CS 189: Introduction to Machine Learning - Discussion 10

1. Decision Trees

Recall that training a decision tree requires looking at every feature to find the best split, where the best split greedily maximizes the information gain. The information gain is defined as

$$H - \left[\frac{n_1 H_1 + n_2 H_2}{n_1 + n_2}\right]$$

where H is the entropy at the current node, H_1 is the entropy at the "left" split, and H_2 is the entropy at the "right" split. n_1 and n_2 are the number of data points at the "left" and "right" splits.

(a) What are good values to choose to test the splits?

Solution: Imagine we are deciding to split on a specific real-valued feature. Obviously, we should choose every value of the data to be the threshold for the split. Choosing a split value between two values in our data will give the same splitting of the data. For instance, imagine that we have data:

$$\begin{bmatrix} 5 & 0 \\ 6 & 0 \\ 7 & 1 \end{bmatrix}$$

where the first column is the features, and the second column is the labels (we have three data points). Note that splitting on 6.5 and 6.3 will result in the same splits. Thus, we want to choose each feature value for a split.

(b) What is the running time for the naive approach to finding the best split (just finding the split, not training the entire tree)?

Solution: We must search through every possible split to find the best one, and there are dn of them, where d is the dimensionality of our data and n is the number of data points. For every split value, we must walk through all n data points and classify them accordingly. We can calculate the information gain from these two new subsets of our data in linear time. Thus, we have $\mathcal{O}(dn^2)$.

(c) What is a smarter way to search for the best split, and what is the running time of this?

Solution: Before we start scanning through a feature, we can sort the data with respect to that feature value. Then, every time we choose a new split value, only one data point will be classified differently. Calculating the new entropies can be done in constant time. Assuming we use a comparison based sorting algorithm, our new running time is $\mathcal{O}(dn \log n)$.

Now consider decision trees for regression. We can no longer use our notion of entropy as a measure of how well our data is split, since our values for our labels are continuous. We want the data at the leaves to be spread as little as possible.

(a) What is a good measure to use to determine how well our data is spread? (Hint: think back to all of our real valued problems. What error measure did we use?)

Solution: We can use sample variance to determine how well our data is split. This is the sum of the squared error from the average.

(b) Write down the equation we want to maximize when searching over splits for regression trees.

Solution: This equation looks very similar to our information gain equation, except the entropies are replaced with variances. We have

$$\sigma^2 - \left[\frac{n_1\sigma_1^2 + n_2\sigma_2^2}{n_1 + n_2}\right]$$

where σ^2 is the variance at the current node, σ_1^2 is the variance at the "left" split, and σ_2^2 is the variance at the "right" split.

2. Maximum Entropy Distribution

   Suppose we have a discrete random variable that has a Categorical distribution described by the parameters $p_1, p_2, \ldots, p_d$. Recall that the definition of entropy of a discrete random variable is

$$H(X) = E[-\log p(X)] = -\sum_{i=1}^{d} p_i \log p_i$$

   Find the distribution (values of the p_i) that maximizes entropy. (Hint: remember that $\sum_{i=1}^{d} p_i = 1$. Don't forget to include that in the optimization as a constraint!)

Solution:

For simplicity, assume that log has base e, i.e. $\log = \ln$ (the solution is the same no matter what base we assume). The optimization problem we are trying to solve is:

$$\begin{aligned} \underset{\vec{p}}{\operatorname{argmin}} \quad & \sum_{i=1}^{d} p_i \log p_i \\ \text{s.t.} & \sum_{i=1}^{d} p_i = 1 \end{aligned}$$

Formulating the Lagrangian, we get

$$\mathcal{L}(\vec{p}, \lambda) = \sum_{i=1}^{d} p_i \log p_i + \lambda \left(1 - \sum_{i=1}^{d} p_i\right)$$

Taking the derivative w.r.t. p_i and λ:

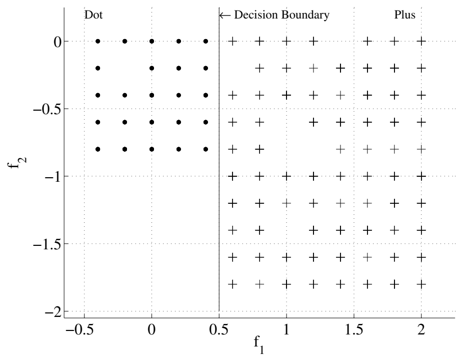
$$\begin{aligned} \frac{\partial}{\partial p_i}\mathcal{L}(\vec{p}, \lambda) &= \log p_i + \frac{p_i}{p_i} - \lambda \implies \lambda - 1 = \log p_i \\ \frac{\partial}{\partial \lambda}\mathcal{L}(\vec{p}, \lambda) &= \sum_{i=1}^{d} p_i = 1 \end{aligned}$$

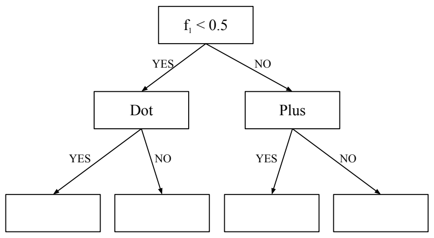
This says that $\log p_i = \log p_j, \forall i, j$, which implies that $p_i = p_j, \forall i, j$. Combining this with the constraint, we get that $p_i = \frac{1}{d}$, which is the uniform distribution.

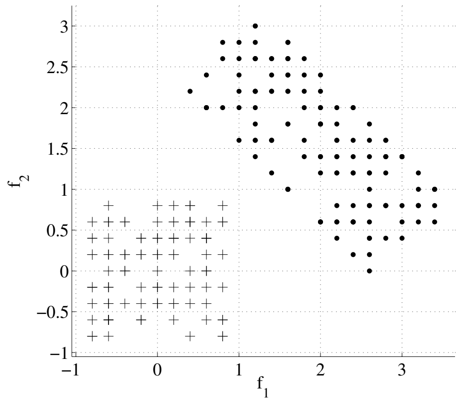
3. You are given points from 2 classes, shown as $+$'s and $\cdot$'s. For each of the following sets of points,

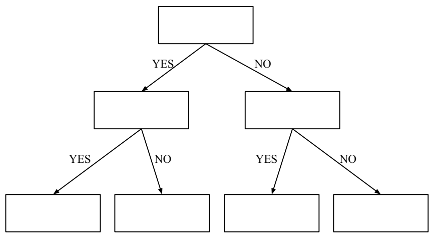
   1. Draw the decision tree of depth at most 2 that can separate the given data completely, by filling in binary predicates (which only involve thresholding of a *single* variable) in the boxes for the decision trees below. If the data is already separated when you hit a box, simply write the class, and leave the sub-tree hanging from that box empty.
   2. Draw the corresponding decision boundaries on the scatter plot, and write the class labels for each of the resulting bins somewhere inside the resulting bins.

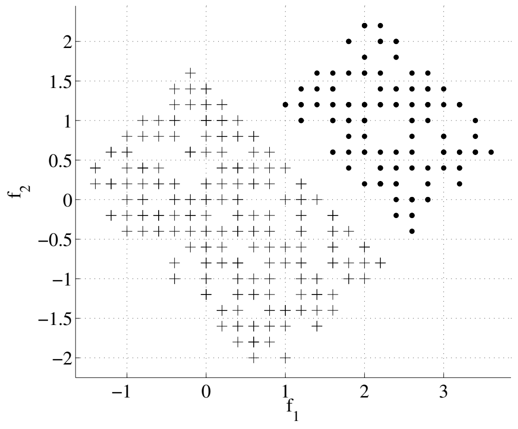
   If the data can not be separated completely by a depth 2 decision tree, simply cross out the tree template. We solve the first part as an example.

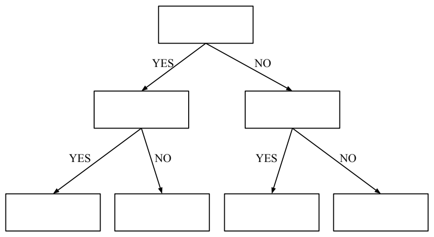

Dot
← Decision Boundary
Plus
f1
f2

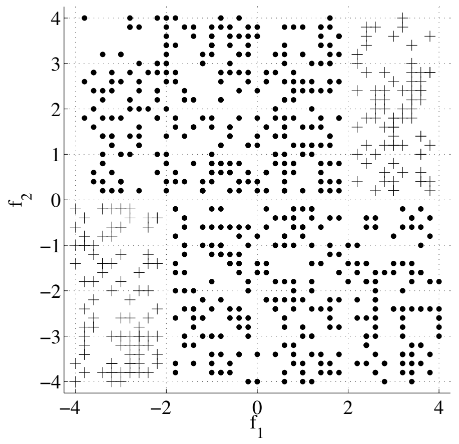

f1 < 0.5
YES
NO
Dot
Plus
YES
NO
YES
NO

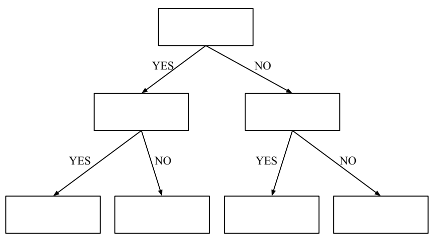

f1
f2


YES
NO
YES
NO
YES
NO


f1
f2


YES
NO
YES
NO
YES
NO


f1
f2


YES
NO
YES
NO
YES
NO