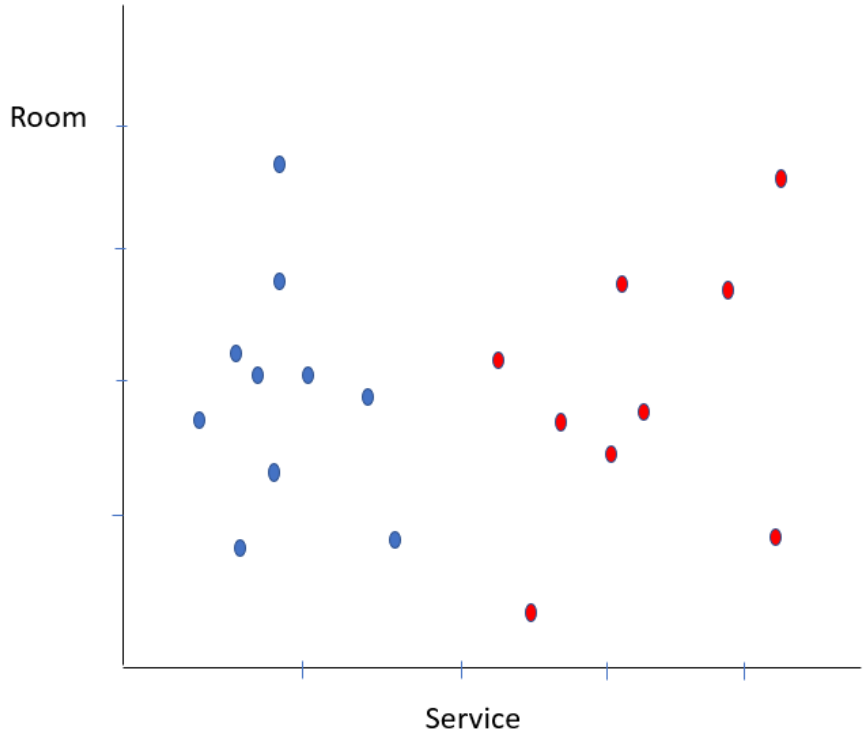
Text Classification- Decision Trees

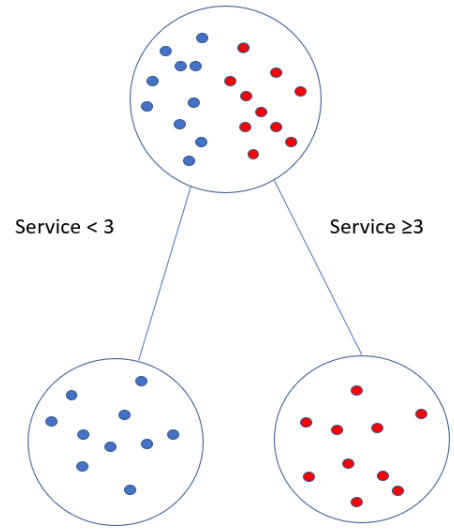
Promothesh Chatterjee*

Decision Trees

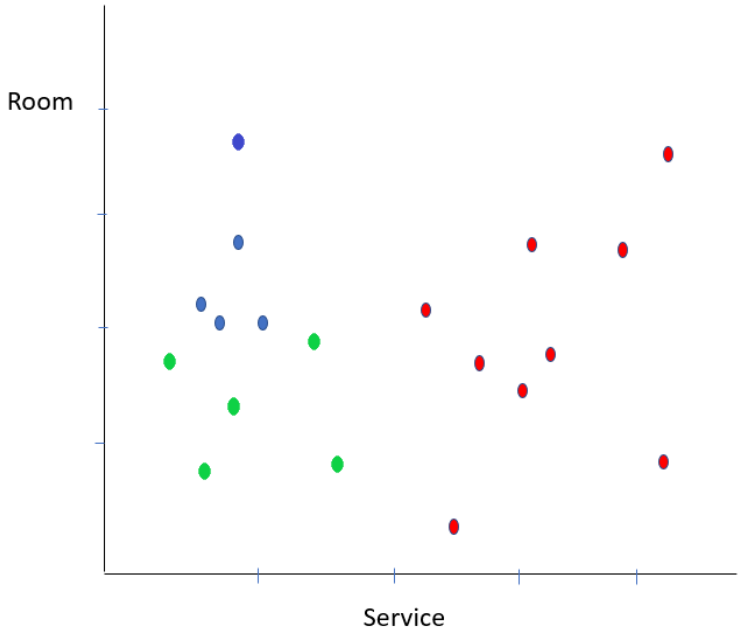
Let's take a previous example from the Vector Space model notes. Let's consider two words-'room' and 'service' from our reviews. Assume, we have reviews consisting only of these two words. We have divided the reviews using some criteria into two colors- blue and red.



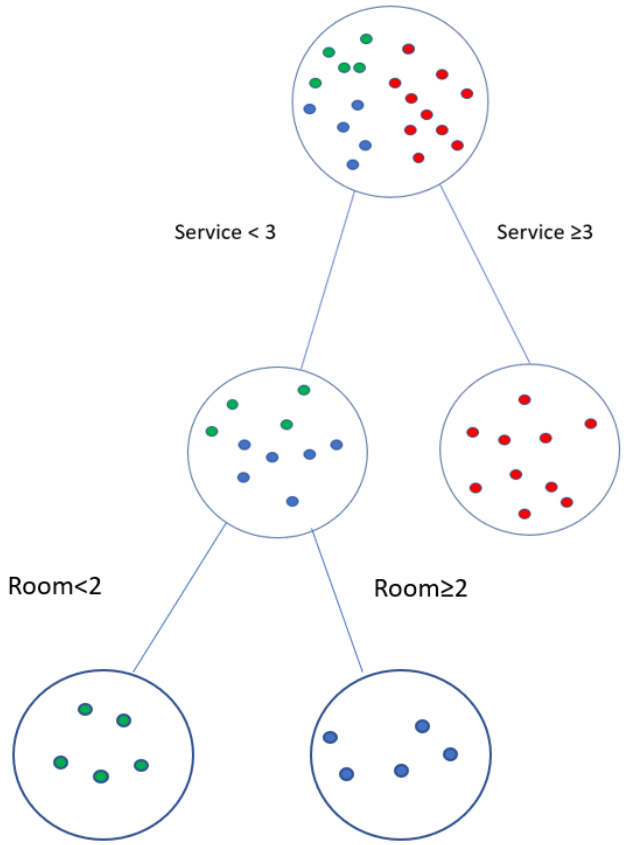
If another review comes in with coordinate value of 3 for service, what color would you predict for the review? If you said, red, you have performed the task of a simple decision tree.



This is an example of a simple decision tree with one decision node that tests the criteria service< 2. If the answer to the decision rule is yes, we take the left branch and pick Blue. If the answer is no, we take the right branch and pick Red. In our case, we picked Red. Essentially, what we have just done is a kind of supervised learning, given a labeled dataset, how should we classify new samples? Because our existing dataset had labels, Blue and Red, we could classify the incoming sample to a particular category. If there were three categories, say Blue, Red and Green, we could easily extend the concept.



We can easily see, that the second node needs to be split further. This is the basic idea underlying a decision tree.



How can we train a Decision Tree?

We first want to find out what should be the root node in our tree? Should the feature 'service' or 'room' be tested first, and what should be the threshold? In the previously discussed example, the root node in our tree was 'service' with a threshold of 3.

Since, the objective of classification is to split observations into different classes, we want the decision node to make best split possible. What decision criteria should we use for the best split? One criterion is Gini impurity.

Gini Impurity The basic idea underlying Gini impurity is simple. Assume you randomly pick a point from the dataset, and randomly assign it to a class based on the existing distribution in the dataset. The probability that we have classified the datapoint incorrectly is Gini impurity.

$G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$

$C = number\ of\ classes\ p(i) = the\ probability\ of\ picking\ a\ datapoint\ with\ class\ i$

For the example above, assuming we have 5 blue, 5 green and 10 red balls, we have C = 3, p(blue)=1/5, p(green)=1/5, p(red)= 1/10.

$G = p(blue) \times (1 - p(blue)) + p(green) \times (1 - p(green)) + p(red) \times (1 - p(red))$

$G = 5/20 \times 15/20 + 5/20 \times 15/20 + 10/20 \times 10/20 = 250/400$

Go back to the first instance of our example with only two classes, red and blue. What is the Gini impurity after the first split here (that is when we have correctly identified all the datapoints to respective classes)? Since we have correctly identified the classes of each datapoint, the Gini impurity of both branches will be 0.

$G_{left} = 1 \times (1-1) + 0 \times (1-0) = 0$

$G_{right} = 0 \times (1-0) + 1 \times (1-1) = 0$

A Gini Impurity of 0 is the lowest and best possible impurity.

What if we had incorrectly assigned a blue datapoint to red class? Since, the left Branch has only blues, so we know that $G_{left} = 0$

But, the right branch will now have 1 blue point and 10 red points. Let's calculate the Gini impurity for the right branch now.

$G_{right} = 1/11 \times 10/11 + 10/11 \times 1/11 = 20/121$

Splitting decision trees can be based on Gini impurity. For a split to take place, the Gini index for a child node should be less than that for the parent node. Among other measures of node impurity is entropy (entropy is minimum when the probability is 0 or 1).

Use of Entropy in Decision-Trees

Let's summarize the basic idea of decision tree again. A decision tree is like a flowchart that helps us make decisions. Imagine you're trying to decide whether to play tennis based on the weather. A decision tree helps us figure this out by asking a series of questions, one at a time, and making decisions based on the answers.

How It Works:

1. **Nodes and Branches**:
    - **Nodes**: These are the questions we ask (e.g., "Is it sunny?").
    - **Branches**: These are the possible answers (e.g., "Yes" or "No").
    - **Leaves**: These are the final decisions (e.g., "Play Tennis" or "Don't Play Tennis").
2. **Splitting**:
    - At each node, we split the data based on a question.
    - We want to ask questions that best separate the data into pure groups (where each group mostly contains the same kind of outcome).
3. **Purity**:
    - Purity means that a group mostly contains either all "yes" or all "no".
    - For example, imagine you have a mixed bag of blue and red balls. Splitting the bag so that each new bag contains only one color is making it pure.

Entropy

Entropy is a way to measure how mixed up or impure the data is. In decision trees, we use entropy to decide the best way to split the data.

Simple Explanation of Entropy:

1. **Concept**:
   - Entropy measures disorder or uncertainty.
   - If you have a box of all blue balls (pure), entropy is low because there's no disorder.
   - If the box has a mix of blue and red balls (impure), entropy is high because there's more disorder.
2. **Using Entropy**:
   - When building a decision tree, we calculate entropy for different ways to split the data.
   - We choose the split that gives us the most ordered (pure) groups, reducing entropy the most.

Simple Calculation of Entropy

Imagine we have a small dataset with 10 instances where we want to classify whether people play tennis based on the weather. We have the following data:

PlayTennis	Count
Yes	6
No	4

Step-by-Step Calculation

1. **Total Instances**: 10 (6 Yes + 4 No)
2. **Proportion of Each Class**:
   - $p(Yes) = \frac{6}{10} = 0.6$
   - $p(No) = \frac{4}{10} = 0.4$
3. **Entropy Formula**: Entropy (H) is calculated using the formula:

$$H(S) = -\sum_{i=1}^{n} p(i) \log_2(p(i))$$

   where $p(i)$ is the proportion of class i.

4. **Calculate Each Term**:

- For "Yes":

$$-p(Yes)\log_2(p(Yes)) = -0.6\log_2(0.6)$$

Using a calculator:

$$\log_2(0.6) \approx -0.737$$

So,

$$-0.6 \times (-0.737) \approx 0.442$$

- For "No":

$$-p(No)\log_2(p(No)) = -0.4\log_2(0.4)$$

Using a calculator:

$$\log_2(0.4) \approx -1.322$$

So,

$$-0.4 \times (-1.322) \approx 0.529$$

5. **Sum the Terms**:

$$H(S) = 0.442 + 0.529 = 0.971$$

So, the entropy of our dataset is approximately 0.971. This value tells us how mixed our dataset is. A value of 0 would mean it's completely pure (all "Yes" or all "No"), and higher values indicate more disorder.

Using Entropy in Decision Trees

1. **Calculate Initial Entropy**:
   - The initial entropy tells us how mixed the data is before any splits.
2. **Choose a Split**:
   - We might start by asking, "Is it sunny?"
   - Calculate the entropy for the "Yes" and "No" branches:
     - If it's sunny: how mixed are the "Yes" and "No" answers?
     - If it's not sunny: how mixed are the "Yes" and "No" answers?
3. **Information Gain**:
   - Information gain measures how much we reduced the entropy by making a split.
   - Choose the question (split) that gives us the highest information gain, meaning it reduces the most disorder.
4. **Repeat the Process**:
   - Repeat this process for each branch until you reach pure groups or can't split anymore.

Putting It All Together

Imagine you're organizing your toys:

1. You start with a mixed pile of different toys.
2. You ask, "Is it a car?" and split the toys into cars and non-cars.
3. You keep asking questions that best split the toys into neat, organized groups.
4. At each step, you use entropy to decide the best way to split the pile.

By the end, you have a tree that helps you decide the best way to organize (or classify) your toys (or data) based on a series of questions.

This approach helps in making decisions and predictions in business, like whether a customer will buy a product based on their characteristics, by splitting data into more and more organized (less mixed) groups.

Issues with Decision-trees

1) Overfitting: Often over-complex trees on training set do not generalize well to the data well, a problem called overfitting. Overfitting is a big practical problem for decision tree models.

2) Decision trees are often unstable because with minor changes in the data, completely different tree can be generated. We try to reduce this issue by methods like bagging and boosting.