## Problems

## Code implementation (5 points)
Pass test cases by implementing the functions in the `code` directory.

Your grade for this section is defined by the autograder. If it says you got an 80/100, you get 4 points here.

## Free response questions (5 points)

1. (0.5 points) Assume you've used a perceptron to learn a good decision boundary model $\mathbf{w^T x}$ for a two-way classification problem. What is the time complexity to select a class label for a new point using this model? Give your answer as a function of the number of points, n, in the data set the decision boundary was learned from. What is the space complexity of the model, in terms of n and the number of dimensions d in the vector representing each data point? Explain your reasoning.

A:Time complexity for selecting a class label for the new point is O(1), because all you do is plugging the feature vector x into $\mathbf{w^T x}$ and choose a label based on the output. Space Complexity is S(d) because the space of w depends solely on d. The decision boundary that one uses for generating a label from the perceptron's numerical output takes S(d) space, too.

2. (0.5 points) Assume you have a K-nearest neighbor (KNN) classifier for doing a 2-way classification. Assume it uses an L^p norm distance metric (e.g. Euclidean, Manhattan, etc.). Assume a naive implementation, like the one taught to you in class. What is the time complexity to select a class label for a new point using this model? Give your answer as a function of the number of points, n, in the data set. What is the space complexity of the model, in terms of n and the number of dimensions d in the vector representing each data point? Explain your reasoning.

A: Because we don't want to underfit our model, we want to choose a k that is k<<n. If k<<n, time complexity is O(n):

1. **Calculating distance from the new sample to all exisiting samples: O(n)**
2. **finding k nearest neighbors: k*O(n) = O(n) (if k<<n)**
3. **Getting a label through finding median/mode/mean of the neighbors' labels:**
    1. **Median: O(k) on average**
    2. **Mode: O(k)**
    3. **Mean: O(k)**

Therefore, adding all time complexities up will give us O(n), if k<<n

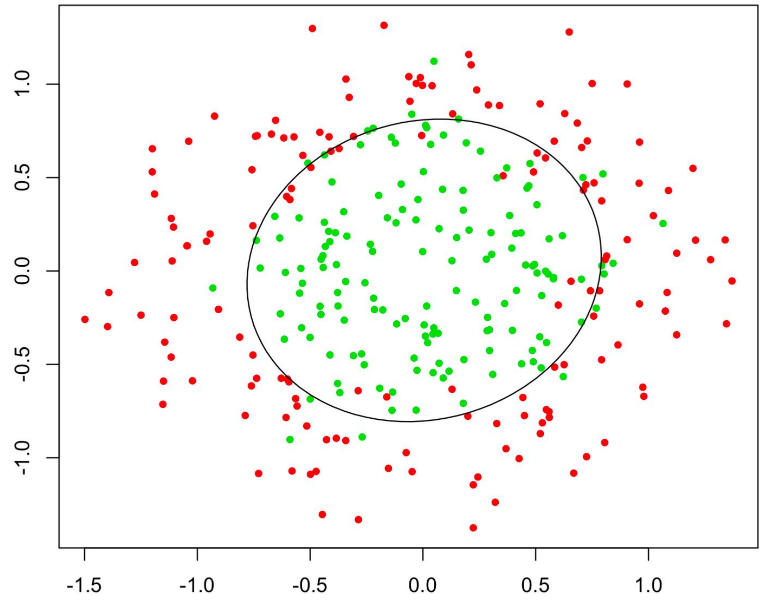
Space complexity: Because we need the entire dataset for labelling the new point, therefore the space complexity is S(nd)

3. (0.5 points) What is the time-complexity of training a KNN classifier, in terms of the number of points in the training data, n, and the number of dimensions d in the vector representing each data point? Is this greater or less than the time-complexity of training a perceptron on the same data? Explain your reasoning.

A: No training is required for KNN, if k is preselected, therefore time complexity is O(1). This is less than training a perceptron on the same data, that is because training a perceptron requires iterating m times through all feature vectors in the dataset and do vector multiplcation on $x^T y$ on each of the d dimensions. If d<<n, then time complexity of training a perceptron is O(nd)

4. (0.5 points) Is a KNN classifier able to learn a non-linear decision surface without using a data transformation prior to inputting the data to the classifier? Why or why not?

A: Yes. The reason is that KNN does not make decisions based on the linear separability of data distribution, hence, instead, it depends on the distribution of classes in its proximity. So unlike regression which requires some prior training, no data transformation is required for KNN. For example, we do not need to make data linearly separable and we can still use KNN to make robust decisions.



(image source: https://i.stack.imgur.com/LZWS8.png)

5. (0.5 points) Assume a Euclidean distance metric. Give an illustration of where the function produced by KNN regression would be very similar to that of a learned linear regressor, but very different when considered on a larger range of input values than exists in the training data. Provide a graph and an explanation.

A: In the illustration below, the linear regressor has degree of 0. Therefore KNN regression will give very similar results as the learned linear regressor. In the bottom chart, the linear regressor has a non-zero slope. Therefore when we have a larger range of input values, KNN will yield a result close to the end points of the input range, which could be very different from the linear regressor's output.

6. (0.5 points) We will build up a *collaborative filter* over the next few questions. The questions presume that you have implemented the `collaborative_filtering` function and passed the corresponding test case `test_collaborative_filtering`. Collaborative filters are essentially how recommendation algorithms work on sites like Amazon ("people who bought blank also bought blank") and Netflix ("you watched blank so you might also like blank"). They work by comparing distances between users. If two users are similar, then items that one user has seen and liked but the other hasn't seen are recommended to the other user. First, frame this problem as a KNN regression problem. How are users compared to each other? Given the K nearest neighbors of a user, how can these K neighbors be used to estimate the rating for a movie that the user has not seen?

A:
The problem can be set up as:

1. **Identify a collection of items as features (e.g, item A, B, C on Amazon)**

2. **Build each user's feature vectors with the number of items they bought, based on their past history (e.g, if user1 bought 2 item A but 0 item B and 0 item C, the feature vector will be [2, 0, 0])**
3. **Now the KNN problem is : for each user's feature vector, what should the zero-value features finally be after applying regression? (e.g, if user1 has bought 0 item B, from user1's K nearest neighbors, how many item B should user 1 buy supposedly?)**

The users similarity can be compared by using a metric to compare their feature vectors.

After setting up the problem, we can estimate the rating of an item that user1 hasn't bought, (or movie user1 hasn't seen) using collaborative filtering.

7. (0.5 points) The MovieLens dataset contains 100k ratings on 1682 movies given by 942 users. First, explore the MovieLens data. For each pair of users, what is the mean number of movies that two users have reviewed in common? What is the median number of movies that two reviewers have reviewed in common? How would the number of movies that two users have both reviewed affect the quality of your collaborative filter? What occurs for those movies that no user has rated?

A: The mean is 18.9, the median is 10.0. The more movies two viewers view together, the better the quality of my collaborative filter, because that will give more information about the average rating of the movie. For the movies that nobody has rated, the imputed values will be zero, which doesn't affect the quality of the collaborative filter.

#### The following questions presume that you have implemented the `collaborative_filtering` function and passed the corresponding test case `test_collaborative_filtering`. You will now need to create additional code to let you explore the effect of different design choices on the performance of a collaborative filter. We will not evaluate this code. We will evaluate the experimental results you produce using this code. The process for this is as follows (note the free parameters `N, K, D, A`):

* Once you load the movielens dataset using your *`load_movielens_data`* function, replace all 0 values for each user with median value of the user's ratings (when computing median, you should only consider non-zero values)

* For each user in MovieLens, take *`N`* *real* ratings at random (the median values you newly filled in the previous step is *NOT real* rating) and replace them with 0. Save this altered data as a new matrix. Be sure to keep the original data around to evaluate your approach.

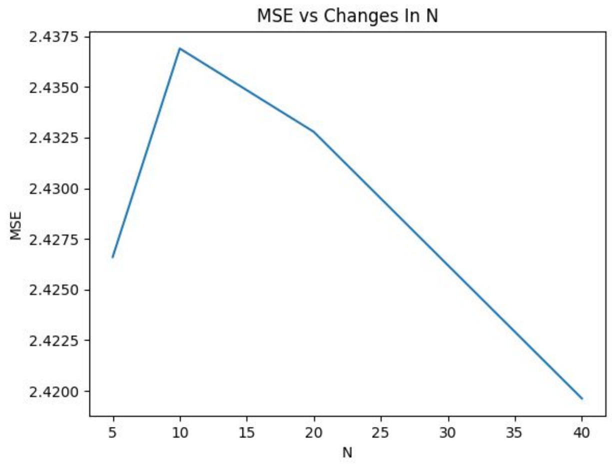
* Use the *`collaborative_filtering`* function to impute values for this new matrix with *`n_neighbors = K`*, *`distance_measure = D`*, and *`aggregator = A`*.

* To evaluate the quality of the collaborative filter, use mean squared error For each *known* rating that you estimated, do: (original_rating - estimated_rating) ** 2. Take the mean of all of these, and then take the square root of that. This is the mean squared error evaluation: error = sqrt(mean((r_i - rhat_i)^2)) where r_i is the original rating (before you substitued a 0) and rhat_i is the rating your system produced, and i iterates over all of the ratings you replaced with 0. Do not estimate the error on ratings that were 0 before you altered them.

HINT: Write ONE function that takes in *`(N, K, D, A)`* as arguments and returns the error of a collaborative filter on the MovieLens dataset. You will be able to use this function for each of the following questions.

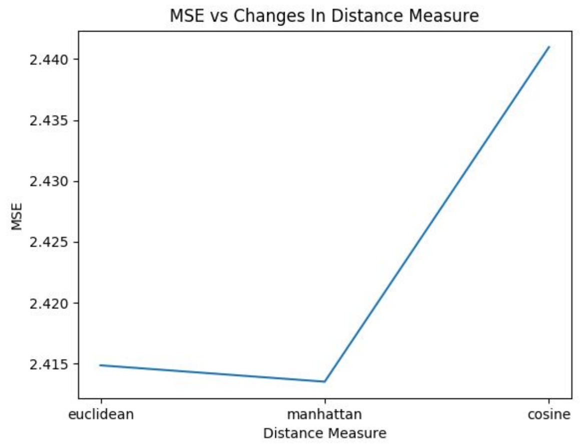
WARNING: Some of these plots can take a very long time to generate as the collobarative filter process is *expensive*. Start early so your computations finish before the deadline! The runtime of the solution (run on a Macbook Pro 2018 laptop) is noted at the end of each question. One tip is to first run the experiments on a small subset of the data to make sure everything works, say just the first 100 users. Once everything works, move up to the entire dataset of 943 users.

8. (0.25 points) For this question, we will measure the effect of changing `N`, the number of ratings we blank out per user. Report the error of your collaborative filter for the values `N` = 5, 10, 20, and 40. Keep `K`, `D`, and `A` fixed at 3, `'euclidean'`, and `'mean'`, respectively. Report the error via a **plot**. On the x-axis should be the value of N and on the y-axis should be the mean squared error of imputating missing values. What happens as N increases? (Estimated runtime: 95 seconds)



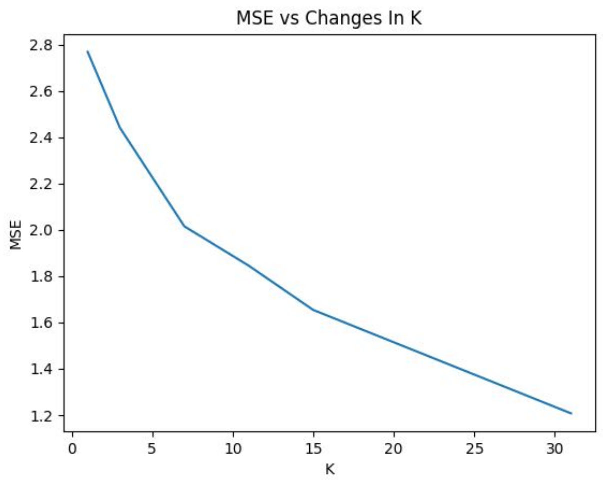
A: As N increases, MSE will first increase, then decrease. I am not entirely sure about this behaviour.

9. (0.25 points) For this question, we will measure the effect of changing `D`, the distance measure we use to compare users. Report the error of your collaborative filter via a **table**. For each possible distance measure, report the error of the filter. Keep `N`, `K`, and `A` fixed at 1, 3, and `'mean'`, respectively. Report the error for each possible distance measure `'euclidean'`, `'cosine'`, and `'manhattan'` distance. (Estimated runtime: 52 seconds)



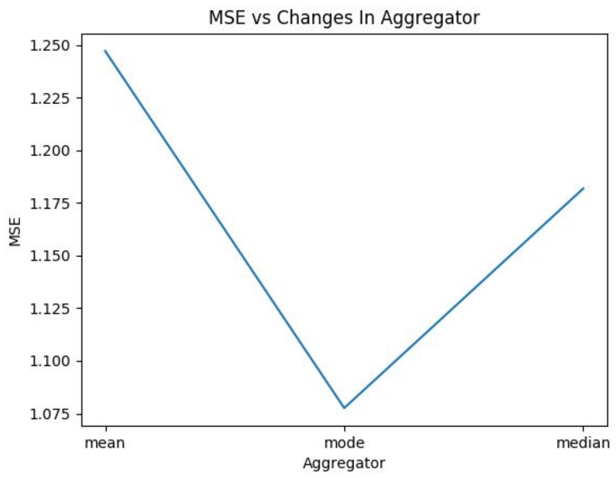
A: Manhattan distance seems to be the best, but euclidean is very close.

10. (0.25 points) Once you find the best performing distance measure, measure the effect of `K`. Report the error of your collaborative filter via a **plot**. On the x-axis should be the value of `K` and on the y-axis should be the mean squared error of the filter. Do this for `K` = 1, 3, 7, 11, 15, 31. Keep `N`, `D`, and `A` fixed at 1, the best performing distance measure found in the previous question, and `'mean'`, respectively. (Estimated runtime: 95 seconds)



A: As anticipated, as K increases, MSE decreases.

11. (0.25 points) Finally, measure the effect of `A`, the aggregator used. Report the error of your collaborative filter via a **table**. For each possible aggregator, report the error of the filter. Keep `N`, `D`, and `K` fixed at 1, the best performing distance measure you found, and the best performing value of `K` that you found, respectively. Set A to be either `'mean'`, `'mode'`, or `'median'`. (Estimated runtime: 110 seconds).



A: mode aggregator is the best aggregator

12. (0.5 points) Now, discuss your results. What were the best settings you found for `D`, `A`, and `K`? For `D`, why do you think that distance measure worked best? For `A`, what about the best setting made it more suitable for your dataset? How did the value of `K` affect your results? Engage critically with the results of your experiments. For each graph you generated, propose an explanation for the behavior of that graph. For each table of values, propose an explanation for the variation in your results.

A:
The best result for D is manhattan, because cosine in high dimensions does not reflect as much “distance” difference, but more “angle difference”. Euclidean provides about the same level of MSE so it is not too different from manhattan.

The best result for A is mode, that is because mode is more robust to outliers than median or mean.

The best result for K is 31. That is because K = 31, compared with the number of users, is not a large enough number to provide an underfitting behaviour. Also, K = 31 is larger than all other K values, therefore the model will consider more neighbours and will provide a less overfitted result. Thus, K=31 is the best value.