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)

Homework 3

You should submit a single file containing your code ("homework3.py") to the autograder

All data files used in this homework are the same as those used in Assignment 1. Start by downloading the Assignment 1 files by following the link in the Assignment 1 spec.

Rating prediction

1. Compute the global average rating in the training set, as well as the validation MSE for a model that always predicts that value.
2. Implement the update equations described on Slide 83 of the recommendation lecture slides. The autograder will run the model for a single iteration.
3. Improve upon the above solution (e.g. by running the code for multiple iterations, regularizing differently, etc.). Improvement will be measured on the validation set.

Read prediction

Since we don't have access to the test labels, we'll need to simulate validation/test sets of our own. So, we'll split the training data ('train_Interactions.csv.gz') as follows: (1) Reviews 1-190,000 for training (2) Reviews 190,001-200,000 for validation (3). For the first assignment, you'll want to upload to Gradescope for testing only when you have a good model on the validation set. This homework will help you to build such a validation set correctly, which will significantly speed up your testing and development time.

4. Although the runner builds a validation set, it only consists of positive samples. We also need examples of user/item pairs that *weren't* read. For each (user,book) entry in the validation set, sample a negative entry by randomly choosing a book that that user hasn't read.
5. The Assignment 1 baseline file provides a simple implementation (which is repeated in the homework stub). Evaluate the performance (accuracy) of the baseline model on the validation set you have built. Modify the strategy (in the function "improvedStrategy") to improve upon its performance, e.g. by setting a different popularity threshold.
6. A stronger baseline than the one provided might make use of the Jaccard similarity (or another similarity metric). Given a pair (u, b) in the validation set, consider all training items b ′ that user u has read. For each, compute the Jaccard similarity between b and b′, i.e., users (in the training set) who have read b and users who have read b ′. Predict as 'read' based on the condition provided (a combination of Jaccard similarity and a popularity threshold). Your solution will be marked correct if it matches the reference 95% of the time, so there is room for some error.

Category prediction

The stub contains code to build training/validation sets consisting of 9,000/1,000 reviews (a small fraction of the complete dataset). We want to build features that represent common words. The stub starts by removing punctuation and capitalization, and finding the 500 most common words across all reviews ('review_text' field) in the training set. See the 'text mining' lectures for code for this process.

7. Build bag-of-words feature vectors by counting the instances of these 500 words in each review. The 501st dimension should be the offset term ("1")
8. Try to improve upon the performance of the above classifier by using different dictionary sizes, or changing the regularization constant C passed to the logistic regression model.

Homework 4

You should submit a single file containing your code ("homework4.py") to the autograder

Each question is worth two marks.

Text mining

We'll use part of the review corpus for training and the rest for testing (code to read the data is provided in the stub). Process reviews **without capitalization or punctuation** (and without using stemming or removing stopwords).

1. Build a sentiment analysis model that estimates star ratings from a 1,000 word bag-of-words model (based on the most popular words). Compare models based on:
    a. the 1,000 most common unigrams;
    b. the 1,000 most common bigrams;
    c. a model which uses a combination of unigrams and bigrams (i.e., some bigrams will be included if they are more popular than some unigrams, but the model dimensionality will still be 1,000).

    You may use a Ridge regression model (sklearn.linear model.Ridge) with a regularization coefficient of λ = 1). Compute the MSE on the test set for each of the three variants.
2. Find the reviews with the highest cosine similarity compared to the first review in the dataset, in terms of their tf-idf representations (using only the training set, and considering unigrams only).

Content, Structure, and Sequences

For these tasks, you may consider the entire dataset (i.e., no train/test splits).

3. Using the word2vec library in gensim, fit an item2vec model, treating each 'sentence' as a temporally-ordered list of items per user. Use parameters `min_count=1,size=10,window=3,sg=1` (already in stub). Find the most similar items to the book from the first review along with their similarity scores (your answer can be the output of the `similar_by_word` function).
4. The above model's item representations can be accessed via `model.wv[itemID]`. Implement a rating prediction function of the form

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

    using the cosine similarity between item representations as your similarity function. Compute the MSE of this model. Next, by making modifications to your item2vec model (e.g. changing the model parameters) or otherwise, improve the model from the previous question in terms of the MSE (as usual, it should be better than the above model, *or*

always predicting the average). The easiest improvement is to adjust the normalization (denominator) in the above expression, which is highly unstable.