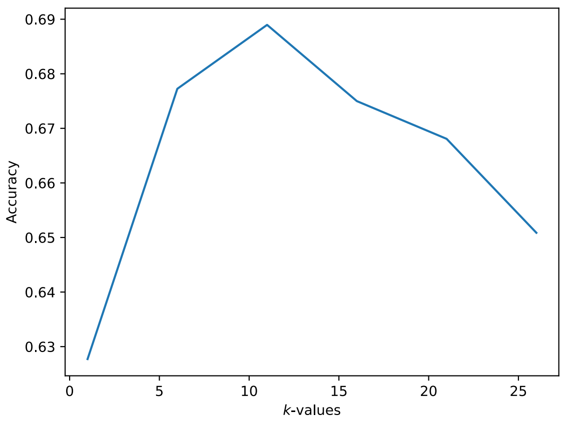
CSC311 Final Report: Project Option 1

Leo Peckham, Longyue Wang, and Gursewak Sandhu

Part A

1. k-nearest neighbours

(a)

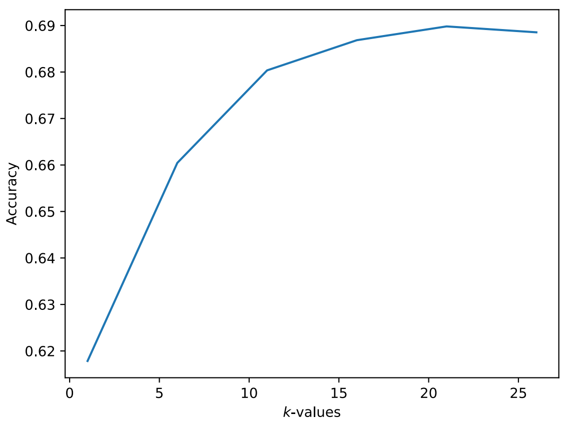


k**-value**	1	6	**11**	16	21	26
Accuracy	0.628	0.677	**0.689**	0.675	0.668	0.651

Figure 1: Accuracies of user-clustering KNN

(b) From this data, we can see that the k^* that gave us the best validation accuracy was $k^* = 11$. Running on test we achieve an accuracy of 0.683.

(c)



k**-value**	1	6	11	16	**21**	26
Accuracy	0.618	0.660	0.680	0.687	**0.690**	0.689

Figure 2: Accuracies of question-clustering KNN

Question-based clustering assumes that if two questions have very similar distributions of answers across the known students, then the questions will behave similarly for new students. Intuitively, if two questions ask about a specific theorem, then the same students who get the first one wrong because they forgot the statement of the theorem will get the second wrong as well.

From the data in Figure 2, we can see that the k^* that gave us the best validation accuracy was $k^* = 21$. Running on test we achieve an accuracy of 0.670.

(d) The performances across the board are similar. The user-based approach does better on test, but only by around a tenth of a percent accuracy. The graphs for different k-values do look quite different, though, with the user-clustering in Figure 1 falling off much faster for high k-values than the question-clustering in figure 2 .

(e) This method relies on a sufficient amount of data to be able to accurately cluster. If there is an example with a significant amount held-out, it will become inaccurate because many nearby users/questions will look similar.

The dimension of the features is also a problem. First, it can be extremely high, being either the number of students or the number of questions. This poses a problem for a KNN approach, especially one that uses a Euclidean distance like this one. Mapping to a smaller latent space first and using another distance function would help alieviate this problem. Second, if we add new students or questions to the dataset, the very *dimensionality* of our data will change. This makes it more difficult to improve our model with new data; mapping to a laten space would also fix this issue.

2. Item response theory

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3. Matrix factorization

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

4. Ensemble

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Part B

1. Motivation

One of the major problems with the KNN model is that the number of dimensions are very high. We have 1774 questions and 542 students, so based on the way we do it we work in either a 1774 dimensional space, or a 542 dimensional space. Additionally, none of the approaches we take are robust to the possibility of adding both more questions and more students, because this increases the dimension in a way that would require retraining.

Ideally we would be able to work in some lower-dimensional latent space that allows us to represent the qualities of the questions and the abilities of the students, and then compute our hypotheses from that. This is similar to what IRT does, where θ is our student ability, and β is our question difficulty. We essentially want to turn θ and β into vectors in this latent space.

2. Design

Motivated by our desire to create a latent space, we lay out our architecture in Figure 3.

We start by constructing the latent space (which we will say has dimension k from now on) with the question metadata. Each question has 388 subjects it can be a part of, and so has 388 features. We take our question metadata Q and we encode the features into a binary representation, giving us a binary question metadata matrix $\mathbf{Q}_1$ of shape $|q| \times |s| = 1774 \times 388$, where $|q|$ is the number of questions and $|s|$ is the number of subjects. To construct our latent question matrix $\mathbf{L}$ ($|q| \times k$), we do SVD rank reduction $\mathbf{Q}_1$. That is:

$$\mathbf{Q}_1 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{1}$$

$$\mathbf{L} = \mathbf{U}_k\mathbf{\Sigma}_k \tag{2}$$

Where $\mathbf{U}$ and $\mathbf{V}$ are the left- and right-singular matrices respectively, and $\mathbf{\Sigma}$ is the eigenvalue matrix. The k subscript means the matrix has been truncated. This formulation falls out from the shapes of the matrices, but it also means that $\mathbf{L}$ is exactly the PCA score matrix, which we are interpreting as a dimensionality reduction from $|s|$ to k. over the rows of $\mathbf{Q}_1$

Next we construct the latent student/user matrix $\mathbf{S}$ of shape $|u| \times k$ where $|u|$ is the number of users/students. We use the intuition that the k features of rows of $\mathbf{L}$ represent information about the skillset required to answer the question corresponding to the given row. If a student answers every question with a certain feature correct, then we will want their magnitude in that feature to be very high. If a student gets half right and half wrong, we will want it to cancel out and for their magnitude in that feature to be very low. A naïve way of getting these traits is to, for each student, pool over the latent question matrix $\mathbf{L}$ based on how they answered the question. To be precise, we use mean-pooling in the form:

$$\mathbf{S}_i = \frac{\sum_j \mathbb{I}[\mathbf{H}_{ij} \neq \text{NaN}](-1)^{\mathbf{H}_{ij}+1}\mathbf{L}_{ij}}{\sum_j \mathbb{I}[\mathbf{H}_{ij} \neq \text{NaN}]} \tag{3}$$

Where $\mathbb{I}$ is the indicator function.

$\mathbb{S}$ and $\mathbb{L}$ now hold the information we want to put into IRT, but do not relate to each other in a way that would let us put it right into our IRT evalutation function. Therefore, we first want to learn a mapping to IRT compatible matrices $\mathbf{\Theta}$ and $\mathbf{B}$ respectively. We set up the simplest such mapping:

$$\mathbf{\Theta} = \mathbf{S}\mathbf{W}_\theta \tag{4}$$

$$\mathbf{B} = \mathbf{S}\mathbf{W}_\beta \tag{5}$$

Were $\mathbf{W}_\theta$ and $\mathbf{W}_\beta$ are matrices we will learn using gradient descent. One would probably get much better results if using a neural-network, or some other more sophisticated form of mapping, but we stick to simplicity here.

For our gradient descent, we minimize negative log-likelihood on the cosine similarity between k dimensional vectors, similar to the treatment of IRT.

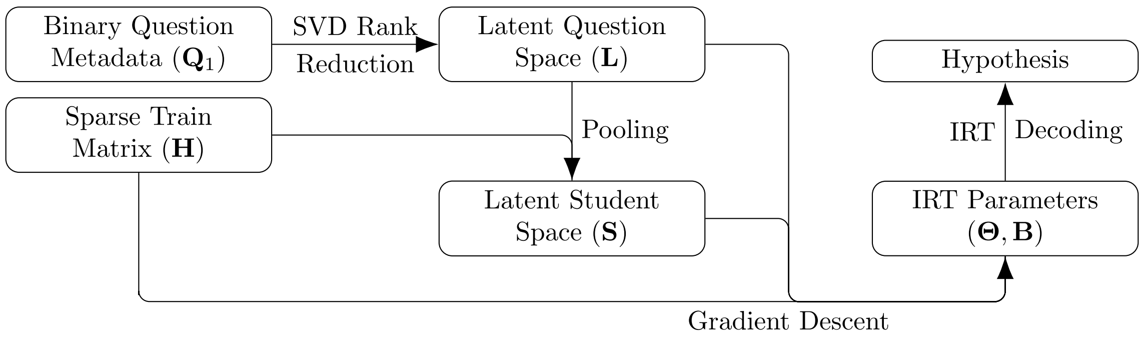

Binary Question Metadata ($\mathbf{Q}_1$)
SVD Rank Reduction
Latent Question Space ($\mathbf{L}$)
Hypothesis
Sparse Train Matrix ($\mathbf{H}$)
Pooling
IRT
Decoding
Latent Student Space ($\mathbf{S}$)
IRT Parameters ($\mathbf{\Theta}, \mathbf{B}$)
Gradient Descent

Figure 3: General architecture of the proposed learning algorithm.

$$\theta_i = \mathbf{S}_i \mathbf{W}_\theta, \beta_j = \mathbf{S}_i \mathbf{W}_\beta \tag{6}$$

$$c_{ij} = \frac{\theta_i \beta_j}{\|\theta_i\|_2 \|\beta_j\|_2} \tag{7}$$

$$g_{ij} = \sigma(a c_{ij} + b) \tag{8}$$

$$\ell = -\sum_{i,j} \left[\mathbf{H}_{ij} \log g_{ij} (1 - \mathbf{H}_{ij}) \log(1 - g_{ij})\right] \tag{9}$$

$$\frac{\partial \ell}{\partial \mathbf{W}_\theta} = \frac{\partial \ell}{\partial \mathbf{\Theta}} \frac{\partial \mathbf{\Theta}}{\partial \mathbf{W}_\theta} = \mathbf{S}^\top \frac{\partial \mathbf{\Theta}}{\partial \mathbf{W}_\theta} \tag{10}$$

$$\frac{\partial \mathbf{\Theta}}{\partial \mathbf{W}_\theta} = \left[\frac{\partial \theta_i}{\partial \mathbf{W}_\theta}\right]_i \tag{11}$$

$$\frac{\partial \theta_i}{\partial \mathbf{W}_\theta} = \sum_j (g_{ij} - H_{ij}) \left(\frac{\beta_i}{\|\theta_i\|_2 \|\beta_j\|_2} - c_{ij} \frac{\theta_i}{\|\theta_i\|_2^2} \right) \tag{12}$$

Where $\|\cdot\|_2$ is the L^2 norm, and σ is the sigmoid activation function. For the last line, a similar result holds exchanging θ_i for β_j. a and b are parameters the model trains along with $\mathbf{W}_\theta$ and $\mathbf{W}_\beta$ to prevent the sigmoid from getting clustered too heavily around 0.5, since the cosine similarity outputs small values.

We run gradient descent for 10,000 steps, using a learning rate of $1e^{-3}$ for everything except a for which we use a learning rate of $1e^{-4}$. We initialize the $\mathbf{W}$ matrices using a normal distribution centered around 0 with a variance of 2 (we tuned this variance as a hyper-parameter). We start a at 5 and b at 0.

For the dimension k of our latent space, we did a hyperparameter search and found that $k^* = 320$ was optimal. In Figure 4 we see the graph of the search with increments of 40. Not shown is another search which we repeated at increments of 10 zoomed in around 320 which also showed 320 to be optimal.

Figure 5 shows the first round of training we did. The blue line shows us the value of ℓ on the train set, and the red line shows us the "score" (which is the same as for IRT) on the valid set. This got accuracies of 79.9% for train, and 64.8% for both valid and test. This is less than we hoped for, and as can be seen in the figure, shows clear signs of overfitting.

To fix this, we added dropout on $\mathbf{B}$ and $\mathbf{B}$ at a 30% chance, which leads to the training curve in 6. This fixed the problem, and by doing early stopping based on the valid score we got a final train accuracy of 72.0%, valid accuracy of 66.9%, and test accuracy of 66.6%.

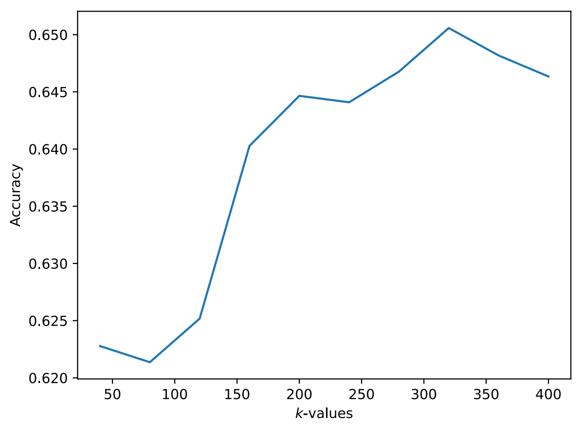


Figure 4: k^* search for SVD rank reduction based on final accuracies of valid

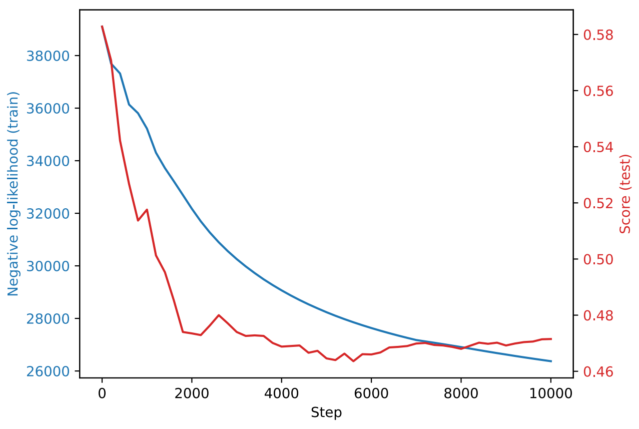


Figure 5: k^* search for SVD rank reduction based on final accuracies of valid

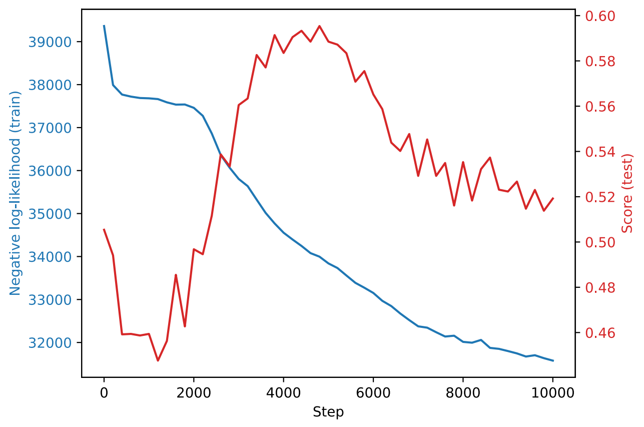


Figure 6: k^* search for SVD rank reduction based on final accuracies of valid