kNN

k Nearest Neighbours is a classifier that says similar things occur in closer proximity than dissimilar things.

Algorithm

A naive(brute) implementation of kNN is to just load the data (*) and whenever you want to classify a data point do the following. 1. Calculate the distance of this point from all the points in the training set, the distance could be one of but not limited to euclidian, manhattan, or just any minkowaski distance, even cosine similarity works in some cases. The distance should be chosen based on the needs. For example manhattan distance is often preferred when the dimensionality is rather high. Euclidian distance is commonly used for most usecases

2. find the first k data points in the training set
3. Once you have obtained the first k data points, you can again do a bunch of things. A simple way is to just take mode of the k labels. A weighted mode is also often used, where weights are inversely proportional to the distance.

Let's address the asterisk in the room

It's also often good to normalize your data, think of an example where you have data with features like weight and height, the dimensions and scales are completely different. One feature may affect the distance much more than other, unless you know that a feature is more important normalizing is good. In practice, you can get better results if you don't scale too.

How to find k?

Run the classifier for a bunch of different values of k, typically we only iterate over odd values to not encounter same number of votes from different classes, that is if you have a binary label. Whichever value of k gives better results, go ahead with that.

Disadvantages (not really)

Since, we have to find distance with all the data points for each classification, the time complexity is $n \log(k)$ for each data point that we classify, where n is the number of training samples. This makes kNN really slow, or perhaps our implementation really slow. This is different from most other classifiers which take time to train but are fast when at classification.

k-D tree: Bonjour

k-D tree as the name suggests is a k dimensional tree. iterate through the dimensions (features), possibly multiple times, each time take the median value and divide the dataset into two halves, this is very similar to BST except it's multidimensional, and we are chosing a different dimension at each level.

When classifying simply go through the k-D tree, maintain a heap of the points that are encountered, select the first k.

Disadavtanges (Really!)

1. k-D tree does perform good in practice, but it doesn't ensure that the selected neighbours are actually the closest ones!
2. When the dimensionality is rather high, it faces what is popularly known as *curse of dimensionality*
3. Feature Scaling. As we discussed earlier, while it is good to scale features, you can get better results without scaling as it is hard to tell which feature is more important.

Questions

1. Explain Curse of dimensionality.
2. Is there any other implementation apart from k-D tree and brute?
3. should you use cosine distance?
4. is KNN parametric? Explain
5. On a small dataset, which is better brute or k-D tree implementation?

Answers

1. Exponential increase of computational complexity, points tend to lie on a hyper sphere.
2. ball tree
3. Typically no, knn is literally just birds of a feather flock together, cosine similarity doesn't capture that, it may still work
4. Yes and No, depends on implementation
5. brute