1 Written Exercises

1. The $K$-nearest neighbors method makes the prediction $y$ for a new test point $x$ by either taking a majority vote (for classification), or by doing averaging (for regression) using the labels of its $K$ most similar training examples. Assume $\mathcal{N}$ to be the set of these examples. For the classification setting, finding the majority amounts to finding the most frequent labels in the set $\mathcal{N}$. For binary classification (assuming $y_i \in \{-1, +1\}$) this is simply equivalent to $y = \text{sign}(\sum_{j \in \mathcal{N}} y_j)$. For the regression setting (assuming $y_i \in \mathbb{R}$), the averaging rule is simply $y = \frac{1}{K} \sum_{j \in \mathcal{N}} y_j$. However, both these rules assume that we trust each of the nearest neighbors equally which may not be what we want. For example, assuming $K = 3$, we may have a case that a test example has one very very similar training example with label -1 and two relatively much less similar training examples with label +1. In such a case, although it might make more sense to assign label -1 to this test example but the standard $K$-nearest neighbor algorithm would say +1. A way to address this issue would be to assign a weight $w_i$ to each training example’s label $y_i$. $w_i$ basically is a measure of how similar $x_i$ is to the test example. We can then somehow use these weights in making the prediction for the test point. How would you compute these weights, and how would you use these for $k$NN based binary classification and regression (write the rules in form of equations)?

An alternative to the $K$-nearest neighbors is called the $\epsilon$-ball method. Instead of finding the $K$ nearest neighbors in the training data, you construct a ball with radius $\epsilon$ around the test point and take the majority vote (for classification) or do averaging (for regression) of all training examples lying inside this ball. The parameter $\epsilon$ (just like $K$) is user specified. Do you think this alternative fixes the above-mentioned problem of the $K$-nearest neighbors? If yes, do you think this is better than the weighting scheme of $K$-nearest neighbors we talked about above? If yes, why? If no, why not?

2. For real-valued features, the “standard” notion of distance that we talked about in the class is the Euclidean distance. In particular, we measure the distance between two vectors $x$ and $y$ by the Euclidean norm of the vector $z$ defined by $z = x - y$. The Euclidean norm is, of course, defined by $||z|| = (\sum_d z_d^2)^{1/2}$. There are other norms that one can define. In fact, there’s a whole class of them, called the $\ell_p$ norms. The $\ell_p$ norm, $||\cdot||_p$ is defined below (for $p > 0$):

$$||z||_p = \left(\sum_d |z_d|^p\right)^{1/p}$$

Here, $|a|$ means the absolute value of $a$. It’s easy to see that the Euclidean norm is exactly the $\ell_2$ norm. The $\ell_1$ norm is just $||z||_1 = \sum_d |z_d|$. This is also known as the Manhattan norm because it measures distances by the number of “blocks” that one would have to walk to get between two points, when roads only run along axes (think of the roads in the Salt Lake City :)).

Consider the case of using a $k$NN classifier, but with the $\ell_1$ norm to measure distances rather than the $\ell_2$ (Euclidean) norm. Draw in two dimensions (i.e., each example has only two features) a simple case of a binary classification problem for which the $\ell_1$ classifier would return a different class for a test point than an $\ell_2$ classifier. In particular, draw $\geq 1$ training points (one for each class) and a test point that would be classified differently according to the two distance metrics.

What properties of a data set do you imagine would influence whether the $\ell_1$ distance would work better or worse than the $\ell_2$ distance?

3. (6350 only) An important notion for a classifier is that of consistency. Roughly, a classification algorithm is consistent if, whenever it has access to infinite amounts of training data, its error rate
approaches the optimal error rate (aka, Bayes optimal). Consider the noise-free setting. Here, the Bayes optimal error rate is zero. Is the one-nearest-neighbor algorithm consistent in this setting?

4. Assume that we are learning the ID3 decision tree to do classification, and at one stage, there are nine training examples remaining, five positive and four negative. We have the choice of splitting on two binary features A or B. When A is true, there are two positive and three negative examples; when A is false, there are three positive and one negative examples. On the other hand, when B is true there is three positive and three negative examples; when B is false there are two positive and one negative examples. Which feature will ID3 choose to split on? Show the information gain calculations.

5. Recall that under any subtree of the decision tree, a feature that has already been tested before need not be tested. Assume that there are $F$ binary-valued features in the data. How many information gain calculations would be needed to construct the full decision tree (i.e., assuming no pruning)?

6. (6350 only) Intuitively, the information gain criteria prefers to use features that split a node such that the child nodes are as homogeneous as possible. For the classification setting, it means that the split would ideally like each of the child nodes to have one of the labels dominate the others (i.e., -1 labels dominating +1 labels, or vice-versa). Using the similar intuition, what would be a good criteria (other than information gain) for splitting if we were doing regression instead of classification (so the labels are real-valued numbers instead of discrete variables like +1/-1)?

2 Programming Exercises

1. Your task is to implement the prediction phase of $K$-NN for classification. There is shell code for this in `KNN.m`. Your job is to implement the subroutine called `KNNpredict`. This takes as input the training data points (with labels), a value for $K$ and a single test point. Its output should be the most frequent label among the $K$ closest points in the training data to the test point. Your implementation should be robust to handle the multiclass classification.

   Apply this classifier to the provided dataset (it’s a binary classification dataset). The dataset has three parts: training, development, and test set. In each file, each row represents an example. The first value in each row is the label and the rest of the values are the features. To load this data in MATLAB, simply do `data = load('hw-train');` To get the labels, do `Y = data(:,1);` and to get the features do `X = data(:,2:end);`. Likewise for the development and the test data.

   For your implemented classifier, allow $K$ to range over all odd integers in $[1; 20]$ and plot validation, and test error as a function of $K$. At which value of $K$ is the development error minimized? At which value of $K$ is the test error minimized?

2. Re-implement the `KNNpredict` routine so that it allows the weighting strategy we discussed in the written exercise (1) for the binary classifications setting (where the unweighted majority rule can be expressed as $y = \text{sign}(\sum_{j \in N} y_j)$). Test your implementation on the provided dataset from part-1. Again allow $K$ to range over all odd integers in $[1; 20]$ and plot development, and test error as a function of $K$. At which value of $K$ is the development error minimized? At which value of $K$ is the test error minimized?