CSC311 Project Writeup

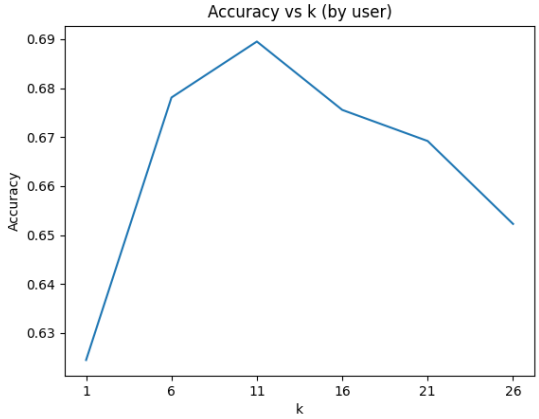
Tian Shu Li, Richard Yan, Ziqian Gao

November 2022

Part A

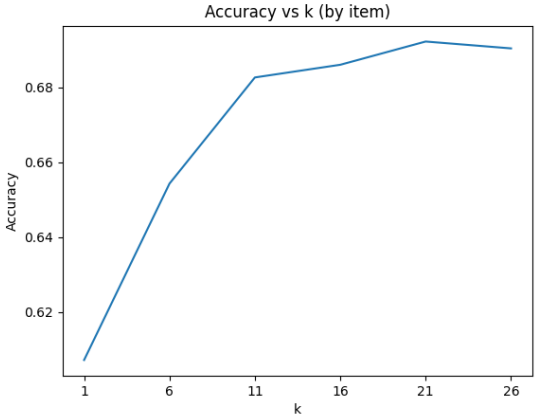
1. **k-Nearest Neighbor**

   (a)



   (b) The k^* will be 11, which get the highest validation accuracy of 0.6895286480383855. The test accuracy on $k^* = 11$ is: 0.6841659610499576.

   (c) The underlying core assumption is that, if question A has the same amount of students answering correct and incorrect with question B (other than the one student we are filling in), then that single student answering question A should match that same student answering question B.



The k^* will be 21, which get the highest validation accuracy of 0.6922099915325995. The test accuracy on $k^* = 11$ is: 0.6816257408975445.

(d) Using the performance data we get in part a and c, we can see that with smaller k values (1, 6, 11), user-based out-performed item-based, with user-based achieves its highest at $k = 11$. However with larger k values (16, 21, 26), item-based out-performed user-based, with item-based achieves its highest at $k = 21$. For $k = 1$, both did worst respectively. Overall, the user-based achieves higher accuracy on the test data, so I will say that user-based performs better, since it gets probably the best result using smaller k values than item-based (which improve runtime slightly).

(e) One limitation of kNN on this question is that no matter using user-based or item-based, it seems that it only consider and compare the amount of correct and incorrect answers, for example if we use the user-based method, it can be the case that two students answered pretty much different set of questions with only a few overlaps (including the question that we are filling in), and coincidentally the amount of correct and incorrect answers they have are close. The method will predict that they answer the question we are filling in similarly, but according to they have mostly answered a different set of questions, this might not be the case. An other example can be a question answered by a group of younger students has been compared to a question answered by a group of older students, where we want to predict how a young student will perform on the question that might be correctly answered more by older students. The kNN algorithm just didn't take in account of the features of different questions and different students.

Another limitation of kNN on this question is that we always fill out the whole matrix in prior to predict, however in practice we may not

really need want to know the whole matrix, but just only a subset (few entries) of it. I believe we have around 5% of matrix filled as given data, and in practice we may just want to fill about 50% or so, but not likely 100% (because we may not be interested in an grade 1 kid answering questions that are more likely to be taught in grade 6).

2. **Item Response Theory**

   (a) We have that

$$\begin{aligned} p(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) &= \prod_{i,j} p(c_{ij}|\theta_i,\beta_j) \\ &= \prod_{i,j} (\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)})^{c_{ij}} (1-\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)})^{(1-c_{ij})} \\ &= \prod_{i,j} (\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)})^{c_{ij}} (\frac{1}{1+\exp(\theta_i-\beta_j)})^{(1-c_{ij})} \end{aligned}$$

   Then

$$\begin{aligned} \log p(\mathbf{C}|\boldsymbol{\theta},\boldsymbol{\beta}) &= \sum_{i=1}^{542}\sum_{j=1}^{1774} c_{ij}\log(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}) + (1-c_{ij})\log(\frac{1}{1+\exp(\theta_i-\beta_j)}) \\ &= \sum_{i=1}^{542}\sum_{j=1}^{1774} c_{ij}(\theta_i-\beta_j-\log(1+\exp(\theta_i-\beta_j)))- \\ &\quad (1-c_{ij})\log(1+\exp(\theta_i-\beta_j)) \\ &= \sum_{i=1}^{542}\sum_{j=1}^{1774} c_{ij}(\theta_i-\beta_j) - \log(1+\exp(\theta_i-\beta_j)) \end{aligned}$$

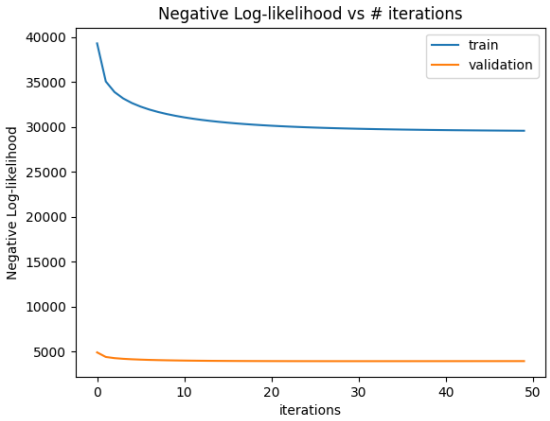
   And the derivatives:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\beta})}{\partial \theta_i} = \sum_{j=1}^{1774} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

$$\begin{aligned} \frac{\partial \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\beta})}{\partial \beta_j} &= \sum_{i=1}^{542} -c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} \\ &= -\sum_{i=1}^{542} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} \end{aligned}$$

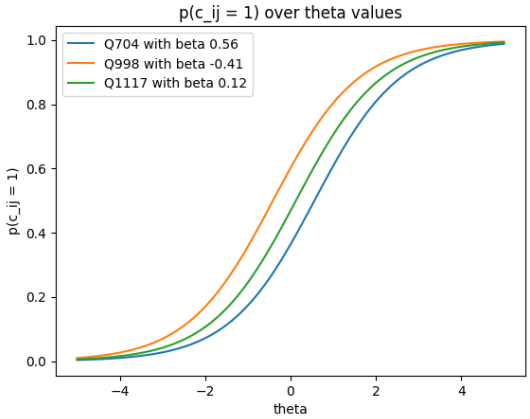
   (b) The hyperparameters we used were:
   Learning rate: 0.01

Num iterations: 50
θ, β initialized with all 0.



(c) Validation accuracy: 0.7058989556872707
Test accuracy: 0.7067456957380751

(d)



We can see that these curves look liked the sigmoid curve, which is very intuitive because the probability $p(c_{ij})$ is indeed calculated using the sigmoid function. Those curves say that, when theta value is low (i.e. trained model student ability is low), then the probability this student answer correct that question is low, and vice versa. Furthermore, as we can see that our model trained that Q998 is the easiest among those 3, and Q704 is the most difficult, and the curves show that Q998 has an higher probability to be answered correctly then Q704 with the same theta value, which is also very intuitive.

3. **Option Neural Network**

(a) 1. ALS fix all other variables and perform gradient descent on 1 unfixed variable, in an alternative fasion, where as neural network perform backward propagation without explictly "fixing" variables since it just does back prop in the backward order.

2. ALS have to be linear, where as neural networks can be non-linear given non-linear activation functions.

3. ALS may be more efficient than neural network due to linear nature.

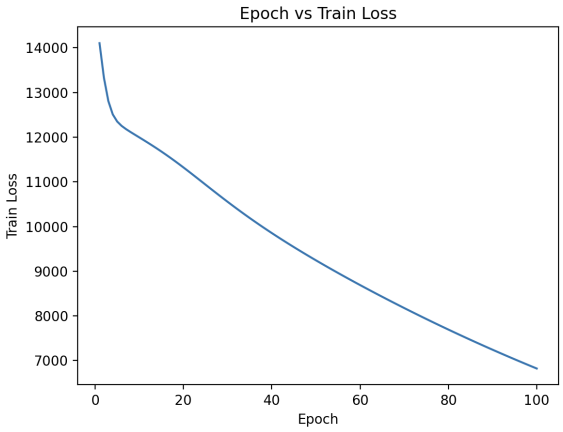
(b) In neural_network.py

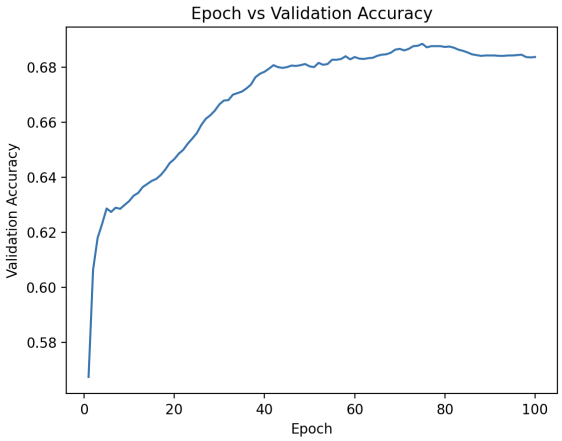
(c) After tuning, we chose the optimization hyperparameters to be lr = 0.005 and num_epoch = 100.

With this setup, we choose k* to be 50 with highest validation accuracy of 0.6837425910245555 among [10, 50, 100, 200, 500].

(d) By fixing k* as 50, we used num_epoch = 75 with validation accuracy of

0.6885407846457804 for test set. Test data achieved accuracy of 0.6819079875811459.





(e) The best lambda value is 0.001, achieving validation accuracy of 0.6850127011007621 and test accuracy of 0.6751340671747107. It seems like regularizer penalty made the model perform worse, since the test accuracy without regularizer is 0.6819079875811459.

4. **Ensemble**

The model we used in part 4 is the neural network implemented in part 3. We first randomly sliced train_data into matrices using np.random.randint(). Then we generated zero_train_matrix named trainz, and we torch.FloatTensored both of them. We set all the parameters required for train() in neural_network.py. Output from part A $q3$ was referred to since they produced the best outfit. And we performed train() for each of them. We re-organize the predictions and acquire the average for final evaluation, we input source prediction and calculate the accuracy.

The parameters we chose for part 4:

lr = 0.005

num_epoch = 75

lambda = 0.001

k = 50

The final valid and test accuracy is reported in the figure as follows:

```
Epoch: 65       Training Cost: 6983.690344      Valid Acc: 0.6009031893875247
Epoch: 66       Training Cost: 6938.124884      Valid Acc: 0.6017499294383291
Epoch: 67       Training Cost: 6893.497798      Valid Acc: 0.6014676827547276
Epoch: 68       Training Cost: 6849.782530      Valid Acc: 0.6017499294383291
Epoch: 69       Training Cost: 6806.953306      Valid Acc: 0.6017499294383291
Epoch: 70       Training Cost: 6764.984551      Valid Acc: 0.6014676827547276
Epoch: 71       Training Cost: 6723.852227      Valid Acc: 0.6014676827547276
Epoch: 72       Training Cost: 6683.532537      Valid Acc: 0.6009031893875247
Epoch: 73       Training Cost: 6644.002645      Valid Acc: 0.6006209427039232
Epoch: 74       Training Cost: 6605.239904      Valid Acc: 0.6003386960203217
valid accuracy: 0.5230031047135196
test accuracy: 0.6172734970364098
```

The valid accuracy is 0.5230031047135196; the test accuracy is 0.6172734970364098.

Comparing to part 3 where we achieved validation accuracy of 0.6850127011007621 and test accuracy of 0.6751340671747107, part 4 has acquired relatively lower accuracy in both. Hence, we didn't obtain better performance using the ensemble method. One reason that likely happened is that the matrix is already very sparse, with around 95% empty space. Therefore, randomly bagging from the sparse matrix from trainm and randint resulted in less data being used in each model. Although this ensemble method helped to reduce variance in the data, more biases could have been induced that caused additional under-fitting, which results in a net decrease of accuracy. Nonetheless, it does seem that the ensemble method converges faster than that in part *c*.

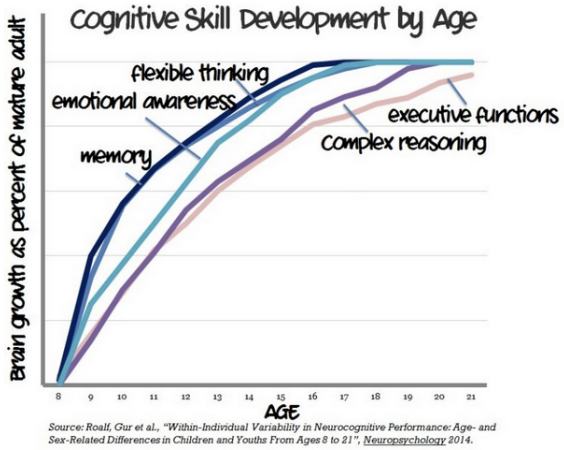
Part B

1. **Formal Description**

   Our group decided to extend upon part A question B: Item Response Theory.

   We realized the only data that were used in question B was the primary metadata. In order to fully explore the potential of machine learning, we want to look into more data as they're available: student metadata and question metadata.

   Source: https://v1.mindprintlearning.com/blog/cognitive-skills/



   First of all, since data is very sparse, we want to add a "prior distribution" by imposing an assumption on the data, hoping to combat the sparsity of the data. We are grouping students into age groups based on their date of birth in student metadata to rank them in terms of cognitive performance. Research have shown that teenagers develops cognitive skills rapidly as shown in the Figure above. Therefore, we suspect there's a positive correlation between users' age and their performance. To calculate this, we first split students into 5 age groups: "2002 or earlier", "2003-2004", "2004-2005", "2005-2006", "2009 or later", naming them group 1, 2, 3, 4, 5, respectively (students without date of birth are discarded as of now i.e. not in any group). The split of the groups are determined by appropriate time interval while ensuring there are sufficient students in each group. Then, we find the accuracy rate of answers based on each group:

   $\forall k \in \{1,2,3,4,5\}, \alpha_k = \frac{\#\ of\ correct\ answers\ submitted\ by\ students\ in\ group\ k}{\#\ of\ answers\ submitted\ by\ students\ in\ group\ k}$

   Then, we normalize this each group's accuracy by the minimum of accuracy of all groups, so that the group with least accuracy has $acc = 1.0$.

   So α_k to $\alpha_k / min(\alpha_1, ..., \alpha_5), \forall k \in \{1,2,3,4,5\}$.

The implementation of this preprocessing procedure can be found in

part_b/age_filter.py.

Then we apply to the original formula, multiplying this constant to i-th student ability θ_i, so our IRT model formula becomes:

$p(c_{ijk} = 1|\theta_i, \beta_j, \alpha_k) = \frac{exp(\alpha_k * \theta_i - \beta_j)}{1+exp(\alpha_k * \theta_i - \beta_j)}$, where student i is in age group k.

Secondly, we want to simulate a more realistic behaviour onto the IRT. When a person don't know an answer, they may choose to guess. Then we should incorporate in the chance that a student would guess and get the correct answer for a question, naming it parameter $p = 0.25$, in which everyone has been given a 'base chance' to get the question right. In addition, we want to distinguish questions by giving them an discriminative index, where a more discriminating questions means a user is either very right or very wrong, and there's not really an spectrum of understanding the question, we name this parameter k_j for each question j. Now, our IRT model formula becomes:

$p(c_{ij} = 1|\theta_i, \beta_j, p, k_j) = p + (1 - p) * sigmoid(k_j(\theta_i - \beta_j))$

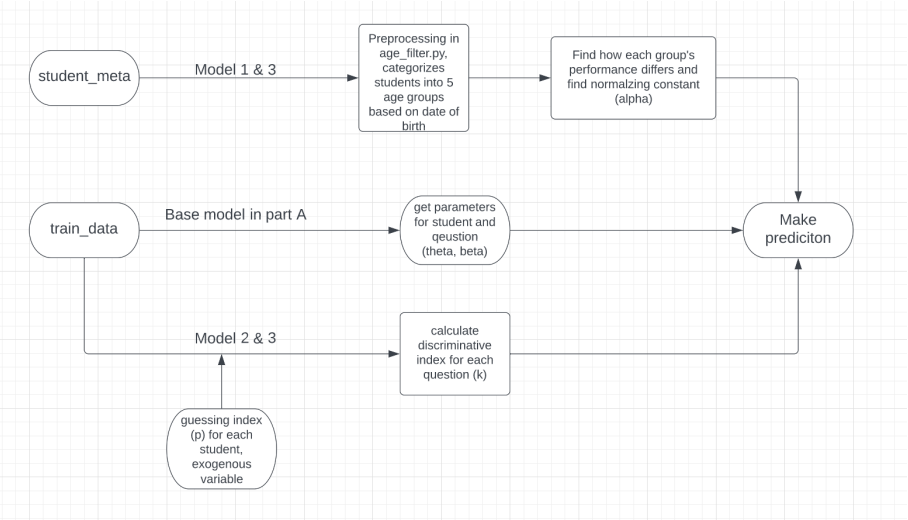
As a result, we will be constructing 3 new models. First model contains the age prior distribution; second model contains the guessing index and the discriminating index; third model contains both from model 1 and model 2. We hope to see how these factors contribute to the performance individually, as well as how they interact with each other.
The combined IRT model formula is:

$p(c_{ij} = 1|\theta_i, \beta_j, p, k_j, \alpha_k) = p + (1 - p) * sigmoid(k_j(\alpha_k * \theta_i - \beta_j))$, where student i is in age group k.

The implementation of the 3 models is in **part_b/new_models.py**.

2. **Figure or Diagram**

The flow chart below shows the workflow and our models in addition to the base model, which is described in detail above in Part B section 1. Looking at the Figure, we can quickly grasp the overall idea of our model improvements. For model 1 and 3, we used student_meta to get split to age groups and makes a prior assumption on each group's ability. For model 2 and 3, we used a prior exogenous variable, p, for the purpose of student randomly guessing the correct answer, as well as calculating the discriminative index for each question k using gradient descent, similar to using GD to find *theta* and *beta* in the base model.



3. **Comparison or Demonstrate**

Codes for carrying out these experiments can be found in the main function of **part_b/age_filter.py**.

Comparisons between models:

	Original Model	Model 1	Model 2	Model 3
Validation accuracy	0.7059	0.7063	0.7087	0.7082
Test accuracy	0.7067	0.7070	0.6850	0.6864

We can see that among the validation data, all 3 modified model perform better than the original part a IRT model, with model 2 having the best score. While among the test data, only our model 1 outperformed the original model, while model 2 & 3 didn't perform as well.

For validation set, all models gave a higher accuracy than that of original model. This is an indicator that the assumptions we made and the parameters added are indeed useful and relevant to the problem.

For test data, it seems that model 1 is indeed giving improvements over the original model, it shows that there's some optimization improvements that generalizes well on the test data, despite the improvements aren't very big. This, in general, support our hypothesis that among the majority of the students age groups, cognitive skills might be a considerably important feature to consider when predicting their performance on answering questions. On the other hand, test accuracy is is lower for model 2 and 3 comparing with the original model. Higher validation accuracy and lower test accuracy presumably means the model is learning more about the training data with the addition of new parameters p and k, but didn't generalize well due to more potential bias induced by more learning parameters.

4. **Limitations**

   To begin with, it is worth to note that there's a data induced limitation that all model will suffer from. Since there's only about 5% of informative data, while the rest is unknown, then it is very difficult for any model to generalize, since most of the data are completely unknown. For instance, there may be question that was done by no one or only a few students, or a student who answered no or only a few questions in the training data. Then there's completely no or few information about this question/student, making it much harder to make predictions.

   By the definition of IRT, it assumes local independence, which is very unlikely in terms of the data, since students could be correlated if they know each other, and questions could be correlated if they are made by the same person. Then, this naive assumption of Independence may negatively affect the accuracy of all 3 models, since they all are based on IRT.

   For model 2 and 3, the guessing index p was exogenous, meaning it is the same for all students. A more realistic situation would be assigning each student i an unique p_i, given their likelihood of conducting a guess. However, it seems that there's no intuitive way to compute unique p_i for each student. Therefore, we've made this simplifying assumption that all students are equal likely to perform a guess, which could led to less performance. As a result, a viable improvement would be come up with a heuristic to compute guessing index p_i for each student i, and update it using gradient descent.

   For model 1 and 3, we've added prior distribution by assuming ability difference based on age groups. We did this because data is very sparse, and we hope more assumptions will direct the model to a more realistic situation. However, this is an double-edged sword. Although it seems that our assumption is correct based on the train, validation, and test data, it could be the case that real world data in general don't follow this trend.

Moreover, it can totally be the case that students only answer questions related to their age group, for example, it might be the case that grade 9 students only answers questions that are for grade 9, rather than those for grade 12, etc. Then in this case, the extra assumption of the age groups may lead to lower accuracy and generalization abilities. The way to find whether it is true is to test more data, and tune the hyper-parameters related to age groups accordingly if necessary.

As for possible extensions, instead of manually selecting age groups, a better approach would be feeding date of birth of students and their performance on questions to a clustering algorithm, such as the EM algorithm. This will enable an unsupervised grouping of the age groups in terms of performance, which would be more accurate than our manual split of the age groups. As of now, we have the age groups: "2003-2004", "2004-2005", "2005-2006", "2009 or later". Letting the machine to learn the intervals that an age groups belongs is better and is very likely to contribute to better results.