**K Nearest Neighbors**

Say we have a set of points classified into groups/clusters, one way or another, designated with some label.

A screenshot of a computer

Description automatically generated with low confidence

Note that we needn’t have necessarily used the Hierarchical Clustering, Kmeans, or DBScan to construct these groups. We just suppose we have them. Given this, we might be interested in classifying a mystery animal via label, given its feature vector. I guess there are lots of different ways to do this.

One way is to interpret the feature vector as a literal vector, i.e., point in some D-dimensional space, and calculate its ‘distance’, i.e. magnitude of its literal displacement vector from all other animals. And then whatever the label of the ‘closest’ animal is, we would choose that as the label of our mystery animal. This is the NN (nearest neighbor) approach, i.e., KNN with K = 1.

Another option finds the labels of the k nearest neighbors, and chooses the label with a plurality of members. Either way, we’ll note that KNN can easily handle multiple classes. You’d just find the K nearest neighbors, and choose the class with the plurality of members. Another way is to classify it accoding to the nearest cluster centroid. In this case, maybe we find the centroids of the classes in the training data, and then we classify test points according to which class has the nearest centroid.

Note it’s probably best to scale the data first, so that no one feature exerts more influence on the ‘distance’ metric than the others.

**Exploring the Model and Hyperparameters**

Here’s a graph I made showing how the smaller K is, the more responsive the model is to the data. But of course this also makes it more likely to overfit the data. In contrast, larger values of K would make it robust for generalization I guess, but also can be overly simplistic. So there’s a tradeoff – we can consider K to be a hyperparameter we can tune. The demarcated line is the decision boundary. There are 500 data points. First we have a linear boundary,

A diagram of a graph

Description automatically generated with medium confidence A diagram of a red and blue graph

Description automatically generated

Performance is arguably the same in each case. But can see that K = 1 has less leakage near the boundaries. And then a quadratic boundary,

A diagram of a graph

Description automatically generatedA diagram of a diagram of a curve

Description automatically generated with medium confidence

Again, can see K = 1 is more responsive to the actual boundary in the quadratic case. Now let’s do a 3-class example. I used 500 points again, and separated into 3 classes: 1 = red, 2 = green, 3 = blue. And we’ll do on a linear-ish boundary,

A diagram of a graph

Description automatically generated with medium confidence A diagram of a number of dots

Description automatically generated with medium confidence

and on a circular boundary,

A diagram of a number of dots

Description automatically generated A diagram of a number of dots

Description automatically generated with medium confidence

But now suppose we have outliers. I made 10% of samples outliers. Then a small K parameter could be bad, because, like we said, it tracks too closely the data. And larger K would be better. Here’s the linear boundary. Can see K = 1 tracks the boundary a little better, but is also way more prone to following outliers.

A diagram of red and blue dots

Description automatically generated A diagram of red and blue dots

Description automatically generated

And here’s the quadratic boundary,

A diagram of red and blue dots

Description automatically generated A diagram of a diagram of a red and blue circle

Description automatically generated with medium confidence

Again, larger K does better for ignoring outliers, but not as responsive. Now let’s do a 3 class guy. I used 500 points again, 15% outliers, and separated into 3 classes: 1 = red, 2 = green, 3 = blue. Here it is with linear-ish boundaries,

A colorful diagram with red blue and green dots

Description automatically generated A diagram of a graph

Description automatically generated with medium confidence

and then with circular boundaries,

A colorful dots on a pink background

Description automatically generated A diagram of a number of dots

Description automatically generated with medium confidence

Can see smaller K, again, overfits, picking up a bunch of outliers. But then larger K, while getting rid of outliers, tends to underfit the surface, although, it might have done a better job than K = 1 actually.

**Comparison to other Models**

One drawback that I would see is that it would be very prone to getting unduly influenced by outliers. Can see that KNN classification with N > 1 tends to redress this issue. Also, I feel like the KNN approach would get really iffy if you were far from *any* data points – notice the edges of of the decision boundary parabola. And related to those two points, I feel like the decision boundary it draws is kind of unnatural. Usually, a boundary (I think) should be expressable mathematically, in terms of the feature values. So it should be a line, or curve, or hypersurface in feature space that would be fit to the feature values. But KNN does not do that. To make a regression analogy, it’s like the difference between plotting a best fit regression line (or curve) through the data vs. just connecting all the points, or even taking the average of all the points within some intervals (xi, xi+1). The connect-the-dots, and average-in-interval methods are too responsive to outliers (though the latter does better than connect-the-dots), and the curves they draw are kind of unnatural. And I feel that’s like what KNN is for classification – too responsive to outliers, and decision boundaries are unnatural. SVC and logistic regression try more for the boundary regression hypercurve approach. On the other hand, the advantage of KNN is that you don’t have to know what the ‘regression curve’ should be.