CSE 158/258, Fall 2023: Homework 2

Instructions

Please submit your solution **by Monday Oct 30.** Submissions should be made on **gradescope**. Please complete homework **individually**.

You should submit two files:

`answers_hw2.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.

`homework2.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.**

You will need the following files:

Homework 2 stub : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/stubs/`

Chapter 4 workbook : `https://cseweb.ucsd.edu/~jmcauley/pml/code/chap4.html`

Beer Reviews : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/data/beer_50000.json`

Amazon Music Instruments : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/data/amazon_reviews_us_Musical_Instruments_v1_00.tsv.gz`

Please include the code of (the important parts of) your solutions.

Further code examples for regression and classification are available on the class and textbook webpages. Executing the code requires a working install of Python 2.7 or Python 3 with the scipy packages installed.

Each question is worth one mark unless otherwise specified.

Tasks — Diagnostics (week 2):

We'll start by building a classifier that predicts whether a beer is highly alcoholic (ABV greater than 7 percent). First, shuffle the data and split it into 50%/25/%/25% train/validation/test fractions (code to do so is provided in the stub).

1. We'll use the *style* of the beer to predict its ABV. Construct a one-hot encoding of the beer style, for those categories that appear in more than 1,000 reviews. You can build a mapping of categories to feature indices as follows:

```
categoryCounts = defaultdict(int)
for d in data:
    categoryCounts[d['beer/style']] += 1

categories = [c for c in categoryCounts if categoryCounts[c] > 1000]
catID = dict(zip(list(categories),range(len(categories))))
```

Train a logistic regressor using this one-hot encoding to predict whether beers have an ABV greater than 7 percent (i.e., `d['beer/ABV'] > 7`). Train the classifier on the training set and report its performance in terms of the accuracy and Balanced Error Rate (BER) on the validation and test sets, using a regularization constant of $C = 10$. **For all experiments use the `class_weight='balanced'` option** (2 marks).

2. Extend your model to include two additional features: (1) a vector of five ratings (`review/aroma`, `review/overall`, etc.); and (2) the review length (in characters). Scale the 'length' feature to be between 0 and 1 by dividing by the maximum length seen during training. Using the same value of C from the previous question, report the validation and test BER of the new classifier.

3. Implement a complete regularization pipeline with the balanced classifier. Split your test data from above in half so that you have 50%/25%/25% train/validation/test fractions (using code from the stub). Consider values of C in the range $\{0.001, 0.01, 0.1, 1, 10\}$. Report the **validation** BER for each value of C. Report which value of C you would ultimately select for your model, and that model's performance on the validation and test sets.

4. An *ablation study* measures the marginal benefit of various features by re-training the model with one feature 'ablated' (i.e., deleted) at a time. Considering each of the three features in your classifier above (i.e., beer style, ratings, and length), and setting $C = 1$,[1] report the *test* BER with only the *other* two features and the third deleted (2 marks).

Tasks — Rating Prediction:

For these questions we'll use the *Amazon Musical Instruments* data. Code to read the dataset is included in the stub.

5. Which 10 items have the highest Jaccard similarity compared to item 'B00KCHRKD6'? Report both similarities and item IDs for the 10 most similar items.[2]

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

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

(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). Split the data into 90% train and 10% testing portions (data to do so is provided in the stub). When computing similarities return the item's average rating if no similar items exist (i.e., if the denominator is zero), or the global average rating if that item hasn't been seen before. *All averages should be computed on the training set only.* Report the MSE of this rating prediction function *on the test set* when $\text{Sim}(i,j) = \text{Jaccard}(i,j)$.

7. Later in the quarter, we'll explore recommender systems based on temporal dynamics. Here, we'll explore a simple form of temporal recommendation known as *time-weight collaborative filtering.* Here, interactions are weighted in terms of their recency, i.e.,

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

where $t_{u,j}$ is the timestamp associated with the rating $R_{u,j}$. For example a decay function might be based on exponential decay, i.e., $f(t) = e^{-\lambda t}$, where λ is a controllable parameter. Design a decay function that outperforms (in terms of the MSE) the trivial function $f(t_{u,j}) = 1$, documenting any design choices you make. You are welcome to consider the timestamp of the target rating $R_{u,i}$ in your function (e.g. you might choose a function of the form $f(|t_{u,i} - t_{u,j}|)$, or similar).[3]

[1] Ideally we would rerun the full pipeline and select a different value of C for each ablation, but here we'll stick to a single value for simplicity.

[2] Return any item in case of ties, you will only be graded on the similarity scores.

[3] Any minimal improvement in the MSE is acceptable.