CSC311 Project

Elaine Sui (1004830613); Ian Jiang (1005415668)

December 15, 2020

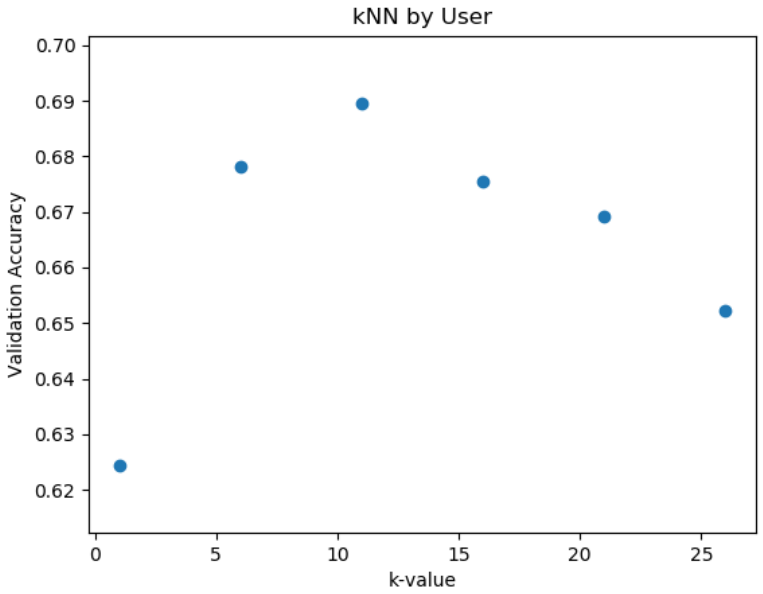
Contributions:

Part A: We did all the questions independently, and then discussed the solutions and put the report together. We eventually used Elaine's code, and both of us produced and edited part of the report (Work split: **50/50**)
Part B: Initially, both of us experimented and tested out different approaches. Once we decided our extension, Ian produced the diagram, the limitations, and a small part of formal description and demonstration. Elaine produced the majority of formal description and demonstration, and the code. Both had the chance to edit each others work. (Work split: **50/50**)

Part A

1. (a)

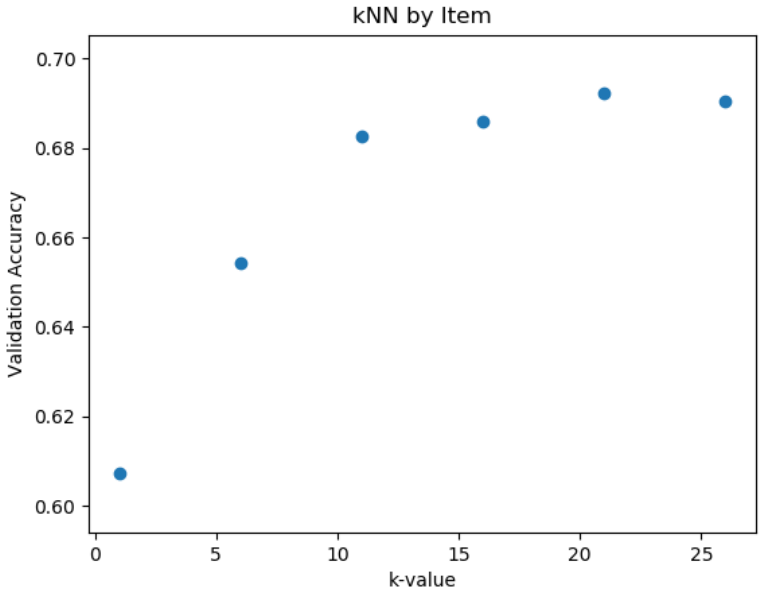
k-value	1	6	11	16	21	26
Validation Accuracy	0.62447	0.67810	0.68953	0.67556	0.66920	0.65227



(b) $k^* = 11$
Final test accuracy: 0.68416

(c) The core underlying assumption is that if question A has the same correct and incorrect answers by other students as question B, the specific student's correctness on question A would match that of his/her correctness on question B.

k-value	1	6	11	16	21	26
Validation Accuracy	0.60711	0.65424	0.68261	0.68600	0.69220	0.69038



$k^* = 21$
Final test accuracy: 0.68163

(d) User-based collaborative filtering performs slightly better on the test set in comparison to its item-based counterpart, suggesting that similar users are more predictive of performance than similar questions.

(e) Firstly, we know kNN-imputer with NaN Euclidean distance works best under the assumption that the missing data are missing completely at random (MCAR) so we don't introduce any more biases. However, this is probably not the case. If students encounter a difficult question, it is very likely that they will simply not answer that question because they don't know how to approach it at all. In other words, it is proper to assume that the majority of students who answer difficult questions are more confident in their ability. This means these hard questions will most likely have more missing data. Thus, if the number of students who answered such a difficult question is less than our tuned value of k, if we impute by user, every student who did not answer that question would be assigned the average correctness of those who responded to it, which are most likely students with higher competence. Therefore, the algorithm will think everyone can answer that question relatively well when in fact only competent students chose to answer the question in the first place, causing us to have inaccurate predictions for that particular question.

Secondly, our current implementation of kNN scales each dimension equally. More specifically, if we are imputing by user, each question is considered equally important in its ability to distinguish "good" students from "bad" students. This is not reflective of real questions, as for instance, thinking questions are designed to reveal students' actual understanding of a topic whereas knowledge and understanding questions only reveal that the students can memorize facts. Ideally, we want those more "discriminative" questions to be weighed more with basic or easy questions weighed less. This way, we are much more likely to get predictions with higher accuracy.

2. (a) Let $\mathcal{O} = \{(i,j) : \text{entry } (i,j) \text{ of matrix } \mathbf{C} \text{ is observed}\}$

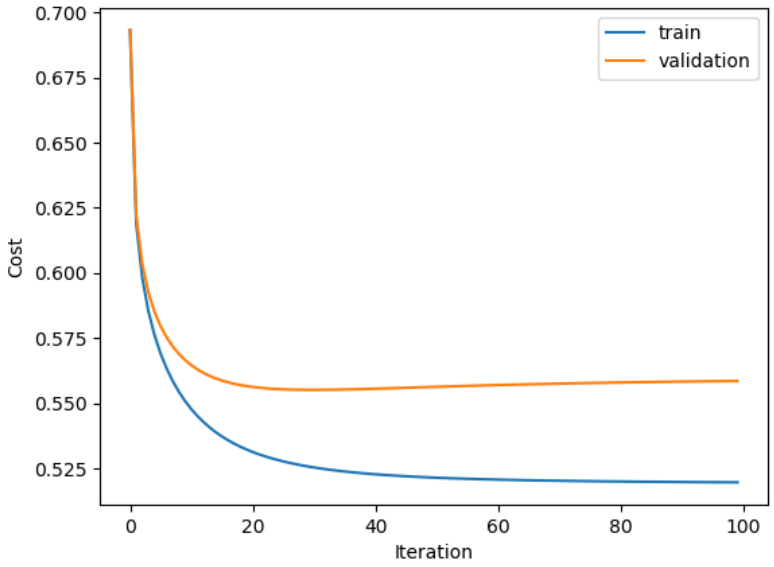
$$
\begin{aligned}
p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) &= \prod_{(i,j)\in\mathcal{O}} p(c_{ij} \mid \theta_i, \beta_j) \\
&= \prod_{(i,j)\in\mathcal{O}} p(c_{ij} = 1 \mid \theta_i, \beta_j)^{c_{ij}} p(c_{ij} = 0 \mid \theta_i, \beta_j)^{1-c_{ij}} \\
&= \prod_{(i,j)\in\mathcal{O}} \Big(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\Big)^{c_{ij}} \Big(1 - \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\Big)^{1-c_{ij}} \\
&= \prod_{(i,j)\in\mathcal{O}} \Big(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\Big)^{c_{ij}} \Big(\frac{1}{1+e^{\theta_i-\beta_j}}\Big)^{1-c_{ij}} \\
\log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) &= \sum_{(i,j)\in\mathcal{O}} \Big[c_{ij}\Big(\theta_i - \beta_j - \log\big(1+e^{\theta_i-\beta_j}\big)\Big) + (1-c_{ij})\Big(-\log\big(1+e^{\theta_i-\beta_j}\big)\Big)\Big] \\
&= \sum_{(i,j)\in\mathcal{O}} \Big[c_{ij}(\theta_i - \beta_j) - \log\big(1+e^{\theta_i-\beta_j}\big)\Big] \\
&= \sum_{i=1}^{N_S} \theta_i \sum_{j:(i,j)\in\mathcal{O}} c_{ij} - \sum_{j=1}^{N_Q} \beta_j \sum_{i:(i,j)\in\mathcal{O}} c_{ij} - \sum_{(i,j)\in\mathcal{O}} \log\big(1+e^{\theta_i-\beta_j}\big)
\end{aligned}
$$

$$
\frac{\partial}{\partial\theta_i} \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j:(i,j)\in\mathcal{O}} \Big(c_{ij} - \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\Big)
$$

$$
\frac{\partial}{\partial\beta_j} \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i:(i,j)\in\mathcal{O}} \Big(-c_{ij} + \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\Big)
$$

(b) **Hyperparameters:**
Learning rate: 0.01 **Number of iterations:** 100
Initial theta: Vector of 0's **Initial beta:** Vector of 0's

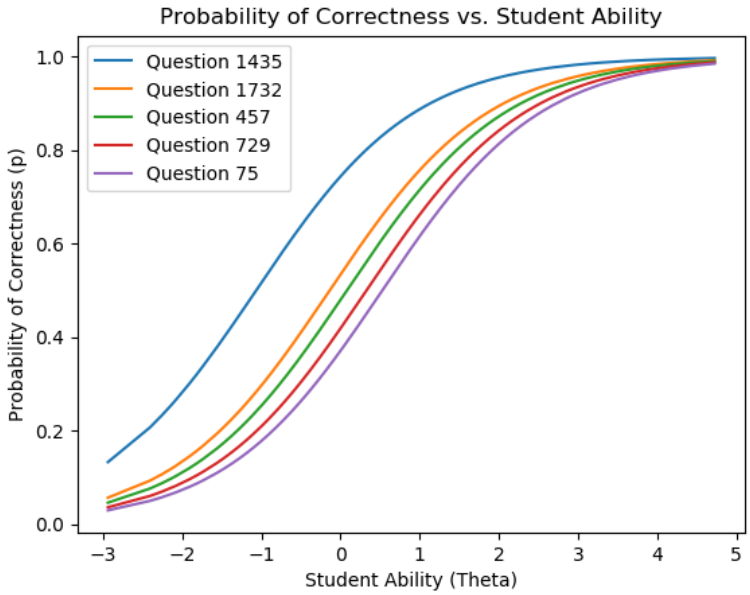
We tuned these hyperparameters by running the model with learning rates in {0.001, 0.005, 0.01, 0.05, 0.1} while keeping the number of iterations arbitrarily fixed at 200 and initializing theta and beta to be vectors of 0's. We plotted the validation cost and accuracy curves for each learning rate and found that the validation cost curve for learning rate of 0.01 seemed to converge fastest and with good stability (too high, validation cost oscillates; too small, validation cost converges too slowly). Then, from the accuracy plots, we observed that the model with learning rate = 0.01 converges at around 100 iterations, so we set number of iterations to 100. Afterwards, we ran the tuned model with fixed (vector of all 0's) and random initialization of theta and beta (with values between -1 and 1), and found that the cost and accuracy curves converge approximately to the same values. So, we opted for the fixed initialization of zeroes for simplicity.



Average Negative Log-Likelihood

(c) **Validation accuracy:** 0.70590 **Test accuracy:** 0.70731

(d)



The shape of these curves resemble relatively flat S's. These curves represent the probability that a given question is responded correctly at each ability level. More specifically, we see that these curves illustrate a positive correlation between student ability and the probability of correctly responding to a given question. Also, the higher curves would represent easier question, as students across all abilities have a higher probability of getting the correct answer. We also see that starting from the lowest student ability, as student ability increases, the probability of correctness first increases at a slow rate, then at an higher rate and then at a slower rate. Thus, if student ability is either really low or really high, the model assumes that (possibly great) improvement in student ability will only slightly increase the probability

of correctness. On the contrary, if student ability is roughly average (on this scale, between -1 and 1), the model assumes that only slight improvement in ability can lead to a much greater increase in the probability of correctness.

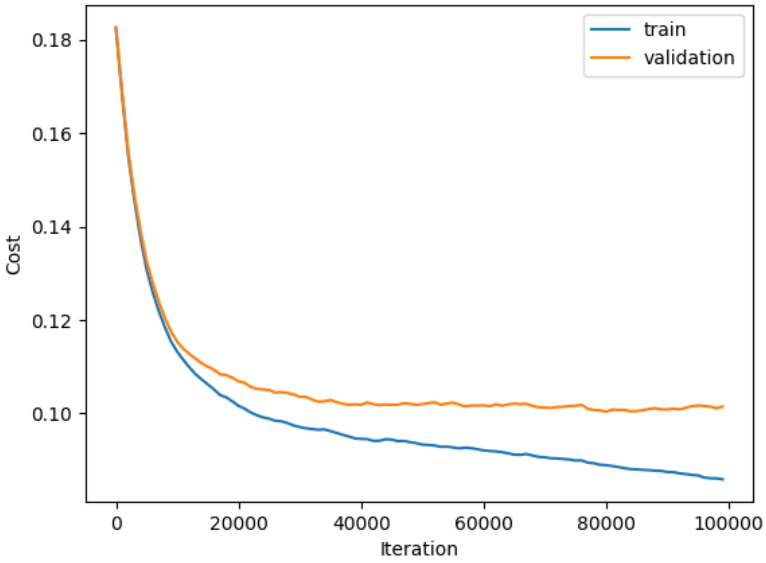
3. (a) **Best k:** 9
Final validation accuracy: 0.66130
Final test accuracy: 0.65876

(b) SVD requires a completed matrix. In our task, we are treating the missing entries as the average of the entries in its column. That is, we are treating them to be the average user correctness on that question among those who have answered it. That means, the way in which we fill these missing entries depends heavily on the correctness of the questions that have been answered and thus, may be improperly skewed when there are limited responses available. If only a single user answered a particular question and answered it correctly, we would treat all users to have answered it correctly, even though that particular user may have been an outlier. Moreover, if no one answers a particular question, we fill those entries as 0. That is, we assume all those students answered it incorrectly. Thus, when we decompose our filled matrix, these very heavily skewed entries will directly affect the "principal components" chosen, leading to less accurate results.

(d) **Hyperparameters:**
Learning rate: 0.1 **Number of iterations:** 100,000 $k^* = 20$

Since **U** and **Z** are initialized randomly, we used a random seed in order to compare between different models. For the learning rate, we first fixed the number of iterations to 15,000 and tried different learning rates in $\{0.5, 0.3, 0.1, 0.05, 0.01\}$. We found out the cost function for learning rate at 0.05 and 0.01 are decreasing too slowly and the validation accuracy is low as well. For learning rate at 0.5, the validation accuracy is also low as the cost function starts to oscillate. Then, for both learning rates at 0.1 and 0.3, we tried to increase the number of iterations to $N \in \{30,000,\ 50,000,\ 100,000,\ 150,000\}$. We observed that at learning rate at 0.3, the algorithm reaches its peak with validation accuracy of 0.65 around 15,000 iterations and never improves. For learning rate at 0.1, the validation accuracy continuously improved until it reached its peak at 100,000 iterations. Therefore, we decide to use a learning rate of 0.1 and 100,000 iterations. Then, fixing the learning rate and number of iterations, we tried k values in $\{10, 20, 40, 60, 80\}$ and discovered that $k = 20$ achieved the highest validation accuracy.

(e)



Both training and validation cost curves show a steep descent in the first 20,000 iterations. The training curve continues to descend at a relatively slower, but constant rate from 20,000 to 100,000 iterations.

The validation curve descends at a much slower rate than the training curve, plateauing at around 90,000 iterations.

Final Validation Accuracy: 0.69616 **Final Test Accuracy:** 0.69433

(f) We would make the loss function a logistic-cross-entropy-loss. That is,

$$\begin{aligned}\mathcal{L}(\mathbf{U},\mathbf{Z},\mathbf{C}) &= \sum_{(i,j)\in\mathcal{O}} -C_{ij}\log\big(\sigma(\mathbf{u}_i^{\mathrm{T}}\mathbf{z}_j)\big) - (1-C_{ij})\log\big(1-\sigma(\mathbf{u}_i^{\mathrm{T}}\mathbf{z}_j)\big)\\ &= \sum_{(i,j)\in\mathcal{O}} C_{ij}\log\Big(1+e^{-\mathbf{u}_i^{\mathrm{T}}\mathbf{z}_j}\Big) + (1-C_{ij})\log\Big(1+e^{\mathbf{u}_i^{\mathrm{T}}\mathbf{z}_j}\Big)\end{aligned}$$

where $\sigma(z)=\frac{1}{1+e^{-z}}$, $\mathbf{C}$ is the sparse matrix and $\mathcal{O}=\{(i,j):\text{entry }(i,j)\text{ of matrix }\mathbf{C}\text{ is observed}\}$

As this is a classification problem, we do not want very confident predictions to be penalized and thus, we constrain the predictions to be within [0, 1] by applying the logistic function on $\mathbf{u}_i^{\mathrm{T}}\mathbf{z}_j$. Moreover, we also want the penalty for predictions to be proportional to its degree of incorrectness. That is, we want to penalize more incorrect predictions more heavily. We accomplish this by using the cross-entropy loss function.

4. We implemented bagging using three IRT models. For each base model, we sampled the training set to get the same number of training points with replacement. Since the bootstrapped training set can have duplicates, we also kept a *count_matrix* which counted the number of times each $(user, question)$ pair was sampled. In order to train the model in a training set with duplicate points, we modified the base IRT model so that the duplicates were counted accordingly. That is, the log-likelihood function and the gradients were the following:

$$\log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i=1}^{N_S} \theta_i \sum_{j:(i,j)\in\mathcal{B}} (c_{ij} \cdot count_{ij}) - \sum_{j=1}^{N_Q} \beta_j \sum_{i:(i,j)\in\mathcal{B}} (c_{ij} \cdot count_{ij}) - \sum_{(i,j)\in\mathcal{B}} (\log\big(1 + e^{\theta_i - \beta_j}\big) \cdot count_{ij})$$

$$\frac{\partial}{\partial \theta_i} \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{j:(i,j)\in\mathcal{B}} \Big(c_{ij} - \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\Big) \cdot count_{ij}$$

$$\frac{\partial}{\partial \beta_j} \log p(\mathbf{C} \mid \boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i:(i,j)\in\mathcal{B}} \Big(- c_{ij} + \frac{e^{\theta_i - \beta_j}}{1 + e^{\theta_i - \beta_j}}\Big) \cdot count_{ij}$$

where $\mathcal{B}$ is the bootstrapped training set (with duplicates) and $count_{ij}$ is the number of times $(user, question)$ pair (i, j) appears in $\mathcal{B}$.

We generate three bootstrapped training sets and we train the base model independently on each of these sets, resulting in three prediction matrices that we average. We ran the ensemble 5 times, getting the following results:

Ensemble	Val Acc 1	Val Acc 2	Val Acc 3	Bagged Val Acc	Bagged Test Acc
1	0.69715	0.69362	0.69546	0.70406	0.70844
2	0.69433	0.69179	0.69927	0.70265	0.70195
3	0.69743	0.69108	0.70040	0.70223	0.71183
4	0.69828	0.70110	0.70040	0.70717	0.69941
5	0.69828	0.69207	0.69800	0.70124	0.70533

Avg Bagged Val Acc	Avg Bagged Val Std	Avg Bagged Test Acc	Avg Bagged Test Std
0.70347	0.00206	0.70539	0.00444

Given the low standard deviations, the ensemble has high stability, as modifications to the bootstrapped training sets does not change the observed test accuracies by much. As observed from its mean loss graph in question 2, validation error curve increases only slightly as training continues, so the base model is likely to have low variance. Since bagging serves to reduce variance, this expected reduction in variance essentially has no effect on the model performance. Although the averaged bagged validation and test accuracies are slightly lower than the corresponding statistics for the base model, we can attribute this to the randomness of bagging. As well, we many attribute this to the scarcity of the bootstrapped data, causing the model to have even less unique data to learn relevant features from. That said, from the standard deviation of the averaged bagged validation and test accuracies, we see that this decrease from the average is negligible.

Part B

Formal Description

We extend the Item-Response Theory base model by modeling the interaction between student ability and question difficulty as a dot product of latent vectors with biases rather than a difference of scalar entries. That is, we assumed the probability that question j is answered correctly by student i is:

$$p(c_{ij} = 1 \mid \mathbf{u}_i, \mathbf{z}_j, \mathbf{b}_{S,i}, \mathbf{b}_{Q,j}) = \frac{e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}}}{1 + e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}}},$$

where $\mathbf{b}_{S,i}$ and $\mathbf{b}_{Q,j}$ are scalars that represent the biases or fixed characteristics for student i's ability and question j's difficulty. These two parameters correspond to θ_i and β_j from the base model. The $k \times 1$ vectors $\mathbf{u}_i$ and $\mathbf{z}_j$ parametrize student i's and question j's additional latent characteristics, respectively. This is an extension to the base model of 1-parameter IRT (Rasch Model) because if we set $\mathbf{u}_i$ and $\mathbf{z}_j$ to be zero vectors, $\mathbf{b}_{S,i} = \theta_i$, and $\mathbf{b}_{Q,j} = -\beta_j$, we arrive at the base model from Part A.

Thus, the probability that we observe the entire correctness matrix $\mathbf{C}$ would be:

$$p(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = \prod_{(i,j) \in \mathcal{O}} \left(\frac{e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}}}{1 + e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}}} \right)^{c_{ij}} \left(\frac{1}{1 + e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,} + \mathbf{b}_{Q,j}}} \right)^{1 - c_{ij}}$$

where $\mathbf{U}$ is the $n \times k$ matrix of student latent traits, $\mathbf{Z}$ is the $m \times k$ matrix of question latent traits, $\mathbf{b}_S$ is the $n \times 1$ vector describing student biases or abilities and $\mathbf{b}_Q$ is the $m \times 1$ vector describing question biases or difficulties.

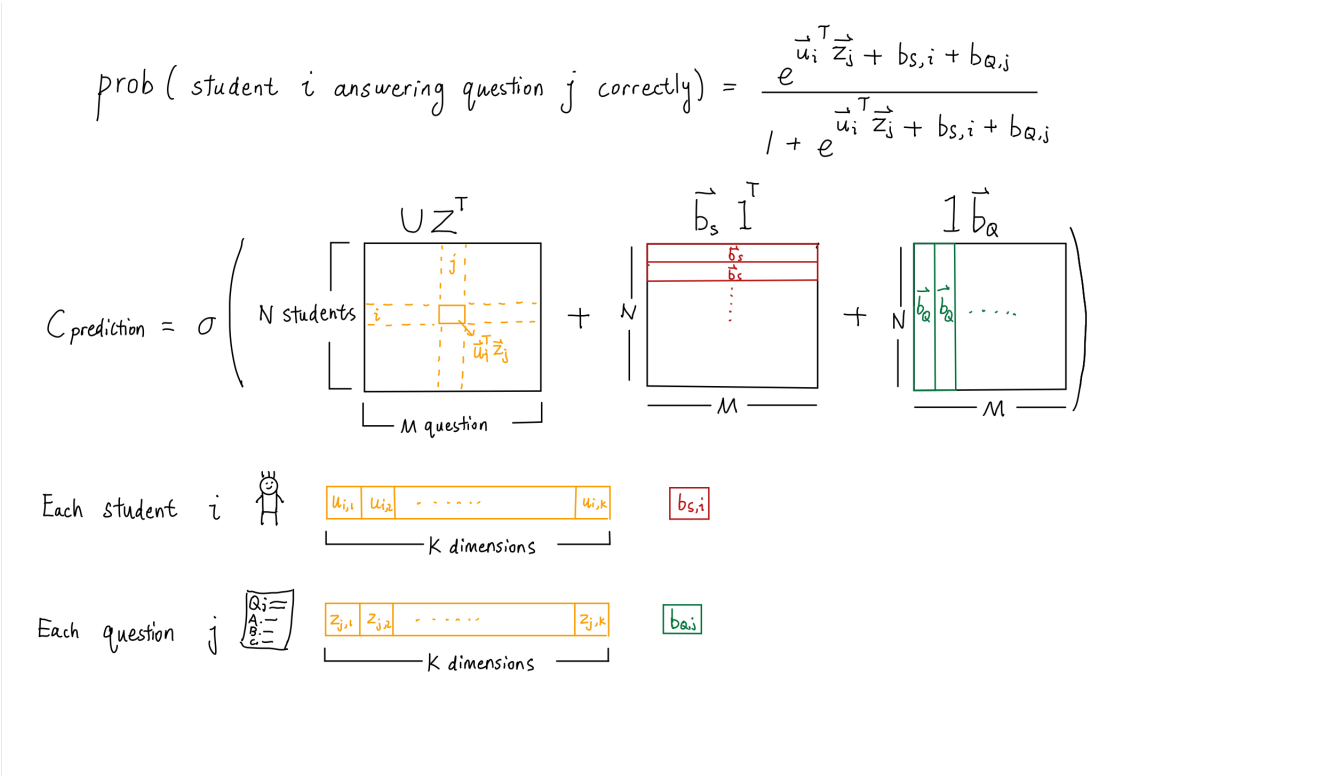


Figure 1: Model Diagram Explanation

Interestingly, we can also interpret this model as a Matrix Factorization model with biases for the latent factors fitted inside a sigmoid function. Equivalently, we are assuming that $\mathbf{C} \approx \sigma(\mathbf{U}\mathbf{Z}^T + \mathbf{b}_S \mathbf{1}^T + \mathbf{1}\mathbf{b}_Q^T)$.

Looking at results from Part A, we suspect that the IRT model is underfitting, as the mean negative log-likelihood seemed to plateau at just under 0.525, indicating that the model has not learned all the relevant features of the data. This might be because one scalar parameter isn't enough to capture a student's or a question's characteristics. As a result, by increasing the number of features that parametrize a student's ability or other latent traits and a question's difficulty or other latent traits, we expect that the model to have a greater capacity, be more expressive, and will learn more correct nuances of the training data. However, as adding more parameters increases the risk of overfitting, we also added L2 regularization on all our parameters. By forcing all parameters to be relatively small, we attempt to prevent the model from memorizing the training data.

Log-likelihood:

$$\log p(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = \sum_{(i,j)\in\mathcal{O}} \Big[c_{ij}(\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}) - \log\Big(1 + e^{\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}}\Big)\Big]$$

Loss function:

$$\mathcal{L}(\mathbf{C}_{train}, \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = -\log p(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) + \lambda_U \|\mathbf{U}\|_F^2 + \lambda_Z \|\mathbf{Z}\|_F^2 + \lambda_S \|\mathbf{b}_S\|^2 + \lambda_Q \|\mathbf{b}_Q\|^2$$

We optimized the above using gradient descent by subtracting the original parameters by the following gradient multiplied by the learning rate for N iterations (vectorized for simplicity):

$$\frac{\partial}{\partial \mathbf{U}} \mathcal{L}(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = -\Big((\mathbf{C}_{train} - \sigma(\mathbf{U}\mathbf{Z}^T + \mathbf{b}_S \mathbf{1}^T + \mathbf{1}\mathbf{b}_Q^T)) \otimes \mathbf{V}\Big)\mathbf{Z} + \lambda_U \mathbf{U}$$

$$\frac{\partial}{\partial \mathbf{Z}} \mathcal{L}(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = -\Big((\mathbf{C}_{train} - \sigma(\mathbf{U}\mathbf{Z}^T + \mathbf{b}_S \mathbf{1}^T + \mathbf{1}\mathbf{b}_Q^T)) \otimes \mathbf{V}\Big)^T \mathbf{U} + \lambda_Z \mathbf{Z}$$

$$\frac{\partial}{\partial \mathbf{b}_S} \mathcal{L}(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = -\Big((\mathbf{C}_{train} - \sigma(\mathbf{U}\mathbf{Z}^T + \mathbf{b}_S \mathbf{1}^T + \mathbf{1}\mathbf{b}_Q^T)) \otimes \mathbf{V}\Big)\mathbf{1} + \lambda_S \mathbf{b}_S$$

$$\frac{\partial}{\partial \mathbf{b}_Q} \mathcal{L}(\mathbf{C}_{train} \mid \mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q) = -\Big((\mathbf{C}_{train} - \sigma(\mathbf{U}\mathbf{Z}^T + \mathbf{b}_S \mathbf{1}^T + \mathbf{1}\mathbf{b}_Q^T)) \otimes \mathbf{V}\Big)^T \mathbf{1} + \lambda_Q \mathbf{b}_Q$$

where $\otimes$ is element-wise multiplication, σ is the sigmoid function, $\mathbf{C}_{train}$ is the sparse training matrix with NaNs zeroed out and $\mathbf{V}$ is the matrix whose entries are 1 if the corresponding entry in $\mathbf{C}_{train}$ is finite (not NaNs), 0 otherwise. We've omitted the algorithm box because the algorithm is almost identical to the base model and the description above should already give a very clear picture.

Comparison and Demonstration

Tuned Hyperparameters:
Learning rate: 0.01
Number of iterations: Early stopping when validation negative log-likelihood increases (18 iterations)
Latent dimensions k: 5
Regularization constants: $\lambda_U = \lambda_Z == \lambda_S = \lambda_Q = 0$
Initial student latent matrix (U): Random numbers between 0 and $1/\sqrt{k}$
Initial question latent matrix (Z): Random numbers between 0 and $1/\sqrt{k}$
Initial student biases ($\mathbf{b_S}$): Vector of 0's **Initial question biases ($\mathbf{b_Q}$):** Vector of 0's

Refer to the Appendix to see more details on how we tuned our hyperparameters.

The following graphs should the training and validation loss and accuracy curves for the baseline model and the extended model. **Note that the solid lines go up until early stopping and the dashed lines show the trends if training were to continue.**

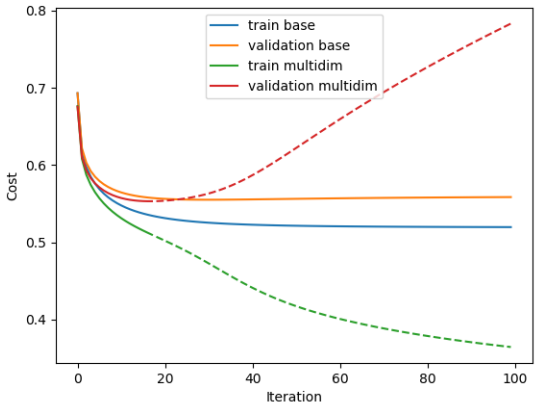


Figure 2: Cost Curves

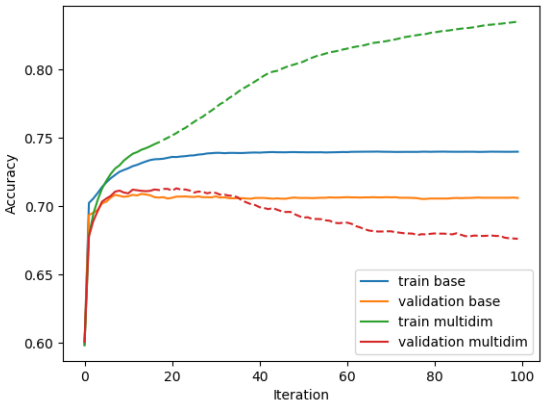


Figure 3: Accuracy Curves

The following tables show the numerical loss and accuracy statistics after training using the tuned hyperparameters for each respective model. Since our regularized mutlidimensional IRT model uses random initialization of entries in the **U** and **Z** latent matrices, we report the average statistics after running our model 5 times.

Model	**Training**	**Validation**
Base (1-parameter IRT)	0.51976	0.55863
Multidimensional IRT	0.51031	0.55268

Table 1: (Averaged) Cost

Model	**Training**	**Validation**	**Test**
Base (1-parameter IRT)	0.73996	0.70590	0.70731
Multidimensional IRT	0.74833	0.71197	0.71166

Table 2: (Averaged) Accuracies

Previously, we have argued that our extension should help reduce underfitting by adding more parameters, that adding more parameters increases the risk of overfitting, and that regularization on our parameters should help reduce the overfitting caused by adding more parameters. We will carry out experiments to test these hypotheses. As the entries of latent matrices **U** and **Z** are initialized at random, we use a random seed to control for this randomness when comparing different models.

Hypothesis 1: Adding more parameters to the baseline model will reduce underfitting

In order to test this hypothesis, we will compare the training cost and accuracy for different values of k (where $k = 0$ is equivalent to the baseline model). We specifically choose the unregularized version of our extended model in order to isolate the effect of adding more parameters. We expect that as training continues, the training cost curves of our extended models should decrease and converge to a lower cost than the baseline and the training accuracy to increase and converge to a higher accuracy than baseline. This would illustrate that the increase in the number of features that parametrize a student and a question increases the model's capacity to learn. Moreover, we expect that for larger values of k, the training loss and training accuracy converge at a faster rate and at lower and higher values, respectively.

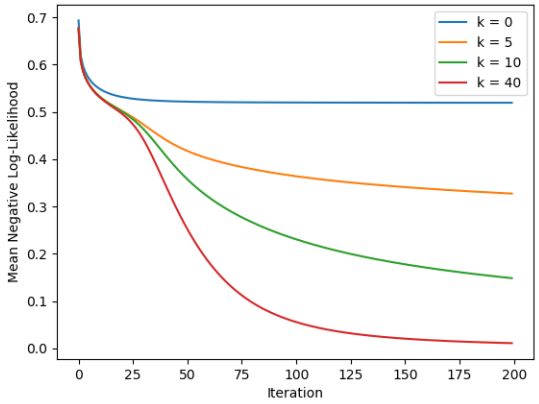


Figure 4: Experiment 1 Training Cost Curves

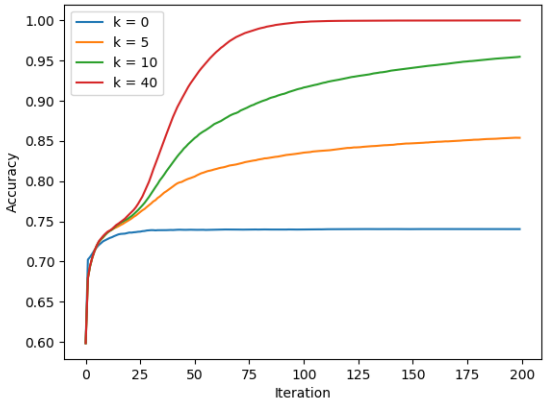


Figure 5: Experiment 1 Training Accuracy Curves

Indeed, we see what we expect, demonstrating how adding more parameters increases the model's capacity to learn. Moreover, we see that having too many parameters and/or training for too long results in the model memorizing the training data, as shown by training loss converging to 0 and training accuracy converging to 1.

Hypothesis 2: Increasing the number of parameters in the model leads to overfitting

This may seem to be a trivial claim, as from Experiment 1, adding more parameters leads to underfitting and there is a bias-variance trade-off. That said, overfitting caused by this addition of parameters was what led us to use regularization, in hopes of reducing it. Refer to the Appendix for more detail on the experiment to test Hypothesis 2. From Figures 2 and 3, it is evident that early stopping prevents overfitting, as validation cost is minimized when validation accuracy is maximized.

Hypothesis 3: Regularization helps reduce overfitting

In order to test this hypothesis, we will compare the training and validation cost and accuracy curves for our extended model ($k = 5$) with and without regularization. If there is a reduction in overfitting due to regularization, we expect to see the validation cost curve for the regularized model increase later and at a slower rate than that of the unregularized model. Moreover, we expect to see training cost converge to 0 at a slower rate, indicating slower memorization of the training data. Additionally, we expect to see validation accuracy to be higher and decreasing at a later iteration and at a slower rate, while training accuracy converges to 1.0 at a slower rate. For the plots below, we used the regularization coefficients that achieved the maximum validation error at the point of early stopping. In the plots below, we use the regularization constants [0.01, 0.1, 0.1, 0.1], but all permutations of {0, 0.01, 0.1} illustrates roughly the same trends for cost and accuracy.

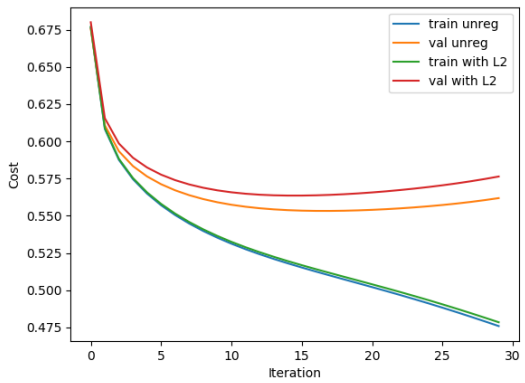


Figure 6: Experiment 3 Cost Curves

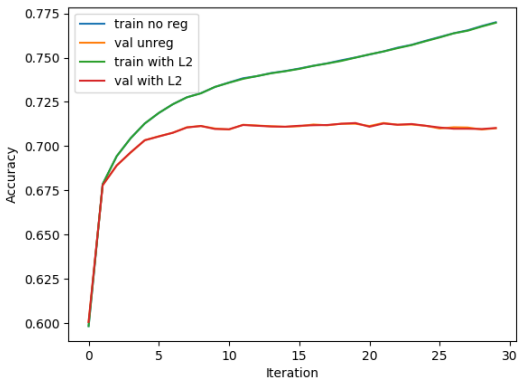


Figure 7: Experiment 3 Accuracy Curves

Note that Figure 7 actually has 4 curves but the orange and red lines overlap almost perfectly, and the green and blue lines overlap almost perfectly. We see that the regularized curves for training and validation costs are a little bit above their unregularized counterparts, which is expected with the added penalty. However, the regularized curves and their unregularized counterparts are almost parallel to each other. In other words, the rate of increase/decrease are fairly equal with and without regularization. Moreover, the unregularized model hits its minimum validation loss at 18 iterations, whereas the regularized model hits its minimum validation cost at 15 iterations. Thus, we do not see regularization positively affecting generalization from a cost standpoint, noting that this difference is minimal. We also see that the unregularized and regularized curves for training and validation accuracies overlap each other and thus, L2 regularization does not cause any significant change in accuracy. Hence, we have altogether proved our hypothesis to be false.

Thus, from our experiments, we can conclude that adding more parameters to the baseline model reduces underfitting while increasing overfitting, confirming the bias-variance trade-off. Moreover, we have observed that early stopping is a more effective technique to reduce overfitting in our model as opposed to L2 regularization. Since the initialization of $\mathbf{U}, \mathbf{Z}$ are random, during some runs, the unregularized model actually outperforms the regularized one, illustrating the ineffectiveness of L2 regularization. Overall, we see that our increase in model performance is due more in part to the addition of parameters to reduce underfitting than is regularization to reduce overfitting.

Limitations:

The main challenge faced by our multidimensional Item-Response Theory model and other generalized IRT models are their fairly high complexities. Especially with our model, overfitting begins quite early (at roughly 18 iterations) and thus, requires early stopping to prevent from further overfitting. However, stopping this early in the training process means that the model may suffer from underfitting, as it does not have enough time to learn all the relevant features of the training data. Thus, we have a bias-variance trade-off. Overall, fitting this model appropriately requires much more data in order to characterize the student and questions accurately. In other words, when we have a scarce dataset, a simpler model is expected to perform much better, as it would be less sensitive to noise and outliers. For example, if a competent student just beginning to use the e-learning software accidentally answered an easy question incorrectly, the model will most likely rate the student as incompetent, at least initially. Thus, a possible extension to this model would be to have a variable for each student or question that quantifies how sparse the data is for that particular student or question and adjust the learning rate for that student/question appropriately to minimize the effect of noisy data.

Another limitation with our current IRT model is that it only assumes student ability and question difficulty features can fully explain why a student can correctly or incorrectly answer a question. However, with multiple choice questions, the act of guessing a correct response from all or a subset of the choices should also be included as a contributing factor. With our current model, when $(\mathbf{u}_i^T \mathbf{z}_j + \mathbf{b}_{S,i} + \mathbf{b}_{Q,j}) \to -\infty$, the probability of student answering question correctly becomes very close to 0, as observed from the shape of a sigmoid. That said, randomly guessing an answer out of n choices will give a probability $\frac{1}{n} > 0$ for the student to correctly answer the question (for n reasonably small). In fact, there exist IRT models that specify a guess parameter γ_j that quantifies how easily someone can correctly guess a question. The 3-parameter IRT model has the following equation:

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

Currently, there is not much research on the effectiveness of incorporating a guessing parameter into multidimensional IRT models. The added parameter can cause the model to further overfit the data, so one must also come up effective regularization strategies, which may be a difficult task. This is definitely a potential area of research.

Lastly, we can extend and improve our model by using maximum a posteriori (MAP) or expected a posteriori (EAP) models to address some of the data sparsity issues mentioned earlier, as adding a prior may be likened to regularizing model parameters. We could also use quasi-Newton (QN) or Expectation Maximization (EM) algorithms to train our model parameters instead of gradient descent to see if the model will converge with more stability.

Appendix

Hyperparameter Tuning:

We tuned our learning rate by first running our model with learning rate in {0.001, 0.005, 0.01, 0.05, 0.1} while fixing the number of iterations to be 200, $k = 20$, initializing $\mathbf{U}, \mathbf{Z}, \mathbf{b}_S, \mathbf{b}_Q$ to have all zero entries. During tuning, we noticed that after a certain point, validation loss seems to diverge, so we trained each model until validation loss hit its minimum. We first set the regularization constants to 0 and found that higher learning rates caused validation loss to oscillate whereas lower learning rates caused validation loss to reach its minimum too slowly. Thus, we settled at learning rate = 0.01. Then, we ran the model on k in $\{2, 5, 10, 20, 40\}$ and found that all k's achieve the same validation accuracy (0.70717) when validation cost is at its minimum. So, we chose $k = 5$, as we thought there would be less risk of overfitting and underfitting with a decently small, but not too small, value. Then, we ran the model on combinations of regularization constants in {0, 0.01, 0.1} and found that no regularization achieved the highest validation accuracy. Finally, we randomized the initialization of $\mathbf{U}$ and $\mathbf{Z}$ to be between 0 and $1/\sqrt{k}$ and found that it achieves higher accuracy than the fixed initialization on average.

Hypothesis 2: Increasing the number of parameters in the model leads to overfitting

In order to test this hypothesis, we will compare the validation cost and accuracy curves of our unregularized extended model for different values of k. To show that these models lead to overfitting, we must show that as training continues, the model will memorize the training data and not generalize well on the validation set. From Experiment 1, we have shown that increasing the number of parameters increases the model's ability to memorize the training data. Thus, we simply need to show that the model does not generalize well. A visual indication of this is the validation curve increasing consistently at a certain point after an initial decrease. We expect that for higher values of k, this increase in validation loss happens at an earlier iteration and at a faster rate. Moreover, we expect to see validation accuracy decreasing at roughly the same time validation error increases, where for higher values of k, a earlier and faster decrease in validation accuracy is observed.

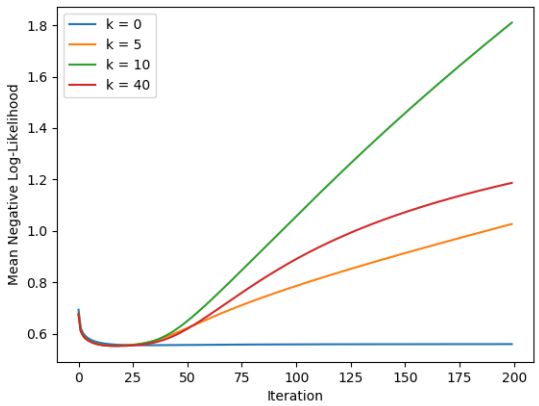


Figure 8: Experiment 2 Validation Cost Curves

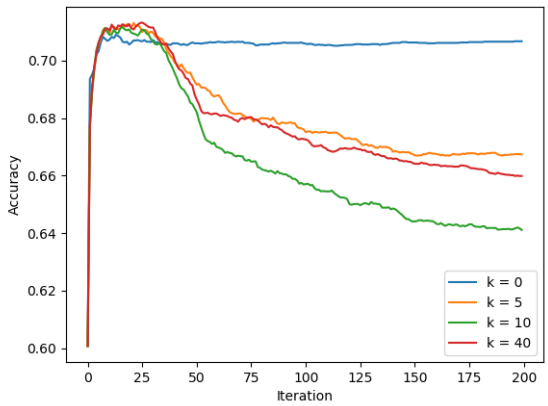


Figure 9: Experiment 2 Validation Accuracy Curves

We see that overall, higher values of k result in models that overfit more, as shown by its steeper increase in loss and steeper decrease in accuracy after hitting its minimum and maximum points, respectively. However, interestingly, we find that this argument does not hold for much higher values of k (in this case, $k = 40$). That said, looking back at the training cost and accuracy curves in Figures 4 and 5 (page 10), we see that the model with $k = 40$ has more or less completely memorized the training data at roughly 100 iterations, resulting in slower learning thereafter, as illustrated by the change in concavity in the validation cost curve. Moreover, the initial steep drop in validation accuracy ending at around 50 iterations corresponds to the second inflection point in the training accuracy. This is where training accuracy starts to increase at a slower rate, indicating that the model is learning less overall and hence, learning less irrelevant features of the training data. Nevertheless, we still see that adding parameters to the baseline model results in diverging validation cost and accuracy.

Thus, it is clear that early stopping will help reduce overfitting by limiting training to the point validation error hits its minimum, which is approximately where validation accuracy hits its maximum. In other words, stopping

at this point during training is when the model generalizes the best. That said, from the plots, we see that these minimum and maximum points for each of the models is roughly the same value, with slightly smaller values of $k > 0$ reaching higher validation accuracies. Thus, in general, the value of k does not seem to play a huge role in the model when using early stopping.