COS 424: Interacting with Data

Lecturer: Robert Schapire
Scribe: Anirudh Badam

Lecture # 4
February 15, 2007

The previous lecture defined the Nearest Neighbor Algorithm and discussed how it suffers from the curse of dimensionality. This means that as the number of dimensions increase the Nearest Neighbor algorithm performs poorer and poorer. To better understand the curse of dimensionality with regard to the Nearest Neighbor algorithm one must understand what higher dimensions look like. The following discussion demonstrates how higher dimensions ($n >> 3$) are qualitatively different from lower dimensions (2 *or* 3).

1 Higher Dimensions

1.1 Hypersphere Volume

Imagine a hypersphere in n dimensions of unit radius. Let's call it S_n. Now imagine the smallest hypercube that contains the sphere with in its volume. Call this cube C_n (the side of this cube would be 2 (why?)). Now imagine a smaller cube c_n at the center of S_n whose side length is 1. The smaller cube is placed such that the centers of S_n, C_n and c_n are co-incident and the edges of the cubes are parallel. Figure 1 shows the setting for $n = 2$ and Figure 2 shows it for $n = 3$. Now let us look at the volume of this hyper sphere. For $n = 2, 3$ we see that the smaller cube is contained entirely in the sphere. Intuitively we would expect this to be true with higher dimensions. Table 1 outlines the volumes of S_n, C_n and c_n for some values of n. The results suggest that as n increases the volume of S_n moves closer to zero and in the limit reaches zero. (Observation: distance of a corner point of c_n from the center is $\sqrt{\Sigma_n \frac{1}{2}} = \sqrt{\frac{n}{4}}$, which is greater than one (outside the hyper sphere) for $n > 4$).

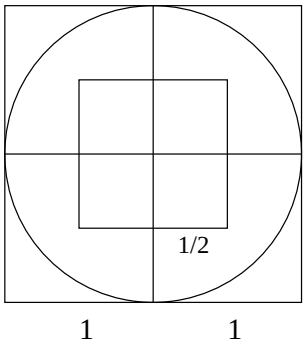

Figure 1: In 2 Dimensions

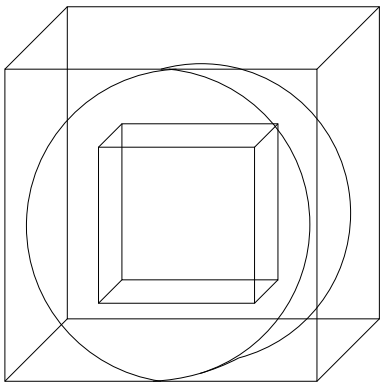

Figure 2: In 3 Dimensions

1.2 Shell Volume

Imagine a hypershell (H_n) of unit outer radius with a shell of thickness ϵ. Figure 3 depicts such a shell in 2 dimensions. Volume of a hypersphere in n dimensions is $k_n \cdot r^n$ (why?), where k_n is some constant for n dimensions and r is the radius of the hypersphere. Hence,

n	$Volume\ of\ c_n$	$Volume\ of\ S_n$	$Volume\ of\ C_n$
2	1	$\pi \approx 3.14$	4
3	1	$4\pi/3 \approx 4.12$	8
4	1	4.93	16
5	1	5.26	32
6	1	5.16	64
7	1	4.72	128
8	1	4.06	256
..	..	..	..
∞	1	0	∞

Table 1: Hypersphere Volume

$$Volume\ of\ H_n = k_n \cdot (1^n - (1 - \epsilon)^n)$$

$$\frac{Volume\ of\ H_n}{Volume\ of\ S_n} = \frac{k_n \cdot (1^n - (1 - \epsilon)^n}{k_n}$$

$$= 1 - (1 - \epsilon)^n$$

$$\rightarrow 1\ as\ n \rightarrow \infty$$

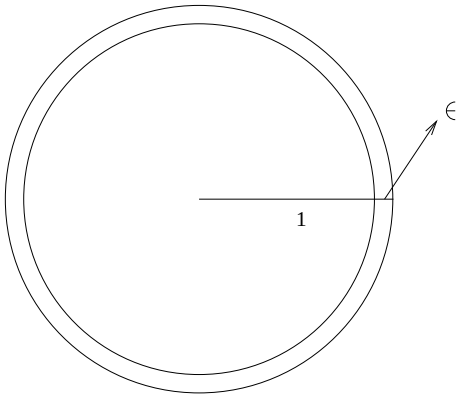

Figure 3: Hypershell in 2 Dimensions

1.3 Conclusion

The results in the previous sections mean that in higher dimensions, the points are "as far as they can be" even in the average case (why?)! This means that the estimate that Nearest Neighbor makes in higher dimensions can be wrong, in the sense that our intuition of a test point being similar to a nearest training point breaks down due to the distances being huge.

As we know the three requirements of a good classifier are Enough Data, Low Training Error and Simplicity, we can say that Nearest Neighbor has no control over the first one. The second one can be true in cases of smaller dimensions (infact training error = 0 for $n = 2, 3$).

Nearest Neighbor can be quite complex! As was demonstrated in the class, it can have

a very complex separation between what it would label as positive and negative (the area distribution of a certain Nearest Neighbor classifier was shown in class to demonstrate this).

2 Decision Tree

2.1 An interesting example

This section introduces a new kind of classifier. A decision tree is a tree where each node is used to make a decision and each edge is used to pursue further path based on the previously made decision. The process stops when a node is reached which is a label. Let us consider the following example. Table 2 shows the training data needed for a classifier and Table 3 shows the test data for the classifier. The apparel of a person such as tie, cape, mask, etc. are called the features/attributes of the data (people in this case). Figure 4 shows a decision tree based on this training data. The node marked tie makes a decision whether the person wears a tie or not. Based on this decision, the person is further sent on to the left or right edge of the tree.

Thus, it is clear to see that Batgirl is labeled as good while Riddler is labeled as bad

	sex	smokes	tie	mask	cape	ears	class
Batman	male	no	no	yes	yes	yes	good
Robin	male	no	no	yes	yes	yes	good
Alfred	male	no	yes	no	no	no	good
Penguin	male	yes	yes	no	no	no	bad
Catwoman	female	no	no	yes	no	yes	bad
Joker	male	no	no	no	no	no	bad

Table 2: Training Data

	sex	smokes	tie	mask	cape	ears	class
Batgirl	female	no	no	yes	yes	yes	???
Riddler	male	no	no	yes	no	no	???

Table 3: Test Data

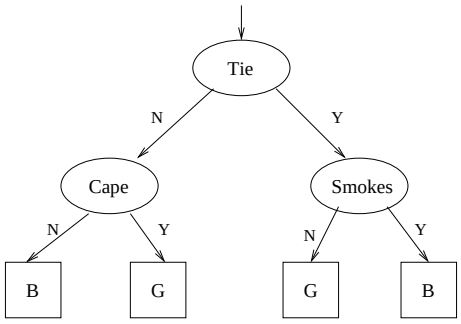

Figure 4: Decision Tree

Figure 5: Choice of Feature for Root

by this decision tree. (Not a bad classifier, but everybody in the class would be labeled bad by this tree (no tie, no cape)). Nevertheless, how did we build this tree?

2.2 Building a Decision Tree

It is clear to see that it is always possible to build a decision tree which will fit the entire training data (why?). Hence, the objective must be to build the smallest possible decision tree which classifies the entire training data correctly. The following discussion describes a recursive procedure to find a decision tree. Although this technique does not explicitly concentrate on finding the smallest decision tree which satisfies training data, it does find a decision tree which would satisfy the entire training data. A very high level definition of the algorithm is that, we need to figure out what to place at a particular node and recursively call the similar procedure to find the children of the node.

Let us first figure out what to put at the root. There are six possibilities in all. Figure 5 shows how choosing tie as the root node would split the data and also shows how choosing cape as the root would split the data. Now we need to decide which is a better choice. By "better" choice we mean the one which splits the data to a "better" extent. It is easy to see that choosing tie would split the data so that the number of bad labels on each side is equal to the number of good labels on that side. On the contrary, choosing cape would split it better in the sense that the right hand side has all good people and the left side has only one good person. This intuition can be formalized in the following manner.

Define an impurity function to gauge a split of the data. The best feature to choose is the one that leads to lesser impurity. In this case we can define the impurity function I as a function of the fraction of good people in a group or block of examples. Hence, we need the impurity to be low when the proportion of good people in a split is low or high and we need it to be the highest when the proportions of good and bad people are comparable. Figure 5 shows the impurities that have to be calculated for each of the choice. Figure 6 shows one such function, where r is the fraction of good people. The following functions are widely used ones to get such impurity functions.

$$\text{Entropy: } I(r) = -r \cdot \log(r) - (1-r) \cdot \log(1 - r)$$
$$\text{Gini Index: } I(r) = r \cdot (1 - r)$$

Now we need to estimate the total impurity of a choice based on the impurities of the left side and the right side. One simple way is to take their sum, but this would bias the

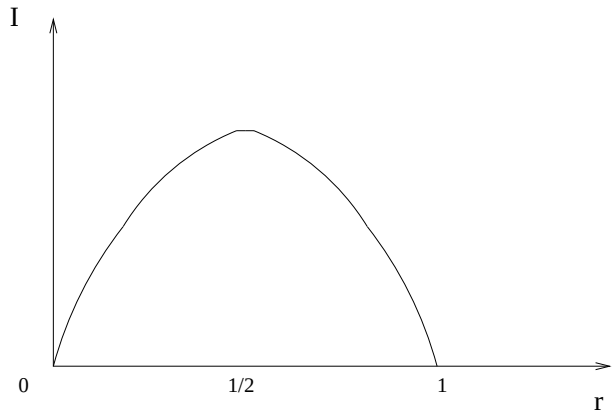

Figure 6: Typical Impurity Function

side with the lower number of entries. Hence, a weighted average is the sensible choice. Algorithm 1 defines a very high level pseudo code for the algorithm.

Algorithm 1 Recursive Decision Tree

for every attribute **do**
consider split with respect to this attribute
compute impurity

$$\mathrm{I} = \frac{p_1 \ + \ n_1}{p_1 \ + \ n_1 \ + \ p_2 \ + \ n_2} \cdot I\left(\frac{p_1}{p_1 \ + \ n_1}\right) + \frac{p_2 \ + \ n_2}{p_1 \ + \ n_1 \ + \ p_2 \ + \ n_2} \cdot I\left(\frac{p_2}{p_2 \ + \ n_2}\right)$$

/*where p_1, n_1, p_2, n_2 are the splits of positives and negatives on the left and the right sides*/
end for
choose state with minimum impurity
recursively build the subtrees