CSC311 Winter 2024 Final Project Report

April 5, 2024

Part A

Collaborative Filtering with k-Nearest Neighbor (kNN)

(a) User-Based Collaborative Filtering

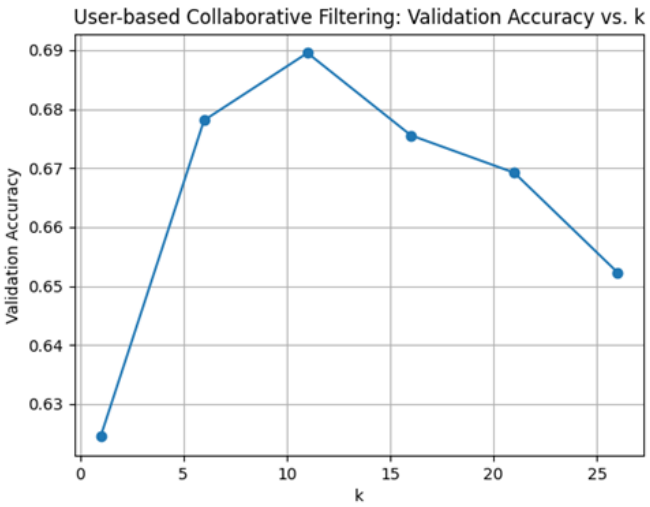

Figure 1: User-Based Validation Accuracy

Best k value for user-based: 11
Validation Accuracy: 0.6895286480383855
Test accuracy: 0.6841659610499576

(b) Underlying assumption for item-based collaborative filtering

Similar items will receive similar ratings or responses from the users. For example, if two items (e.g., questions in a diagnostic test) have been answered

similarly by a set of users, they are likely to be answered similarly by other users as well.

(c) Repeat part (a) with item-based collaborative filtering.

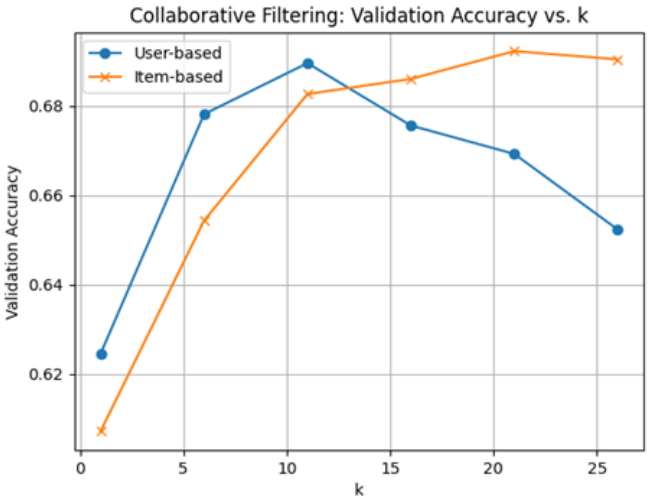

Figure 2: Enter Caption

Best k value for item-based: 21
Validation Accuracy: 0.6922099915325995
Test accuracy: 0.6816257408975445

(d) Compare user-based and item-based collaborative filtering

We compared these two models using validation accuracy, F1 Score, MCC, AUC-PR... But these values are very similar between item-based and user-based, but when we tried to measure the running time of these two models, we found out that the running time for the item based model is larger than the time for the user base. This is expected, because the shape of the sparse matrix is (542, 1774), where 542 is num of users and 1774 is num of the questions. So for item based there are more calculations expected.
So for the same accuracy, user based have less running time, so we can say user based is better in this scenario.

(e) potential limitations of kNN for the task

1.Scalability and Efficiency: kNN can be computationally expensive, especially as the dataset grows. Since it requires calculating the distance between a

point and all other points in the dataset, it can be slow for large datasets, making it impractical for real-time predictions in an big scale online education platform.

2.Sparsity of Data: The effectiveness of kNN relies on the assumption that similar students will answer similarly to similar questions. However, in a sparse matrix where many students may have answered only a few questions, finding the 'nearest neighbors' can be misleading. This sparsity can lead to inaccurate predictions, as the algorithm might rely on a very small set of answers to determine similarity.

Item Response Theory (IRT) Model

We derived the log-likelihood and its derivatives for the IRT model. After tuning the hyperparameters, we plotted the training and validation log-likelihoods as a function of iteration. The validation and test accuracies for our final model were reported.

(a) Log-likelihood and its derivatives

Given the probability of a correct answer:

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

The log-likelihood of observing the entire set of student responses C is:

$$\log p(C|\theta, \beta) = \sum_{i,j} c_{ij} \log \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) + (1 - c_{ij}) \log \left(\frac{1}{1 + \exp(\theta_i - \beta_j)} \right)$$

For the correct answers, where $c_{ij} = 1$, the log-likelihood becomes:

$$c_{ij} \log \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right)$$

Which simplifies to:

$$c_{ij}(\log(\exp(\theta_i - \beta_j)) - \log(1 + \exp(\theta_i - \beta_j)))$$

$$c_{ij}((\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j)))$$

For the incorrect answers, where $c_{ij} = 0$, the log-likelihood becomes:

$$(1 - c_{ij}) \log \left(\frac{1}{1 + \exp(\theta_i - \beta_j)} \right)$$

Which simplifies to:

$$(1 - c_{ij})(-\log(1 + \exp(\theta_i - \beta_j)))$$

Combining both terms, we get the simplified log-likelihood for all student-question pairs:

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

This expression represents the log-likelihood of the entire dataset C for the IRT model.

Now, let's start to do the **derivatives of the log likelihood.**

Starting with the log-likelihood function:

$$\log p(C|\theta,\beta) = \sum_{i,j} [c_{ij}(\theta_i - \beta_j) - \log(1+\exp(\theta_i - \beta_j))]$$

The derivative with respect to θ_i is:

$$\begin{aligned}\frac{\partial}{\partial \theta_i}\log p(C|\theta,\beta) &= \sum_j \left[c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right] \\ &= \sum_j [c_{ij} - p(c_{ij}=1|\theta_i,\beta_j)]\end{aligned}$$

And the derivative with respect to β_j is:

$$\begin{aligned}\frac{\partial}{\partial \beta_j}\log p(C|\theta,\beta) &= \sum_i \left[-c_{ij} + \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right] \\ &= \sum_i [p(c_{ij}=1|\theta_i,\beta_j) - c_{ij}]\end{aligned}$$

These derivatives are used to perform gradient ascent to find the values of θ_i and β_j that maximize the log-likelihood.

(b)

We implemented all the missing functions in item_response.py.

(c) Tuning hyperparameters

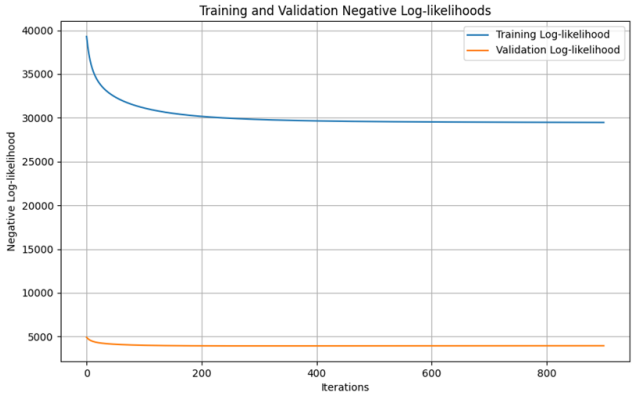

Figure 3: negative logliklihood

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)]
LR: 0.0005, Iterations: 100, Validation Accuracy: 0.7039232289020604, Training Accuracy: 0.7170300592718035
LR: 0.0005, Iterations: 300, Validation Accuracy: 0.7075924357888794, Training Accuracy: 0.7333474456675134
LR: 0.0005, Iterations: 500, Validation Accuracy: 0.7066045723962744, Training Accuracy: 0.7370342929720576
LR: 0.0005, Iterations: 700, Validation Accuracy: 0.7050522156364663, Training Accuracy: 0.7389923793395428
LR: 0.0005, Iterations: 900, Validation Accuracy: 0.7056167090036692, Training Accuracy: 0.7393275472763196
LR: 0.001, Iterations: 100, Validation Accuracy: 0.7071690657634773, Training Accuracy: 0.7271556590460062
LR: 0.001, Iterations: 300, Validation Accuracy: 0.7066045723962744, Training Accuracy: 0.739045300592718
LR: 0.001, Iterations: 500, Validation Accuracy: 0.7060400790290714, Training Accuracy: 0.7396450747953711
LR: 0.001, Iterations: 700, Validation Accuracy: 0.7066045723962744, Training Accuracy: 0.7399802427321479
LR: 0.001, Iterations: 900, Validation Accuracy: 0.70575783234547, Training Accuracy: 0.7398214789726221
Best LR: 0.0005, Best Iterations: 300
Final Test Accuracy: 0.7025119954840531
```

Figure 4: tuning hyperparameters

Our best parameter are:
Learning rate: 0.0005
Iteration: 300
With
Final Validation Accuracy: 0.7075924357888794
Final Test Accuracy: 0.7025119954840531

(d)

We selected three questions j1, j2, and j3. For each question, we plot a curve showing the probability of the correct response.

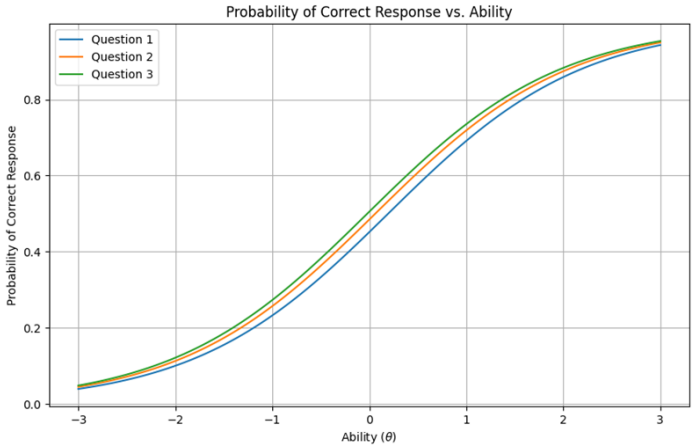

Figure 5: question ability curve

The curves displayed the characteristic sigmoid shape typical of Item Response Theory models, indicating the **probability of a correct response increases with a test-taker's ability.** The closeness of the curves for questions j_1, j_2, and j_3 suggests that these items have similar difficulty levels, as the inflection points where the probability of a correct response is 50% are nearly identical. This implies that none of the questions are significantly harder or easier than the others across the ability range shown. The moderate steepness of the curves indicates a fair level of discrimination, meaning the questions can differentiate between test-takers of varying abilities reasonably well, without being too sensitive or too broad. Overall, the questions would likely offer a balanced challenge and provide useful information about test-takers' abilities around the average difficulty level represented by these questions.

Neural Network

(a)

We completed the AutoEncoder class to perform a forward pass of the autoencoder.

(b) Training and tunning autoencoder

Best parameter without regularization: k=10, lr=0.01, num_epoch=120

(c) Plot

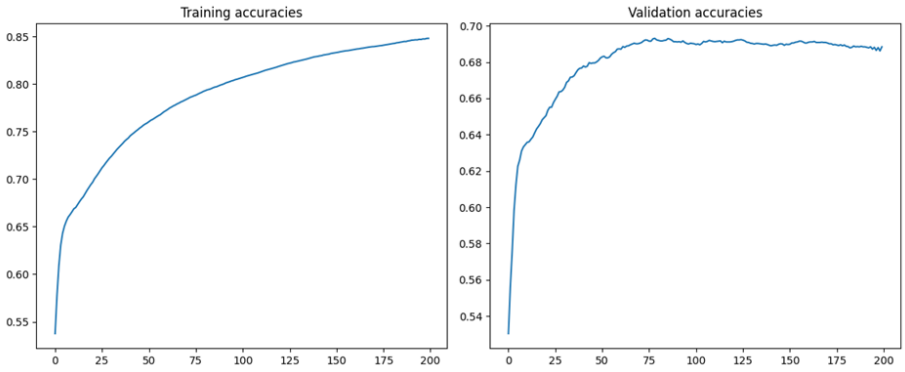

Figure 6: training loss and validation accuracy (no regularization)

Test accuracy: 0.6754163138583121

(d) Regularization introduced

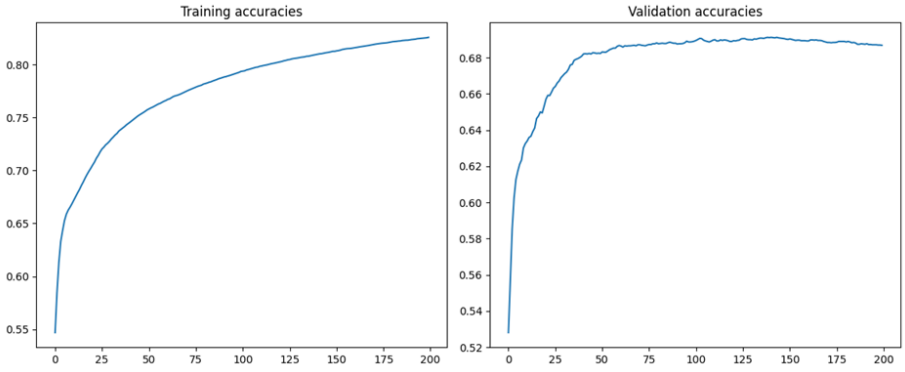

Figure 7: training loss and validation accuracy (with regularization)

Best parameter with regularization: **k**=10, **lr**=0.01, **num_epoch**=120, **lamb**=0.001
Test accuracy: 0.6830369743155518

Our regularized model performs slightly better then the unregularized one, but only within 1% of accuracy.

Part B

In Part B, we modified one of the algorithms implemented in Part A to improve its performance. We formally described our modified algorithm, provided figures to illustrate it, and compared it with the baseline models. Our analysis showed that our modified algorithm performed better in certain aspects, and we explained the reasons behind its performance. We also discussed one limitation of our model.

Introduction of Modified Algorithm

In Part B, we mainly modified our algorithm by utilizing the metadata in our dataset and modifying our AutoEncoder to accommodate the enhanced data.

Metadata

There are two parts of metadata in our dataset: question metadata and user metadata. We aim to find the underlying mechanism that similar people will perform similarly on similar questions. So our first step is to find a way to express the similarity of the questions and students.

Question Metadata

In total there are 388 subjects in the dataset, and we first removed some subjects that has no questions, and subjects that has all 1774 questions, there are still 288 left, which is too large of a dimension if we use multi-hot vector as our neural network, so we need to find a way to reduce the dimension for efficiency and preventing the model to be too large. We tried keeping the subjects with most questions, using PCA to reduce the dimension, and using embedding to represent each of the questions in terms of its subjects, and found that using embeddings worked best in our model. As a result, we decided to use embeddings to transform the multi-hot vectors into lower-dimension vectors that preserve the similarity. We use an EmbeddingBag layer from pytorch to do the transformation. EmbeddingBag is a simple lookup table that stores a unique vector (called the embedding) for each subject in our 288 subjects (called the dictionary of the EmbeddingBag) When given a multi-hot vector for a question (which can be seen as a set of subjects), the EmbeddingBag finds the embedding for each subject, and computes the mean of the embeddings. With this, the questions with similar subjects will be close to each other in terms of Euclidean distance. We hope that given these embeddings, the model will be able to find the underlying connections between different questions.

Student Metadata

We changed the gender and premium_pupil columns to one-hot features and calculated the relative age based on the date_of_birth column.

The Model

Embedding

We first use k-means clustering to separate the questions into 20 clusters and set up a small network to train an optimal EmbeddingBag that represents the clustering. We connect the EmbeddingBag's output to a fully connected layer with 20 outputs, use softmax on the output, and train the network using cross-entropy loss. We choose the embedding dimension to 30 after experiments on both lower and higher dimensions as a balance between information loss and training efficiency.

AutoEncoder

After finding an optimal embedding, we compute the embedding for each question and concatenate them as a long vector to form our embedding vector v. In the input of the network, we repeat each entry of the input x so that the dimension meets our embedding vector v. We make a vector x' such that for each $0 \leq i \leq 1773$, $x'_{i*embedding\_dim}, \ldots, x'_{(i+1)*embedding\_dim} = x_i$ then we perform an element-wise multiplication, so that only the embeddings for correct answers are passed to the neural network, all other subjects get 0's as input. We also enlarged the autoencoder to accommodate the enlarged input. In the decoder, we use 4 layers that reduce the dimension to 10000, 1000, 500, 200, respectively, with ReLU activation function in between and dropout layer as regularization. We then append the student metadata to the 200-D bottleneck and increase the dimension to 500, 1000, 1774 using three layers with ReLU activation function in between and dropout layers, and finally a sigmoid activation function after the last output. Adding student metadata late ensures that it is not neglected by the neural network due to its low dimension.

Figures and Diagrams

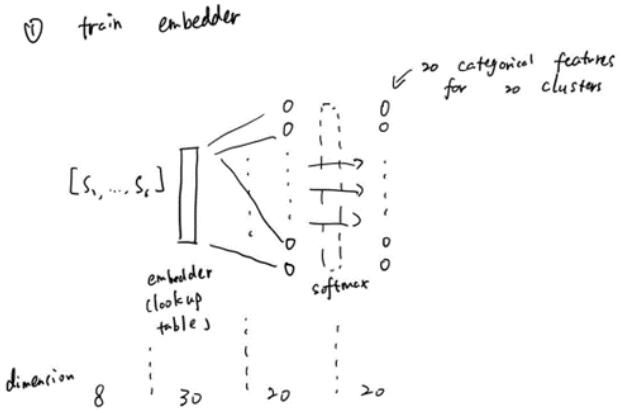

Figure 8: embedder training

Figure 8: we first train embedder using a simple neural network, the input is every subjects that each question belongs to, with padding to ensure size match.

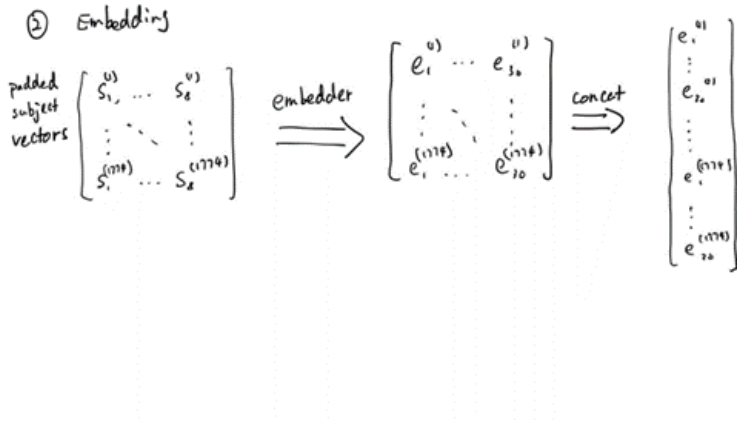

Figure 9: compute embeddings

Figure 9: we used the train embedder to compute embeddings for each of the questions, and concatenate them to form a 1-D vector with all the embeddings.

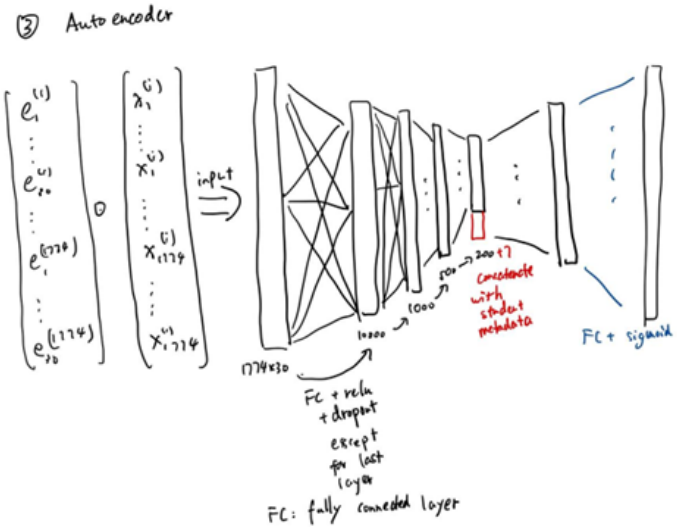

Figure 10: prediction network

Figure 10: the embedder takes in the Haderman product of x' and the embedding vector v, the vector x' acts as an indicator variable. Then it passes through various layers, with student metadata appended in middle, and finally forms the prediction vector

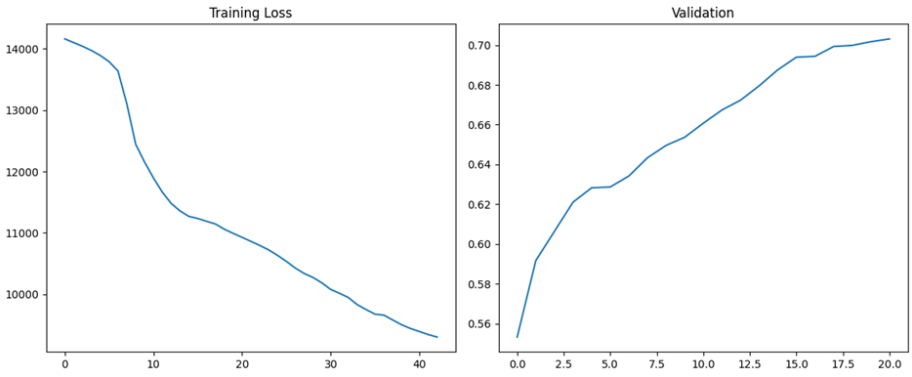

Figure 11: Enter Caption

Figure 11: training loss and validation accuracies plot for our model.

Modified Model versus Baseline Models

Testing performance for the models

First, we thoroughly test their performances and visually compare them:

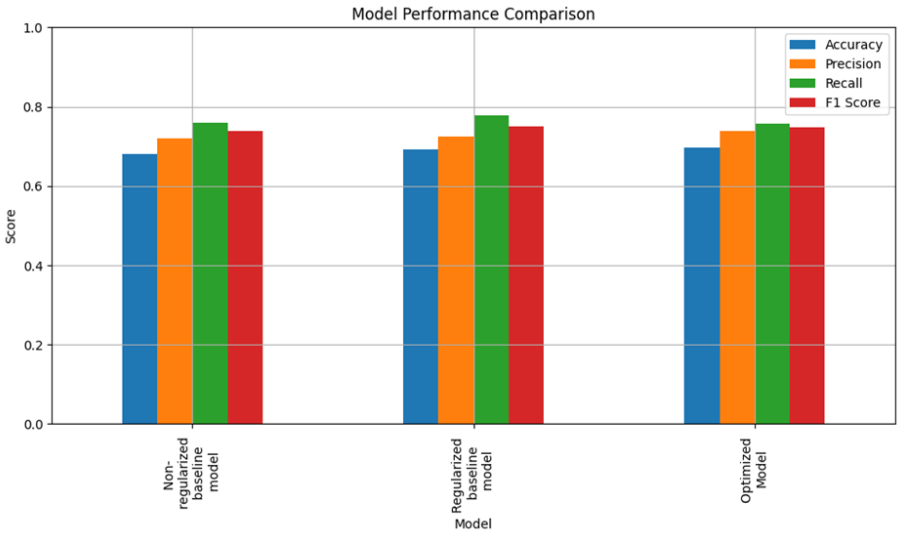

Figure 12: performance comparison

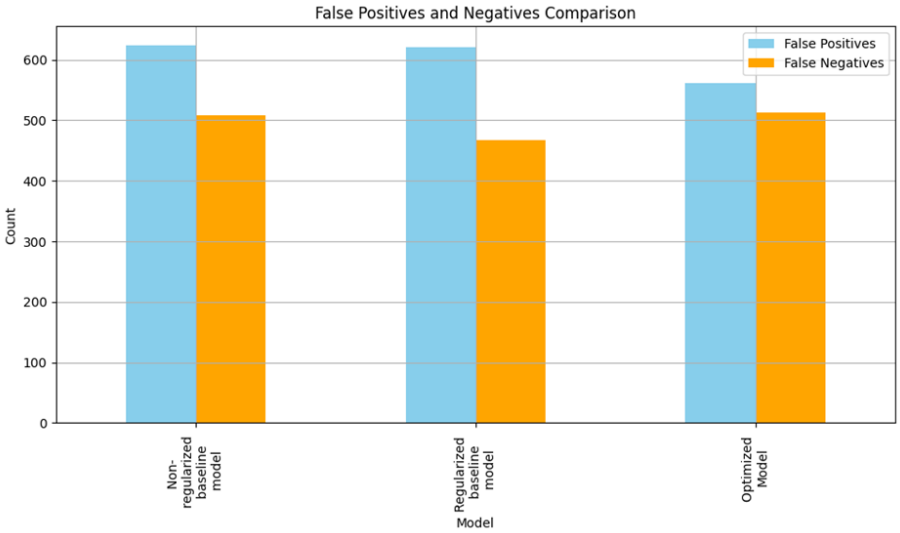

Figure 13: false positive and false negative counts

Looking at the problem domain:

In the domain of educational technology, the need to minimize false positives takes on a unique importance. A false positive occurs when a model incorrectly predicts that a student can answer a question correctly when, in reality, they cannot. This misprediction can lead to missed learning opportunities, as students might not be provided with the set of practice necessary to master content areas where they are struggling. Compared to false negatives, which represent an underestimation of a student's understanding and lead to unnecessary practice recommendations, false positives are considered more detrimental. They directly contribute to gaps in knowledge that could hinder a student's overall learning progression.

Thus, our part (b) optimized model has better performance than the baseline models. It achieves the highest balance of precision in predicting when students truly cannot answer questions, coupled with a more favorable rate of false positives compared to the baseline models. This distinction makes our optimized model particularly well-suited to the domain's focus on accurately identifying and addressing students' learning needs.

Detailed Analysis

Accuracy and Precision

Though the three models exhibit similar levels of overall accuracy, our optimized model in part (b) stands out for its precision. This metric is critical in this domain, as it reflects the model's capability to correctly identify instances where students are likely to struggle with questions. Our optimized model's superior precision suggests it is the most reliable at ensuring students receive practice for the questions they truly need help with, thereby minimizing false positives.

Recall and F1 Score

Recall, the ability of the model to capture all instances where additional practice is needed, shows the Regularized baseline model with a slight edge. However, given the emphasis on minimizing false positives, our optimized model's balanced performance, as reflected in its F1 Score, demonstrates its effectiveness in maintaining a strong precision-recall balance. This balance is crucial for optimizing the learning experience by providing targeted interventions without overwhelming students with unnecessary practice.

False Positives and Negatives

Our optimized model's lower rate of false positives is a key factor in its selection as the preferred model. By reducing the likelihood of underestimating students' learning needs, our optimized model helps ensure that critical gaps in knowledge are identified and addressed through recommended question sets, aligning with the educational goals of adaptive learning platforms.

Comparison Conclusion

Given the paramount importance of minimizing false positives to prevent overlooked learning opportunities, our optimized model emerges as the optimal choice for integration into adaptive learning systems. Its ability to accurately predict when students cannot answer questions correctly supports a more effective and efficient personalized learning experience.

Further refinement of our optimized model could aim to enhance its recall capabilities, ensuring an even broader capture of students' learning needs without significantly increasing false positives. Such advancements would further solidify our optimized model's role in supporting adaptive learning technologies, underscoring the critical role of precise predictive modeling in advancing educational outcomes.

Model's Performance Increased

Our revised model have a better performance than the base line model, here are some possible points that we get benefits from:

1.Increased depth of the neural network

Our model added extra hidden layers compared with the base-line model. This increase in depth allows for more complex representations of the data, enabling the model to capture higher-order interactions between features. The additional layers provide a greater capacity for learning, which can lead to improved performance, especially when there is sufficient data and regularization techniques are used to prevent overfitting. We also added dropout layers as regularization to prevent over-fitting and improve generalization.

2.Additional training data

In the base-line model, we are only using the primary data sets to train the model, but in our modified one, we introduced some new meta-datasets such as subject_meta.csv and question_meta.csv. This additional data provides more context and information about the subjects and questions, allowing the model to make more informed predictions. The inclusion of meta-data can help the model understand the underlying structure of the data better, leading to improved generalization and performance.

3.Usage of embedding layer

Our new model used an embedding layer, which enables us to represent the data of the subject of a particular question as a dense vector. This representation allows the model to capture the relationships between different subjects and questions in a more nuanced way. Embeddings can help the model learn semantic similarities and differences between subjects, which can be crucial for making accurate predictions in tasks involving natural language or categorical data.

4.Change of activation functions

In the base-line model, all the activation functions used are sigmoid functions. By comparison, our new model used ReLU (Rectified Linear Unit) in every hidden layer and sigmoid at the output layer. ReLU activation functions help alleviate the vanishing gradient problem, allowing for deeper networks to be trained more effectively. They also tend to converge faster in practice compared to sigmoid functions. Using ReLU in the hidden layers and sigmoid at the output layer is a common practice in binary classification tasks, as it combines the benefits of fast training and the ability to output probabilities.

Limitation

There are many limitations with our models, but one thing which stands out is the autoencoder's sensitivity to input data distribution. Our model may perform poorly in settings where not a lot of input data are given, or it is not clean, i.e. there exists a lot of outliers or is highly skewed. For example, if we are given a dataset where a significant portion are "outliers", the model may encounter difficulty reconstructing these outliers.

This limitation exists because autoencoders try to compress and reconstruct the data based on the dominant patterns in the training data, so when the data is skewed it may not be able to learn a representation that adequately captures the characteristics of these atypical examples. Some possible modification includes implementing a more robust loss function that reduces the influence of outliers on the training process, after research, I've found loss functions such as Huber loss which seems to be more resilient to anomalies in the data that MSe, improving its ability to generalize across a wider range of inputs.

Conclusion

In conclusion, our project explored various machine learning algorithms for predicting the correctness of students' answers in an online education platform. We successfully implemented and analyzed these algorithms and proposed modifications to enhance their performance.

Team contribution

Every team member actively participated in the project. We held weekly meetings to discuss progress and address any challenges that occurred. In addition to our regular meetings, we are attending office hours together whenever we encountered complex problems that required further clarification or guidance. Everyone did a great job, from the initial research and planning stages to the final implementation and testing. For part A, Jackson is responsible for the KNN implementation, Alysa and Jackson worked together on the IRT section, Alwnyn and David focused on the neural network part. For part B Alwnyn

came up with the idea of embedding and successfully implemented it into our model, which resulted in improved performance. For the part B report Alwnyn described our modified model and generated various graphs, Alysa conducted the comparison test between our model and the baseline model, Jackson explained why our model performed better than the baseline model, David points out the limitation of our model and its reason.

References

[1] PyTorch Tutorials. (n.d.). *Word Embeddings: Encoding Lexical Semantics*. Retrieved from `https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`

[2] Shan, Y., Hoens, T. R., Jiao, J., Wang, H., Yu, D., & Mao, J. (2016). *Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16) (pp. 255-262). New York, NY, USA: Association for Computing Machinery. `https://doi.org/10.1145/2939672.2939704`

[3] Course materials and project instructions.

[4] OpenAI. (n.d.). *ChatGPT*. Retrieved from `https://chat.openai.com`