Final Project

Deadline: Friday, Dec.2 2022, at 11:59pm

Submission: You need to submit the following files through MarkUs[1]:

- Your answers to Part A and B, as a PDF file titled `final_report.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

- The Python codes you used for both Part A and B, as a zip file `code.zip`. You may exclude the folder `/data` from the starter code.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Collaboration: You should form teams of 2-3 students. More than 3 students will not be permitted. There will not be a separate marking scheme should you chose to work alone. **Your final report should list the contributions of each team member**.

1 Introduction

One of CSC311's main objectives is to prepare you to apply machine learning algorithms to real-world tasks. The final project aims to help you get started in this direction. You will be performing the following tasks:

- Try out existing algorithms to real-world tasks.

- Modify an existing algorithm to improve performance.

- Write a short report analyzing the result.

The final project is not intended to be a stressful experience. It is a good chance for you to experiment, think, play, and hopefully have fun. These tasks are what you will be doing daily as a data analyst/scientist or machine learning engineer.

2 Background & Task

Online education services, such as Khan Academy and Coursera, provide a broader audience with access to high-quality education. On these platforms, students can learn new materials by watching a lecture, reading course material, and talking to instructors in a forum. However, one disadvantage of the online platform is that it is challenging to measure students' understanding of the course material. To deal with this issue, many online education platforms include an assessment component to ensure that students understand the core topics. The assessment component is often composed of diagnostic questions, each a multiple choice question with one correct answer. The diagnostic question is designed so that each of the incorrect answers highlights a common misconception. An example of the diagnostic problem is shown in figure 1. When students incorrectly answer the diagnostic question, it reveals the nature of their misconception and, by understanding these misconceptions, the platform can offer additional guidance to help resolve them.
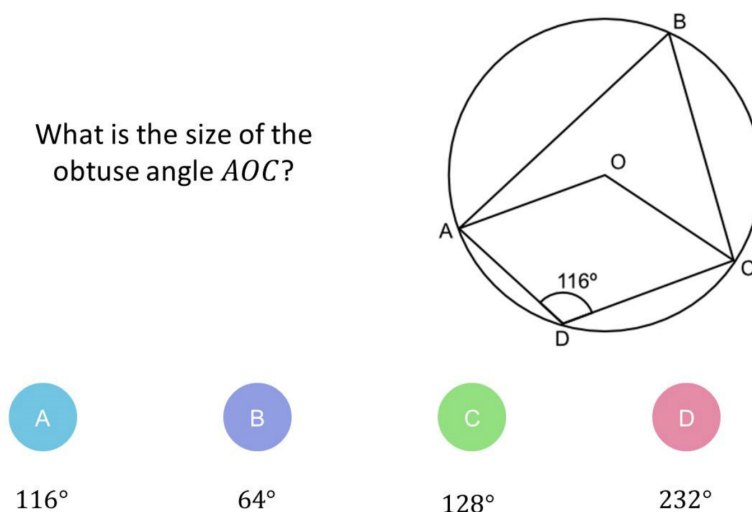
---

[1] https://markus.teach.cs.toronto.edu/2022-09

Figure 1: An example diagnostic question [1].

In this project, you will build machine learning algorithms to predict whether a student can correctly answer a specific diagnostic question based on the student's previous answers to other questions and other students' responses. Predicting the correctness of students' answers to as yet unseen diagnostic questions helps estimate the student's ability level in a personalized education platform. Moreover, these predictions form the groundwork for many advanced customized tasks. For instance, using the predicted correctness, the online platform can automatically recommend a set of diagnostic questions of appropriate difficulty that fit the student's background and learning status.

You will begin by applying existing machine learning algorithms you learned in this course. You will then compare the performances of different algorithms and analyze their advantages and disadvantages. Next, you will modify existing algorithms to predict students' answers with higher accuracy. Lastly, you will experiment with your modification and write up a short report with the results.

You will measure the performance of the learning system in terms of prediction accuracy, although you are welcome to include other metrics in your report if you believe they provide additional insight:

$$\text{Prediction Accuracy} = \frac{\text{The number of correct predictions}}{\text{The number of total predictions}}$$

3 Data

We subsampled answers of 542 students to 1774 diagnostic questions from the dataset provided by Eedi[2], an online education platform that is currently being used in many schools [1]. The platform offers crowd-sourced mathematical diagnostic questions to students from primary to high school (between 7 and 18 years old). The truncated dataset is provided in the folder /data.
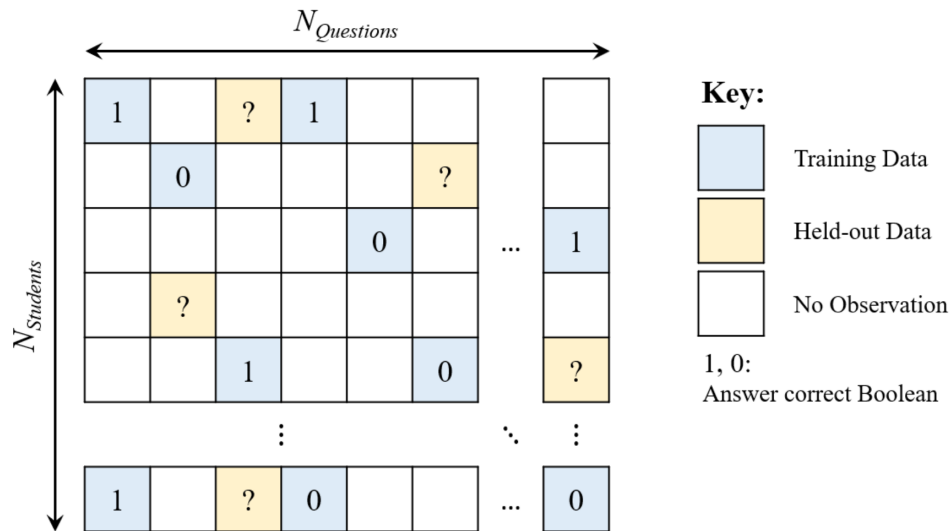
---

[2]https://eedi.com/

Figure 2: An example sparse matrix [1].

3.1   Primary Data

The primary data, `train_data.csv`, is the main dataset you will be using to train the learning algorithms throughout the project. There is also a validation set `valid_data.csv` that you should use for model selection and a test set `test_data.csv` that you should use for reporting the final performance. All primary data `csv` files are composed of 3 columns:

- **question_id:** ID of the question answered (starts from 0).

- **user_id:** ID of the student who answered the question (starts from 0).

- **is_correct:** Binary indicator whether the student's answer was correct (0 is incorrect, 1 is correct).

We also provide a sparse matrix, `sparse_matrix.npz`, where each row corresponds to the **user_id** and each column corresponds to the **question_id**. An illustration of the sparse matrix is shown in figure 2. The correct answer given a pair of (**user_id**, **question_id**) will have an entry 1 and an incorrect answer will have an entry 0. Answers with no observation and held-out data (that will be used for validation and test) will have an entry NaN (`np.NaN`).

3.2   Question Metadata

We also provide the question metadata, `question_meta.csv`, which contains the following columns:

- **question_id:** ID of the question answered (starts from 0).

- **subject_id:** The subject of the question covered in an area of mathematics. The text description of each subject is provided in `subject_meta.csv`.

3.3   Student Metadata

Lastly, we provide the student metadata, `student_meta.csv`, that is composed of the following columns:

- **user_id:** ID of the student who answered the question (starts from 0).

- **gender:** Gender of the student, when available. 1 indicates a female, 2 indicates a male, and 0 indicates unspecified.

- **data_of_birth:** Birth date of the student, when available.

- **premium_pupil:** Student's eligibility for free school meals or pupil premium due to being financially disadvantaged, when available.

4   Part A

In the first part of the project, you will implement and apply various machine learning algorithms you studied in the course to predict students' correctness of a given diagnostic question. Review the course notes if you don't recall the details of each algorithm. For this part, you will only be using the primary data: `train_data.csv`, `sparse_matrix.npz`, `valid_data.csv`, and `test_data.csv`. Moreover, you may use the helper functions provided in `utils.py` to load the dataset and evaluate your model. You may also use any functions from packages `NumPy`, `Scipy`, `Pandas`, and `PyTorch`. Make sure you understand the code instead of using it as a black box.

1. [**5pts**] **k-Nearest Neighbor.** In this problem, using the provided code at `part_a/knn.py`, you will experiment with k-Nearest Neighbor (kNN) algorithm.

   The provided kNN code performs collaborative filtering that uses other students' answers to predict whether the specific student can correctly answer some diagnostic questions. In particular, the starter code implements user-based collaborative filtering: given a user, kNN finds the closest user that similarly answered other questions and predicts the correctness based on the closest student's correctness. The core underlying assumption is that if student A has the same correct and incorrect answers on other diagnostic questions as student B, A's correctness on specific diagnostic questions matches that of student B.

   (a) Complete a function `main` located at `knn.py` that runs kNN for different values of $k \in \{1, 6, 11, 16, 21, 26\}$. Plot and report the accuracy on the validation data as a function of k.

   (b) Choose k^* that has the highest performance on validation data. Report the chosen k^* and the final test accuracy.

   (c) Implement a function `knn_impute_by_item` on the same file that performs item-based collaborative filtering instead of user-based collaborative filtering. Given a question, kNN finds the closest question that was answered similarly, and predicts the correctness basted on the closest question's correctness. State the underlying assumption on item-based collaborative filtering. Repeat part (a) and (b) with item-based collaborative filtering.

   (d) Compare the test performance between user- and item- based collaborative filtering. State which method performs better.

   (e) List at least two potential limitations of kNN for the task you are given.

2. [**15pts**] **Item Response Theory.** In this problem, you will implement an Item-Response Theory (IRT) model to predict students' correctness to diagnostic questions.

The IRT assigns each student an ability value and each question a difficulty value to formulate a probability distribution. In the one-parameter IRT model, β_j represents the difficulty of the question j, and θ_i that represents the i-th students ability. Then, the probability that the question j is correctly answered by student i is formulated as:

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

We provide the starter code in `part_a/item_response.py`.

(a) Derive the log-likelihood $\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ for all students and questions. Here $\mathbf{C}$ is the sparse matrix. Also, show the derivative of the log-likelihood with respect to θ_i and β_j (Hint: recall the derivative of the logistic model with respect to the parameters).

(b) Implement missing functions in `item_response.py` that performs alternating gradient descent on $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ to maximize the log-likelihood. Report the hyperparameters you selected. With your chosen hyperparameters, report the training curve that shows the training and validation log-likelihoods as a function of iteration.

(c) With the implemented code, report the final validation and test accuracies.

(d) Select three questions j_1, j_2, and j_3. Using the trained $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, plot three curves on the same plot that shows the probability of the correct response $p(c_{ij} = 1)$ as a function of θ given a question j. Comment on the shape of the curves and briefly describe what these curves represent.

3. [**15pts**] **Matrix Factorization OR Neural Networks.** In this question, please read both option (i) and option (ii), but you only need to do one of the two.

   (i) **Option 1: Matrix Factorization.** In this problem, you will be implementing matrix factorization methods. The starter code is located at `part_a/matrix_factorization`.

      (a) Using a function `svd_reconstruct` that factorizes the sparse matrix using singular-value decomposition, try out at least 5 different k and select the best k using the validation set. Report the final validation and test performance with your chosen k.

      (b) State one limitation of SVD in the task you are given. (Hint: how are you treating the missing entries?)

      (c) Implement functions `als` and `update_u_z` located at the same file that performs alternating updates. You need to use SGD to update $\mathbf{u}_n \in \mathbb{R}^k$ and $\mathbf{z}_m \in \mathbb{R}^k$ as described in the docstring. To be noted, this is different from the alternating least squares (ALS) we introduced in the course slides, where we used the direct solution. As a reminder, the objective is as follows:

$$\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(n,m) \in O} \left(C_{nm} - \mathbf{u}_n^\top \mathbf{z}_m \right)^2,$$

      where $\mathbf{C}$ is the sparse matrix and $O = \{(n, m) : \text{entry } (n, m) \text{ of matrix } \mathbf{C} \text{ is observed}\}$.

      (d) Learn the representations $\mathbf{U}$ and $\mathbf{Z}$ using ALS with SGD. Tune learning rate and number of iterations. Report your chosen hyperparameters. Try at least 5 different values of k and select the best k^* that achieves the lowest validation accuracy.

(e) With your chosen k^*, plot and report how the training and validation squared-error-losses change as a function of iteration. Also report final validation accuracy and test accuracy.

(ii) **Option 2: Neural Networks.** In this problem, you will implement neural networks to predict students' correctness on a diagnostic question. Specifically, you will design an autoencoder model. Given a user $\mathbf{v} \in \mathbb{R}^{N_{questions}}$ from a set of users $\mathcal{S}$, our objective is:

$$\min_{\boldsymbol{\theta}} \sum_{\mathbf{v} \in \mathcal{S}} \|\mathbf{v} - f(\mathbf{v}; \boldsymbol{\theta})\|_2^2,$$

where f is the reconstruction of the input $\mathbf{v}$. The network computes the following function:

$$f(\mathbf{v}; \boldsymbol{\theta}) = h(\mathbf{W}^{(2)} g(\mathbf{W}^{(1)} \mathbf{v} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \in \mathbb{R}^{N_{\text{questions}}}$$

for some activation functions h and g. In this question, you will be using sigmoid activation functions for both. Here, $\mathbf{W}^{(1)} \in \mathbb{R}^{k \times N_{questions}}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{N_{questions} \times k}$, where $k \in \mathbb{N}$ is the latent dimension. We provide the starter code written in `PyTorch` at `part_a/neural_network`.

(a) Describe at least three differences between ALS and neural networks.

(b) Implement a class `AutoEncoder` that performs a forward pass of the autoencoder following the instructions in the docstring.

(c) Train the autoencoder using latent dimensions of $k \in \{10, 50, 100, 200, 500\}$. Also, tune optimization hyperparameters such as learning rate and number of iterations. Select k^* that has the highest validation accuracy.

(d) With your chosen k^*, plot and report how the training and validation objectives changes as a function of epoch. Also, report the final test accuracy.

(e) Modify a function `train` so that the objective adds the L_2 regularization. The objective is as follows:

$$\min_{\boldsymbol{\theta}} \sum_{\mathbf{v} \in \mathcal{S}} \|\mathbf{v} - f(\mathbf{v}; \boldsymbol{\theta})\|_2^2 + \frac{\lambda}{2}(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2)$$

You may use a method `get_weight_norm` to obtain the regularization term. Using the k and other hyperparameters selected from part (d), tune the regularization penalty $\lambda \in \{0.001, 0.01, 0.1, 1\}$. With your chosen λ, report the final validation and test accuracy. Does your model perform better with the regularization penalty?

4. **[15pts] Ensemble.** In this problem, you will be implementing bagging ensemble to improve the stability and accuracy of your base models. Select and train 3 base models with bootstrapping the training set. You may use the same or different base models. Your implementation should be completed in `part_a/ensemble.py`. To predict the correctness, generate 3 predictions by using the base model and average the predicted correctness. Report the final validation and test accuracy. Explain the ensemble process you implemented. Do you obtain better performance using the ensemble? Why or why not?

5   Part B

In the second part of the project, you will modify one of the algorithms you implemented in part A to hopefully predict students' answers to the diagnostic question with higher accuracy. In particular, consider the results obtained in part A, reason about what factors are limiting the performance of one of the methods (e.g. overfitting? underfitting? optimization difficulties?) and come up with a proposed modification to the algorithm which could help address this problem. Rigorously test the performance of your modified algorithm, and write up a report summarizing your results as described below.

You will not be graded on how well the algorithm performs (i.e. its accuracy); rather, your grade will be based on the quality of your analysis. Try to be creative! You may also optionally use the provided metadata (`question_meta.csv` and `student_meta.csv`) to improve the accuracy of the model. At last, you are free to use any third-party ideas or code as long as it is publicly available. You must properly provide references to any work that is not your own in the write-up.

The length of your report for part B should be 3-4 pages. Don't be afraid to keep the text short and to include large illustrative figures. The guidelines and marking schemes are as follows:

1. [**15pts**] **Formal Description:** Precisely define the way in which you are extending the algorithm. You should provide equations and possibly an algorithm box. Describe way your proposed method should be expected to perform better. For instance, are you intending it to improve the optimization, reduce overfitting, etc.?

2. [**10pts**] **Figure or Diagram:** that shows the overall model or idea. The idea is to make your report more accessible, especially to readers who are starting by skimming your report.

3. [**15pts**] **Comparison or Demonstration:** Include:

   - A comparison of the accuracies obtained by your model to those from baseline models. Include a table or a plot for an illustrative comparison.

   - Based on the argument you gave for why your extension should help, design and carry out an experiment to test your hypothesis. E.g., consider how you would disentangle whether the benefits are due to optimization or regularization.

4. [**15pts**] **Limitations:** of your approach.

   - Describe some settings in which we'd expect your approach to perform poorly, or where all existing models fail.

   - Try to guess or explain why these limitations are the way they are.

   - Give some examples of possible extensions, ways to address these limitations, or open problems.

6   Optional: Competition

Optionally, you will take part in a competition where you will submit the predictions of your model on a private test dataset. We will be using a platform called Kaggle[3], which is a popular online community of data scientists and machine learning practitioners to solve data science challenges.

---

[3]https://www.kaggle.com/

You will first form a team on Kaggle and submit your prediction as a `csv` file on the project Kaggle website.

As you prep for it, your submission should be a `csv` file with two columns **id** and **is_correct**. You should use the provided functions `load_private_test_csv` to load the dataset and `save_private_test_csv` to save the predictions as a `csv` file both located at `utils.py`. Once you save the `csv` file, you can directly upload it to the competition page. After the submission, you can find your model's performance on the public leaderboard, which displays an accuracy of your predictions on 70% of private data. The full leaderboard that uses all private data will be released after the competition.

You are not required to use your model from Part B for this part.

Performance on the competition will not affect your grade, not even as extra credit. The top 3 teams with the highest performance on all private data will receive a small gift from the instructors.

The submission link for Kaggle can be found here:

<div align="center">

https://www.kaggle.com/competitions/csc311-fall-2022

</div>

7 Friendly Advice

- **Read carefully!** Make sure you are following the guidelines. Read this document carefully to understand the dataset, what you are asked to implement and demonstrate, etc. Ask questions on Piazza and visit office hours if you are unsure of any requirements.

- **Be honest!** You are not being marked on how good the results are. It doesn't matter if your method is better or worse than the ones you compare to. What matters is that you clearly describe the problem, your method, what you did, and what the results were. Just be scientific.

- **Be careful!** Don't do things like test on your training data, set hyperparameters using test accuracy, compare unfairly against other methods, include plots with unlabeled axes, use undefined symbols in equations, etc. Do sensible crosschecks like running your algorithms several times to understand the between-run variability, performing gradient checking, etc.

References

[1] Wang, Zichao, et al. *Diagnostic Questions: The NeurIPS 2020 Education Challenge.* arXiv preprint arXiv:2007.12061 (2020)