General guidelines:

- These homeworks are tentatively due on Monday of week 3, 5, 7, and 9. If the corresponding lecture material (module 1 for homework 1 etc.) has not been completed by the deadline, the deadline will be moved by one week. But the final deadlines are always **as posted on gradescope.**
- Welcome to start early, but homeworks should be considered in "draft form" until the submission page is active on gradescope
- **Each homework is worth 8 marks, and one HW is dropped, so that all HWs are worth 24 marks**
- Homeworks must be completed individually

Homework stubs and files:
https://drive.google.com/drive/folders/1f5xFMJYdjzh0cU-XLK3nL07flhPlANwK?usp=sharing
Solutions will be posted to the same folder.

The "runner" files exhibit similar behavior to what the autograder is doing, and should run without error once you've implemented your solution.

Homework 1

You should submit a single file containing your code ("homework1.py") to the autograder

Regression (week 1):

First, using the book review data (see the "runner" code for the exact dataset names), let's see whether ratings can be predicted as a function of review length, or by using temporal features associated with a review.

1. Train a simple predictor that estimates rating from review length, i.e.,

   $$\text{star rating} \approx \theta_0 + \theta_1 \times [\text{review length in characters}].$$

   Rather than using the review length directly, scale the feature to be between 0 and 1 by dividing by the maximum review length in the dataset. Return the value of θ and the Mean Squared Error of your predictor (on the entire dataset).
2. Extend your model to include (in addition to the scaled length) features based on the time of the review. The runner contains code to compute the weekday. Compute features using a one-hot encoding for the weekday and month. Be careful not to include any redundant dimensions: e.g. your feature vector, including the offset term and the length feature, should contain no more than 19 dimensions.
3. Train models that
   a. use the weekday and month values directly as features, i.e.,

star rating ≃ θ0 + θ1 × [review len in chars] + θ2 × [t.weekday()] + θ3 × [t.month]

b. use the one-hot encoding from Question 2.

Return the MSE of each.

4. Repeat the above question, but this time split the data into a training and test set. You should split the data into 50%/50% train/test fractions (first half / second half) **following the random shuffle of the data used by the code stub (or runner).** After training on the training set, compute the MSE of the two models (the one-hot encoding from Question 2 and the direct encoding from Question 3) on the test set.

Classification (week 2):

Next, using the beer review data, we'll try to predict ratings (positive or negative) based on characteristics of beer reviews. Load the 50,000 beer review dataset (done in the runner), and construct a label vector by considering whether a review score is four or above, i.e.,

y = [d['review/overall'] >= 4 for d in dataset]

5. Fit a logistic regressor that estimates the binarized score from review length, i.e.,

   p(rating is positive) ≃ σ(θ0 + θ1 × [length])

   Using the class weight='balanced' option, compute the number of True Positives, True Negatives, False Positives, False Negatives, and the Balanced Error Rate of the classifier.
6. Compute the precision@K of your classifier for K ∈ {1, 100, 1000, 10000}.
7. Improve your predictor (specifically, reduce the balanced error rate) by incorporating additional features from the data (e.g. beer styles, ratings, features from text, etc.). The BER should be ~3% higher than the solution from Q5.

Homework 2

You should submit a single file containing your code ("homework2.py") to the autograder

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 (this is already done by the runner/autograder).

1. We'll use the style of the beer to predict its ABV. First we construct a one-hot encoding of the beer style, for those categories that appear in more than 1,000 reviews (already done in the runner). 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. 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 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 model's validation and test performance for the value of C that works best on the validation set.
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, report the test BER with only the other two features and the third deleted (2 marks).

Rating Prediction:

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

5. Implement a function to retrieve the most similar items to a given query item. Report both similarities and item ID.
6. Implement a rating prediction model based on the similarity function

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

(there is already a prediction function similar to this in the textbook code, you can either start from scratch or modify the solution there). Split the data into 90% train and 10% testing portions (data to do so is provided in the runner). 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.

7. Implement a predictor that works better than either the predictor from Q6 (or is better than a naive solution in the event that the solution from Q6 works poorly)