Midterm Project: Predicting Student Performance

Machine Learning Course

Fall 2024

1 Introduction

The purpose of this project is to build and refine machine learning models that can estimate a student's likelihood of correctly answering specific diagnostic questions, based on their past performance and that of other students. Such models are crucial for educational platforms where personalized feedback and adaptive learning paths are used to enhance learning experiences.

In this project, you will implement machine learning algorithms, evaluate their performance, and propose improvements. You will work with an existing dataset, where your task is to predict whether a student will correctly answer a set of questions based on historical data.

2 Project Objectives

In this project, you are required to:

- Implement machine learning models covered in the course and apply them to a real-world dataset.
- Compare the performances of different models and identify areas where improvements can be made.
- Propose modifications to the models that enhance their prediction accuracy.
- Document your findings in a concise report, including analysis and explanation of any modifications.

3 Background and Dataset

The dataset provided for this project is a subset of a real-world educational platform, consisting of student responses to mathematical diagnostic questions. The objective is to predict student performance on future questions by analyzing their past answers and comparing them with other students' responses.

The dataset is composed of:

- **train_data.csv:** Contains three columns:
  - *question_id*: Identifies each diagnostic question.
  - *user_id*: Identifies each student.
  - *is_correct*: A binary indicator (0 or 1) specifying whether the student answered correctly.
- **valid_data.csv:** A validation dataset used to fine-tune the models.
- **test_data.csv:** A test dataset for evaluating final model performance.
- **sparse_matrix.npz:** A matrix representation of the responses, where rows represent students, columns represent questions, and the entries indicate whether the student answered correctly (1), incorrectly (0), or if data is missing (NaN).

In addition, metadata is provided for further analysis:

- **question_meta.csv:** Describes the subject matter of each question.
- **student_meta.csv:** Includes details about the students, such as gender and eligibility for free school meals.

4 Part A: Machine Learning Models

4.1 k-Nearest Neighbor (kNN)

In this part, you will implement the k-Nearest Neighbor (kNN) algorithm to predict whether a student will answer a question correctly based on the answers of other students with similar response patterns. You will explore both user-based and item-based collaborative filtering.

- **Step 1:** Implement the kNN algorithm for user-based collaborative filtering. In this approach, the idea is that students who answered similar questions in a similar way are likely to perform similarly on future questions. Use the provided code framework in `knn.py`.
- **Step 2:** Vary the value of k (e.g., $k = 1, 6, 11, 16, 21, 26$) and observe how the accuracy of predictions changes with different numbers of nearest neighbors. Use the validation dataset to choose the optimal value of k. Report your findings with a plot that shows accuracy as a function of k.
- **Step 3:** Modify the code to implement item-based collaborative filtering. In this case, you predict student performance based on how similar the current question is to other questions the student has already answered. Repeat the analysis with different values of k and compare the performance of item-based filtering to user-based filtering.
- **Step 4:** Write a brief comparison discussing which method (user- or item-based) performed better and suggest reasons why. For example, did certain types of students or questions perform better with one approach over the other?

4.2 Item Response Theory (IRT)

You will now implement an Item Response Theory (IRT) model, which assigns a difficulty parameter to each question and an ability parameter to each student. The goal is to model the probability that a student will answer a question correctly based on these two factors.

- **Step 1:** The IRT model assumes that the probability of a correct answer is given by:

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

  where θ_i represents the ability of student i, and β_j represents the difficulty of question j.
- **Step 2:** Derive the log-likelihood function for the model, which describes how likely it is that the observed data matches the model's predictions. Implement the gradient of the log-likelihood with respect to the parameters θ and β for optimization purposes. Use alternating gradient descent to estimate the parameters.
- **Step 3:** Train your IRT model using the provided training data and report the log-likelihoods on both the training and validation datasets as a function of the number of iterations. Adjust the learning rate and the number of iterations as necessary to achieve a stable training curve.
- **Step 4:** Choose three distinct questions and plot the probability of a correct response as a function of student ability (θ) for these questions. These plots will help you visualize how the difficulty of questions affects their likelihood of being answered correctly by students of varying abilities.
- **Step 5:** Summarize the results and discuss what these probability curves indicate about the differences between easy and difficult questions, as well as students with varying levels of ability.

4.3 Ensemble Learning

In this section, you will implement an ensemble method to combine the predictions of multiple models, which can improve the overall accuracy and stability of the predictions. The specific ensemble technique you will use is **bagging**.

- **Step 1:** Choose three base models that you have already implemented, such as k-Nearest Neighbor, Item Response Theory (IRT), or Matrix Factorization. Train these models independently on the training data.

- **Step 2:** For each model, apply **bootstrap aggregating (bagging)**. In this technique, you will generate multiple versions of the training set by sampling with replacement and train each base model on these different versions of the data. Use the training data to generate at least 3 bootstrapped datasets.
- **Step 3:** For each test sample, predict whether the student will answer a question correctly using each of the three base models. To obtain the final prediction, average the predictions from the three models (or take the majority vote if using classification).
- **Step 4:** Evaluate the performance of the ensemble by calculating the accuracy on the validation and test datasets. Compare the results of the ensemble with the individual base models and explain whether the ensemble improved the overall prediction accuracy and why that might be the case.
- **Step 5:** Discuss the advantages of using ensemble methods, such as their ability to reduce overfitting and improve prediction stability. Also mention any limitations of your ensemble model in this context.

4.4 Matrix Factorization or Neural Networks

You now have the choice to either implement a matrix factorization method or a neural network model to predict student responses. Select one approach and follow the corresponding steps.

4.4.1 Matrix Factorization

- **Step 1:** Use singular value decomposition (SVD) to factorize the response matrix into two smaller matrices. These matrices represent latent factors for students and questions. Test several values of the latent dimension k, such as $k = 5, 10, 20, 50, 100$, to find the best performing k using the validation set. Report the validation and test accuracy for the best k.
- **Step 2:** Alternatively, implement an alternating least squares (ALS) approach using stochastic gradient descent (SGD) to optimize the latent factors. Report the training and validation loss as a function of the number of iterations.
- **Step 3:** Compare the results of the SVD and ALS methods. Discuss any limitations of matrix factorization for this problem, such as how missing data is handled.

4.4.2 Neural Networks

- **Step 1:** Implement an autoencoder using neural networks to reconstruct the input data (student responses). The autoencoder should have two layers: a latent layer and an output layer. You will use the sigmoid activation function for both the hidden and output layers.
- **Step 2:** Train the network for different values of the latent dimension k (e.g., $k = 10, 50, 100, 200, 500$) and evaluate the model's performance using the validation dataset. Adjust hyperparameters like the learning rate and the number of iterations to optimize performance.
- **Step 3:** Add L2 regularization to the network's objective function to prevent overfitting. Test different regularization strengths ($\lambda = 0.001, 0.01, 0.1, 1$) and observe how they affect the validation accuracy. Report the final validation and test accuracy with the best k and λ.

5 Part B: Algorithm Modification

Based on the results from Part A, you are now expected to modify one of the algorithms you implemented to improve its prediction accuracy. For example, you could adjust the model to handle overfitting, include additional metadata (e.g., student gender or question subject), or experiment with new optimization techniques.

- Clearly define the modification you are making and explain why you expect it to improve performance.
- Rigorously evaluate the modified algorithm, comparing its accuracy to the baseline models from Part A.

- Provide visualizations (e.g., tables or graphs) to support your findings.
- Discuss any limitations of your approach and suggest potential improvements.