Machine Learning Final Project

Pragnavi Pragnavi Ravuluri Sai Durga
pr2370@nyu.edu

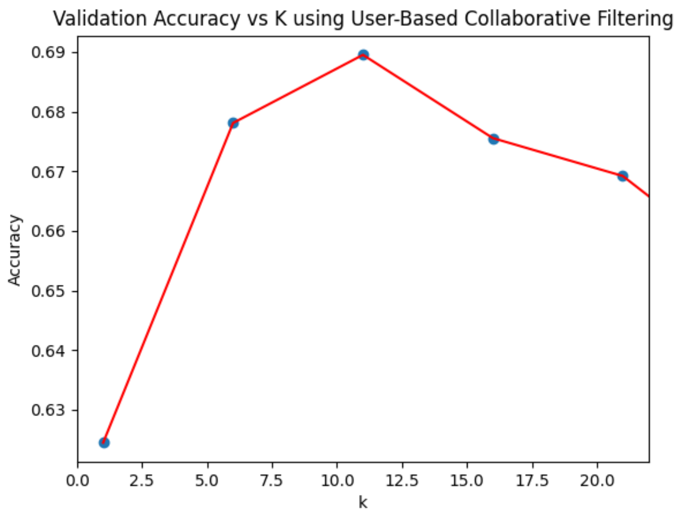
Leuber Leuterio
ll4407@nyu.edu

December 19, 2022

Part A

1. k-Nearest Neighbor

a)

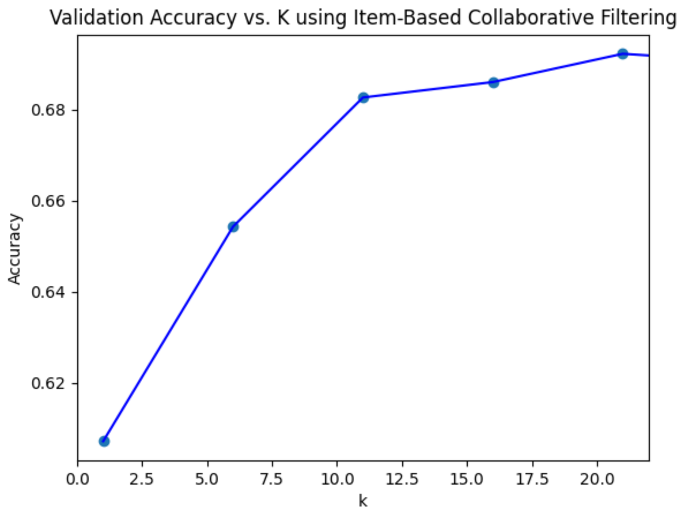


b)

k* = 11
Final Test Accuracy = 0.6841659610499576

c)

KNN finds the k closest questions and the correctness of the student is predicted by averaging the correctness of the k closest questions. Therefore, the underlying assumption for item-based collaborative based filtering is that questions that were answered similarly in the past are more likely to have similar answers of correctness in the future than compared to a randomly chosen question



k* = 21 Final Test Accuracy = 0.6816257408975445

d)

The accuracy for user-based collaborative filtering is 0.6842, whereas the accuracy for item-based collaborative filtering is 0.6816. The test performance for user-based and item-based collaborative filtering is very close, but user-based performs slightly better.

e)

The first limitation of KNN is that it is a slow algorithm. As the data set grows, the memory cost becomes very large and the speed of the algorithm declines very fast. A second limitation of KNN is that it is not very good at dealing with sparse data. If there are questions that have been only answered by a few students, or if there are students that have only answered a few questions, then that limits the ability of KNN to accurately calculate similarities.

2. Item Response Theory

a)

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

$$p(c_{ij} = 0|\theta_i, \beta_j) = \frac{1}{1 + exp(\theta_i - \beta_j)}$$

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

$$p(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{i=1}^{n}\prod_{j=1}^{d} p(c_{ij}|\theta_i, \beta_j)$$

$$\log p(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{n}\sum_{j=1}^{d} \log p(c_{ij}|\theta_i, \beta_j)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{d}(c_{ij}(\theta_i - \beta_j) - \log(1 + exp(\theta_i - \beta_j)))$$

$$\frac{\partial}{\partial \theta_i} \log p(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j=1}^{d}(c_{ij} - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})$$

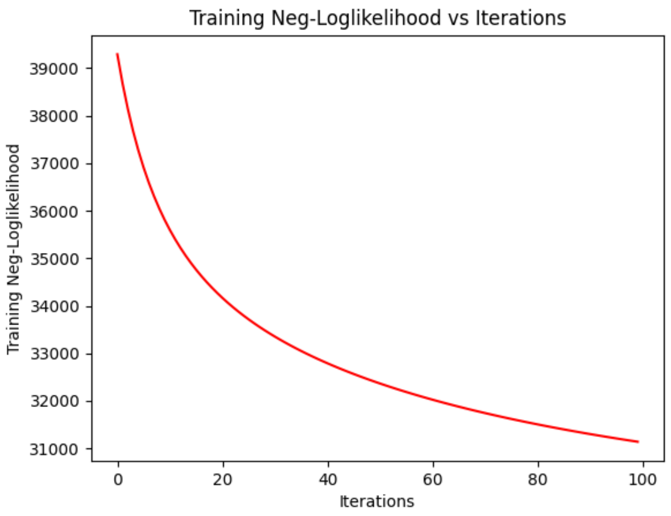
$$\frac{\partial}{\partial \beta_j} \log p(\boldsymbol{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{n}(-c_{ij} + \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})$$

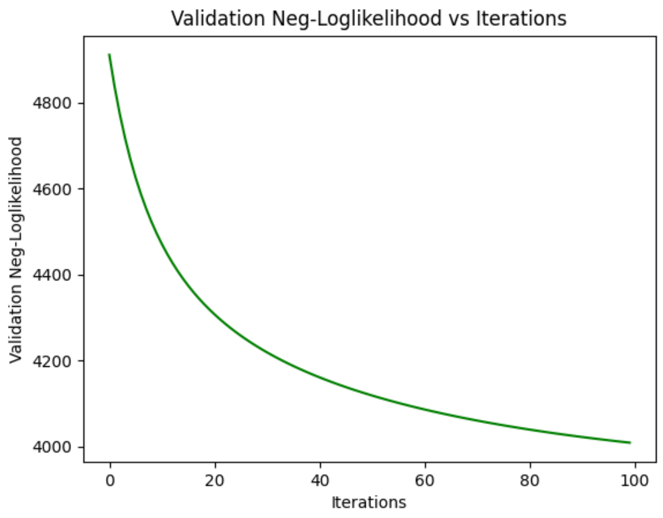
b)

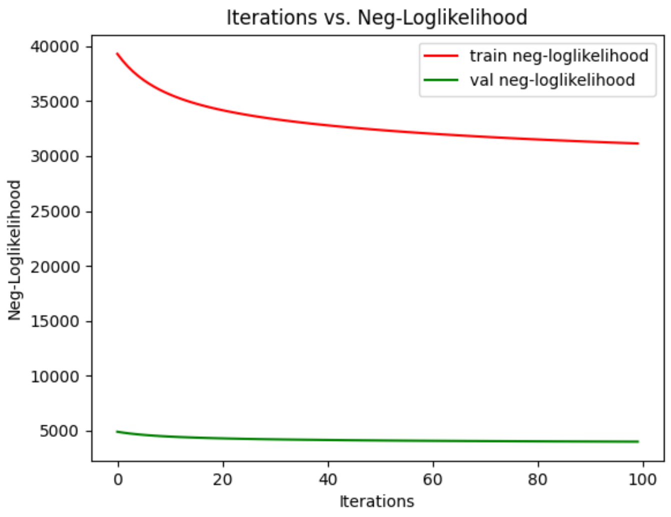
Hyperparameters:

$$lr = 0.001$$

$$k = 100$$



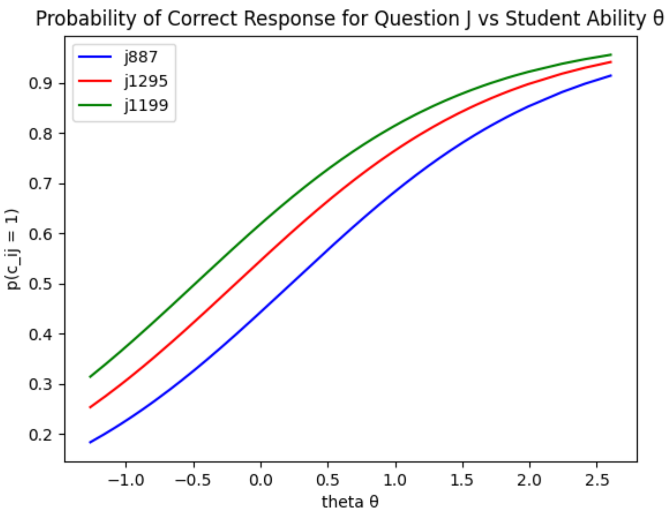




c)

Final Validation Accuracy: 0.7071690657634773
Final Test Accuracy: 0.6994072819644369

d)



We expect the shape of the curves to be sigmoid in shape since the sigmoid function is used to calculate the probability, and we can see that the shapes of the curves are indeed sigmoid. Each question has it's own sigmoid function since they have different levels of difficulty. We observe that user ability theta is positively correlated with answering the question correctly. Therefore, as user ability increases, the probability of answering the question correctly increases. We also observe that:

$$p(c_{i1199} = 1|\theta_i, \beta_{1199}) > p(c_{i1295} = 1|\theta_i, \beta_{1295}) > p(c_{i887} = 1|\theta_i, \beta_{887})$$

Since difficulty is negatively correlated to probability, we conclude that:

$$\beta_{1199} < \beta_{1295} < \beta_{887}$$

3. Neural Networks

a)

```python
class AutoEncoder(nn.Module):
    def __init__(self, num_question, k=100):
        """ Initialize a class AutoEncoder.

        :param num_question: int
        :param k: int
        """
        super(AutoEncoder, self).__init__()

        # Define linear functions.
        self.g = nn.Linear(num_question, k)
        self.h = nn.Linear(k, num_question)

    def get_weight_norm(self):
        """ Return ||W^1|| + ||W^2||

        :return: float
        """
        g_w_norm = torch.norm(self.g.weight, 2)
        h_w_norm = torch.norm(self.h.weight, 2)
        return g_w_norm + h_w_norm

    def forward(self, inputs):
        """ Return a forward pass given inputs.

        :param inputs: user vector.
        :return: user vector.
        """
        #####################################################################
        # TODO:                                                             #
        # Implement the function as described in the docstring.             #
        # Use sigmoid activations for f and g.                              #
        #####################################################################
        #Encode the inputs and apply sigmoid
        coded = torch.sigmoid(self.g(inputs))
        #Decode the encoded repressentation and apply sigmoid
        out = torch.sigmoid(self.h(coded))

        #####################################################################
        #                        END OF YOUR CODE                           #
        #####################################################################
        return out
```

b)

Optimal Hyperparameters:

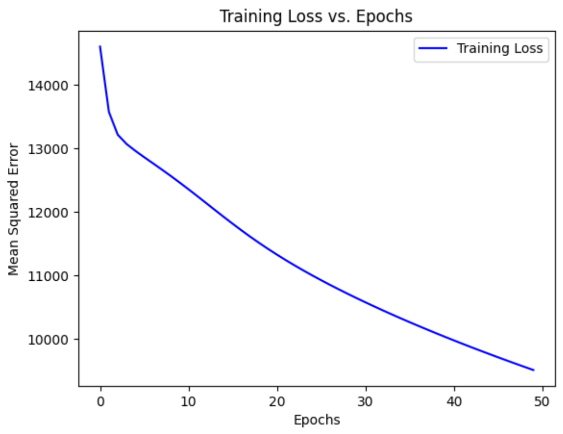
$$lr = 0.01$$

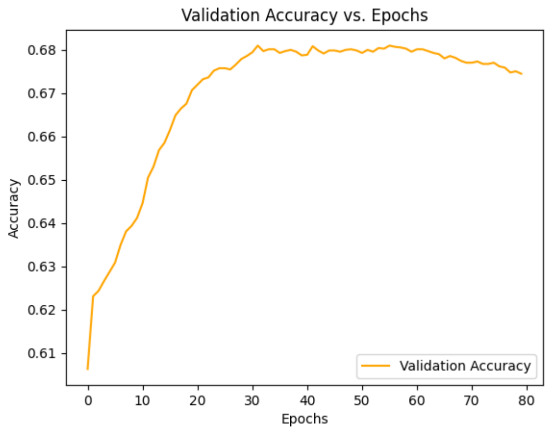
$$\text{num epochs} = 80$$

k* = 50 has the highest validation accuracy
Validation Accuracy = 0.6884

c)

k* = 50
Final Test Accuracy = 0.6895





We observe that training loss decreases as the number of epochs increase. Validation accuracy increases as the number of epochs increase. It is apparent from looking at the graph that the accuracy plateaus at around 0.68 after 30 epochs

d)

Test Accuracy is optimized when $\lambda = 0.001$. The regularized model does not perform better than the unregularized model. The validation and test accuracy are both slightly lower in the regularized version of the model than the unregularized version: accuracy = 0.6802 in the regularized model; accuracy = 0.6895 in the unregularized model.

λ	Validation Accuracy	Test Accuracy
0.001	0.6809	0.6802
0.01	0.6854	0.6749
0.1	0.6853	0.6768
1	0.6246	0.6254

Part B

Neural Networks

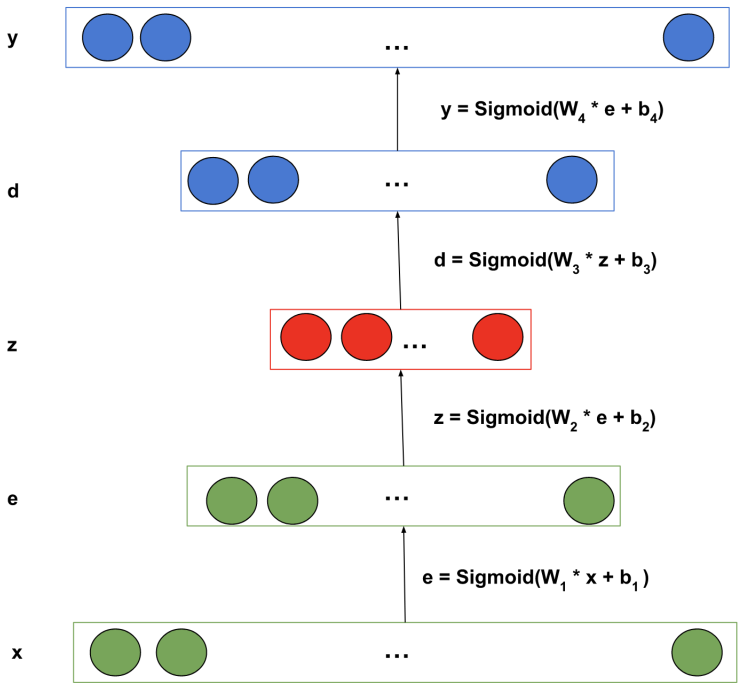
The baseline Neural Network model is an Autoencoder that consists of 3 layers, with the middle layer being the hidden layer. The Auto Encoder has 2 parts: The encoder and the decoder. The Encoder encodes the input into lower dimensions and then the decoder decodes the encoded representation and produces the original input. The input is the student's binary values to each question that indicates whether the students answer was correct.

We extend the Neural Network algorithm in Part A by adding 2 additional hidden layers with activation functions. Whereas the baseline model contained 3 layers, our improvised model will contain 5 layers, with 3 of those being hidden layers. The Autoencoder consists of an encoder layer e, and a decoder layer d, both of which are fused together on the representation layer z. For each activation function, we used the Sigmoid function.

$$e = Sigmoid(W_1 * x + b_1) \tag{1}$$
$$z = Sigmoid(W_2 * e + b_2) \tag{2}$$
$$d = Sigmoid(W_3 * z + b_3) \tag{3}$$
$$y = Sigmoid(W_4 * d + b_4) \tag{4}$$

The additional hidden layers enable the Autoencoder to learn mathematically more complex underlying patterns in the data. As a result of this, we expect that the improvised model will achieve a higher accuracy compared to the baseline model.

Figure or Diagram



Forward pass: During the forward pass, the model takes user input x from the training set. **x** is a vector of binary indicators that indicate the student's correctness to each question. The layers in this model are: x, e, z, d, y. Layers e, z, and d are hidden layers. The sigmoid function is applied on all layers. After traversing the neural network, **y** is output.

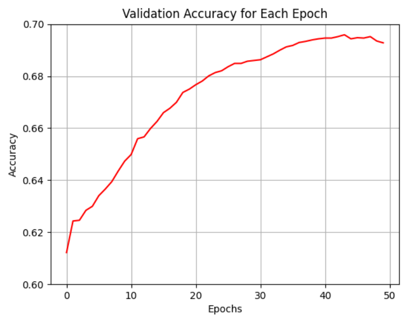
Comparison or Demonstration

Table 1 compares the validation and test accuracies of the models in part A with the validation and test accuracies of the extended model. We observe that the extended model has a higher performance in terms of test accuracy than all other models, including the baseline model.

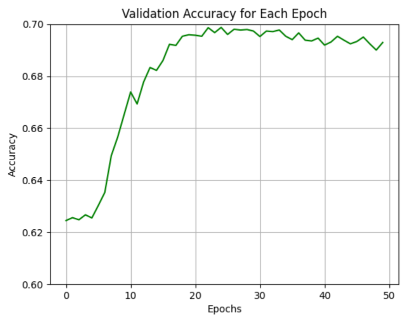
Figure 1 compares the validation accuracies of the extended model and the baseline model. Figure 2 compares the validation and training losses between those models. We observe that validation Accuracy is higher in the extended model than in baseline. Also, training loss has significantly improved in the extended model over a number of epochs. As a result, we conclude that the extended model performs better than the baseline model.

Model	Validation Accuracy	Test Accuracy
KNN	0.6895	0.6842
IRT	0.7072	0.6994
Baseline Neural Network	0.6943	0.6966
Extended Neural Network	**0.6945**	**0.7042**

Table 1: Accuracies of Extended Model Versus Models in Part A

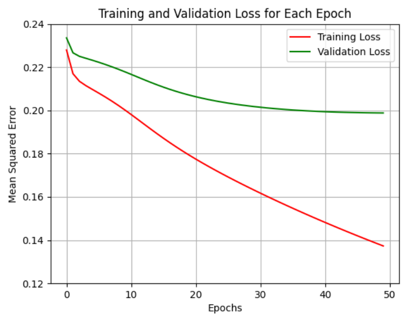


(a) Baseline Model

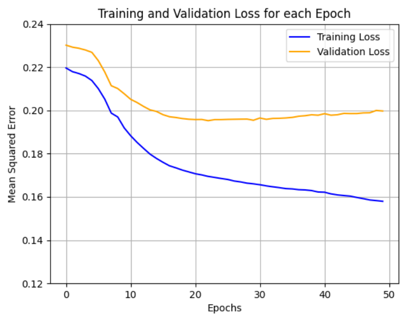


(b) Extended Model

Figure 1: Validation Accuracy in Baseline and Extended Models



(a) Baseline Model



(b) Extended Model

Figure 2: Training and Validation Loss in Baseline and Extended Models

The additional hidden layers in the extended model enables it to learn more patterns in the data. Therefore, we hypothesize that the increase in accuracy observed in the extended model is mainly due to optimization rather than regularization. In order to test this, we conducted an experiment by removing the regularization penalty.

Model	λ	Validation Accuracy	Test Accuracy
Baseline	0.001	0.6809	0.6802
Extended	0.001	0.6929	0.7031
Extended	0.000	0.6938	0.7019

Table 2: Comparing the Accuracies in the Extended Model after removing the regularization penalty

When removing the regularization penalty in the extended model, the test accuracy decreases slightly. However, the test and validation accuracy in the unregularized version of the extended model is higher than the regularized baseline model. This supports our claim that the benefits in the extended model are mainly due to optimization.

Limitations

One of the challenges faced by our model and other neural network models is the tendency to over fit. With our model, as training loss continues to decrease, the validation loss decreases initially, but then starts to increase around 30 epochs (Figure 2). This pattern is indicative of overfitting, which requires the model to stop early in order to prevent. However, stopping early means that the model may suffer from underfitting since the model may not have enough time to learn the relevant features in the data.

Another limitation of our model is that we would not expect it to perform well if data sparasity of the training matrix is large. This occurs when there is a large number of hold-outs or unknown responses.

We guess the reason why the extended model is prone to overfitting, is because of the fact that the number of parameters in the the model grows linearly with both the number of users and the number of questions. This makes it more prone to overfitting.

Our model performs poorly when the data is too sparse because if a user has a large number of nulls, then the weights and biases of our model will not be changed after training on that particular user. As a result, our model will not be able to learn much from this user.

A possible extension of our model is to add additional hidden layers to further increase accuracy. Adding more hidden layers means that the number of parameters grows, and the model is more prone to overfitting. A way to address this is to hyptertune parameters as additional layers are added. Also, we could employ other methods, such as early dropout and considering using different activation functions.