CSE 158/258, Fall 2022: Midterm

Instructions

Midterm is due at 9am, on **Wednesday November 2**. Submissions should be made via gradescope.

The midterm is worth 26 marks.

You can base your solution on the stub code provided here:
`https://cseweb.ucsd.edu/classes/fa22/cse258-a/files/Midterm_stub.ipynb`
`https://cseweb.ucsd.edu/classes/fa22/cse258-a/files/Midterm_stub.html`

You should submit two files:

`answers_midterm.txt` should contain a python dictionary containing your answers to each question. Its format should be like the following:

```
{ "Q1": 1.5, "Q2": [3,5,17,8], "Q2": "b", (etc.) }
```

The provided code stub demonstrates how to prepare your answers and includes an answer template for each question.

`midterm.py` A python file containing working code for your solutions. The autograder *will not execute your code*; this file is required so that we can assign partial grades in the event of incorrect solutions, check for plagiarism, etc. Your solution should **clearly document which sections correspond to each question and answer**. We may occasionally run code to confirm that your outputs match submitted answers, so **please ensure that your code generates the submitted answers.**

All questions in this midterm will make use of a small dataset of *spoiler* data from *Goodreads*, which can be downloaded from:
`https://cseweb.ucsd.edu/classes/fa22/cse258-a/files/spoilers.json.gz`

Each data point consists of a review, broken into several sentences, each containing a tag for whether that sentence contains a 'spoiler' (i.e., reviews key plot details). A few relevant fields include:

`user_id`, `book_id` The ID of the user and the book in each review.

`timestamp` When the review was entered.

`rating` The rating associated with the review.

`review_sentences` The review, as a list of sentences. Each entry in the list contains a binary (0/1) indicator for whether the review contains a spoiler, followed by the text of the sentence.

Note: none of these experiments should require more than a few seconds to run on modest hardware. Please only use a smaller fraction if absolutely stuck;[1] we may only give partial credit if we cannot verify the correctness of your solution.

Section 1: Regression

Autoregression refers to predicting the next entry in a sequence based on the previous values. In this section, we'll use autoregression to predict the next rating associated with a user or item based on the previous ratings.

Implement an autoregression based model to predict the rating of each review in terms of the previous ratings for each user or each item. Data structures have been provided containing a (time-sorted) list of reviews per-user and per-item. We'll compare two approaches:

- Model 1: Predict the most recent rating based on the user's previous ratings
- Model 2: Predict the most recent rating based on the item's previous ratings

Complete the following tasks:

1. For each user and item, estimate the average across all of their previous ratings, and use that to estimate their final rating (i.e., predict `ratings[-1]` from `ratings[:-1]`). Report the MSE of:
   (a) Model 1: For each user, predict based on the average of their previous ratings;
   (b) Model 2: For each item, predict based on the average of its previous ratings.

   Note that a correct implementation should compute the average by *excluding the rating you are trying to predict*. You may skip any users or items that have only a single rating in their history. (2 marks)

2. Rather than computing the average across all ratings, implement a model which only computes the average across the last N ratings (again you may discard instances with only a single rating, and truncate instances with fewer than $N+1$ ratings).[2] Report the performance (MSE) of a user-based model (Model 1) for window sizes of $N \in \{1, 2, 3\}$. (3 marks)

3. Implement an *autoregression*-based model that uses the user's last N ratings as features (i.e., Model 1) in a regressor (in addition to a constant feature). That is, for $N = 2$ your model would be:

$$\text{next rating} = \theta_0 + \theta_1 \times [\text{most recent rating}] + \theta_2 \times [\text{second most recent rating}]$$

   Compute:

   (a) The feature vectors associated with user in the first review (i.e., `dataset[0]`) given window sizes of $N = 2$ and $N = 3$. (2 marks)
   (b) The performance of the autoregressor (MSE) for window sizes of $N \in \{1, 2, 3\}$ (in this case you may discard instances for which too few historical ratings exist).[3] (2 marks)

[1] If doing so, use the first 10% of the data
[2] I.e., when you truncate, just take an average over as many ratings as are available.
[3] As in Question 1 you are still computing `ratings[-1]` from `ratings[:-1]`.

4. As you increase the window size N, it will become increasingly likely that too few entries are available, and entries will be discarded. Instead of discarding such entries, here we'll explore two strategies to deal with missing features:

   - Mean-value imputation: For each index that is missing, replace it by the average for that user (again discarding the rating you are trying to predict when computing the average).[4]
   - Missing-value indicator: For each index, use *two* features as follows:
     - `[1,0]` if the feature is missing;
     - `[0,x]` if the feature (i.e., the historical rating x) is available

     (meaning that for a window size of N, you will have $2N$ dimensions, plus a constant feature)

   Your answer should contain:

   (a) The feature vector for the first user in the dataset under both representations (mean-value and missing-value), for a window size of $N = 10$; (2 marks)

   (b) The MSE for a user model (Model 1) under both representations. (2 marks)

Section 2: Classification

In this question we'll focus on predicting *spoiler* labels for individual sentences in reviews.

5. Build a predictor that estimates whether a sentence contains a spoiler. The label for each sentence is the first value in each entry of `review_sentences`. Your predictor should take the form:

$$p(\text{sentence contains spoiler} = \\ \sigma(\theta_0 + \theta_1[\texttt{length in chars}] + \theta_2[\texttt{number of `!'}] + \theta_3[\texttt{number of capital letters}])$$

   Implement a classifier on all sentences; use the `class_weight='balanced'` option and a regularization constant of `C=1` (i.e., follow the code in the stub). Report:

   (a) The feature vector associated with the first sentence (including the constant feature). (1 mark)

   (b) The number of True Positives,[5] True Negatives, False Positives, False Negatives, and Balanced Error Rate (BER) of the model. (2 marks)

6. Much like we used an autoregression-based approach to estimate ratings in Section 1, we can use a similar approach to build *classifiers*. Build a classifier whose feature vector is the spoiler label for the first $N = 5$ sentences in order to predict the spoiler label for the 6th sentence in each review (you may discard any reviews with fewer than six sentences). Concatenate the autoregressive features with those from Question 5 (so that your feature dimension should be 9). Again use the `class_weight='balanced'` option and a regularization constant of `C=1`. Report:

   (a) The feature vector associated with the first review (including the constant feature). (1 mark)

   (b) The Balanced Error Rate of the model. (1 mark)

7. Build a *regularization pipeline* for your model from Question 6. Use the first 50% of the data for training, the next 25% for validation, and the final 25% for testing (follow the data split outlined in the stub). Consider values of C in the range $\{0.01, 0.1, 1, 10, 100\}$. Report the validation performance (BER) for each value of C; the best value of C; and its performance (BER) on the test set. (2 marks)

[4]If there are *no* historical ratings for a user, use the global average.

[5]This should be the *number of true positives*, rather than the true positive *rate*.

Section 3: Recommendation

For these questions you should use the first 75% of ratings as a training set and the last 25% as a test set (though there are no models to fit).

8. Implement a rating prediction model based on the similarity function

$$r(u,i) = \bar{R}_i + \frac{\sum_{j \in I_u \setminus \{i\}} (R_{u,j} - \bar{R}_j) \cdot \mathrm{Sim}(i,j)}{\sum_{j \in I_u \setminus \{i\}} \mathrm{Sim}(i,j)},$$

   (there is already a prediction function similar to this in the starter code; you can either start from scratch or modify the solution in the starter code). Report the MSE of this rating prediction function (on the test set) when $\mathrm{Sim}(i,j) = \mathrm{Jaccard}(i,j)$. Note that predictions should be made on the *test* set while quantities (averages and similarities) should be computed on the training set. (2 marks)

9. On the test set, report the MSE (i.e., average across all pairs (u,i)) for:

   (a) Those instances where i never appeared in the training set;

   (b) Those instances where i appeared at least once but no more than five times in the training set;

   (c) Those instances where i appeared more than five times in the training set.

   (2 marks)

10. The above is referred to as a *cold-start* issue in recommender systems (i.e., the model performs worse on users/items not seen during training). Make a modification to your prediction function such that it will perform better for unseen items; describe (in a couple of sentences) what modification you implemented and report its MSE *for those items that never appeared in the training set* (i.e., what you reported Q9a).[6] (4 marks)

[6] Your solution needn't have better performance than your model from Question 9, but should have better performance for these 'cold' items.