CSE 158/258, Fall 2024: Midterm

Instructions

Midterm is due at 9am, on **Tuesday November 5**. Submissions should be made via gradescope.

The midterm is worth 25 marks.

You can base your solution on the stub code provided here:
`https://cseweb.ucsd.edu/classes/fa24/cse258-b/stubs/Midterm_stub.ipynb`
`https://cseweb.ucsd.edu/classes/fa24/cse258-b/stubs/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 the video game review data from *Steam*: `https://cseweb.ucsd.edu/classes/fa24/cse258-b/files/steam.json.gz`

Each data point consists of a record of a video game review. A few relevant fields include:

`userID`, `gameID` The ID of the user and the game.

`hours` The amount of time the user played the game.

`hours_transformed` $\log_2(\texttt{hours} + 1)$; i.e., the above, log-transformed.

`text` The text of the user's review.

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; we may only give partial credit if we cannot verify the correctness of your solution.

Section 1: Regression

1. Fit a linear regressor of the form

$$\texttt{time\_played} = \theta_0 + \theta_1 \dot{(}\texttt{review\_length})$$

   where the review length is the number of characters in the review (i.e., `len(d['text'])`) and 'hours' is the original (i.e., not transformed) hours variable.

   Report the value of θ_1 and the Mean Squared Error of the prediction (2 marks).[1]

2. Split your data into train and test portions with ratios 80/20, following the code in the stub. Train a model as in Q1 using the training set and report (a) its Mean Squared Error; (b) how often the model underpredicts (i.e., prediction less than label); and (c) how often the model overpredicts (i.e., prediction greater than label) ***(all quantities on test set)*** (3 marks).

3. You (probably) found that the model overpredicts much more often than it underpredicts. Let's try to 'fix' our model to correct this behavior. Attempt the following interventions:

   (a) Delete outliers from the training set. Specifically, keep only those instances with target values among the lowest 90% (among values in the training set).

   (b) Use the `hours_transformed` variable as the target (i.e., $\log_2(\text{hours} + 1)$).

   (c) (Hard) Fit a model with parameters θ_0 and θ_1 (as in Q2) such that:

   - θ_0 is the same value as in your solution to Q2
   - θ_1 is chosen such that the prediction for the *median* review length in the training set is equal to the *median* number of hours.[2]

   (that is, θ_1 is chosen such that your line passes through $(0, \theta_0)$ and $(\text{median\_length}, \text{median\_hours})$)

   For each of the above interventions, report *how many times the model overpredicts and underpredicts* on the corresponding *test set* (6 marks).

Section 2: Classification

4. Following the training/test splits you used in Q2, solve the same problem using a *classifier*, i.e., use review length to predict whether the time played is above the median. Use a `linear_model.LogisticRegression` model with a regularization strength of $C = 1$. Report (a) the number of True Positives; (b) the number of True Negatives; (c) the number of False Positives; (d) the number of False Negatives; and (e) the Balanced Error Rate (BER) of the classifier on the test set (2 marks).

5. Using the same model, report

[1]Be careful to use `fit_intercept='False'` if manually including an offset feature.

[2]Hint: it is probably easiest to just solve this analytically, i.e., don't try to use library function.

- How often the model *overpredicts* (i.e., predicts 'above median' when the target is below the median).[3]
- How often the model *underpredicts* (i.e., predicts 'below median' when the target is above the median).

(2 marks).[4]

6. The dataset spans years from 2010-2018 (the year of each entry can be found by taking `int(d['date'][:4])`). Using the same type of model as in Q4, compute the BER for:

   (a) those reviews written in 2014 or earlier.

   (b) those reviews written in 2015 or later.

   (c) How well does a model trained only on reviews written in 2014 or earlier (i.e., as a training set) perform on reviews written in 2015 or later (i.e., as a test set)?

   (d) How well does a model trained only on reviews written in 2015 or later perform on reviews written in 2014 or earlier?

   (all models should be trained using (an appropriate fraction of) the training set and evaluated using (an appropriate fraction of) the test set) (4 marks).

Section 3: Recommendation

For these questions you should use the first 80% of samples as a training set and the last 20% as a test set, i.e., the same splits as we constructed in previous questions.

7. Using the training set, compute the 10 users most similar to first user in the dataset (i.e., the user from the first entry) in terms of Jaccard similarity. Report the Jaccard similarity of the first and tenth most similar users (2 marks).

8. Implement a function to predict `hours_transformed` from based on a weighted average of Jaccard similarities, i.e.,:

$$\texttt{hours\_transformed}(u,i) = \frac{\sum_{v \in U_i} R_{v,i} \cdot \text{Sim}(u,v)}{\sum_{v \in U_i} \text{Sim}(u,v)}. \quad (1)$$

   Implement two versions, one based on the similarity user-to-user similarity (i.e., as in the above equation) and one based on item-to-item similarity. Handle 'edge cases' (unseen user/item or zero denominator) by returning the global average (on the training set). Report the MSE of each predictor on the test set (2 marks).[5] Note: a correct implementation should run in less than a minute.

9. Adjust your definition above so that the similarity weights recent actions more highly. Use the similarity function

$$\texttt{hours\_transformed}(u,i) = \frac{\sum_{v \in U_i} R_{v,i} \cdot \text{Sim}(u,v) \cdot e^{-|y(u,i)-y(v,i)|}}{\sum_{v \in U_i} \text{Sim}(u,v) \cdot e^{-|y(u,i)-y(v,i)|}}, \quad (2)$$

   where $y(u,i)$ is the (integer) year in which the review occurred (2 marks).

[3] Just report the *number* of overpredctions rather than e.g. the rate; the autograder will accept answers roughly in the correct range.

[4] Hint: rather than computing these values manually, they can be expressed in terms of *rates* (or counts, i.e., true positives, true negatives, etc.).

[5] Be careful to base similarity measurements on the training set only.