CS 6220: Song Recommendation System

By: Jason Miller and Shefali Khatri

1. INTRODUCTION

Recommendation systems are valuable as they enhance the user experience for both a seller and the consumer. Recommendation systems enhance the user experience for the seller because their product is better able to reach the targeted consumer, while it also enhances the user experience for the consumer because the consumer does not need to spend time sifting through irrelevant/undesirable products. A system that is able to successfully recommend desired songs to its customers will lead to greater user engagement, decreased customer churn and ultimately increased revenue for a business.

Our goal for our project was to develop a reliable song recommendation system that would be able to use previous user behavior to infer songs with a strong likelihood of being heard by each user. The nature of the problem was predictive and we used supervised learning as we had information regarding songs that were previously heard. Recommendation systems often deal with two types of data which can either be implicit or explicit. Explicit data is data in which user ratings are known and therefore, user preference is directly understood. Implicit data is data in which user preference must be inferred from user actions such as purchase history, browsing patterns, search patterns and click rates. While explicit data is easier to work with, it often introduces bias as individuals are more inclined to give ratings only when they have extreme feelings towards the product. Conversely, implicit data is more difficult to work with, but widely available as it requires no additional input from the user. Our problem used the Echo Nest Taste Profile Dataset[1] which contained implicit data in the form of listening counts for each user and song. We used a series of models that combined traditional Matrix Factorization techniques as well as Neural Networks to identify an appropriate approach to song recommendation.

2. RELATED WORK

The literature on recommender systems suggests that most recommendation systems follow either a collaborative filtering approach, content based filtering approach, or a hybrid of the two methods. These are the most widely known approaches. However, as the field of Deep Learning has evolved, new methods have been developed which utilize neural networks. For example, in 2016 Google published a paper that suggested Youtube was using two neural networks to provide video recommendations. The first neural network was used to generate candidates, while a second neural network was used for ranking.[2] Commonly used

[1]Million Songs Subset: Getting the dataset | Million Song Dataset

[2] Covington, Paul et al. "Deep Neural Networks for YouTube Recommendations." *Proceedings of the 10th ACM Conference on Recommender Systems* (2016): n. pag.

collaborative filtering approaches include matrix factorization, K-nearest neighbors, and Pearson correlation while the most commonly used content based filtering approach uses vector space representation.

Our methods utilized a combination of collaborative filtering approaches and neural networks. Traditional matrix factorization techniques rely on ratings and explicit data. However, our data was implicit and therefore, we relied on a Matrix Factorization technique called Alternating Least Squares which attempts to learn information contained within the missing data in the user item interaction matrix. This is a problem that does not exist with explicit data as the missing data contains no information. Alternating least squares differs from traditional Matrix Factorization as it uses a different approach for minimizing the loss function.

Additionally, we used Bayesian Personalized Ranking which attempts to predict how a user would rank items in a catalog. This method is unlike others that attempt to solely predict item ratings for each user. The method is derived from Bayes' Theorem and the ultimate goal is to then maximize the posterior probability. This technique can be used in combination with many other methods and is used to obtain a ranking. Our Bayesian Personalized Ranking was used with a Neural Network similar to the approach taken by Youtube.

There are also more sophisticated approaches such as feature-rich recommendation systems, factorization machines, and deep factorization machines. While we did consider these approaches, we ultimately did not have the time and resources needed to build them due to the model complexity. A sequence-aware recommendation system was also considered, but our data did not contain timestamps for user interactions.

3. DATA AND EXPERIMENTS

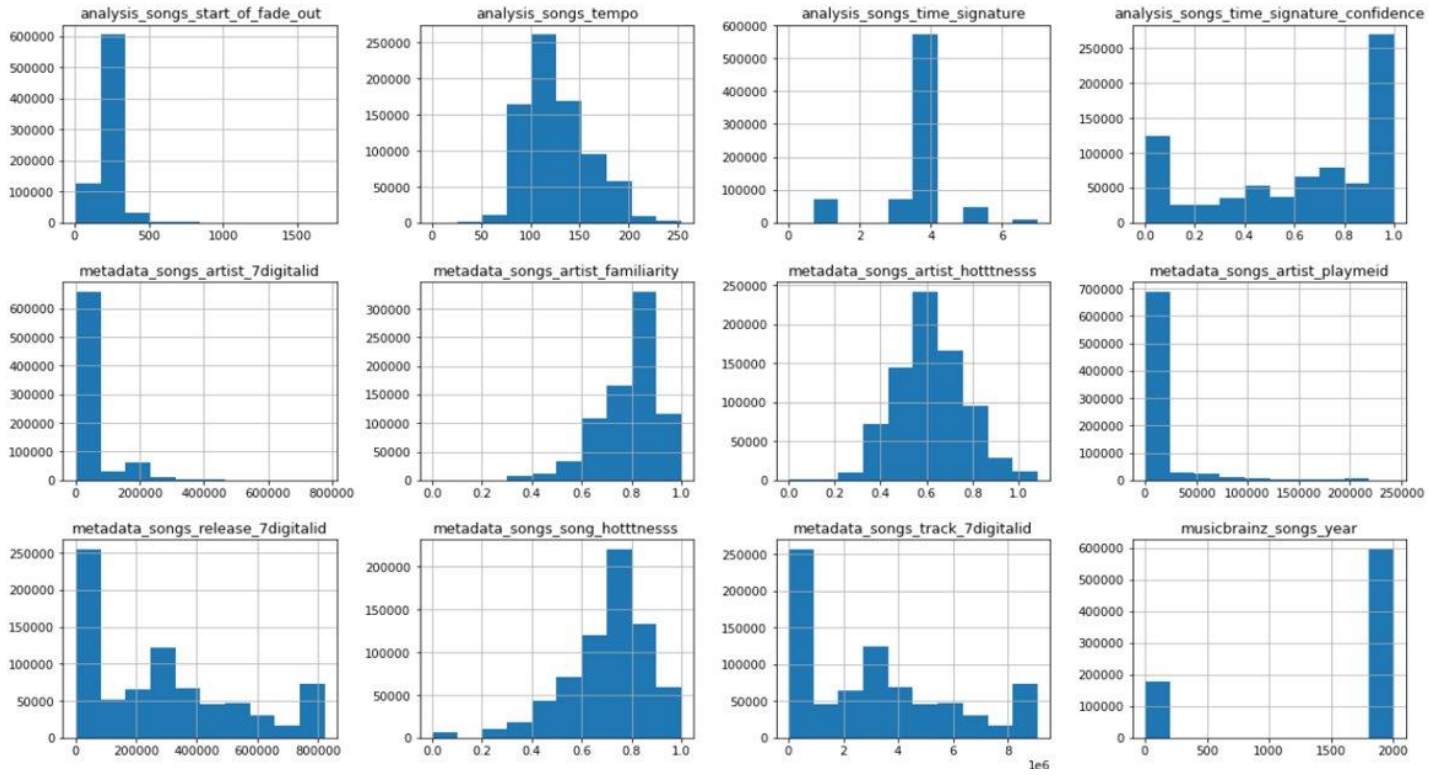
For the purpose of our experiment, we did not work with the full 300gb Million Song Dataset but instead with a 1.8gb subset containing tabular data for 10,000 songs. This data included rich feature data including pure audio analysis data, artist location, genre tags, and release date. The data was provided in Hierarchical Data Format (HDF5), and scripts were written to extract the data from these H5 files and reformat it into a DataFrame. As part of preprocessing, 49 features were eliminated from the dataset for the following reasons:

- 2 features had over 60% of data missing.
- 21 features had 0 standard deviation across all observations.
- 24 features had inconsistent data types or information contained in arrays
    - *We believe this information is valuable, but much preprocessing and feature engineering would be needed to work with it and this can be done in future iterations of this project.*
- 2 features were found to contain irrelevant data, such as nominal data, labels to link other datasets, or index values for file storage organization.

Additionally, a significant number of numeric features were found to have a high proportion of values of zero. This is shown in Figure 3.1.1. To understand why so many zeroes were

occurring, an analysis was conducted to determine whether or not zeroes were naturally occurring or represented missing data. Several features containing many zeroes were determined to represent categorical data, therefore a value of 0 represented a specific encoded category. An example of this was song mode: minor and major song modes corresponded to 0 or 1. For other features such as year released or tempo, zeroes appeared to represent missing values. It would not be possible for a song to be released in year 0 or to have a tempo of 0 beats per minute. We decided to perform binning on these features. In the case of tempo, equal-width binning was used with 10 bins and 0 values were binned into an unknown category. In the case of year, equal-width binning was used, but with decade intervals. Binning by decade helps to naturally represent different types of songs. The missing values for year were binned into an unknown category.

Figure 3.1.1. *Histograms showing distributions of values in numeric features. Many features have significant amounts of zeroes.*



Other features ranged from 0 to 1 and appeared to represent confidence, percentage, or “algorithmic” score values. Song artist hotness was a feature that had values ranging from 0-1.08. The Million Songs Dataset website described this feature as ‘algorithmic estimation.’ Two other features were also found with the same description. Upon deeper analysis of the features computed with algorithmic estimation, it appeared that all of these except for song artist hotness had a range from 0-1. Therefore, an assumption was made that song artist hotness should also have a range of 0-1 and min-max scaling was performed on this feature. Subsequently, in order to clean up the data for use, mean imputation was performed on one feature, song hotness, which was found to have a small percentage of observations

missing. Our final step of preprocessing was to standardize the data to a Gaussian distribution.

Additionally, we worked with the Echo Nest Taste Profile Subset[3] which contains user listen data for songs in the full Million Song Dataset. This dataset contained user ID's, song ID's, and track listen counts for each user. The original intention was to generate a model that leveraged this user data in addition to the rich feature data of the Million Song Subset. The Million Song Subset was merged with the Echo Nest dataset to create a subset of users who listened to songs for which we had rich feature data.

Although we intended to work with the Million Songs Subset dataset, the initial models we found that worked with implicit data did not require additional information beyond the data contained in the Echo Nest Taste Profile. We decided to use the subset we created so that in future iterations of the project, we could easily leverage the rich feature data from the Million Song Subset.

4. SOLUTION/METHOD

Our approach to this problem used two different techniques to identify a successful solution for song recommendation. The two methods used were as follows:

1. Alternating Least Squares Matrix Factorization
2. Bayesian Personalized Ranking

We did not attempt any additional innovation as these are widely accepted methods used in the industry.

4.1. ALTERNATING LEAST SQUARES MATRIX FACTORIZATION

Alternating Least Squares is a Matrix Factorization technique first introduced in 2008 in a paper titled "Collaborative Filtering for Implicit Feedback Datasets."[4] The goal of Alternating Least Squares is to enable Matrix Factorization with implicit data. The user-item interaction matrix in this problem is defined as the user list multiplied by an item list. For each user-item pair that exists, the Alternating Least Squares method has an associated confidence measure which helps to define a user's preference towards an item. An item with more interactions will have a stronger confidence than an item with fewer interactions. The confidence measure is computed as:

$$C_{ui} = 1 + \alpha r_{ui}$$

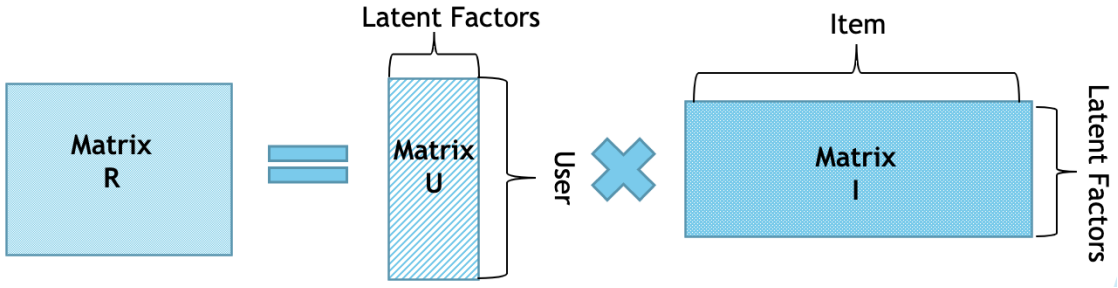
Where C_{ui} represents the confidence for a given user and item, r_{ui} represents the number of interactions and alpha is a constant. In the paper, the authors used an alpha

[3] The Echo Nest Taste Profile Subset

[4] *Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08). IEEE Computer Society, USA, 263–272.*

of 40. Preferences are assumed to be the inner product of vectors for each user and each item. This technique attempts to map the item and user vectors into a common latent factor space. The latent factors represent the user and item vectors in lower dimensions. The vectors are obtained by decomposing the user-item interaction matrix into the product of lower dimensional vectors.

Figure 4.1.1. *An illustration of matrix factorization.*



Unlike traditional matrix factorization, this technique also accounts for the unobserved user item interactions as these also contain information within them about a user's preference. Therefore, this method uses a different loss function than traditional matrix factorization. Alternating least squares minimizes two different loss functions by alternating between re-computing the user factors and item factors with stochastic gradient descent until convergence. This results in one set of factors being held constant at each iteration. Additionally, the authors included L2 regularization to prevent overfitting.

ALS model implementation:

Our approach used the ALS model defined in the implicit package.[5] We used a subset of our dataset to train the model. Our training dataset contained only observations in which we had at least 5 observations per user to avoid the cold-start problem. The cold-start problem arises when you have a new user on whom you have no previous information. In these situations, most models are not able to provide a recommendation. For the purposes of this project, we did not focus on the cold-start problem, but instead tried to identify songs to recommend to users for whom we did have data. We then conducted a stratified k-fold validation in which we defined k as 5 and stratified our data based on the user to ensure that the ALS model would not encounter the cold-start problem. We used the validation to tune our hyper-parameters and found that performance in our model did not generally increase beyond values of 15 for alpha, l2 regularization of 0.01, and 35 latent factors. These were the factors we ultimately chose for our final ALS model. We compared performance of each of these measures against the model's hit ratio. The hit ratio is defined as the number of songs correctly recommended over all observations. Figure 4.1.2 depicts how model performance changed based on different values of alpha, lambda (l2 regularization), and latent factors.

[5] Implicit — Implicit 0.4.4 documentation

Figure 4.1.2. *Hit ratio values when adjusting different parameters for the ALS model.*

Figure 4.1.2(a). *Adjusting L2 regularization*

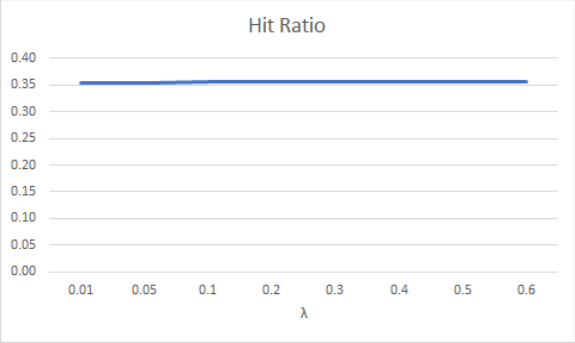
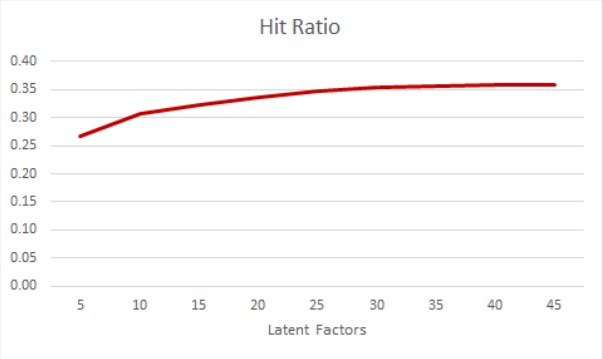


Figure 4.1.2(b). *Adjusting latent factors*



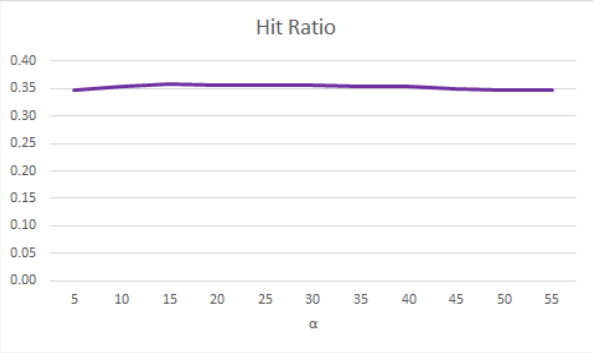


Figure 4.1.1(c). *Adjusting alpha values*

An additional parameter in the model that we did not tune for was the number of iterations. The number of iterations were not tuned for as the iterations do not impact the model performance. The ALS model is known to converge within 20 iterations or less. However, as a buffer, the number of iterations for our model was set to 30.

4.2. BAYESIAN PERSONALIZED RANKING

The concept of Bayesian Personalized Ranking was introduced in the 2009 paper "BPR: Bayesian Personalized Ranking from Implicit Feedback."[6] Bayesian Personalized Ranking is a pairwise Learning to Rank (LtR) algorithm that takes two items as input and returns an ordering of the two. Whereas other recommendation techniques attempt to predict the ratings users would give for unseen items, pairwise LtR algorithms attempt to

[6] Rendle, Steffen et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback." *UAI* (2009).

predict how users would rank unseen items. Further, BPR attempts to optimize specifically for this ranking.

Some formalisms are needed to make BPR work: let $>_u$ represent a personalized total ranking for a user of all items. $>_u$ must meet the following properties to be a total order/ranking:

$$\forall i, j \in I : i \neq j \Rightarrow i >_u j \vee j >_u i \qquad (totality)$$
$$\forall i, j \in I : i >_u j \wedge j >_u i \Rightarrow i = j \qquad (antisymmetry)$$
$$\forall i, j, k \in I : i >_u j \wedge j >_u k \Rightarrow i >_u k \qquad (transitivity)$$

BPR is typically only used on implicit feedback. Intuitively, it makes sense that if a user likes a song, the user will listen to that song many times. However, the reasons why a user listened to a song a few times or not at all are unclear. In our dataset, each user does not provide ratings for each song listened to, so this data can be treated as implicit feedback. In BPR, we assume that if an item was not bought—a song was not listened to—it is worse than if the item was bought. Further, we assume that if a user listened to one song but not another, the user would rank the one listened to over the other.

More formally, we define that given two songs ***i***, ***j***:

- If a user listened to ***i*** and did not listen to ***j***, we assume that the user preferred ***i*** over ***j*** - we call this a **positive pair**
- If a user listened to ***j*** and did not listen to ***i***, we assume that the user preferred ***j*** over ***i*** - we call this a **negative pair**
- If a user did not listen to song ***i*** or ***j***, we don't know which song the user preferred

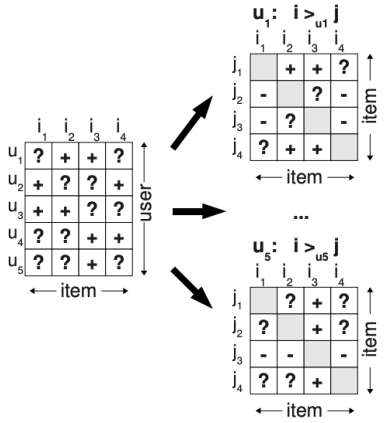


Figure 4.2.1. *Generating positive and negative item pairings for each user.*

We are trying to predict how the user would rank the songs that the user did not listen to. For each user, we find all possible positive and negative pairs, and we feed these pairs into our model as training data. We can see this illustrated in Figure 4.2.1. Each instance

in the data is called a ***triplet*** composed of a *user, i,* and *j* where *i* and *j* represent songs. These triplets are fed into the model.

BPR uses an optimization function $\mathrm{BPR\text{-}OPT}$ that is derived from Bayes' theorem. It assumes that if a perfect total ranking $>_u$ exists, then there must be some model that produces the perfect total ranking. This is described by the following equation:

$$p(\Theta|>_u) \propto p(>_u|\Theta)p(\Theta)$$

where Θ is the vector of parameters for a model. $p(>_u|\Theta)$ is expanded using the likelihood function and the prior probability $p(\Theta)$ is assumed to be normally distributed. With some clever mathematics beyond our scope, $\mathrm{BPR\text{-}OPT}$ equation becomes

$$\sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2$$

This is proven in the original 2009 paper by Rendle et al. The gradient of this equation is used in stochastic gradient descent.

BPR model implementation:

First, the dataset was filtered to only contain users that had listened to 4 or more songs. Positive and negative triplets were created for each user. The ***j*** item (song that was not listened to) for each triplet was randomly selected. One song was held out for each user. The idea of this was to see if the model can predict the withheld song. Figure 4.2.2 illustrates the architecture of the neural network used to implement BPR. The BPR loss function was implemented in the Lambda layer. This architecture is largely based on this Medium article. The processes that cost the most computational time were the generation of triplets during preprocessing and the generation of recommendations after training each model.

In order to properly tune the model, several models with different hyperparameter combinations were tested. Different latent dimension values for the embedding layers were tested, as well as different learning rate values. Each of the models ran for 3 epochs at a batch size of 2048. Due to computational constraints, these hyperparameters were not adjusted. In future iterations of this project, larger epoch values can be tested. The evaluations of different combinations of latent dimensions and learning rates are shown in Section 5.2.

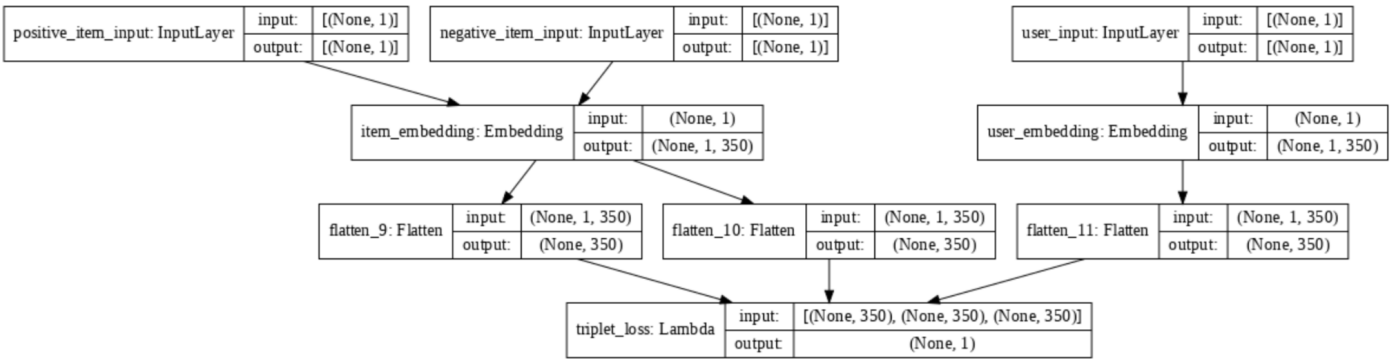


Figure 4.2.2. *BPR neural network layer architecture for 350 latent dimensions.*

5. EVALUATION AND RESULTS

The performance of our models were evaluated on three metrics which include hit ratio, mean average precision and mean average recall. These metrics were calculated as followed:

$$\text{Hit Ratio} = \frac{\text{Number of songs correctly recommended to all users}}{\text{Number of observations in the dataset}}$$

$$\text{Precision} = \frac{1}{|U|}\sum_{u \in U} \frac{\text{Number of songs correctly recommended to user } u}{\text{Number of songs recommended to user } u}$$

$$\text{Recall} = \frac{1}{|U|}\sum_{u \in U} \frac{\text{Number of songs correctly recommended to user } u}{\text{Number of observations for user } u \text{ in dataset}}$$

The primary metric that was used for evaluation and model comparison was the hit ratio. Although tuning attempted to maximize all three metrics, hit ratio was the main focus.

5.1 ALTERNATING LEAST SQUARES MATRIX FACTORIZATION

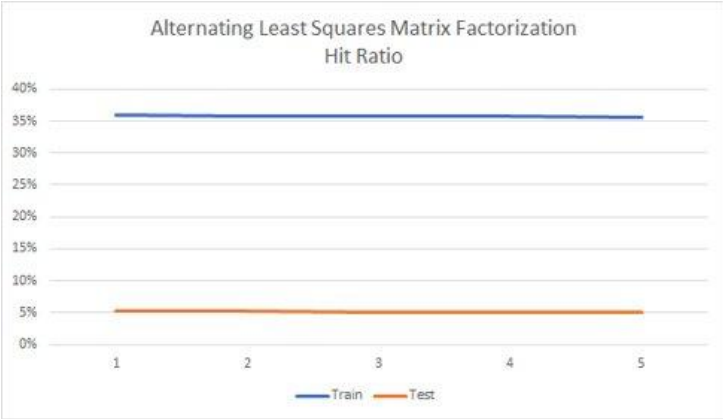
Evaluation of the Alternating Least Squares Matrix Factorization was performed against a recommendation of the top 10 most popular songs. The top 10 most popular songs were used as a baseline for comparison. A subset of the test dataset which contained at least 5 interactions per user was used for evaluation to ensure that the cold-start problem did not occur. A stratified 5-fold cross validation was then performed on the test dataset to evaluate the model against the top 10 most popular songs.

The ALS model was found to recommend correctly approximately 5% of the time, while the top 10 most popular songs contained a correct recommendation approximately 2.5% of the time. A paired t-test was conducted on the hit ratios obtained from 5-fold cross validation and a statistically significant difference was not

found between the two approaches. Additionally, when the validation data was compared against the test data, the ALS model appeared to have been severely overfit. The validation data resulted in a hit ratio of 35% across all 5 folds while the test data resulted in a hit ratio of 5% across all folds.

Two reasons that could account for the poor model performance is that the model was trained on a dataset containing significantly fewer songs than contained in the test data. This is because a subset of the data was used for training given the time required to tune a model. The second reason that the model may not have performed well was the sparsity of the user-item interaction matrix. Typically, ALS models work well on matrices that have a sparsity of 99.5% or less. However, the user-item interaction matrix for the Echo Nest Taste Profile data had a sparsity of 99.9% which could have contributed to the poor performance.

Figure 5.1.1. *Comparing hit ratio on training vs. test data for ALS MF model.*



5.2 BAYESIAN PERSONALIZED RECOMMENDER

Precision, recall, and hit ratio were all calculated using the top 10 recommendations for each user. Additionally, prior to training the model, 1 record was removed for users that had more than 3 records in the dataset. The idea of this is that we can evaluate the goodness of our recommender by withholding a song a user did listen to and seeing if our recommender listened to that withheld song. These hold-out items became in essence our testing data for this model.

$$\text{Hold out hit ratio} = \frac{\text{Number of withheld items correctly recommended}}{\text{Total number of items withheld}}$$

Figure 5.2.1 shows the results for various hyperparameter combinations. It is clear that models with learning rates of 0.1 performed worse than models with lower learning rates, regardless of latent dimension values. All four metrics increased directly with latent dimensions except for hold out hit ratio. This trend is highlighted in Figure 5.2.2.

Figure 5.2.1. *Metrics obtained for different combinations of hyperparameters.*

	latent_dim	learning_rate	precision	recall	hit_ratio	hold_out_hits
0	200	0.001	0.898046	0.157440	0.852244	0.189554
1	200	0.010	0.964685	0.168766	0.913556	0.147237
2	200	0.100	0.646932	0.093420	0.505693	0.046645
3	250	0.001	0.916698	0.161128	0.872210	0.188296
4	250	0.010	0.968306	0.169764	0.918955	0.146971
5	250	0.100	0.657485	0.094732	0.512799	0.050127
6	300	0.001	0.930318	0.163965	0.887564	0.201378
7	300	0.010	0.968883	0.169829	0.919309	0.144650
8	300	0.100	0.655941	0.094416	0.511084	0.051239
9	350	0.001	0.941188	0.166068	0.898947	0.200967
10	350	0.010	0.969723	0.170075	0.920640	0.149897
11	350	0.100	0.658450	0.094121	0.509489	0.045508

Figure 5.2.2. *Models with learning rate = 0.01 and 0.001. Precision, hit ratio, and recall plotted over varying latent dimensions.*

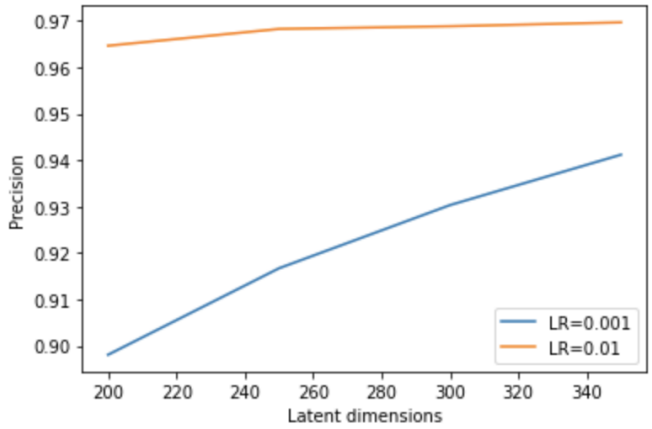


Figure 5.2.2(a). *Precision*

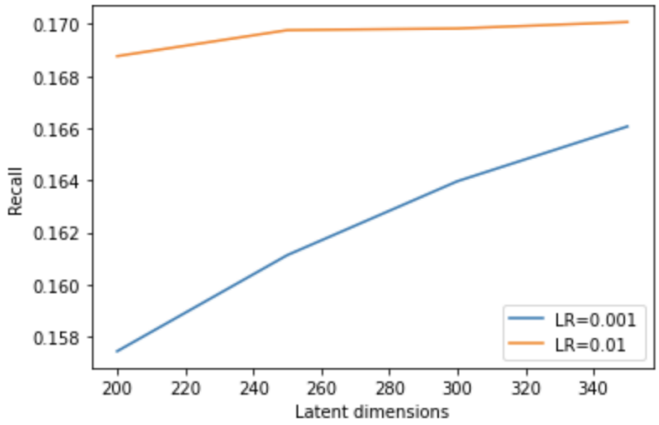


Figure 5.2.2(b). *Recall*

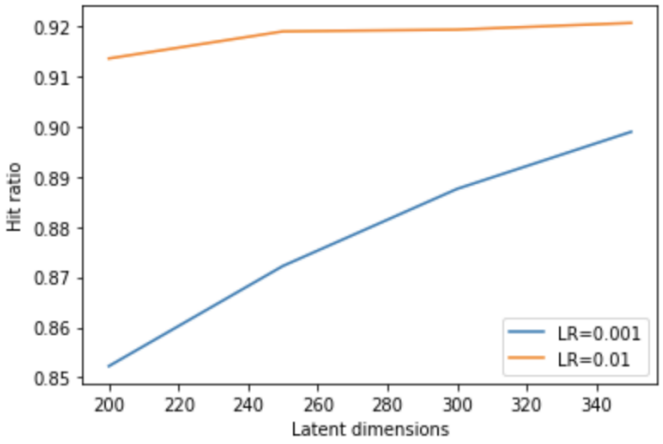


Figure 5.2.2(c). *Hit ratio*

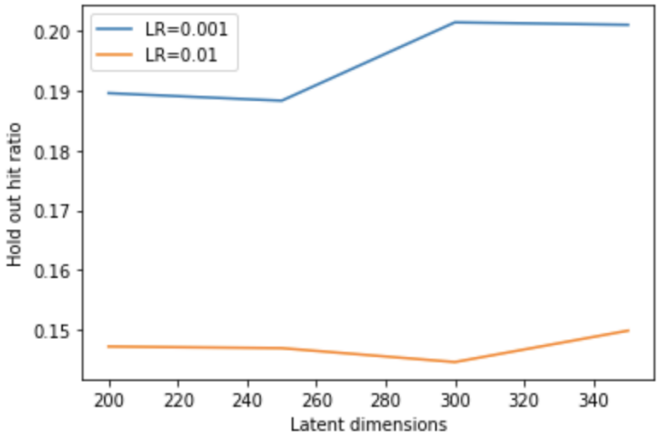


Figure 5.2.2(d). *Hold out hit ratio*

6. CONCLUSION

In this paper we have discussed two approaches to implementing recommendation systems on implicit data. Alternating least squares matrix factorization attempts to use matrix factorization on implicit data by developing an associated measure of preference and a loss function that accounts for the information contained within unobserved interactions. Bayesian Personalized Ranking leverages Bayesian statistics to directly optimize for the correct ordering of recommendations, and we have implemented it with a neural network.

Considerations for future project iterations:

- Determine how to leverage rich feature data for songs.
- Use the entire Echo Nest dataset so we have more data.
- Use only users that have a large number of observations. Similarly, use only songs that occur in a large number of observations.
- BPR model:
    - Experiment with leave-one-out cross validation with holdout item.
    - Train and evaluate models with different batch sizes and epoch values.
    - Experiment with higher latent dimensions and lower learning rates.