

1 Administrivia

- Data files for HW3 are now available from the class webpage.
- Last time, I made a mistake on the board. Mea culpa! I said that the Euclidean distance function can be written $(X^T Y)^{1/2}$, but of course, it should be $((X - Y)^T (X - Y))^{1/2}$. Thanks to everybody who caught that! That's what I get for trying to extemporize math off the top of my head. . .

2 Decision Trees, Cont'd

- Our goal in making a decision tree learning system is to build a tree from data recursively. I.e., at any point in time, pick the attribute that splits the data into the most nearly pure subsets. (See overhead w/ DT algorithm)
- Last time, we finished with a description of the **entropy function** H , which allows us to determine how nearly pure a data set S is. (I.e., how close it is to having only one class in it.):

$$H(S) = - \sum_{c \in \text{classes}(S)} p_c \log_2 p_c$$

- Now we'll use that to construct a criterion for deciding which attribute to split on at any given point in building a tree.
- What we want is to evaluate which attribute drives the data toward the *most pure* state. I.e., the one that improves purity most w.r.t. the original set.
- Problem: Entropy measures the purity of a *single* set, S . But after we choose an attribute A to split on, we have divided S into a number of different subsets $S_1, S_2, \dots, S_{A.v}$. How do we evaluate the total entropy of all of the subsets produced by A ?
- Could just sum individual entropies for each subset, but that leads to distortions – consider when an attribute pulls out only a *single* datum from the set.
- Better: entropy of a group of sets is the *average* of entropies of individual sets. Let S_c be the subset of S defined by attribute $A = c$ ($c \in A.v$). Let $H(S|A)$ be the entropy of the split defined by A , then,

$$H(S|A) = - \sum_{c \in A.v} \frac{|S_c|}{|S|} H(S_c)$$

- Now what we want to ask is, *which attribute improves this the most over the raw entropy of S ?*
- Well, if raw entropy is $H(S)$, then we define the **entropy gain**, $G(S|A) = H(S) - H(S|A)$. Now we just pick the attribute that maximizes gain. I.e., for our split, we select A such that:

$$A = \arg \max_{p \in \text{Attr}} G(S|p)$$

- We've treated only categorical splits here, but you can also do splits on continuous values by making tests of the form $A > 3$ vs. $A \leq 3$. Do this by sorting the attribute and picking the best split point within that attribute.
- Q: if you think of the data as lying in \mathbb{R}^n (as in k -NN), then what kind of geometrical structures does a DT produce? I.e., what sorts of divisions does it use?

3 Perceptrons and Neural Networks

- Perceptrons are described in section 10.2 of your text.
- Ok, we started by talking about k -NN learners, which used a distance function directly to compare points. It led to a piecewise linear division between classes. Then we looked at DTs, which make a succession of axis-orthogonal splits.
- Possibly the simplest form of learner for binary-class data is the **linear decision boundary** classifier.
- This just supposes two data classes, $+$ and $-$, in \mathbb{R}^n and tries to fit a single linear decision surface (i.e., hyperplane) between them.
- If you can fit a hyperplane between classes such that all $+$ fall on one side and all $-$ fall on the other, then the data is called **linearly separable**, otherwise it's called *linearly inseparable*.
- The question now is: if the data is separable, how do you pick a plane that separates the data? Note that, in general, there may be an *infinite* number of planes that will do.
- One way to pick a separating hyperplane is the **perceptron algorithm**, developed in the 1950's. It's a precursor to what we now think of as "neural networks" (properly speaking *artificial* neural networks).
- The general idea of the perceptron alg is that you start with an arbitrarily chosen hyperplane, w , and iterate across the data. If, for each datum, it falls onto the "correct" side of w , then you leave w alone. Otherwise, you nudge w toward the point it got wrong.
- By successively iterating through the data, eventually the plane will settle down somewhere separating the data (assuming that the data are linearly separable, of course)
- Let's look at how you might do that mechanically.
- Start by defining data point $X \in \mathbb{R}^n$ and label $Y \in \{-1, +1\}$. Now note that a hyperplane w is defined by a set of $n + 1$ parameters – $w(0)$ is the offset and $w(1) \dots w(n)$ are the coefficients. You can think of this plane as being orthogonal to the vector w (usually called the *weight vector*).

- Now we'll use a trick. Note that X is n elements long, but w is $n + 1$ elements long. Let's augment X with $X(0) = 1$. Thus, you can now write

$$w^T X = \sum_{i=0}^n w(i)X(i)$$

- Now you have that the quantity

$$w^T X \begin{cases} \geq 0 & \text{if } X \text{ is "above" the hyperplane (i.e., further from origin)} \\ < 0 & \text{if } X \text{ is "below" the hyperplane} \end{cases}$$

- So now you want to move the plane iff $\text{sign}(w^T X) \neq Y$ or, equivalently, if $Y - \text{sign}(w^T X) \neq 0$.