Collaborative Filtering Task

CSC311 – Final Report

Louis Liu – XXXXXXXXXX

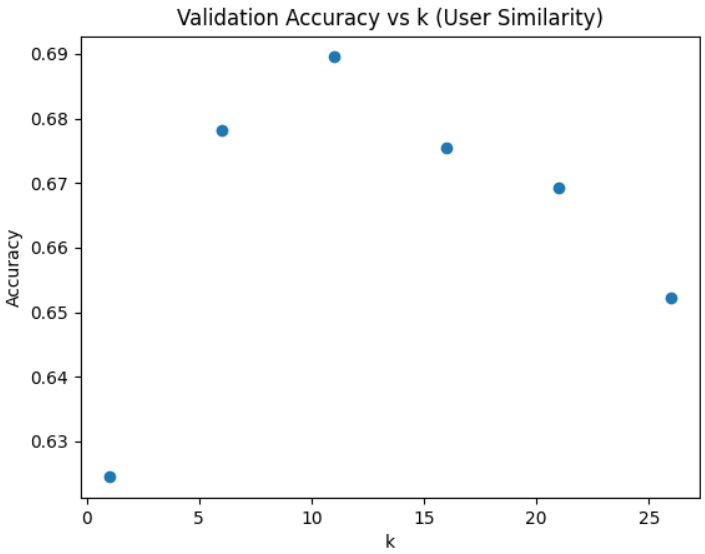
Alastair Sim – XXXXXXXXXX

Eric Ji - XXXXXXXXXX

Part A

Question 1: K Nearest Neighbours

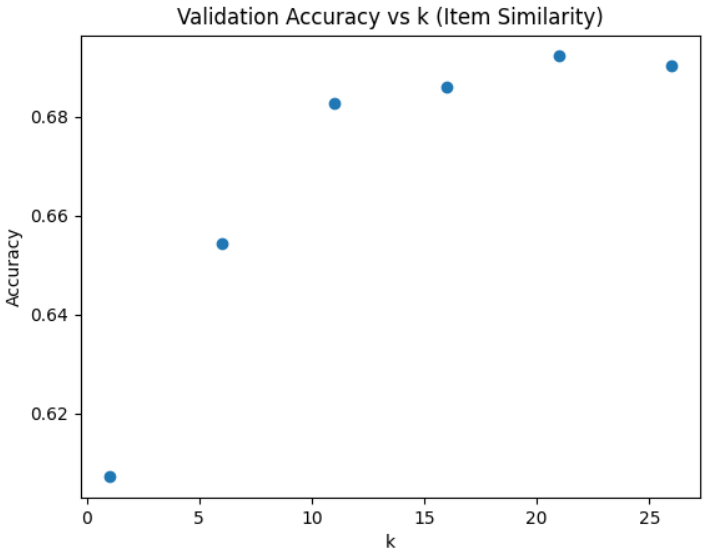
a)



k	Validation Accuracy
1	0.6248
6	0.6781
11	**0.6895**
16	0.6756
21	0.6692
26	0.6523

b) Based on the above results, the optimal k that yielded the greatest validation accuracy is k* = 11. This model attained a test accuracy of 0.6842**.**

c)



k	Validation Accuracy
1	0.6071
6	0.6542
11	0.6826
16	0.6860
21	**0.6922**
26	0.6904

Results and Underlying Assumption

Based on the above results, the k that yielded the greatest validation accuracy is k* = 21. This model attained a test accuracy of 0.6816**.** The underlying assumption for item-based collaborative filtering is that if question A has the same correct and incorrect answers given by other students compared to question B, then question A's correctness given by a specific student matches that of question B.

d) At optimal values of k, user-based collaborative filtering performs slightly better, yielding a test accuracy of 0.6842 over 0.6816 produced by item-based collaborative filtering.

e) Some limitations of the current kNN implementations:

- Slow predictions at inference. For each prediction, the runtime complexity is $O(N_S N_D)$ where N_S is the number of students and N_D is the number of questions.
- Large memory requirement. The entire training data must be available for each test prediction.
- The current kNN implementation only finds nearest neighbours based on questions or students, but does not take into account other types of metadata such as student profile.

Question 2: Item Response Theory

a) Deriving Log Likelihood

The expression for the probability of correctness for a single student-question (i, j) pair is:

$$p(c_{ij} = 1|\theta_i, \beta_j) = \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}$$

$$p(c_{ij}|\theta_i, \beta_j) = \left(\frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{c_{ij}} \cdot \left(1 - \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{1 - c_{ij}}$$

$$p(c_{ij}|\theta_i, \beta_j) = \left(\frac{e^{\theta_i - \beta_j}}{1 + e^{(\theta_i - \beta_j)}}\right)^{c_{ij}} \cdot \left(\frac{1 + e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}} - \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{1 - c_{ij}}$$

$$p(c_{ij}|\theta_i, \beta_j) = \left(\frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{c_{ij}} \cdot \left(\frac{1}{1 + e^{\theta_i - \beta_j}}\right)^{1 - c_{ij}}$$

Apply likelihood expression from correctness probability above:
(Note: the multiplier is strictly for existing i, j pairs)

$$p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{(i,j)} \left(\frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{c_{ij}} \cdot \left(\frac{1}{1 + e^{\theta_i - \beta_j}}\right)^{1 - c_{ij}}$$

Take the log of the expression above to obtain the log-likelihood and manipulate:

$$\log p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \log \prod_{(i,j)} \left(\frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right)^{c_{ij}} \cdot \left(\frac{1}{1 + e^{\theta_i - \beta_j}}\right)^{1 - c_{ij}}$$

$$\log p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{I} \sum_{j=1}^{J} \left[c_{ij} \cdot \log\left(\frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right) + (1 - c_{ij}) \cdot \log\left(\frac{1}{1 + e^{\theta_i - \beta_j}}\right)\right]$$

$$= \sum_{i=1}^{I} \sum_{j=1}^{J} \left[c_{ij} \cdot \left[\log(e^{\theta_i - \beta_j}) - \log(1 + e^{\theta_i - \beta_j})\right] + (1 - c_{ij}) \cdot \left[\log 1 - \log(1 + e^{\theta_i - \beta_j})\right]\right]$$

$$= \sum_{i=1}^{I} \sum_{j=1}^{J} \left[c_{ij} \cdot \left[\theta_i - \beta_j - \log(1 + e^{\theta_i - \beta_j})\right] + (1 - c_{ij}) \cdot \left[0 - \log(1 + e^{\theta_i - \beta_j})\right]\right]$$

$$= \sum_{i=1}^{I} \sum_{j=1}^{J} \left[c_{ij}\theta_i - c_{ij}\beta_j - c_{ij}\log(1 + e^{\theta_i - \beta_j}) - \log(1 + e^{\theta_i - \beta_j}) + c_{ij}\log(1 + e^{\theta_i - \beta_j})\right]$$

The expression above is the final log-likelihood. The c_{ij} elements in the data that are equal to NaN are avoided with the indicator function.

$$\log p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{I} \sum_{j=1}^{J} \left[c_{ij}\theta_i - c_{ij}\beta_j - \log(1 + e^{\theta_i - \beta_j})\right] \cdot \mathbb{I}(c_{ij} = 1 \,||\, c_{ij} = 0)$$

Find the derivative of the log-likelihood expression with respect to θ_i

$$\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\partial}{\partial \theta_i} \sum_{i=1}^{I}\sum_{j=1}^{J} \left[\, c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1 + \, e^{\theta_i - \beta_j}\right)\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\partial}{\partial \theta_i} \sum_{j=1}^{J} \left[\, c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1 + \, e^{\theta_i - \beta_j}\right)\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{j=1}^{J} \left[\, c_{ij} - \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \left(\sum_{j=1}^{J} c_{ij} - \sum_{j=1}^{J} \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right) \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

Find the derivative of the log-likelihood expression with respect to β_j

$$\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{I}\sum_{j=1}^{J} \left[\, c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1 + \, e^{\theta_i - \beta_j}\right)\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{I} \left[\, c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1 + \, e^{\theta_i - \beta_j}\right)\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \sum_{i=1}^{I} \left[\, -c_{ij} + \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right] \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

$$\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \left(\sum_{i=1}^{I} -c_{ij} + \sum_{i=1}^{I} \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right) \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

The derivative of the log-likelihood with respect to θ_i is:

$$\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \left(\sum_{j=1}^{J} c_{ij} - \sum_{j=1}^{J} \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right) \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

The derivative of the log-likelihood with respect to β_j is:

$$\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta},\boldsymbol{\beta}) = \left(\sum_{i=1}^{I} -c_{ij} + \sum_{i=1}^{I} \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\right) \cdot \mathbb{I}\left(c_{ij} = 1 \,||\, c_{ij} = 0\right)$$

b) See Code

The expressions derived on the previous page are for log-likelihood and the code implements the algorithm using negative log-likelihood. As a result, the expression to update θ_i and β_j are given by:

$$\theta_i \leftarrow \theta_i + learning\_rate \cdot \frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta})$$

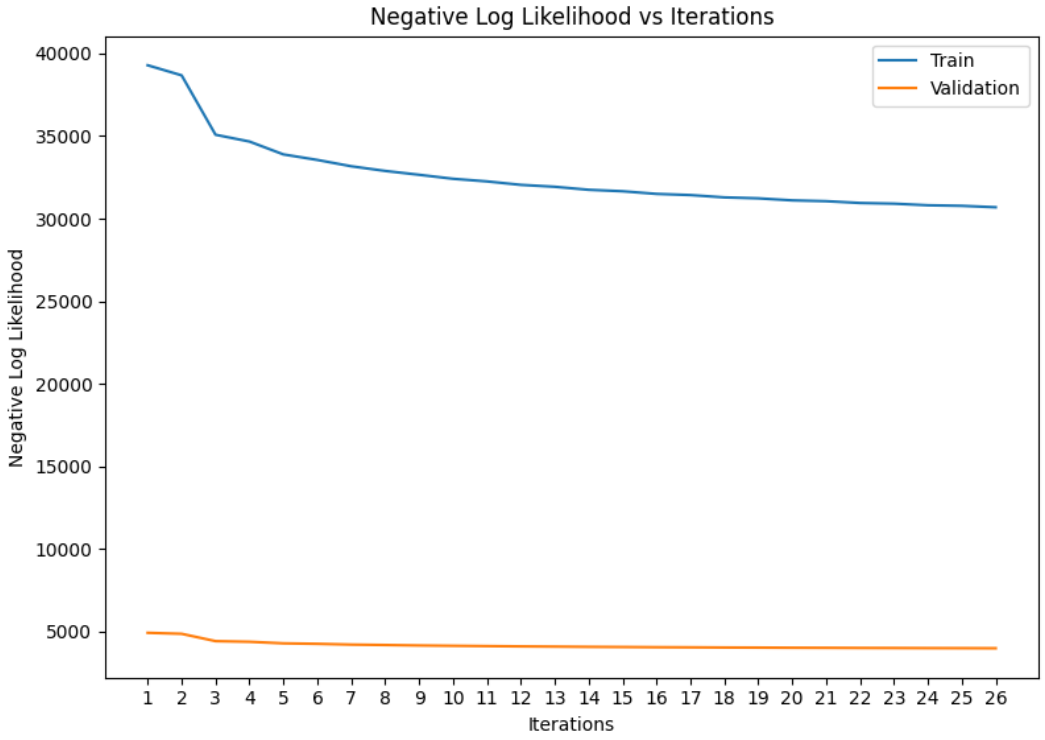
$$\beta_j \leftarrow \beta_j + learning\_rate \cdot \frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta})$$

The expressions for $\frac{\partial}{\partial \theta_i} \log\, p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta})$ and $\frac{\partial}{\partial \beta_j} \log\, p(c_{ij}|\boldsymbol{\theta}, \boldsymbol{\beta})$ are derived on the previous page. Alternating gradient descent was implemented which means only θ or β was updated in one iteration. The next iteration would update the other set of weights and so on.

From a simple grid search, the best validation accuracy obtained is 0.709. The hyperparameters used to achieve these results are tabulated below. θ_i and β_j were both initialized as vectors of zeros.

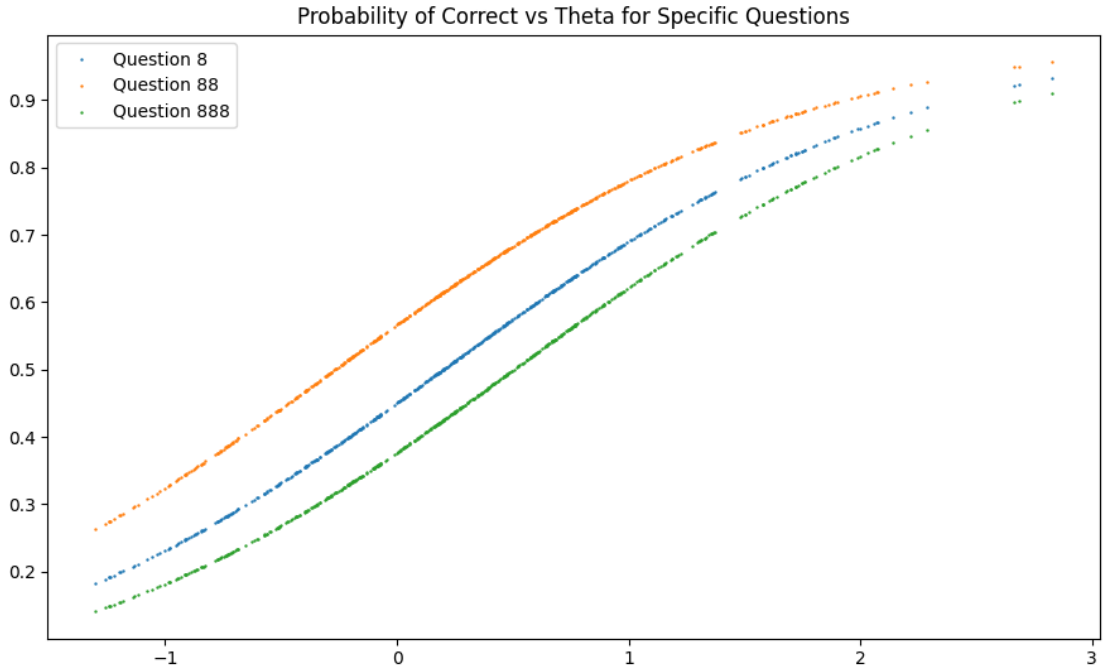
Number of Iterations	Learning Rate
26	0.01

The training and validation curves for negative log-likelihood vs iterations is shown below.



c) Using the optimal hyperparameters given above, the validation and test accuracies are approximately 0.7085 and 0.7028 respectively.

d) Scatter Plot



The figure above shows a scatter plot for three arbitrarily selected questions. The x-axis represents $\boldsymbol{\theta}$, which is a representation of a student's ability (θ_i represents the ability of student i). The y-axis represents the probability of a student i getting the selected question correct. As shown in the plot, the value of θ_i is positively correlated with the probability of correct for all questions. This logically makes sense because a more capable student is more likely to get a question correct, and a less capable student is less likely to get a question correct. The probabilities also seem to show signs of flattening at the extremities. This makes sense because the probabilities are bounded by 0 and 1. More importantly, the flattening at the extremity shows that after a certain threshold, more ability no longer helps increase the probability of getting a question correct and vice versa.

Note: A scatter plot is chosen because the θ_i values in $\boldsymbol{\theta}$ are not sorted. Furthermore, it also helps visualize the distribution of student abilities. Th θ_i values are most abundant around the average value which is an example of the central limit theorem in action.

Question 3: Neural Network

a) Differences Between ALS and Neural Networks

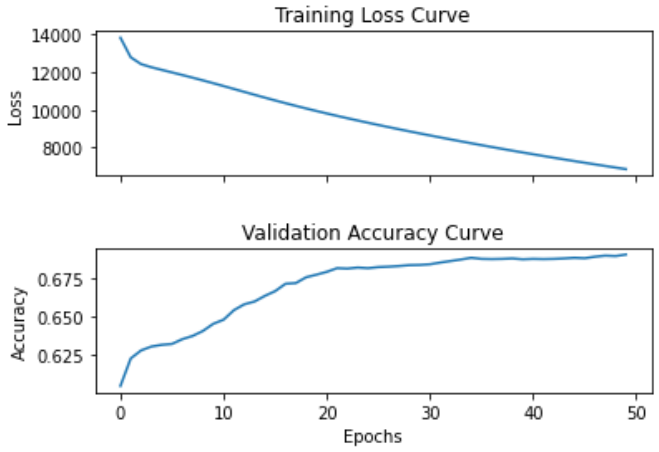
1. ALS algorithm always factorizes an input matrix R into two factors matrix U and matrix V such that $R = U^TV$. As a result, ALS minimizes two loss functions (one for U and one for V) instead of just one loss function in neural networks
    - For ALS, gradient descent is run on the U, while V remains constant. Then gradient descent is run on V, while U remains constant
    - For NN, gradient descent has one loss function to optimize, so every iteration focuses on optimizing the same loss value
2. In general, neural network such as MLPs are used in supervised machine learning algorithms because truth labels are used to compare against predictions. ALS is used in matrix completion which is an unsupervised learning task since there are no truth labels available
3. As mentioned earlier, ALS algorithm always factorizes an input matrix into two matrix factors to optimize, there is not much flexibility in that process. On the other hand, NN have more overall flexibility because the number of hidden layers is completely customizable

b) See code

c) The optimal k that resulted in the greatest validation accuracy is k*=50.

k	Optimal Hyperparameters		Validation Accuracy
	Learning Rate	# Epochs	
10	0.1	10	0.6876
50	**0.01**	**50**	**0.6901**
100	0.01	45	0.6892
200	0.01	30	0.6808
500	0.01	20	0.6717

d)

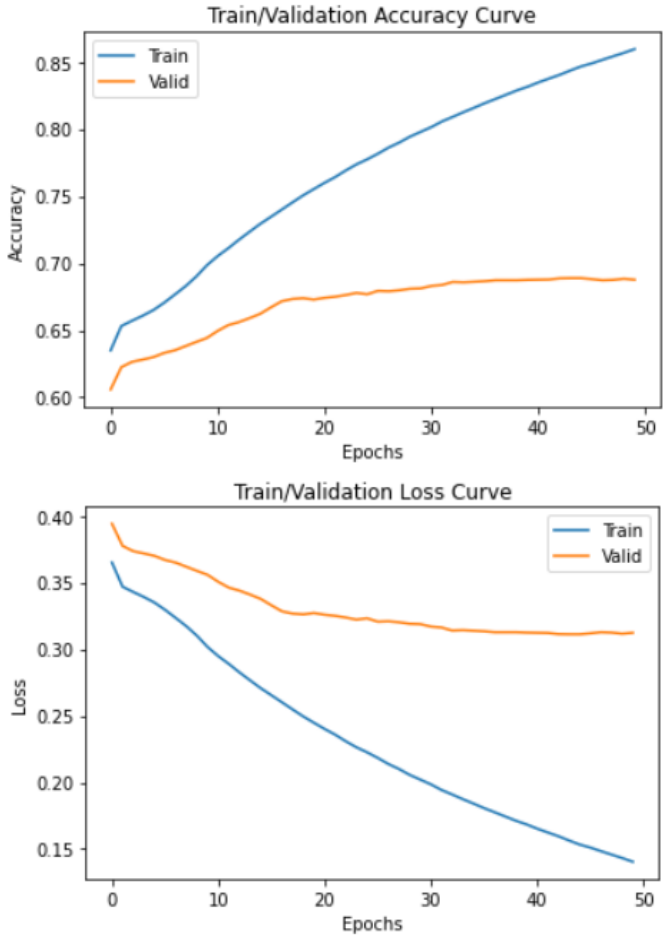


This model achieved a test accuracy of 0.6777.

e)

k	Optimal Hyperparameters			Validation Accuracy
	Learning Rate	# Epochs	Regularization Constant	
50	0.01	50	**0.001**	**0.6909**
			0.01	0.6834
			0.1	0.6247
			1	0.6250

The optimal regularization constant that resulted in the greatest validation accuracy is **λ*= 0.001**. In this model, the added regularization penalty helped reduce overfitting, allowing it to outperform the previous model by attaining a test accuracy to **0.6917**.



Accuracy (top) and loss (bottom) training curves for autoencoder model with regularization

Question 4: Ensemble

The final validation and test accuracies from the ensemble with bagging is shown in the table below.

Final Validation Accuracy	Final Test Accuracy
0.6898	0.6954

Models in the Ensemble

Model	Hyperparameters
kNN	k=11
IRT	Learning Rate = 0.01 Epochs = 26
Neural Network	K=50 Learning Rate = 0.01 lambda = 0.001 Epochs = 60

The ensemble consisted of one of each of the models shown in the figure above. The best hyperparameters found during training were respectively used for each model. Each of the models trained on different datasets created by bootstrapping. The bootstrap function iterates over the length of the original training set and generates three new datasets by randomly sampling (i.i.d.) from the original dataset with replacement. Predictions were generated from each of the three models by passing in the validation and test sets into each model. Using the predictions obtained by each model, an average is calculated to output the final prediction of the ensemble. All the predictions were floats between zero and one; these values were rounded accordingly and compared against the targets to obtain a final accuracy value.

The final accuracies given by the ensemble was worse than the best performance of the IRT model. The reason that there was no improvement was likely because the IRT model already performed the best. Factoring in the outputs from the slightly worse performing KNN and neural network models with equal weightings might have caused more predictions that would be predicted as correct from the IRT to be predicted as incorrect due to the KNN and NN “voting” against IRT.

Part B

1. Introduction

In the previous section, a collaborative filtering problem in the context of predicting the correctness of a student's answer to an unseen question, was attempted to be solved using an ensemble of different machine learning algorithms: kNN, IRT, and Autoencoder Neural Network. After training the models, the ensemble was able to hit a test accuracy of around 68.6%. The training data was given as a sparse matrix with each student representing a row and each question representing a column. Due to the nature of collaborative filtering, around 95% of the sparse matrix is incomplete and each algorithm finds a way to impute the missing entries. Upon analysis of each algorithm, it was believed that the neural network model could be improved the most effectively.

2. Existing Implementation

The current implementation of the neural network model first zeroes out all missing entries in the sparse matrix (NaN becomes 0). Next, it takes each row in the sparse matrix – a vector representing the correctness of all questions for a specific student – and feeds it into an autoencoder. The autoencoder encodes and decodes the input question vector and attempts to reconstruct it with predicted entries. In the training process, a masked MSE loss is applied around the input and output vectors to update model weights. After concluding the training process, the weights would ideally be values that can perfectly reconstruct an input vector from the same data distribution. To make a prediction of the correctness of question j answered by student i, the algorithm passes row i of the sparse training matrix through the autoencoder with the final prediction being element j in the output vector.

2.1 Problems with the Model

The main problem with the existing model is overfitting. It can be seen in the training curve that the training loss continually decreases while the validation loss plateaus and slightly increases after a certain number of epochs. This indicates that although that model is complex enough to continue improving on the training set, it is fitting to the noise of the dataset and therefore unable to generalize well in the validation/test sets. We speculate some model-specific factors that could have contributed to the overfitting or limiting of its performance:

1. The NaN values within the training matrix are replaced with zeros because the autoencoder model cannot work with NaN values. Since zero interprets as an incorrect answer, this inherently adds a bias towards incorrect answers. Formally speaking, the model's inductive bias is that it assumes that all questions with unknown correctness were answered incorrectly. Therefore, the model will learn a poor representation of the true data distribution.

2. The model uses a relatively simplistic architecture, where the autoencoder consists of a single hidden layer – the bottleneck layer. In addition, vanilla techniques such as SGD optimizer and sigmoid activation layers are used, along with a sole L2 regularizer. Potential improvements may be found from exploring more complex models and additional tools to tackle overfitting.

3. Proposed Neural Network Model

From the analysis of the neural network algorithm, the model faces issues with overfitting and poor optimization. The two proposed ways of improving the model performance include augmenting the input data and changing the autoencoder model architecture. The final addition to the current algorithm is stopping training early to further prevent overfitting.

3.1 Metadata Injection

Since the autoencoder takes some row i of the training matrix as input, it is essentially receiving the question data for a specific student i. The proposed model involves concatenating 2 additional pieces of information about each student i to the end of the input vector, specifically their age and premium status, to give the model more input parameters to learn from. The idea is to try to bias the input vector to a certain predisposition about the student based on their given student information. For example, a student will receive an additional 1.0 on their input vector if they are a premium student. The model may learn to associate this 1.0 with a more intelligent/capable student, and therefore reconstruct the input vector such that there are more predictions labeled as correct. In a way, this forces the model to try to encode the additional user data (age and premium status) into the output vector. See Appendix A for how the extra parameters for each student were calculated.

3.2 Model Architecture and Training

The model architecture remains the same except for two key differences. The first difference is the change in dimensions for the input vector from metadata injection. The second difference attempts to address the overfitting by adding dropout. By ignoring some of the units, dropout makes the training process noisier. This forces the other nodes within a layer to probabilistically have a greater effect on the decision-making [1]. Another proposed change is to switch the optimizer from SGD to Adam, which should help decrease training time by utilizing adaptive learning rates and momentum [2]. Additionally, Adam should be able to approximate the generalizability of SGD if the hyperparameters are tuned well.

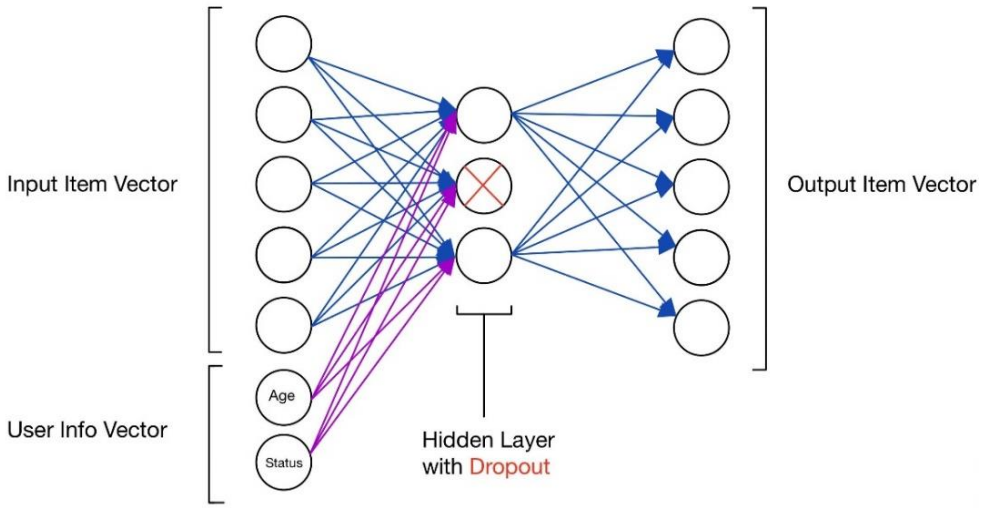


Figure 3.1: Proposed model diagram

3.3 Early Stopping

Previously, we adopted an inefficient approach of constantly re-training models with different epochs in hopes of finding one that converges to the lowest validation loss and immediately concludes, or early stops, to prevent further overfitting. The variability of SGD causing the optimal epoch to be different each time further added to our training overhead. One minor improvement to our code is automating a model save at various checkpoints during training – whenever the validation loss of the current epoch is lower than the previous low. To train a model, we now set a high epoch number for training and load the model with the latest checkpoint afterward for evaluation.

4. Results and Discussion

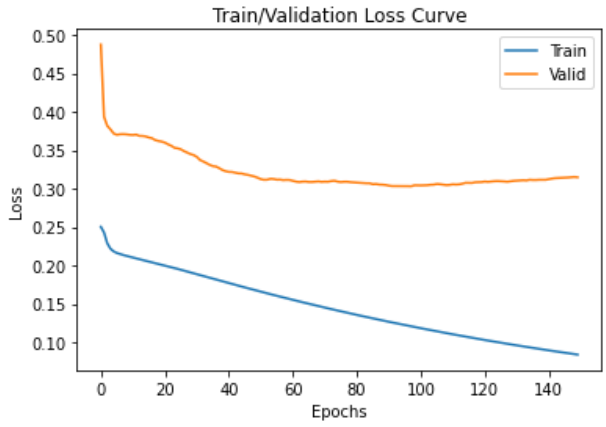


Figure 4.1: Training curve of final model

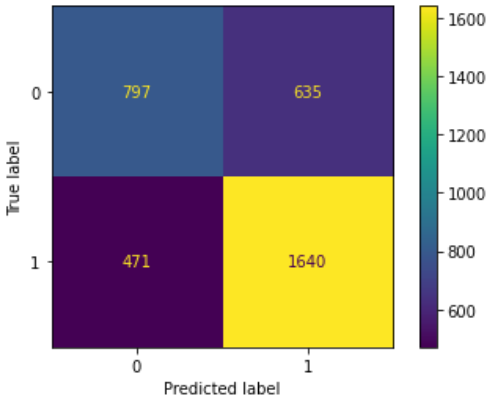


Figure 4.2: Confusion matrix of final model

Table 4.1: Results and hyperparameters of final model

Validation Accuracy	Test Accuracy	Sensitivity	Specificity	Hyperparameters
0.6969	**0.6878**	0.6285	0.7209	- Bottleneck layer size = 50 - Learning rate = 0.0001 - Reg. constant = 0.0001 - Dropout rate = 0.8 - # Epochs = 97

Table 4.2: Results and hyperparameters of final model

Models	Test Accuracy
kNN	0.6842
IRT	**0.7028**
Neural Network	0.6917
Ensemble	0.6954
Proposed	0.6876

The purposed model scores a test accuracy of 0.6878, behind the IRT, ensemble and previous NN baseline models, but beating the kNN model. The highest performing model is the IRT, scoring 0.7028 test accuracy. However, directly compared with our original NN model, the test accuracy is only ~0.4 lower, which could easily be attributed to random variations from the optimizer. Finally, the sensitivity and specificity values are found to be fairly close to one another, suggesting a reasonable tradeoff that supports our current evaluation threshold of 0.5.

Unfortunately, it seems that our changes did not improve the NN model performance. Consistent with our hypothesis however, switching from SGD to Adam sped out training time dramatically. This enabled us to drop the learning rate down one order of magnitude to 0.0001 while being time-efficient, and so that the model could consistently attain the current performance rather than potentially jumping out of the minimum. The best dropout rate was found to be 0.8. Lower values reduced model performance, likely because dropping too many weights in an already small network is not beneficial.

4.1 Limitations of Proposed Model

The decision to implement the user metadata into the training data was based on the assumption that an older user would have more experience with difficult questions and would be likely able to guess questions done by younger users correctly. However, the data fails to represent the different fields each user is comfortable with. For example, an engineering student may be more comfortable with questions involving mechanics than a finance student.

When implementing dropout, the assumption that the other questions within the vector will significantly affect the decision-making probabilities is naïve. As there is no way to further correlate the relationships between the difficulty within the model based on subject, removing some will not help the model detect any patterned relations based on user performance and age alone.

Furthermore, roughly 95% of entries are missing from the sparse matrix. This issue affects the performance of all models because the lack of data causes overfitting and inability to generalize. Potential solutions to this issue may involve obtaining more data or processing the data in some way such that it becomes more meaningful. An example solution could be clustering the users. To implement this, more user metadata would be needed alongside age and premium status, such as occupation, area of study, or even place of residence.

5. More Attempts to Improve Model Performance

<u>Changing the biasing of inputs</u>

Instead of replacing the NaN values in the sparse training matrix with zeros, we replace them with a value that is more informative or fair. We attempted 4 different initialization values, results of which are tabulated below in Table 5.1. The first attempt was initializing the inputs to 0.5, which represents the middle point of the output predictions. The second and third attempts were initialized to 1 and -1 to arbitrarily test the effect of input biasing. The last attempt was initializing to the probability of question correctness, calculated over the corresponding question vector, and excluding any other incomplete entries. For example, if student i has not answered a specific question j, we can look at how question j was answered by other students by calculating: (# of correct / total who answered question j). This is somewhat analogous to adding a prior.

Table 5.1: Comparison of results for various input biasing (Appendix A)

	Validation Accuracy	Test Accuracy	Test Prediction Ratio (Correct/Incorrect)
Initialize to 0 (baseline)	**0.6909**	**0.6917**	**1.596**
Initialize to 0.5	0.6705	0.6672	1.837
Initialize to 1	0.6399	0.6359	1.767
Initialize to -1	0.6605	0.6686	2.342
Initialize to probability of question correctness	0.6617	0.6619	1.992

We found that changing the input biasing to any value other than 0 yielded worse results. One notable observation is that all these trials produce a prediction ratio (on the test set) greater than that of the baseline model: 1.596. In comparison, the ratio of labels found directly in the test set is 1.474. This shows that all models we have trained so far, and especially ones trialed in this section, tend to predict more correct labels than incorrect.

Increasing number of hidden layers

We also attempted to increase the number of hidden layers in the autoencoder from 1 to 3, however, it resulted in significantly poor performance, scoring around 50% validation even after tuning hyperparameters. It is possible that shallower neural networks are more optimal for collaborative filtering due to lower chances of overfitting.

References

[1] J. Brownlee, "A gentle introduction to dropout for Regularizing Deep Neural Networks," *Machine Learning Mastery*, 06-Aug-2019. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Accessed: 03-Dec-2021].

[2] S. Park, "A 2021 guide to improving cnns-optimizers: Adam vs sgd," *Medium*, 21-Jun-2021. [Online]. Available: https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008. [Accessed: 03-Dec-2021].

Appendix A: Reasoning for Incorporating Meta Data

To implement the user info vector, the factors in the student metadata that were considered included premium status and date of birth. A premium student was interpreted as a student who excels academically, so they were assumed likely to get more questions correct. Students with a confirmed premium status got a one appended to their input vectors and students with a known non-premium status got a zero appended to their input vector. Students with unknown statuses were given a value equal to the number of premium students divided by total number of students with confirmed statuses. Moreover, it was assumed that the older the student was, the more likely they are to solve a question correctly. Therefore, all the dates of birth were converted into ages and the students with unknown ages were given the average age in the data set. The ages were then appended to their respective input vectors as values normalized between zero and one. To avoid any gender bias, the gender metadata was withheld. Overall, this process appended a vector of size two to the input vector for the autoencoder.

Appendix B: Training curves for various input biasing

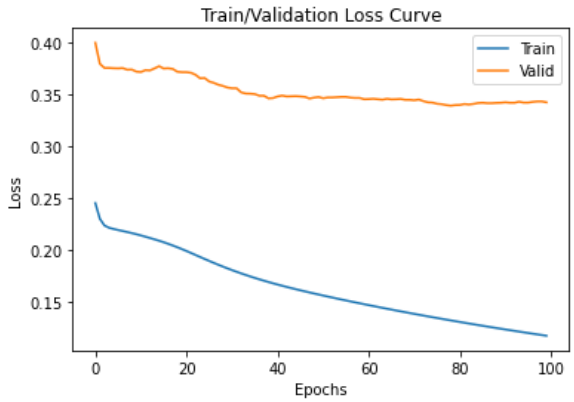


Figure B.1: Initializing missing entries to 0.5

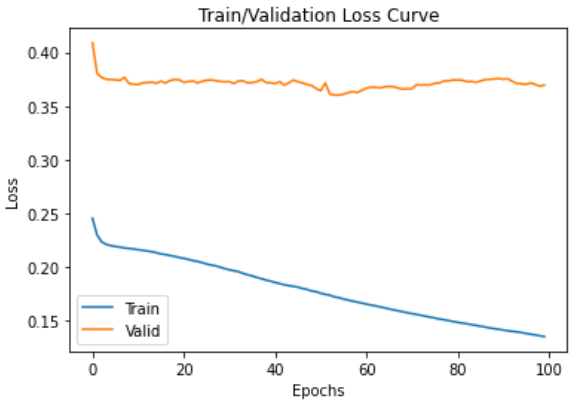


Figure B.2: Initializing missing entries to 1

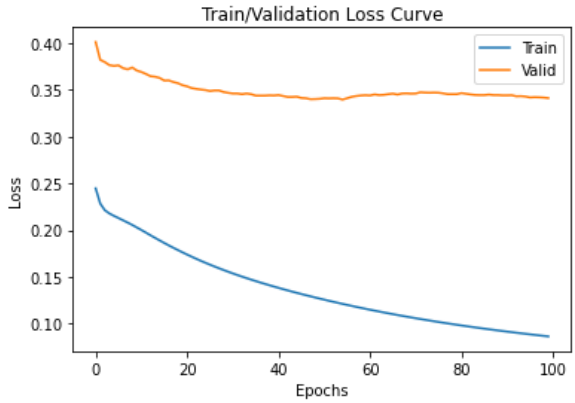


Figure B.3: Initializing missing entries to -1

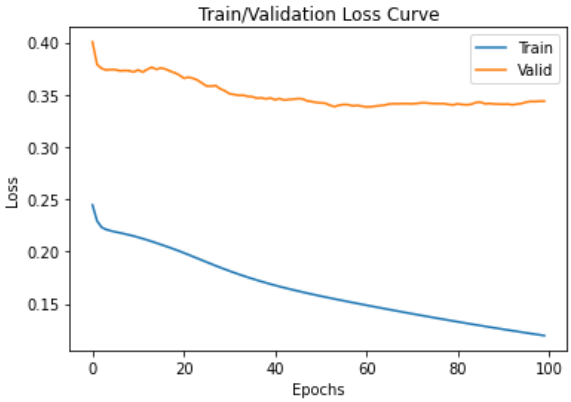


Figure B.4: Initializing missing entries to probability of question correctness