Problems

Code implementation (6 points)

Pass test cases by implementing the functions in the `src` directory.

Your grade for this section is defined by the autograder. If it says you got an 80/100, you get 4.8 points here.

Free response questions (4 points)

1. (0.5 total points) Assume you have a k-Nearest Neighbor (kNN) classifier for doing a 2-way classification. Assume it uses an norm distance metric (e.g., Euclidean, Manhattan, etc.). Assume a straightforward implementation, like the one discussed in class. What is the time complexity to select a class label for (i.e., to classify) a new point using this model? (0.25 points) Give your answer in terms of the number of points, , in the data set. You can answer this using big O notation and explain why you think that is the answer, or you can answer this by simply explaining your reasoning in terms of the number of points. What is the space complexity of the model, in terms of and the number of dimensions, , in the vector representing each data point? (0.25 points) Once again, you can use big O notation with an explanation or give an explanation of your reasoning.

   Time complexity:

   - First, the time complexity of finding the distance between two vectors depends on the dimensionality - d - of the vectors i.e. O(d).
   - Since we are directly calculating the distance between each observation in the training set and the new observation, we have O(nd) time complexity.
   - A rather brute force way to find the k smallest distances would be to go through the data k times, each time selecting the smallest element that hasn't been selected. This would add O(kn). (in my implementation I went for sorting the data, which is O(n log(n)))
   - Finally, calculating the appropriate label for the class would take O(k). Since k <= nk, we can ignore this term.

   Therefore, the time complexity is O(nd + nk)

   Space complexity:

   - We need to store all the training data in the algorithm. If each observation is of size d, then our space complexity would be O(nd)

2. (0.25 total points) What is the time complexity of training a kNN classifier in terms of the number of points in the training data, , and the number of dimensions, , in the vector representing each data point? The same note about big O notation applies here as did for Question 1.

   There aren't any operations that need to be done to "train" a KNN classifier. We simply need to memorize a pointer to the training data (assuming it's already in memory). Therefore, the time complexity is O(1).

3. (0.25 total points) Is a kNN classifier able to learn a non-linear decision surface? Why or why not?

   If we use a kernel to weigh our observations, then kNN can definitely learn a non-linear decision boundary (non linear in a low dimension, but linear in a higher dimension).

   Even without the "kernel trick" I think it's reasonable to say that kNN can learn a non-lienar decision surface. The reason is clearest when thinking about the Voronoi tesselation image shown in the lecture slides. The individual segments separating decision boundaries are linear, but the overall surface certainly isn't.

4. (0.5 total points) Imagine we're building a *collaborative filter*, which we'll learn a lot about later in the quarter. 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? (0.25 points) 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? (0.25 points)

   - Users can be compared to each other in a pairwise manner e.g. User A is closer to user B than to user C. In terms of "closer", we can use a distance measure. For example, we could calculate the Euclidian distance between two users, where the features would be the ratings of the users for movies they've both watched.

   Perhaps a better way of measuring distance would be via cosine distance, which is 1 - cosine similarity. This way our measurements would not be influenced by the way users use the scale. Rather, two users would be closer to each other if they both rated movie E higher than movie F.

   - After we have the k nearest users, we can calculate the average rating they would give for movies that our specific user hasn't seen. Then, we can just recommend the few movies with the highest predicted ratings.

5. (0.5 total points) Your boss gives you a very large dataset. What problems do you see coming up with implementing kNN? Name a few of them.

   kNN can be pretty slow at making predictions for new observations if the training data is very large.

   Moreover, if the data is very large it might not fit into memory.

   Finally, kNN doesn't really tell you what features are important for making predictions. Therefore, it isn't particularly interpretable, because there is no model to be interpreted.

6. (0.5 total points) Discuss possible solutions to the problems you have identified in Question 5.

   The time complexity of kNN can be alleviated by implementing KD-trees to make finding the nearest neighbours quicker, but the size of the data might still be prohibitive. This is similar to creating an index in a database.

   If the data doesn't fit into memory, then any sort of kNN calculations would have to involve a distributed system like Spark or Hadoop MapReduce.

   In terms of interpretability, perhaps providing some summary statistics for teh overall data population as well as the k nearest observations might help put the prediction in context. For example, our new observation is closest to these k observations. The average feature values are x, y, and z while the overall population averages are X, Y, and Z.

7. (1.0 total point) You are given these six training examples as ((a_i, b_i), y_i) values, where a_i and b_i are the two feature values (positive integers) and y_i is the class label: {((1,1),-1),((2, 2), −1), ((2, 8), +1), ((4, 4), +1), ((6, 5), −1), ((3, 6), −1)}. Classify the following test example at coordinates (4, 7) using a kNN classifier with k = 3 and Manhattan distance defined by d((u, v), (p, q)) = |u −p| + |v −q|. (0.5 points) Explain shortly how you came up with the answer. (0.5 points)

I drew a little coordinate system and I could visually see which points are closest. Here are the distances between the 6 points and (4,7) explicitly: 9, 7, 3, 3, 4, 2. We can see that the k=3 closest points are at distance 3,3,2. Their corresponding labels are 1, 1, -1. Since this is a classification task, it makes sense to use the mode. Therefore, the new point should be classified as a 1.

8. (0.5 total points) Assume we increase k in a kNN classifier from 1 to n, where n is the number of training examples. Discuss if the classification accuracy on the training set increases. Consider weighted and unweighted KNN classifiers.

The closest neighbour to any point in the training set is itself (I'm assuming we're not ignoring it). Therefore, when k=1, the accuracy on the training set will be 100%. This is in line with our expectation that the lower the k, the more kNN overfits. If we increase k to n, then any point in the training set will get the same prediction, probably the mode of all the labels in the data. Therefore, the accuracy on the training data will certainly decrease as long as we have at least 2 label values. The accuracy we get will simply the proportion of the majority class, which we can use in general as a lower benchmark for any ML model.

All of the above is in the unweighted kNN scenario.

Things change if we use weighted kNN. With k=1, we would still get 100% accuracy. However, when we go to k=n, each training observation would probably get a different prediction. This is because all other points are weighted differently as we move from point to point. The accuracy, however, will again certainly be less than 100%. The details depend on the type of weighting we use i.e. the type of kernel.