Christopher Tsai

CS 349 HW #2 Free Response Questions

1. The time complexity to compute the distance from one testing example to one training example is O(d), where d is the dimension of each data point (aka the number of features). For n training examples, the time complexity becomes O(nd). Next, one needs to choose k nearest distances, by looping through the n distances k times and removing the distances that don't make the cut. Thus total time complexity to classify a single testing example is O(nd + nk). Space complexity is O(nd) as this is the size of the stored training data that is used when testing one data point.
2. Both time and space complexities are O(1) for training a kNN model because this step technically just involves taking in training data and storing it.
3. Yes, a kNN is able to learn a nonlinear decision boundary/surface, especially at low k values. At low k values, the decision surface is more easily pushed and pulled towards different groups of data points, possibly leading to overfitting. At high k values, the decision boundary is more smooth, but can lead to underfitting if k is too high (the decision boundary looks like a straight line in this case).
4. The kNN algorithm can be used in user-based collaborative filtering to predict a user's rating of an item, in turn predicting if the user will like what is suggested to him/her. To compare users, a similarity measure is chosen, which could be the p-norm, the pearson correlation, or the cosine similarity (among others) between users. The kNN algorithm then outputs the k users with most similar ratings/preferences to the test user based on previous ratings. The test user's predicted rating of the item in question can be calculated as the average of the k-nearest-neighbors' ratings.
5. The biggest problem with a very large dataset will be the long testing time, as the distances to all data values will have to be calculated and compared for every testing example. As shown in the testing complexity question, the time and space required to test a dataset increases very fast as the dataset's size increases. Moreover, as dimension increases, the complexity of the methods that we can use to calculate distance between points increases exponentially and accuracy might decrease as well. Thirdly, methods to

find the optimal k value to use for large datasets might be very exhaustive and inaccurate when using large datasets, leading to long waiting times and over/underfitting.

6. To make kNN fast, we can either reduce the dimension d or reduce the number of training examples n. To reduce d, we can get rid of features that we don't need, using our knowledge of the problem and statistical feature selection strategies (such as trees and RFE). To reduce n, we can select a subset of n n′ where k< n′<<n. To find n′, we can use methods such as K-D trees, inverted lists, and locality-sensitive hashing.
7. First the Manhattan distance from the test example to each training example is calculated as: [9, 7, 3, 3, 4, 2]. Next, since k =3, the 3 closest points to the test point are found. Clearly, the test example is closest to the last training example (with dist. of 2) and the two middle examples (with dist. of 3). Out of these three examples, the most common class label is +1. This becomes the predicted class label of the test point.
8. Choosing the value of k is a problem of its own, and different datasets require different k values. As mentioned before, low k values can lead to overfitting and high k values can lead to underfitting. Weighted kNN classifiers have smoother-looking curves so they generally help deal with noise and overfitting, but the same edge cases remain. To find the optimal k, cross-validation is used.