E-Commerce Recommendation System

Ajinkya Pawala , Swapnil Pardeshi , Sumanth Gopalkrishna, Vijay Iyer
Indiana University Bloomington

ABSTRACT

In Recommendation Systems, statistical and knowledge based techniques are used to analyze interactions with customers in order to recommend specific products. With the rapid development of information technology, the information overload problem on e-commerce sites is becoming increasingly serious. People are unable to obtain the desired information quickly from the vast group of items.The goal of recommendation systems is to alleviate the issue of information overload that occurs on e-commerce sites. Using collaborative filtering, a recommendation or prediction is made about a user's interests based on the preferences and tastes of many other users. In this paper, an e-commerce recommendation system using four different approaches into the collaborative filtering algorithm is presented.

1. INTRODUCTION

A surplus of information has been freely available to everyone over the last few decades, making it hard for users to filter through all this information and select the only required elements. Many online electronic-commerce firms recommend products for users, selling millions of products on a single platform. The overwhelming number of choices offered by today's technology can cause information overload for everyday users. Recommender systems aim to address this problem of information overload through accurate, personalized product recommendations.

An online recommendation system predicts if an item will be useful to a customer based on the available information. This has been steadily increasing in popularity within the last few years, where it is used in companies like eBay, Amazon, and malls like Target. These companies collect massive amounts of user data and personalize product recommendations to suit customers' and businesses' needs.

Recommendation systems are broadly classified into two categories- 1. content based filtering and 2. collaborative filtering. Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating).

In this paper, we present the workings of four different models under the collaborative filtering approach of recommendation systems. The four different models are k-nearest neighbors, stochastic gradient descent, alternating least squares and deep learning based neural collaborative filtering. We have used open-source available amazon datasets containing the user and item metadata and we demonstrate how these models work and perform on this given data to recommend new products to customers. Also, we demonstrate how to evaluate the model's performance using different metrics such as hit ratio and mean absolute error. Section 2 presents the prior work done in this field, Section 3 describes the approaches to these models, Section 4 explains the results and the different evaluation metrics, Section 5 explains the future directions in this project and Section 6 is for the conclusion and Section 7 for the references we used for this paper.

2. RELATION TO PRIOR WORK

Neighborhood models: Neighborhood models are the most commonly used for Collaborative Filtering. Their original method, which was shared by previous Collaborative Filtering systems, is user-oriented. These methods predict unknown ratings while depending on the available ratings of similar users. Later a similar item-item approach became popular.

This method predicts an item's rating based on the known ratings made by the same user on similar items. Its improved accuracy and scalability make it more useful for many applications. The fact that items formerly preferred by users are familiar with them in addition to reportedly like minded users provides a good reason behind predictions for item-oriented methods.

Most item-based tasks employ a similarity measure, in which sij stands for the similarity between i and j. Usually, this measure is calculated using a Pearson correlation coefficient or cosine similarity. By comparing the k-items that are most similar to i and rui - the unknown value by user u to item i - we determine those that are most similar, and thus predict the value at hand. The predicted value is calculated by taking the weighted average of the ratings for neighboring items.

Implicit feedback is an issue with all item-oriented models, they do not allow us to differentiate between user preferences and the assurance we have in those preferences.

Latent factor models: Alternatives to Collaborative Filtering, including Latent Factor Models, provide a natural way to characterize latent features that contribute to observed ratings. Examples include pLSA, neural networks, and the Latent Dirichlet Allocation technique. A new class of models based on Singular Value Decomposition (SVD) of the user-item observations matrix has gained popularity as a result of their high accuracy and scalability.

We can take the example of one of the top e-commerce giants in the world right now, Amazon. Amazon started off with item-item collaborative filtering using matrix factorization to give personalized results to the users.[1]

Eventually they incorporated better ML models like deep Learning which they thought would outperform matrix completion methods because they could take care of non-linearity but to their surprise deep learning (state of the art) performed worse than a popular ranked list(simple best seller). Later, Amazon researchers found that using neural networks to recommend movies for amazon prime worked far better when input was sorted into chronological order and used over a short-term period. This worked better because people or users tend to watch movies which have been released recently over old movies which may have higher ratings.

In this paper, we describe our approach which consists of utilizing four different machine learning models in the collaborative filtering approach to recommend items to users using the Amazon dataset. We plan on implementing an ensemble approach of combining these models to achieve better accuracy.

3. APPROACH

3.1 KNN

We can use nearest neighbour algorithms for building recommendation systems. KNN can be the simplest approach. For user based collaborative filtering, we first build a user similarity matrix based on the products rated by the users. Then for the target user we try to find the K similar users. These K users are the nearest neighbours of the target user. The products rated by these top K users is used to build the recommendation list for target users.

3.2 Matrix Factorization

Latent factor models is an approach that tries to explain the ratings by characterizing both items and users on factors inferred from the ratings patterns.Matrix Factorization is one such Latent Factor model. Matrix factorization characterizes both users and items as a vector of factors inferred from the rating patterns.High correspondence between users and item factors leads to a recommendation due to the high rating obtained from the dot product between the user and item vectors[9]. Matrix Factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products. Accordingly, each item i is associated with a vector $q_i$of size f, and each user u is associated with a vector $p_u$of size f. We get the estimated ratings for each user-item interaction by taking dot products of the corresponding vectors.

The idea of estimating ratings for unknown interactions between users and items, is equivalent to finding missing entries in a user-item matrix, with users as rows and items as columns. This is a common problem known as Matrix Completion. This is sometimes referred to as the Netflix problem. We try to represent the matrix in a lower dimensional form, that is, find a low-rank approximation of the matrix. The SVD of a matrix is the best rank-k approximation possible for that matrix. SVD decomposes the matrix into the form USV^T. We can get a rank-k approximation by keeping only the top-k singular vectors of the obtained components, as depicted in below pic[8].

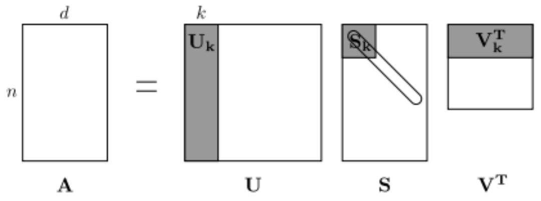


Figure 1: Matrix Factorization

For a matrix A, the proposed rank-k approximation can be written as - $\widehat{A} = \sum_{i=1}^{k} s_i . u_i v_i^T$, where the singular values $s_i$are sorted in descending order($s_1 > s_2 > s_3 > .... s_m$). The rank-k approximation is then - $A_k = U_k S_k V_k^T$. We can think of the above as approximating the raw data A in terms of k 'concepts' (e.g., 'math', 'music' and "sports"), where the singular values of S_k express the signal strengths of these concepts, the rows of V^T and columns of U express the 'canonical row/column' associated with each concept (e.g., a customer that likes only music products, or a product liked only by music customers),and the rows of U (respectively, columns of V^T)approximately express each row (respectively, column) of A as a linear combination
(scaled by S_k) of the 'canonical rows' (respectively, canonical columns). The value $k$is an important parameter, and is pivotal in how closely we can approximate the original matrix.

Although SVD is closely related to our problem, it is not the solution. SVD is undefined for incomplete matrices, where the solution is often to impute the missing entries with some default value and then take SVD to get the components. However, for practical purposes, since this approach works well only with few missing entries[8] and our user-item matrix will be highly sparse, with most users having rated only a few items.Instead of imputing values for the missing entries, we only try to minimize the known ratings. That is we try to minimize the following equation [reference 11]-

$min_{p,q} \sum_{(u,i)\in k} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$, where $r_{ui}$is the known rating user u gives item i. λis the regularization parameter to avoid overfitting, $p_u$and $q_i$are the user and item factors as discussed before. We can think of our user and item matrices $P.Q^T$as similar to the SVD equation USV^T, with the diagonal entries of S dissolved into P and Q. Two approaches to minimizing the above equation are stochastic gradient descent (SGD) and alternating least squares (ALS).

3.2.1 Stochastic Gradient Descent

For stochastic gradient descent, the approach used by Simon Funk[15]. For each training case, we go over each known rating and get the error from the known ratingr_{ui}using $e_{ui} = r_{ui} - q_i^T p_u$. We then update the corresponding user and item vectors p_u and q_i respectively, involved in that rating, using the below equations[9] -

$$q_i = q_i + \gamma(e_{ui} \cdot p_u - \lambda . q_i)$$
$$p_u = p_u + \gamma(e_{ui} \cdot q_i - \lambda . p_u)$$

Above equations can be derived by taking derivative of the objective function defined earlier for minimization,with respect to each p_u and q_i. One epoch corresponds to updating the user and item vectors for all the known ratings in the training set.

3.2.1 Alternating least squares

The optimization problem of finding both sets of vectors p_u and q_i by minimizing the loss function-

$$min_{p,q} \sum_{(u,i)\in k} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

is a non-convex[16]. However, if one set of vectors is considered constant, for example, P, then the objective function for Q is convex, and vice versa. Thus the approach is the fix P and solve for Q, then fix Q and solve for P, until convergence. The following are the update equations -

1. For all users, update corresponding user vectors $$p_u = (\sum_{r_{ui} \varepsilon r_{u*}} q_i q_i^T + \lambda I)^{-1} \sum_{r_{ui} \varepsilon r_{u*}} r_{ui} q_i$$
2. For all items, update corresponding item vectors $$q_i = (\sum_{r_{ui} \varepsilon r_{*i}} p_u^T p_u + \lambda I)^{-1} \sum_{r_{ui} \varepsilon r_{*i}} r_{ui} p_u$$
3. Repeat 1 and 2, till convergence.

Here r_{u*} and r_{*i} are the ratings user u gives for all items and ratings given by users for item i respectively. λterm is for regularization to avoid overfitting.The above equations in step 1 and 2, are similar to Ordinary Least Squares Solution equation when solving for βin the equation $y = X\beta$. ALS can be parallelized as well, as opposed to SGD, as can be seen in the equation, updates to user vector p_u does not depend on other user vectors.

3.3. Neural Collaborative Filtering

Before going to the model architecture of Neural Collaborative filtering, let's look at what embeddings actually are. Let's say we want to represent users according to what they like, in the figure we can see that Bob liked both Shampoo and Toys. Similarly, Alice likes Shampoo but hates toys. Pretty weird example but I hope it makes things easy to understand.

This two dimensional space is known as an embedding. Essentially, the embedding reduces our users such that they can be represented in a meaningful manner in a lower dimensional space. In this embedding, users with similar item preferences are placed near to each other, and vice versa. They are passed as one hot vectors and the outputs that we get are a representation of their likings

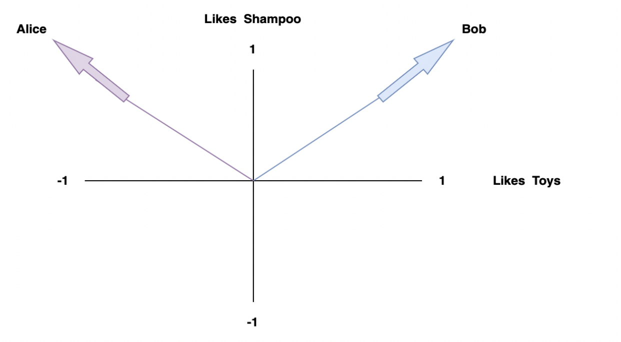


Figure 2: User-Item Embedding

Now coming to the data, we will train the recommender system using implicit feedback. However, the Amazon dataset that we are using is based on explicit feedback. To convert this dataset into an implicit feedback dataset, we'll simply binarize the ratings and convert them to '1' (i.e. positive class). The value of '1' represents that the user has interacted with the item.

It is important to note that using implicit feedback reframes the problem that our recommender is trying to solve. Instead of trying to predict item ratings when using explicit feedback, we are trying to predict whether the user will interact with each item. Now since we just have the data of the items the user has interacted with, we will create the negative samples i.e. 0s by adding 4 negative samples randomly that the user had not interacted with. Even though this is a sweeping assumption that may not be true, it usually works out rather well in practice.

4. EXPERIMENTS AND SETUP

Dataset: We used the Amazon review dataset. This dataset was released by Amazon in 2018 and it is available publicly. The dataset includes reviews (ratings, text, helpfulness votes) and product metadata (descriptions, category information, price, brand, and image features). The data is divided into different subcategories such as electronics / beauty products / software etc. We experimented with different subcategories.

4.1 K Nearest Neighbours

For KNN, we worked with beauty products subcategory. For better results we removed products that are not rated by any of the users and also removed users who have not rated any products. We experimented with different similarity metrics like cosine, euclidean and manhattan ranging on different values of k from 5 to 100 to find the optimum results.

For the evaluation purposes we used hit rate as our metric. From the test instance, we hide some products that are already rated by the user and see if the recommendation list generated for that user can list that hidden product. If it does, then we count it as a hit. And we measure hit rate as the total number of hits divided by total number of users.

4.2 Matrix Factorization

We run our algorithm on 5-core and ratings only subsets found in the amazon reviews datasets page.

The steps followed in our experiments were-
1. Preprocessing step -
    a. In the 5-core dataset links, there are columns in addition to the <uid, iid, rating, timestamp> columns we need - so we remove them.
    b. Some of the 5-core datasets are too small in size (in the order less than 10,000 ratings), so we combine them in some of the runs. This is possible since all dataset subsets have the same columns
    c. Removing duplicate user-item pairs.
    d. We specify 2 values - min_user_rated and min_item_rated during the preprocessing step. min_user_rated is the minimum number of items a user should have rated. If a user has not rated at least these many items, we remove the ratings by that user. After this, step we do a similar step for items having at least as many ratings as min_item_rated
    e. We then split our ratings into training and validation sets. For a recommendation system with a user having rated many items, the validation set will consist of ratings which we know, but will treat as unseen, to evaluate whether our model parameters (P and Q matrices) are generalized. To do this, we want every unique user and every unique item to be present in the training set, so that we can make recommendations for those users and recommend those items to other users.
    f. We group the data frame by users, then for every user, we put a small portion of the ratings to the test set. We optionally do a sorting of all rows by the timestamp column, thus we have a validation set where the items are those about to be rated. We have a fixed validation set size of 0.2. This means of all the items a user has rated, 20% of those will go in the validation set.

2. The next step is to decide the set of hyperparameters. The hyperparameters for our matrix factorization are - no. of factors f, learning rate η, and regularization parameter λ.
3. After deciding the hyperparameter set, we pass both training and validation sets to our SGD or ALS methods. We run the method with all combinations of hyperparameters. Typical learning rate values we experimented with were 0.1, 0.01. Typical regularization rates we experimented with were 0.1, 0.01, 1. Typical factors we experimented with were 2, 5, 10,20, 50.
4. In the SGD or ALS method, we implement early stopping with a best_validation_set value set to whenever we obtained a minimum loss function score. The loss function we used was Mean Absolute Error. We save the model for the best validation error. We can use this saved model later for making recommendations.
5. Once training and validation is over, we can generate estimated rating for any user-item pair, by dot product of p_u and q_i. For making recommendations, we dot this for every q_i for that user p_u, and then return the top-n items with best estimated ratings.

4.3. Neural Collaborative Filtering

The inputs to the model are the one-hot encoded user and item vector. Because this is a positive sample (item actually rated by the user), the true label (interacted) is 1. The user input vector and item input vector are fed to the user embedding and item embedding respectively, which results in a smaller, denser user and item vectors.
The embedded user and item vectors are concatenated before passing through a series of fully connected layers, which maps the concatenated embeddings into a prediction vector as output. At the output layer, we apply a Sigmoid function to obtain the most probable class. In the example below, the most probable class is 1 (positive class), since 0.8 > 0.2.

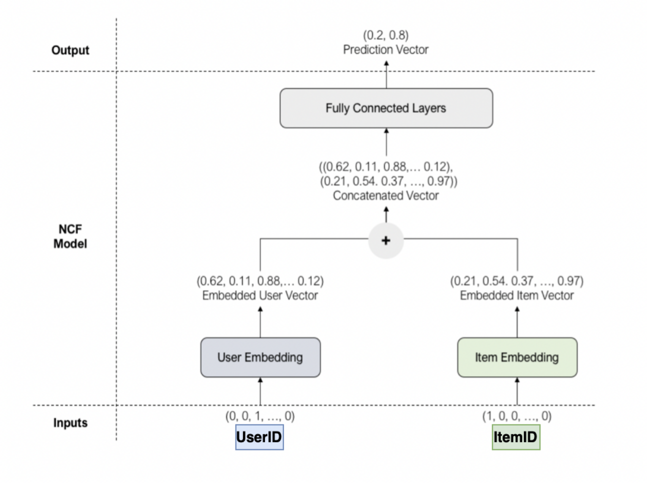
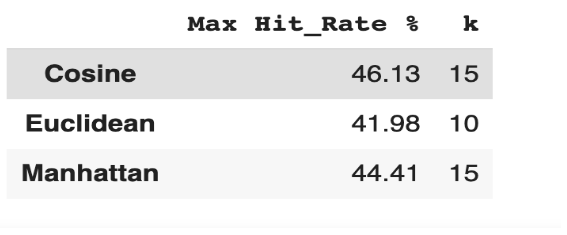


Figure 3: NCF Architecture

5. SUMMARIZE RESULTS

5.1 K Nearest Neighbours



	Max Hit_Rate %	k
Cosine	46.13	15
Euclidean	41.98	10
Manhattan	44.41	15

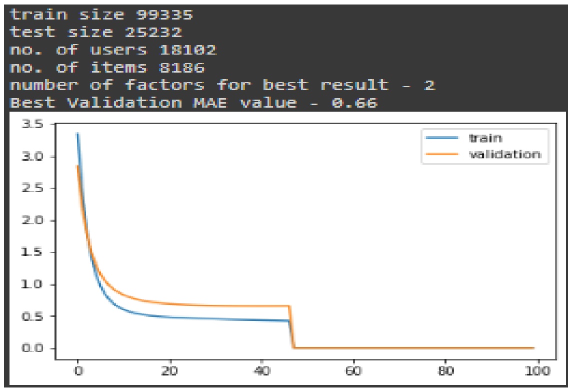
Figure 4: Hit Ratio for KNN

For KNN, the model with cosine similarity and K =15 resulted in maximum hit rate. Among Cosine, Manhattan and Euclidean, cosine gave the best hit rate. We found that hit rate increases with a value of K=15 and then decreases again.

5.2 Matrix factorization

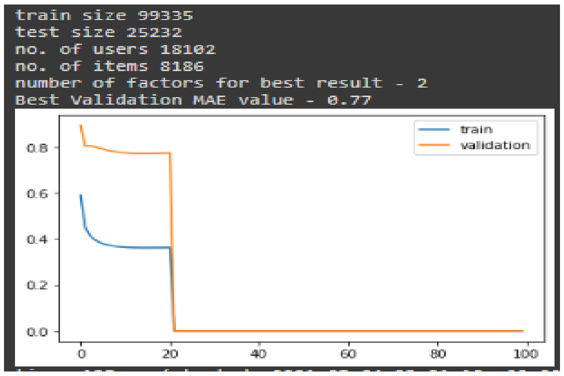
We performed the steps mentioned above for Matrix Factorization on many of the Amazon Reviews dataset. We performed the process described above on single dataset or combination of smaller datasets from the datasets page. We use the metric MAE as the metric for evaluating the saved models. We show few of the results below, in the form of learning curve images as a plot of MAE metric, along with the best MAE values (the epoch where value goes to 0 is the early cutoff epoch). Tabulated results for the same as well as all other datasets can be found in the Matrix Factorization Analysis notebook in the repository. We observe that ALS models overfit with training error reducing very early on, and we had to use large regularization parameter values like 0.1 and 1 for better results.

1. (8 5-core datasets combined)- Amazon Fashion, All Beauty, Appliances, Luxury Beauty, Software, Magazine Subscriptions, Industrial and Scientific, Gift cards



```
train size 99335
test size 25232
no. of users 18102
no. of items 8186
number of factors for best result - 2
Best Validation MAE value - 0.66
```

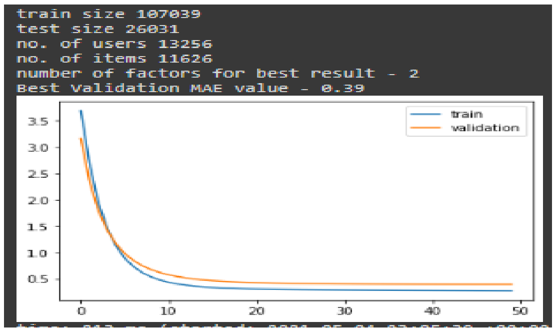
Figure 5: SGD



```
train size 99335
test size 25232
no. of users 18102
no. of items 8186
number of factors for best result - 2
Best Validation MAE value - 0.77
```

Figure 6: ALS

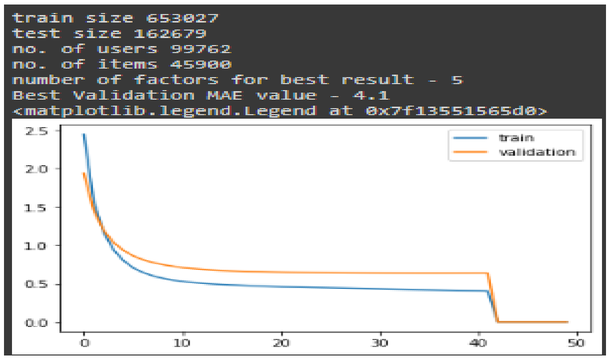
2. Digital Music 5-core dataset (SGD)



```
train size 107039
test size 26031
no. of users 13256
no. of items 11626
number of factors for best result - 2
Best Validation MAE value - 0.39
```

Figure 7: ALS

3. (4 5-core datasets combined SGD) - 1.Software, 2.Patio, Lawn and Garden 3. Digital Music 4. Arts, Crafts and Saving



```
train size 653027
test size 162679
no. of users 99762
no. of items 45900
number of factors for best result - 5
Best Validation MAE value - 4.1
<matplotlib.legend.Legend at 0x7f13551565d0>
```

Figure 8: SGD

5.3 Neural Collaborative Filtering

In traditional Machine Learning projects, we evaluate models using metrics such as Accuracy (for classification problems) and RMSE (for regression problems). However, such metrics are too simplistic for evaluating recommender systems. The goal here is that we don't need the user to interact with every single item in the list of recommendations. Instead, we just need the user to interact with at least one item on the list — as long as the user does that, the recommendations have worked. To simulate this, let's run the following evaluation protocol to generate a list of top 10 recommended items for each user. For each user, randomly select 99 items that the user has not interacted with. Combine these 99 items with the test item (the actual item that the user last interacted with). We now have 100 items. Run the model on these 100 items, and rank them according to their predicted probabilities. Select the top 10 items from the list of 100 items. If the test item is present within the top 10 items, then we say that this is a hit. Repeat the process for all users. The Hit Ratio is then the average hits. This evaluation protocol is known as Hit Ratio, and it is commonly used to evaluate recommender systems.

The model was tested for 3 different optimizers and following were the hit ratios that we got. For RMSprop we got the maximum hit ratio of 0.91, for Adam we got 0.82 and for Nesterov with beta = 0.09 we got 0.80.

Optimizers	**Hit Ratio**
Adam	0.82
Nesterov (beta =0.09)	0.80
RMSprop	0.91

Figure 9: Hit Ratio

6. SUMMARY AND FUTURE DIRECTION

Traditionally, recommender systems are based on methods such as clustering, nearest neighbor and matrix factorization. However, in recent years, deep learning has yielded tremendous success across multiple domains, from image recognition to natural language processing. Recommender systems have also benefited from deep learning's success. In fact, today's state-of-the-art recommender systems such as those at Youtube and Amazon are powered by a combination of traditional approaches and complex deep learning systems.

One of the content-less approaches we thought about for the cold-start problem for new users is to take the mean of each of the k factors of all users, and this can be the k factor for our new user. We can then compare the recommendation made by this user vector to other trivial approaches like recommending the most popular item, items with highest rating, etc.

We plan to implement the ensembling of the four models described above to get better performance and accuracy. Based on the accuracies of the four models, we give weights to them and choose the top recommendations given by these models based on the weights which were assigned to these models, giving higher priority to the better accuracy model.

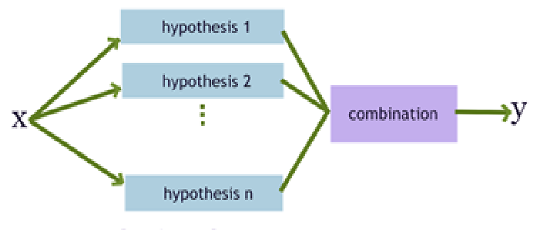


Figure 8: Ensemble [9]

7. PROJECT TAKEAWAYS

Overall, we gained a lot of knowledge about machine learning in the field of recommendation systems by working on this project.

We learned how recommendation engines work and how important they are to alleviating the issue of information overload that occurs on e-commerce sites such as Amazon and Walmart by implementing the different models in the recommendation engine. Working with Amazon's dataset, it provided us with insights into user and item metadata that can be used in a variety of ways to benefit the companies.

As we developed our model, we realized that evaluation metrics such as hit ratio and mean absolute error were vital in helping us evaluate our model. In the early stages, our model overfit, and we only realised that with the help of these evaluation metrics.

Learned different ways to approach the problem of sparse matrix by using item and user mapping instead of the user-item matrix which mostly had null values and also tried different ways to deal with the split of the train-test matrix. As we worked with explicit ratings and also converted explicit ratings into implicit ratings for the neural network model, we realized the difference in the usages of the two types of ratings.

There were two major challenges that we faced: first, because we had a large amount of data to analyze, the user-item pivot table for ALS and SGD took up a lot of space and slowed down the model, so we employed user item mapping to solve that. The second challenge was the lack of user metadata, which prevented us from countering the problem of new users using a content-based approach.

8. REFERENCES

[1] G. Linden, B. Smith and J. York, "Amazon.com Rec- ommendations: Item-to-item Collaborative Filtering", *IEEE Internet Computing* 7 (2003), 76–80

[2] J. Bennet and S. Lanning, "The Netflix Prize", *KDD Cup and Workshop*, 2007. www.netflixprize. com.

[3] M. Deshpande, G. Karypis, "Item-based top-N recom- mendation algorithms", *ACM Trans. Inf. Syst.* 22 (2004) 143-177.

[4] J. L. Herlocker, J. A. Konstan, and J. Riedl. "Explain- ing collaborative filtering recommendations", In Pro- ceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, ACM Press, pp. 241-250, 2000.

[5]]https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea

[6] http://deepyeti.ucsd.edu/jianmo/amazon/index.html

[7]https://www.amazon.science/the-history-of-amazons-recommendation-algorithm

[8]https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/baalbaki.pdf

[9]https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[10]https://arxiv.org/abs/1708.05031