Study Notes for AI (CS 440/520)
Lectures 20-21: Inductive Learning - Decision Trees

Corresponding Book Chapters: Chapters 18.1-18.2-18.3-18.4
Note: These notes provide only a short summary and some highlights of the material covered in the corresponding lecture based on notes collected from students. Make sure you check the corresponding chapters. Please report any mistakes in or any other issues with the notes to the instructor.

1 Inductive Learning

This is a form of deterministic supervised learning

- Given: The correct value of an unknown function for particular inputs
- Output: Recover the unknown function or approximate it.

For inductive learning to be able to operate, we require the availability of examples: $(x(f(x))$.

1.1 Hypothesis

Given then examples of f, return function h that approximates f. The function h that inductive learning will be computing is called a hypothesis:

- Not easy to tell whether any particular h is a good approximation of f.
- A consistent hypothesis is one that agrees with all the available data points/examples.
- A good consistent hypothesis: Generalizes well.

Consider for example the problems shown in Figure 1, where we want to compute the function that has produced the sample points that we have available. In order to solve this problem, we must first decide which is the hypothesis space, the space that we will look for an answer. For example, we can consider polynomials of degree at most K as in cases (a), (b) and (c) in the Figure. For different values of K, however, we have a different hypothesis space. Or we can consider a sinusoidal function as in (d).

It is often the case that multiple hypothesis spaces contain different consistent hypotheses. For example, the data points in parts (a) and (b) in the figure are the same, but as we can see we have polynomials of different degree that are able to fit these points. So the question arises, how do we choose among multiple consistent hypotheses? The answer to the above question is known as Ockham's razor: Prefer the simplest hypothesis consistent with the data.

It might also happen that a hypothesis space does not contain a consistent hypothesis. A learning problem is considered realizable if the hypothesis space contains the true function. Typically, we are faced with the following trade-off between selecting complex and powerful hypothesis spaces and considering simpler spaces:

- Expressiveness of the hypothesis space
- Complexity of finding simple, consistent hypothesis within that space.

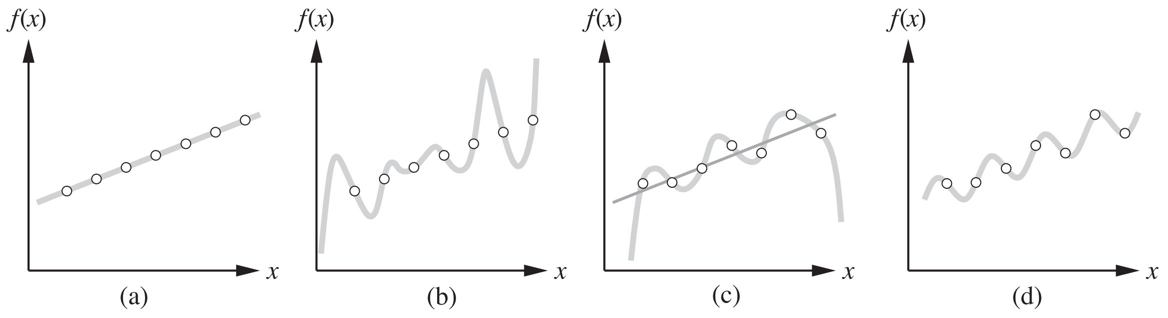


Figure 1: Fitting functions given examples of their outputs.

1.2 Decision Tree approach

- Input: An object / situation described by a set of attributes, where each attribute belong to a particular class.
- Output: A way to take decisions or classify new unseen examples.

The decision tree approach constructs a tree data structure. The internal nodes of the tree correspond to attributes. The edges of the trees correspond to the possible values that the parent attributes can acquire. Once a decision tree is constructed we can use it to correctly classify a new example according to its attribute values by performing a sequence of tests at each node of the tree. The final decision of the tree is stored at the leaf nodes.

1.3 Example

A somewhat simpler example is provided by the problem of whether to wait for a table at a restaurant. The aim here is to learn a definition for the predicate $WillWait$. In setting this up as a learning problem, we first have to state what attributes are available to describe examples in the domain. Let's suppose we decide on the following list of attributes:

1. $Alternate$: whether there is a suitable alternative restaurant nearby
2. Bar: whether the restaurant has a comfortable bar area to wait in
3. Fri/Sat: true on Fridays and Saturdays
4. $Hungry$: whether we are hungry
5. $Patrons$: how many people are in the restaurant (values are None, Some and Full)
6. $Price$: the restaurant's price range ($,$$,$$$)
7. $Raining$: whether it is raining outside
8. $Reservation$: whether he made a reservation
9. $Type$: the kind of restaurant (French, Italian, Thai or burger)
10. $WaitEstimate$: the wait estimated by the host (0-10 minutes, 10-30, 30-60, ¿60)

A possible decision tree for the restaurant example is shown in Figure 2. Notice that the tree does not use the $Price$ and $Type$ attributes, in effect considering them to be irrelevant. Examples are processed by the tree starting at the root and following the appropriate branch until a leaf is reached. For instance, an example with $Patrons = Full$ and $WaitEstimate = 0 - 10$ will be classified as positive (i.e., yes, we will wait for a table.

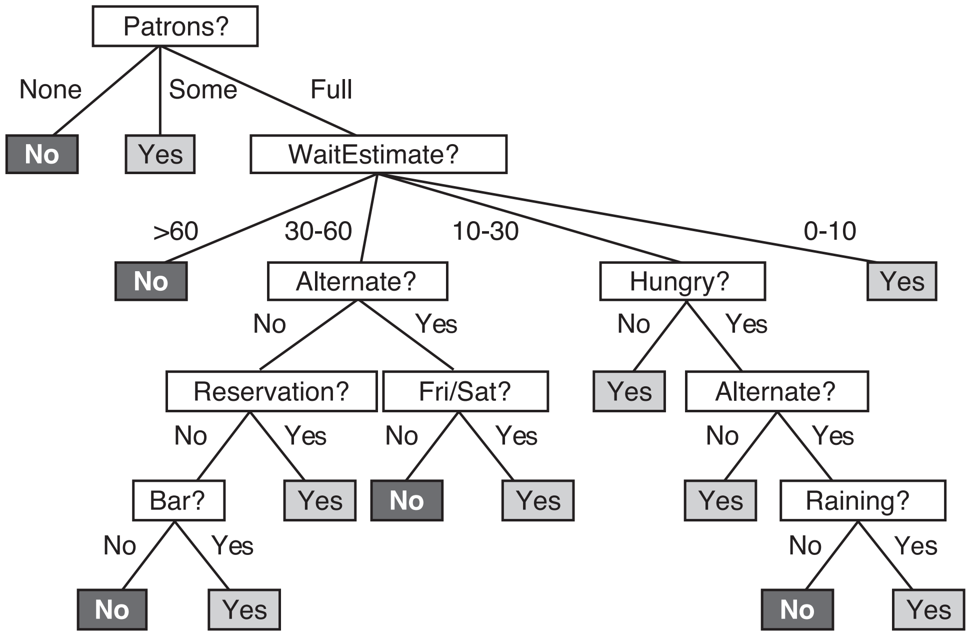


Figure 2: A decision tree for deciding whether to wait for a table.

1.4 How to learn from a training set?

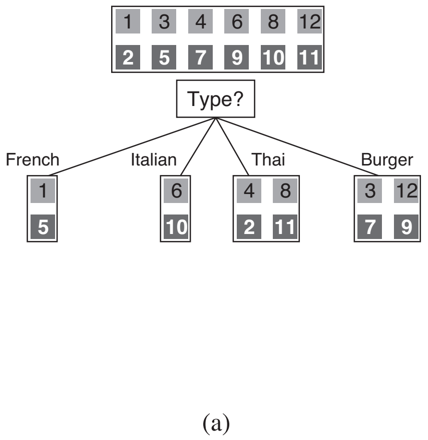
- Simple approach: memorize each observation (1 path for each example). This approach does not generalize.
- Alternative objective: Find the smallest decision tree consistent with the examples by testing the most important attribute first, which is the one that makes the most difference to the classification of an example.

Consider, for example, the Figure 3. If we select the attribute *Type* as the first attribute to split the examples, then the resulting split does not help us in quickly identifying how the examples are classified. Notice that all the resulting subsets in the left part of the figure have examples of both classes. On the right side, however, we see that if we select *Patrons* as the attribute to split the examples, then we know directly that if the value of this attribute is *None*, then the answer to the "Wait for table" question is no.
Overall the operation of a decision-tree follows the following rule:

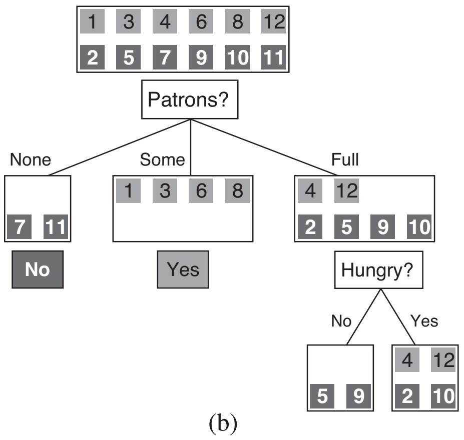
1. If there exist some positive and some negative examples in a subset, choose the attribute that best splits them.
2. If all examples are positive (or all negative), then we are done with this subset of examples.
3. If no examples are left, then return a default value calculated from the majority classification at the node's parent.
4. If no attributes are left and we have both positive and negative examples, this means the either we do not have enough attributes or there is noise is in the data.

2 Choosing attributes

Here, we will describe an automated way for selecting attributes. The objective is to select attributes so as to minimize the depth of the final tree and providing an exact classification as soon as possible. To achieve this objective, we will use as a measure of an attribute's utility the amount of information that it provides. (Shannon and Weaver '49).



(a)



(b)

Figure 3: Different classifications based on the attribute chosen to split the examples.

An attribute's utility is related to the amount of information it provides. If the possible answers v_i have probabilities $P(v_i)$, then the information content I of the actual answer is given by

$$I(P(v_1), ..., P(v_n)) = \sum_{i=1}^{n} -P(v_i) \cdot log_2 P \cdot (v_i).$$

As a simple example, we can think about the tossing of a fair coin. The probability of each outcome is one half, so the information content is

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2} log_2 \frac{1}{2} - \frac{1}{2} log_2 \frac{1}{2} = 1.$$

By the way, bits are used as the unit to represent information content. So, the information content for the tossing of a fair coin calculated as 1 bit.

2.1 Estimate of the Information

An estimate of the probabilities of the possible answers before any of the attributes have been tested is given by the proportions of positive and negative examples in the training set. Given the training set of p positive examples and n negative examples, the estimate of the information contained in a correct answer is

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}.$$

2.2 Remainder

Since a test on a single attribute A will not usually tell us all of information needed to make a decision, we must measure exactly how much information we still need after the attribute test. Each attribute A divides the training set E into subsets $E_1, ..., E_n$ according to their values of A(attribute), where A can have v distinct values. Each subset E_i has p_i positive examples and n_i negative examples. So, if we go along that branch, we need $I(p_i/(p_i+n_i), n_i/(p_i+n_i))$ bits of information to answer the question. A randomly chosen example from the training set has the ith value of the attribute with probability $(p_i + n_i)/(p + n)$. On average after testing attribute A, we need to know $Remainder(A)$

bits of information to classify the example:

$$Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right).$$

2.3 Information Gain

The information gain from any attribute test is the difference between the original information requirement and the information still needed after the test:

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - Remainder(A). \tag{1}$$

From Equation 1 and Figure 3, we can answer which attribute should be tested in the decision tree.

$$Gain(Patrons) = 1 - \left[\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I\left(\frac{2}{6}, \frac{4}{6}\right)\right] \approx 0.541$$

$$Gain(Type) = 1 - \left[\frac{2}{12}I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12}I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12}I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12}I\left(\frac{2}{4}, \frac{2}{4}\right)\right] = 0$$

The information gained from testing *Patrons* is higher than that gained from testing *Type*, so *Patrons* should be used in the decision tree. This result is intuitive considering that no progress is made in classifying the examples using *Type* while half of the examples have been correctly classified after using *Patrons*.

3 Assessing the performance of a learning algorithm

A learning algorithm is good if it produces hypothesis that do a good job of predicting the classifications of unseen examples. Since we are aiming to compute consistent hypothesis, the examples we use to train a learning algorithm (e.g., decision tree) are already correctly classified by the tree. So in order to evaluate the performance of a learning algorithm, we have to use a separate set of examples, called the test set.

Overall, the steps of the methodology to assess the performance of a learning algorithm are the following:

1. Collect a large set of examples.
2. Divide it into 2 distinct sets: Training set and Test set.
3. Apply the learning algorithm to the training set and generate hypothesis h.
4. Measure the percentage of examples in the test set correctly classified by h.
5. Repeat steps 2→4 for different sizes of training vs. test set, and randomly select examples for training and test sets.

What performances measure should we report in this case?

- Not the prediction performance of the best hypothesis on the test data, since the hypothesis now does depend on the test data as well.
- Use a new evaluation set

The goal of the above steps is to design a learning procedure that is able to generalize and avoid **overfitting**. Overfitting occurs when the learning algorithm is trained to detect meaningless "regularity" in the data. For example, consider the problem of trying to predict the roll of a die. Suppose the experiments are carried out during an extended period of time with various dice and that the attributes describing each training example are as follows:

1. Day : the day on which the die was rolled (Mon, Tue, etc).
2. $Month$: the month in which the die was rolled (Jan or Feb).
3. $Color$: the color of the die (Red or Blue).

The decision tree approach we described will be able to find an exact hypothesis from examples as long as no two examples have identical descriptions and different classification. Any such hypothesis, however, will be totally spurious, since the attributes are actually irrelevant to the outcome of the dice.