technique where the interaction function is defined as:

$$\widehat{y_{ul}} = f(u, i|\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u^T\mathbf{q}_i = \sum_{k=1}^{K} p_{ui}q_{ik}$$

Where $\mathbf{p}_u$ and $\mathbf{q}_i$ denote the latent vector for user u and item i, respectively and K denotes the dimension of the latent space.
To estimate parameters θ, existing approaches generally follow the machine learning paradigm that optimizes an objective function. Two types of objective functions are most commonly used – pointwise loss and pairwise loss. As a natural extension of abundant work on explicit feedback, methods on pointwise learning usually follow a regression framework by minimizing the squared loss between $\widehat{y_{ul}}$ and its target value y_{ui}. To handle the absence of negative data, they have either treated all unobserved entries as negative feedback, or sampled negative instances from unobserved entries. For pairwise learning, the idea is that observed entries should be ranked higher than the unobserved ones. As such, instead of minimizing the loss between $\widehat{y_{ul}}$ and y_{ui}, pairwise learning maximizes the margin between observed entry $\widehat{y_{ul}}$ and unobserved entry $\widehat{y_{ul}}$
Here, our NCF (Neural Collaborative Filtering) framework parameterizes the interaction function f using neural networks to estimate $\widehat{y_{ul}}$. As such, it naturally supports both pointwise and pairwise learning.

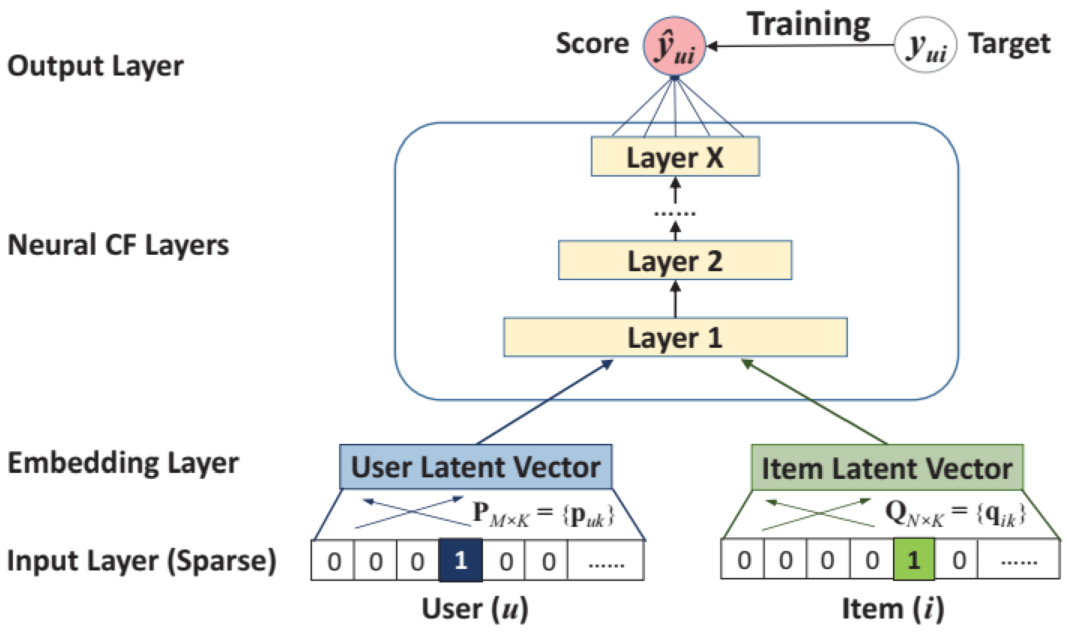


Figure 6: Neural collaborative filtering framework

Neural Collaborative Filtering We first present the general NCF framework, elaborating how to learn NCF with a probabilistic model that emphasizes the binary property of implicit data. We then show that MF can be expressed and generalized under NCF. To explore DNNs for collaborative filtering, we then propose an instantiation of NCF, using a multi-layer perceptron (MLP) to learn the user item interaction function. Lastly, we present a new neural matrix factorization model, which ensembles MF and MLP under the NCF framework; it unifies the strengths of linearity of MF and non-linearity of MLP for modelling the user–item latent structures.
To permit a full neural treatment of collaborative filtering, we adopt a multi-layer representation to model a user–item interaction y_{ui} as shown in Figure 6, where the output of one layer serves as the input of the next one. The bottom input layer consists of two feature vectors $\mathbf{v}_u^U$ and $\mathbf{v}_i^I$ that describe user u and item i, respectively; they can be customized to support a wide range of modelling of users and items, such as context-aware, content-based, and neighbor-based. Since this work focuses on the pure collaborative filtering setting, we use only the identity of a user and an item as the input feature, transforming it to a binarized sparse vector with one-hot encoding. Note that with such a generic feature representation for inputs, our method can be easily adjusted to address the cold-start problem by using content features to represent users and items.
Above the input layer is the embedding layer; it is a fully connected layer that projects the sparse representation to a dense vector. The obtained user (item) embedding can be seen as the latent vector for user (item) in the context of latent factor model. The user embedding and item embedding are then fed into a multi-layer neural architecture, which we term as neural collaborative filtering layers, to map the latent vectors to prediction scores. Each layer of the neural CF layers can be customized to discover certain latent structures of user–item interactions. The dimension of the last hidden layer X determines the model's capability. The final output layer is the predicted score $\widehat{y_{ul}}$ and training is performed by minimizing the pointwise loss between $\widehat{y_{ul}}$ and its target value y_{ui}.
We now formulate the NCF's predictive model as:

$$\widehat{y_{ul}} = f(\mathbf{P}^T\mathbf{v}_u^U, \mathbf{Q}^T\mathbf{v}_i^I|\mathbf{P}, \mathbf{Q}, \theta_f)$$

where $\mathbf{P} \in \mathbb{R}^{M \times K}$ and $\mathbf{Q} \in \mathbb{R}^{N \times K}$, denoting the latent factor matrix for users and items, respectively; and θ_f denotes the model parameters of the interaction function f. Since the function f is defined as a multi-layer neural network, it can be formulated as:

$$f(\mathbf{P}^T \mathbf{v}_u^U, \mathbf{Q}^T \mathbf{v}_i^I) = \phi_{out}(\phi_X(...\phi_2(\phi_1(\mathbf{P}^T \mathbf{v}_u^U, \mathbf{Q}^T \mathbf{v}_i^I))...))$$

where ϕ_{out} and ϕ_X respectively denote the mapping function for the output layer and x-th neural collaborative filtering (CF) layer, and there are X neural CF layers in total.
To learn model parameters, existing pointwise methods largely perform a regression with squared loss:

$$L_{sqr} = \sum_{(u,i) \in \mathbf{y} \cup \mathbf{y}^-} w_{ui}(y_{ui} - \widehat{y_{ul}})^2$$

where $\mathbf{y}$ denotes the set of observed interactions in $\mathbf{Y}$, and $\mathbf{y}^-$ denotes the set of negative instances, which can be all (or sampled from) unobserved interactions; and w_{ui} is a hyperparameter denoting the weight of training instance (u,i)

Question 9

Try to implement an MLP network using the squared loss above and check its performance in terms of different metrics. Also, try to choose some user at random and explain the recommended options with rationality. (Don't forget to split the data by validation and training).

Question 10

While the squared loss can be explained by assuming that observations are generated from a Gaussian distribution, we point out that it may not tally well with implicit data. This is because for implicit data, the target value y_{ui} is a binarized 1 or 0 denoting whether u has interacted with i. In what follows, we present a probabilistic approach for learning the pointwise NCF that pays special attention to the binary property of implicit data.
Considering the one-class nature of implicit feedback, we can view the value of y_{ui} as a label 1 means item i is relevant to u, and 0 otherwise. The prediction score $\widehat{y_{ul}}$ then represents how likely i is relevant to u.

Question 11

Try to implement the probabilistic approach above using an MLP network. Explain in detail the architecture of your network and the activation function that you have used. Try to use a variety of options for different part of you design such activation function to achieve the best results. Evaluate your model like the previous part. Now, we approach the original problem of matrix completion in a different way. In an equivalent picture, matrix completion or recommendation can be cast as a link prediction problem on a bipartite user-item interaction graph. More precisely, the interaction data can be represented by an undirected graph G=(W,E,R) with entities consisting of a collection of user nodes $u_i \in U$ with $i \in \{1, ..., N_u\}$, and item nodes $v_j \in V$ with $j \in \{1, ..., N_v\}$, such that $U \cup V = W$. The edges $(u_i, r, v_j) \in E$ carry labels that represent ordinal rating levels, such as $r \in \{1, ..., R\} = R$, One of the methods to tackle the link prediction problem, is graph auto-encoders. Here, we make use of a setup that makes efficient use of convolutional weight sharing and allows for inclusion of side information in the form of node features. Graph autoencoders are comprised of 1) a graph encoder model $Z = f(X, A)$, which take as input an $N \times D$

feature matrix X and a graph adjacency matrix A, and produce an $N \times E$ E node embedding matrix $Z = [z_1^T, ..., z_N^T]^T$, and 2) a pairwise decoder model $\widehat{A} = g(Z)$, which takes pairs of node embeddings z_i, z_j and predicts respective entries $\widehat{A}_{ij}$ in the adjacency matrix. Note that N denotes the number of nodes, D the number of input features, and E the embedding size.
For bipartite recommender graphs $G = (W, E, R)$, we can reformulate the encoder as $[U, V] = f(X, M_1, ..., M_R)$, where $M_r \in \{0,1\}^{N_u \times N_v}$ is the adjacency matrix associated with rating type $r \in R$, such that M_r, contains 1's for those elements for which the original rating matrix M contains observed ratings with value r. U and V are now matrices of user and item embeddings with shape $N_u \times E$ and $N_v \times E$, respectively. A single user (item) embedding takes the form of a real-valued vector $U_{i,:}(V_{j,:})$ for user i (item j).

Analogously, we can reformulate the decoder as $\hat{M} = g(U, V)$, i.e., as a function acting on the user and item embeddings and returning a (reconstructed) rating matrix $\hat{M}$ of shape $Nu \times Nv$. We can train this graph auto-encoder by minimizing the reconstruction error between the predicted ratings in $\hat{M}$ and the observed ground-truth ratings in M. Examples of metrics for the reconstruction error are the root mean square error, or the cross entropy when treating the rating levels as different classes. The described process is showing in the following figure. In what follows, we present the architecture of the encoder and the decoder. It's also shown in the next figure as well.

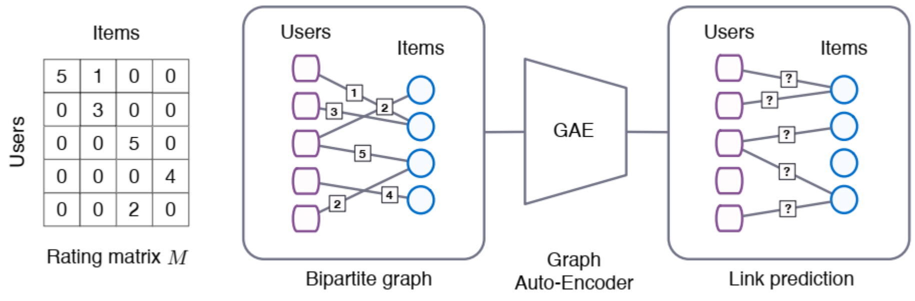


Figure 7: Graph Auto-Encoder for link prediction

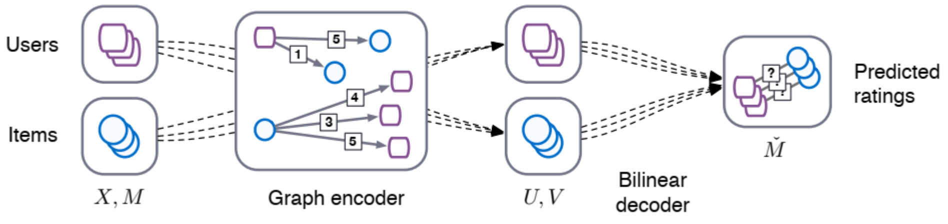


Figure 8: Structure of the graph Auto-Encoder

Graph convolutional encoder
In what follows, we propose a particular choice of encoder model that makes efficient use of weight sharing across locations in the graph and that assigns separate processing channels for each edge type (or rating type) $r \in R$. This type of local graph convolution can be seen as a form of message passing, where vector valued messages are being passed and transformed across edges of the graph. In our case, we can assign a specific transformation for each rating level,

resulting in edge-type specific messages $\mu_{j\rightarrow i,r}$ from items j to users i of the following form:

$$\mu_{j\rightarrow i,r} = \frac{1}{c_{ij}} W_r x_j$$

Here, c_{ij} is a normalization constant, which we choose to either be $|N_i|$ (left normalization) or $\sqrt{(|N_i||N_j|)}$ (symmetric normalization) with N_i denoting the set of neighbors of node i. W_r is an edge-type specific parameter matrix and x_j is the (initial) feature vector of node j. Messages $\mu_(j \rightarrow i, r)$ from users to items are processed in an analogous way. After the message passing step, we accumulate incoming messages at every node by summing over all neighbors $N_{i,r}$ under a specific edge-type r, and by subsequently accumulating them into a single vector representation:

$$h_i = \sigma\left[accum\Big(\sum_{j\in N_{i,1}} (\mu_{j\rightarrow i,1}), ..., \sum_{j\in N_{i,R}} (\mu_{j\rightarrow i,R})\Big)\right] \quad (2)$$

$$u_i = \sigma(W h_i) \quad (3)$$

where $accum(.)$ denotes an accumulation operation, such as $stack(.)$, i.e., a concatenation of vectors (or matrices along their first dimension), or $sum(.)$, i.e., summation of all messages. $\sigma(.)$ denotes an element-wise activation function such as the $ReLU(.) = max(0, .)$. To arrive at the final embedding of user node i, we transform the intermediate output h_i as follows: The item embedding vi is calculated analogously with the same parameter matrix W. In the presence of user- and item-specific side information we use separate parameter matrices for user and item embeddings. We will refer to (2) as a graph convolution layer and to (3) as a dense layer. **Bilinear decoder**

For reconstructing links in the bipartite interaction graph we consider a bilinear decoder, and treat each rating level as a separate class. Indicating the reconstructed rating between user i and item j with $\hat{M_{ij}}$, the decoder produces a probability distribution over possible rating levels through a bilinear operation followed by the application of a soft-max function:

$$p\big(\hat{M}_{ij} = r\big) = \frac{e^{u_i^T Q_r v_j}}{\Sigma_{s\in R} e^{u_i^T Q_s v_j}}$$

with Q_r a trainable parameter matrix of shape $E \times E$, and E the dimensionality of hidden user (item) representations $u_i(v_j)$. The predicted rating is computed as

$$\hat{M}_{ij} = g(u_i, v_j) = \mathbb{E}_{p(\hat{M}_{ij}=r)}[r] = \sum_{r\in R} (rp(\hat{M}_{ij} = r))$$

Question 12

Using the notes above, train the Auto-Encoder. Consider the loss function as bellow.

$$L = -\sum_{i,j;\Omega_{i,j}=1} \sum_{r=1}^{R} I[r = M_{ij}] log\big(p(\hat{M}_{ij} = r)\big)$$

with $I[k = l] = 1$ when $k = l$ and zero otherwise. The matrix $\Omega \in \{0, 1\}^{N_u \times N_v}$ serves as a mask for unobserved ratings, such that one's occur for elements corresponding to observed ratings in M, and zeros for unobserved ratings. Hence, we only optimize over observed ratings. For further information for designing the network, refer to Graph Convolutional Matrix Completion which is in the files given to you.

EASE (Embarrassingly Shallow Autoencoder)

In the previous sections, you were introduced to matrix factorization. The aim of matrix factorization is to find features that explain the user-movie interactions (the rating given to a movie by a particular user). These features are calculated by trying to reconstruct the user-movie interactions matrix X, where each row of X corresponds to a user and each column to a movie. If the user i hasn't watched the movie j, then the ij-th element of the matrix X will be 0.
The user-movie interactions matrix can be reconstructed by other methods. Suppose we know a similarity score for each possible pair of movies. These similarity scores are stored in an M*M matrix named B, where M is the total number of movies.

Question 13

Explain why $S = XB$, where S is a U by M matrix (U is the total number of users), can be used to suggest movies to users (hint: each row in the matrix S corresponds to a user. The similarity scores for a given movie in the matrix B (corresponding to a row of B) are multiplied by the score given to that movie by the user, and this is done for all movies. The resulting rows are then summed together)

Question 14

The aim of the EASE algorithm is to reconstruct the B matrix. B is calculated in a way to satisfy the following goals:

- $S = XB$ should be as close to X as possible.
- The elements of B shouldn't grow too large, and they should model meaningful relationships between movies.

Explain why a matrix B that satisfies the goals above is "good" for our purposes.

Question 15

The goals above are neatly summed up as the following minimization problem:

$$\begin{aligned} \min_{B} \quad & ||X - XB||_F^2 + \lambda \cdot ||B||_F^2 \\ \text{s.t.} \quad & \text{diag}(B) = 0 \end{aligned}$$

Where $||.||_F$ denotes the frobenius norm of a matrix (square root of the sum of the absolute squares of its elements), and the constraint $diag(B) = 0$ requires the diagonal elements of B to be 0. Briefly explain why the constraint $diag(B) = 0$ is important (hint: B=I might satisfy the requirements of the previous part (I is the identity matrix), but it's not a good solution, as $S = XB$ results in movie suggestions that a user has already watched!).

Question 16 continued on next page...

Question 16

Solve the optimization problem above using the lagrangian multipliers method and derive the following expression for B:

$$\hat{B}_{i,j} = \begin{cases} 0 & \text{if } i = j \\ -\frac{\hat{P}_{ij}}{\hat{P}_{jj}} & \text{otherwise.} \end{cases}$$

Where $\hat{P} \triangleq (X^T X + \lambda I)^{-1}$ (hint: the Frobenius norm of a matrix A can be written as $tr(A^T A)$, and $\frac{\partial tr(A^T A)}{\partial A} = 2A$. Use these facts to differentiate the Lagrangian with respect to the matrix B and set it to 0, and find B as a function of X,λ, and the lagrangian multipliers. Then, use the equality constraint.)

Question 17

Download the 1M MovieLens dataset from MovieLens 1M Dataset | GroupLens. Use the ids of the users and movies along with the movie ratings to create the matrix X described in the previous parts. Drop the information of the users with less than 20 submitted ratings.

Question 18

Randomly select 80% of the users for training the model and keep the other 20% for testing.

Question 19

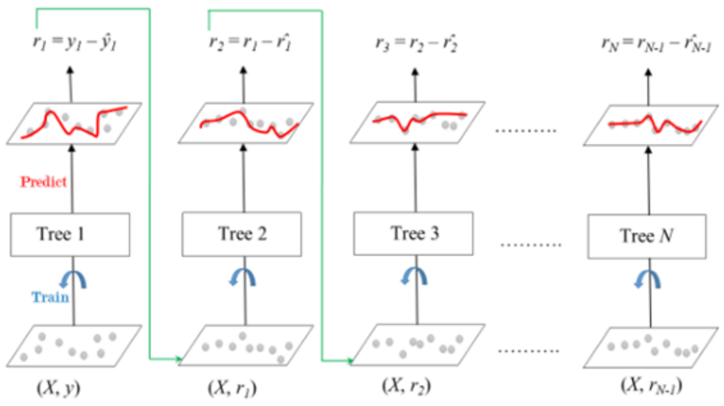
Use 5-fold cross-validation on the training data to find the optimal value for λ. Cross-validation here is done by shuffling the users, then selecting the first 20% of the users for testing and the rest for training. This is repeated for the next 20% and so on. Find the value of λ that results in the best cross-validation score. (important: you are only allowed to use the data of the 80% of the users you selected in the previous part for cross-validation and finding the best value of λ)

Question 20

Train the model using the value of lambda you calculated in the previous part and test it on the test split. Report the R^2 score of your model.

Gradient Boosting

(see Gradient boosting - Wikipedia, A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning - MachineLearningMastery.com, XGBoost - Wikipedia for more information about this section)
Gradient boosting is an ensemble machine learning algorithm. It aims to minimize the loss function by sequentially adding more models to the ensemble, which is in contrast to typical ensemble models that use voting on parallel models. These models are constrained to be very simple to avoid overfitting. A common model used in gradient boosting is a decision stump, which is a one-level binary decision tree.



More information on the algorithm: Ensembles are constructed from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting (source: Extreme Gradient Boosting (XGBoost) Ensemble in Python - MachineLearningMastery.com).

See boosted stumps animation to get a visual understanding of what this algorithm does.

Question 21

Repeat steps 1, 2, 3 and 4 of the tree-based recommendations section.

Question 22

Normalize the X matrix using sklearn.preprocessing.StandardScalar() and then apply PCA to the normalized data (Select an appropriate number of components to be used and report this number). See Importance of Feature Scaling — scikit-learn 1.2.0 documentation for an example of this.

Question 23

Install the XGBoost library and use its XGBRegressor model to predict ratings for the test dataset. Use cross-validation to find the best hyperparameters for the model. Report the R^2 score of your model.

Tree-Based Recommendations

(See Decision tree learning - Wikipedia and Regression Trees, Step by Step. Learn how to build regression trees and… | by Ivo Bernardo | Towards Data Science for more information on tree-based methods)

Tree-based methods partition the data into subsets in such a way that the samples in each subset are *similar*, and then fit a model to each subset. Since the data samples in each subset are similar, a simple model like *averaging* works well on each subset for most purposes.
For example, suppose that you are given the data of the movie ratings on a website. For each rating, you have access to the tags of the movie and some information about the user that submitted the rating. Given the information about a specific user and a specific movie, your task is to predict the rating this user would submit for the movie. One way of doing this would be to find ratings for *similar* movie/user pairs.
A reasonable prediction then would be the average of these similar movie/user pair ratings. This is the idea behind a regression tree. Regression trees learn the best splitting points for the features of the data (splitting points partition the data) and predict the output for the new data points according to these splitting points (the best splitting points are the boundaries that result in the highest accuracy increase for the model. This is a greedy algorithm. The boundaries are calculated by recursively finding the best binary partition for each partition of the data).

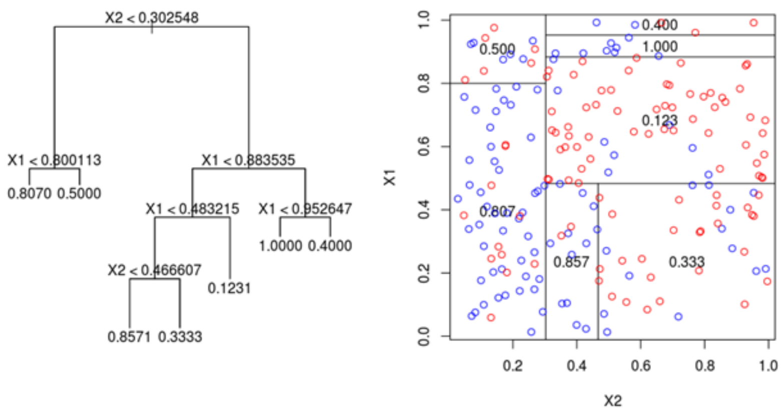


Figure 9: The right plot shows the boundaries for an example dataset. The left diagram shows how the output for a new sample is calculated.

This is similar to how the KNN method works, except for the fact that the KNN calculates distances to all of the training data for each prediction, while the regression tree method learns simple boundaries for the training data and then predicts an output for the data according to these boundaries. Thus, the regression tree method is faster when dealing with fixed training data (because the boundaries are learned once on the training data).

Question 24

Download the 100k-MovieLens dataset from MovieLens 100K Dataset | GroupLens.

Question 25 continued on next page…

Question 25

Use the pandas library to transform the MovieLens dataset into a training data matrix **X** and an output vector **y**, such that each element of y is a rating of the dataset, and the corresponding row of the matrix X contains the information related to the rating. This information includes data of the user that submitted the rating and the data of the rated movie. In other words, each row of X contains information about a sample rating (information about the user and the movie), and each element of y contains a rating.
(hint: use one-hot encoding for the gender and occupation of each user)

Question 26

Shuffle the data and select 80% of the samples as training data and the rest for testing.

Question 27

Features of the dataset require adjustments:

- The ZIP code for each user is provided in the dataset. The first digit of the ZIP code contains geographical information about the region the user is from. You are advised to use categorical encoding (one-hot encoding) on the first digit of the ZIP codes of the users.
- The release date of the movie can be encoded as a signed UNIX timestamp.
- Remove the movies' titles and their IMDB URLs from the features. They require lots of preprocessing to be useful for most classic machine learning models.
- Several features of the data, including the statistical features, might have high predictive power, and extracting them helps decrease error. Some of these statistical features you are advised to use are:
  - Mean, minimum, maximum, and standard deviation of the ratings submitted by a given user
  - Mean, minimum, maximum, and standard deviation of the ratings submitted for a movie

  These features can be augmented as columns of numerical data for each row corresponding to a rating in your training data.

Important: These features should be calculated and extracted separately for the train and test datasets to avoid leakage of the test set information when training the model.

Question 28

Train a regression tree on the training data (apply cross-validation for finding the best hyperparameters of the model). Use the tree.DecisionTreeRegressor model of the sklearn library. Report the R^2 score of your model.

A regression tree model has several important hyperparameters. See this link for tuning your model: Decision Tree Hyperparameters Explained | by Ken Hoffman | Medium