KNN, ID Trees, and Neural Nets

Intro to Learning Algorithms

KNN, Decision trees, Neural Nets are all ***supervised learning*** *algorithms*
Their general goal = make accurate predictions about unknown data after being trained on known data.

Data comes in form of examples with the general form: $(x_1, \ldots, x_n, y)$
$\mathbf{x_1}$, .. $\mathbf{x_n}$ are also known as features, inputs or dimensions **y** is the output or class label.
Both x_i and ys can be **discrete** (taking on specific values) {0, 1}
or **continuous** (taking on a range of values) [0, 1]

In training we are given (x_1, ... x_n, y) tuples. In testing (classification), we are given only (x_1,... x_n) and the goal is to predict y with high accuracy.

Training error is the classification error measured using training data to test.
Testing error is classification error on data not seen in the training phase.

K Nearest Neighbors

1-NN
- Given an unknown point, pick the closest 1 neighbor by some distance measure.
- Class of unknown is the 1-nearest neighbor's label.

k-NN
- Given an unknown, pick the k closest neighbors by some distance function.
- Class of unknown is the **mode** of the k-nearest neighbor's labels.
- k is usually an odd number to facilitate tie breaking.

How to draw 1-NN decision boundaries

Decision boundaries, lines on which it is **equally likely** to be in any of the classes.

1. Examine the region where you think decision boundaries should occur.
2. Find oppositely labeled points (+/-)
3. Draw bisectors. (use pencil)
4. Extend and join all bisectors. Erase extraneously extended lines.
5. Remember to **draw boundaries to the edge of the graph** and indicate it with arrows! (a very common mistake).
6. Your 1-NN boundaries generally should have sharp edges and corners (otherwise, you are doing something wrong or drawing boundaries for a higher k-nn.)

Distance Functions

How to determine what points are "nearest". Here are some standard Distance functions:

Euclidean Distance	$D(\vec{w},\vec{v}) = \sqrt{\sum_i^n (w_i - v_i)^2}$
Manhattan Distance (Block distance) - Sum of distances in each dimension	$D(\vec{w},\vec{v}) = \sum_i^n \lvert w_i - v_i \rvert$
Hamming Distance - Sum of differences in each dimension	$D(\vec{w},\vec{v}) = \sum_i^n I(w_i, v_i)$ I(x,y) = 0 if identical, 1 if different.

Cosine Similarity - Used in Text classification; words are dimensions; documents are vectors of words; vector component is 1 if word i exist.	$D(\vec{w},\vec{v}) = \frac{\vec{w}\cdot\vec{v}}{\|\vec{w}\|\|\vec{v}\|} = \cos\theta$

(Optional) How to Weigh Dimensions Differently

In Euclidean distance all dimensions are treated the same. But in practice not dimensions are equally important or useful!

For example. Suppose we represent documents as vectors of words.
Consider the task of classifying documents related to "Red Sox". If all words are equal, then the word "the" weighs the same as the word "Sox". But almost every english document has the word "the". But only sports related documents have the word "Sox". So we want k-nn distance metrics to weight *meaningful* words like sox more than functional words like "the".

For text classification, a weight scheme used to make some dimensions (words) more important than others is known as: TF-IDF

$$tf \cdot idf(w_i,d) = tf(w_i,d)\cdot idf(w_i)$$

$$tf(w_i,d) = \frac{\#(w_i)\in d}{|d|}$$

$$idf(w_i) = \log\frac{|D|}{\#d\in D\, with\, w_i}$$

Here:
tf: Words that occur frequently should be weighed more.
idf: Words that occur in all the documents (functional-words like the, of etc) should be weighed less.
Using this weighing scheme with a distance metric, knn would produce better (more relevant) classifications.

Another way to vary the importance of different dimensions is to use: Mahalanobis Distance

$$D(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})S^{-1}(\vec{x}-\vec{y})}$$

Here S is a covariance matrix. Dimensions that show more variance are weighted more.

Decision Trees:

Algorithm: Build a decision tree by **greedily** picking the lowest disorder tests.
NOTE: This algorithm is greedy so it does not guarantee that the tree will have the minimum total disorder!

Disorder = Average of Entropy(Split)

Examples:
Disorder equation for a test with two branches (l, r) and each branch having 2 (binary) class.

$$\text{Disorder} = \frac{l}{T}\left(\left[-\frac{a}{l}\lg\frac{a}{l}\right]+\left[-\frac{b}{l}\lg\frac{b}{l}\right]\right)+\frac{r}{T}\left(\left[-\frac{c}{r}\lg\frac{c}{r}\right]+\left[-\frac{d}{r}\lg\frac{d}{r}\right]\right)$$

a = count of class 1 on left side b = count of class 2 on left side
c = count of class 1 on right side d = count of class 2 on right side
a + b = l c + d = r

Note that $H\left(\frac{a}{l}\right)=\left[-\frac{a}{l}\lg\frac{a}{l}-\frac{b}{l}\lg\frac{b}{l}\right]=H\left(\frac{b}{l}\right)$ Note that Binary Entropy is symmetric!

For a test with 3 branches, and 2 binary class outputs:

$$\text{Disorder} = \frac{b_1}{T}H\left(\frac{a}{b_1}\right)+\frac{b_2}{T}H\left(\frac{c}{b_2}\right)+\frac{b_3}{T}H\left(\frac{e}{b_3}\right)$$

a = count of class 1 on branch 1 b = count of class 2 on branch 1
c = count of class 1 on branch 2 d = count of class 2 on branch 2
e = count of class 1 on branch 3 f = count of class 2 in branch 3
$a+b = b_1$ $c + d = b_2$ $e + f = b_3$

More Generally:

$$\sum_b^B \frac{b}{T}H(P_b)$$

Where B is the set of branches, P_b is the "distribution" of classes under branch b. In most case P_b is binary, i.e. have two values; But in the general case it can have multiple values.
For example, we had 3 class values.

$$H(\cdot)=-\frac{a}{T}\lg\frac{a}{T}-\frac{b}{T}\lg\frac{b}{T}-\frac{c}{T}\lg\frac{c}{T}$$

a = count of class 1 ; b = count of class 2 ; c = count of class 2 ; a + b + c = T

Homogeneous Partitioning Trick:

A time-saving heuristic shortcut to picking the lowest disorder test.
1. Pick tests that breaks the space into a *homogeneous portion* and a *non-homogeneous* portion
2. Pick the test that partitions out the **largest** homogeneous portion; that test will likely have the lowest disorder.
Caution! when the homogeneous portions are about the same; you should compute the full disorder. This is where this shortcut might break down!

Table of common Binary Entropy values:

Note because H(x) is symmetric, i.e. H(1/3) = H(2/3). Fractions > 1/2 are omitted.