Machine Learning

CSE 5290 @ FIT Study Guide

This document's content is in part adapted from "Artificial Intelligence: A Modern Approach" (4th Edition) by Stuart Russell and Peter Norvig, and the slides provided with it. Additionally, this document's content is in part adapted from Dr. Debasis Mitra's slides covering the same book.

Table of Contents

Introduction

In **machine learning**, a computer observes some data, builds a **model** based on the data, and uses the model as both a **hypothesis** about the world and a piece of software that can solve problems.

Machine learning can be broken into three quintessential types in this course:

- **Supervised learning**, where the agent observes input-output pairs and learns a function that maps from input to output. The type of supervised learning problem depends on the nature of the outputs:

    - When the output is one of a finite set of values, the learning problem is called **classification**.
    - When the output is a number, the learning problem is called **regression**.
- **Unsupervised learning**, where the agent learns patterns in the input without any explicit feedback. The most popular example is **clustering**, where data is grouped or *clustered* based by "proximity" (more metrics than just Euclidean distance).
- **Reinforcement learning**, where the agent learns from a series of **reinforcements** (rewards and punishments). The agent decides which actions led to the reinforcement, and how to alter it's actions to receive more rewards.

Recommend watching: Neural networks by 3Blue1Brown - https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

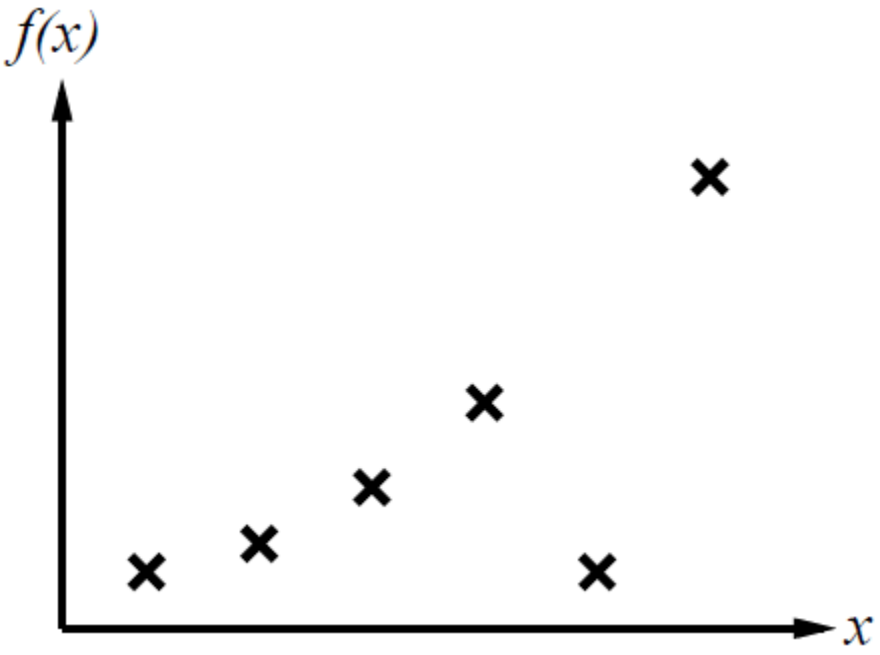
Inductive Learning

Concept

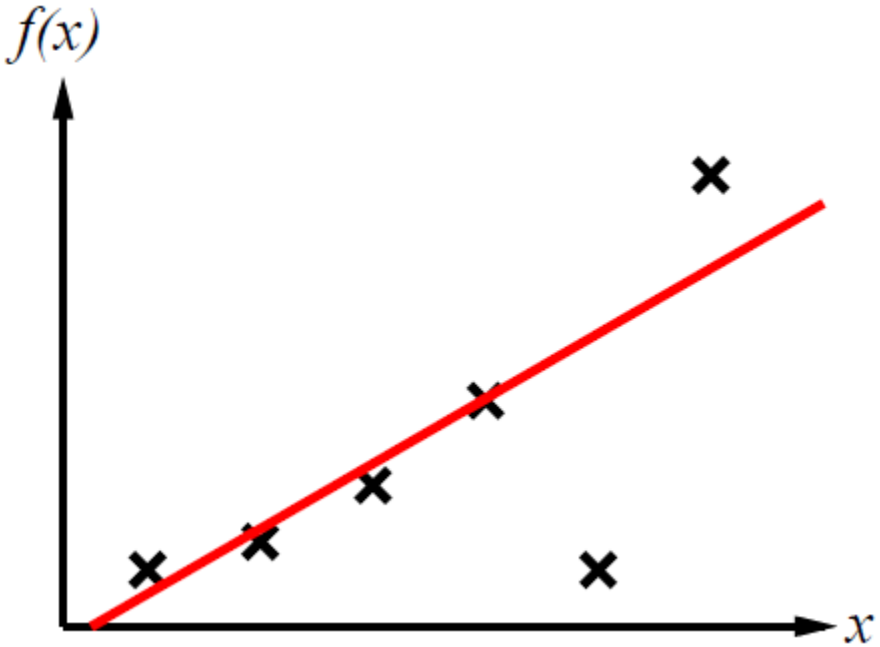
- Given a data set (where each point is on the unknown **target function** f), find a **hypothesis** h such that $h \approx f$ given a **training set** of examples.
- The objective of **inductive learning** is to construct/adjust h to agree with f on the training set.
    - **Induction** = going from a specific set of observations to a general rule
- h is **consistent** if it agrees with f on all examples.
- One strategy for this is **curve fitting**. There are three types of curve fitting:
    - **Underfitting**, where h fails to find or match the pattern in the data.
    - **Best-fit**, where each $h(x_i)$ is close to y_i.
    - **Overfitting**, where h pays too much attention to the training set, causing it to perform poorly on unseen data.
- Remember **Ockham's razor**: "plurality should not be posited without necessity." Simply: be consistent and accurate, but as simple as possible.

Example

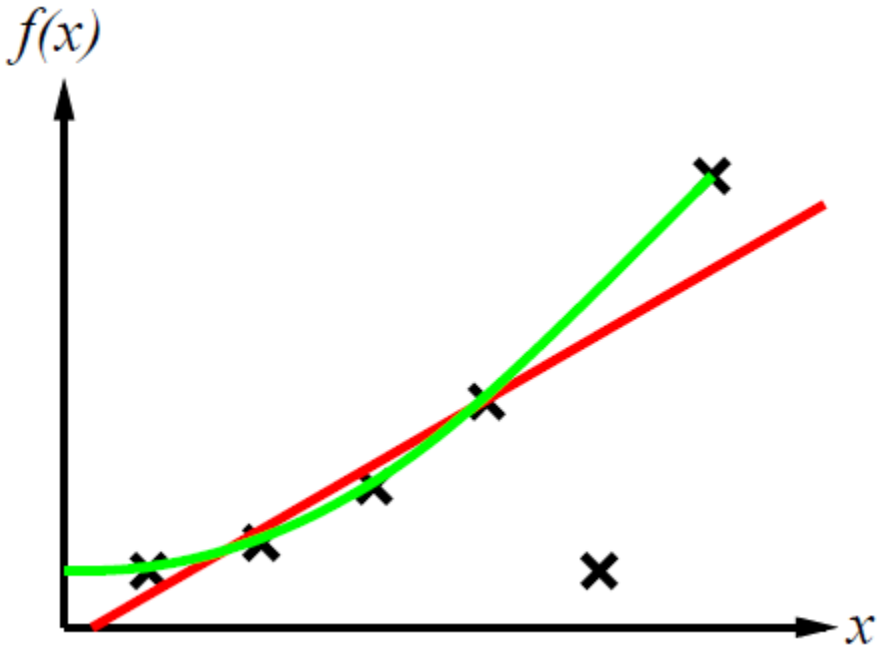
Given the following data set:



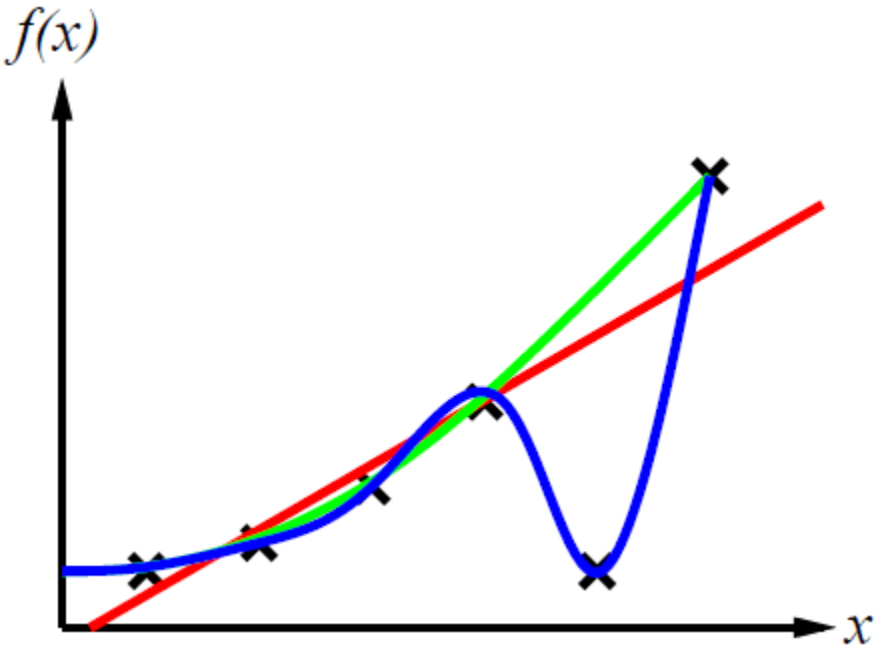
We want to find $h(x)$. We start with the model $h(x) = w_1x + w_0$:



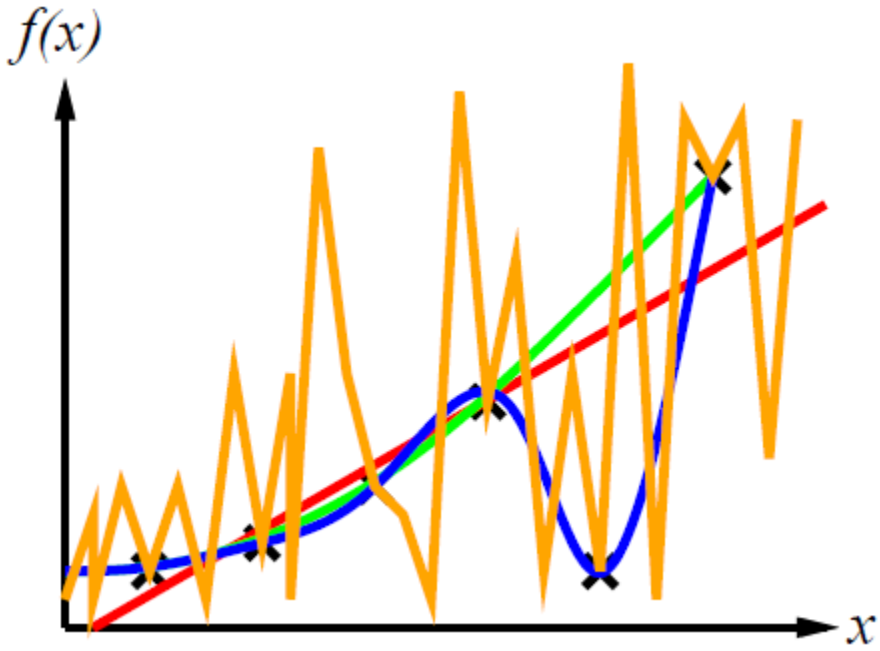
However, this model doesn't support the sufficient shape to match the training set. Next, we try $h(x) = w_2x^2 + w_1x + w_0$:



This is a lot closer, but it still omits the 5th data point from the left on the x axis. We repeat this process until we obtain:



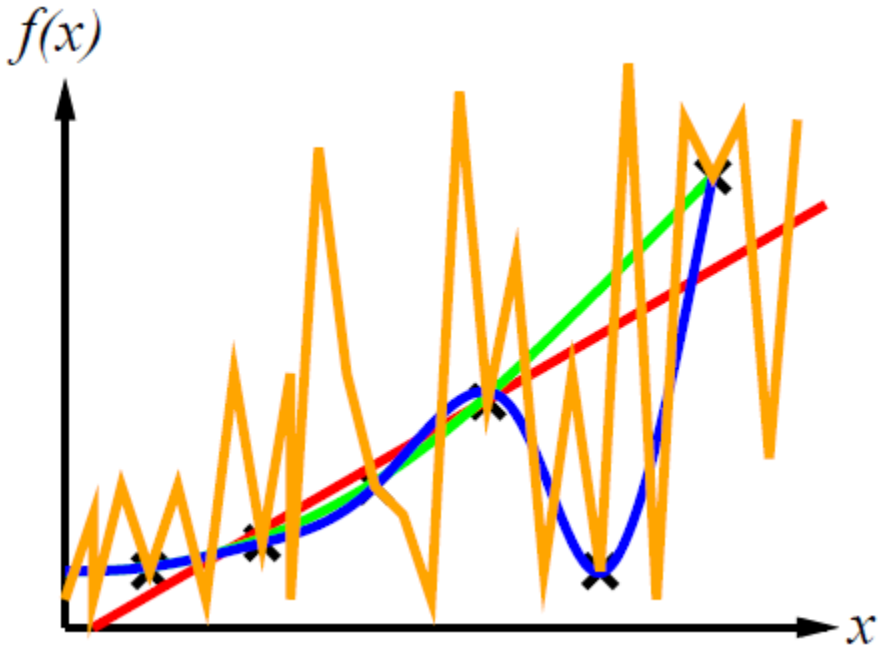
This h is nearly consistent. However, if repeat our process attempting to get a perfect fit:



We get a perfect fit that would perform horribly on any other data points in f.

Problem 1

Select all attributes that apply to h in the following graph:



- ☐ Consistent
- ☐ Underfit
- ☐ Best fit
- ☐ Overfit

Answer(s): Consistent, overfit.

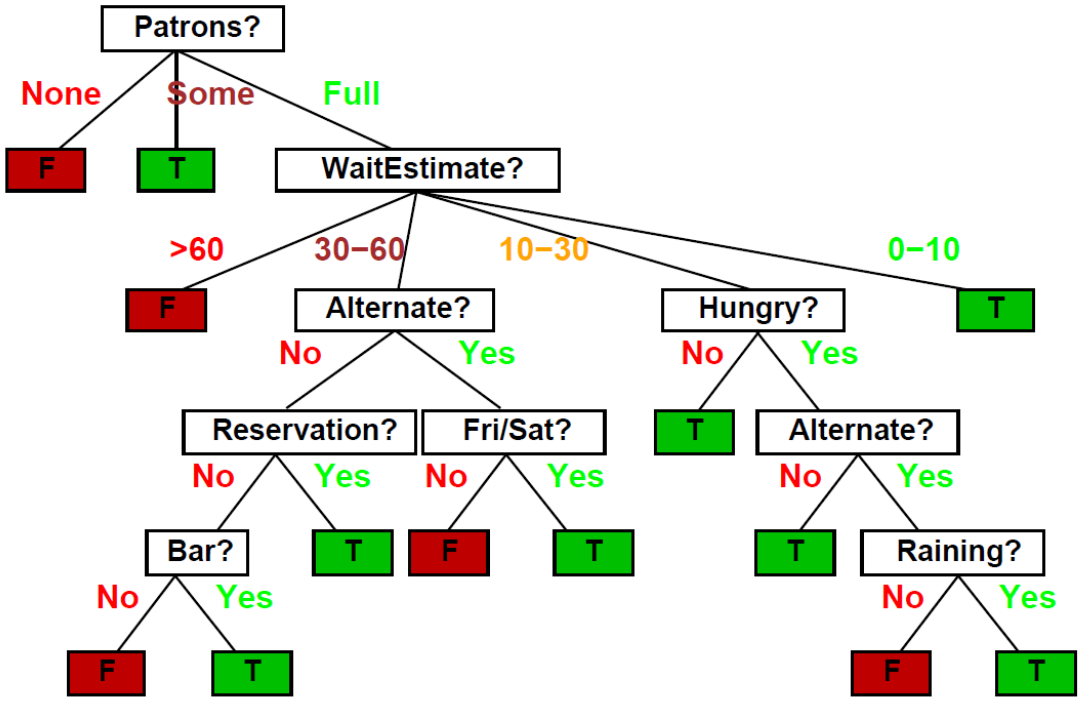
Decision Trees

Introduction

Consider an individual deciding whether to wait for a table at a restaurant, considering a variety of attributes. An example of this is illustrated with the following table:

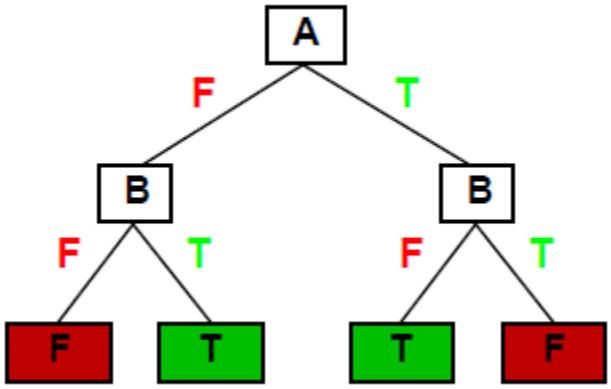
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

The logic behind the decision is illustrated with the following **decision tree**:



A decision tree is a representation of a function that maps a vector of attribute values to a single output value – a "decision." Input and output attributes can be discrete or continuous, and decision trees can express any function of the input attributes:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



One could form a consistent decision tree for any training set by creating a path for each point in the training set. However, not only would this likely be overfitting, but we also prefer to find **compact** decision trees.

Not only can decision trees become large very quickly, there's also many possible decision trees (hypotheses) with relatively few attributes. The number of distinct decision trees with n Boolean attributes is the number of Boolean functions, meaning the number of distinct truth tables with 2^n rows $= 2^{2^n}$. Therefore, with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees!

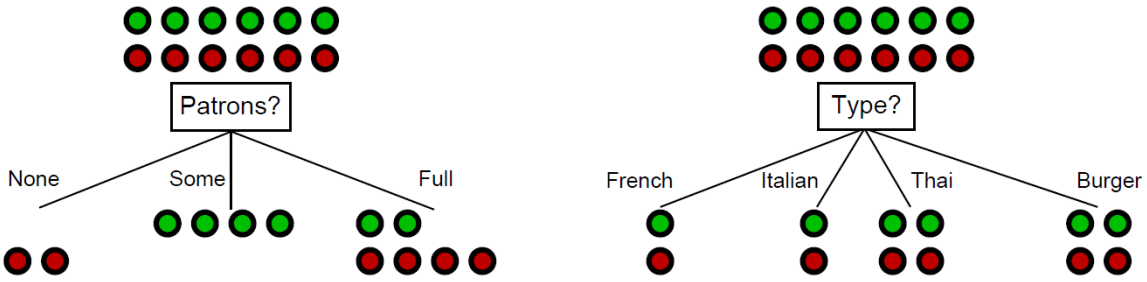
This means that although there's a high chance the target function can be expressed with decision trees, there's a larger number of hypotheses consistent with the training set which may or may not be accurate, meaning that one may get worse predictions.

Recommended watching: Decision Tree Classification Clearly Explained! by Normalized Nerd - https://www.youtube.com/watch?v=ZVR2Way4nwQ

Learning

So, how do we create high quality decision trees? Let's define a quality tree: a small tree consistent with the training examples. We attempt to build a decision tree with the following algorithm by recursively choosing the "most significant"/decisive/informative attribute as root of (sub)tree. Simply: we want to put decisions in the tree based on attributes that will quickly split data points into subsets with the same classification. We will define a systematic metric for this choice shortly.

Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



Patrons? is a better choice—gives **information** about the classification

Algorithm

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return
          the classification
  else if attributes is empty then return MODE(examples)
  else
    best = CHOOSE-ATTRIBUTE(attributes, examples)
    tree = a new decision tree with root test best
    for each value v of best do
      vBestExamples = {elements of examples with best = v}
      subtree = DTL(vBestExamples, attributes – best, MODE(examples))
      add a branch to tree with label v and subtree subtree
    return tree
```

This is a good heuristic; however, we must be able to quantify **information**.

Problem 2

Consider a decision tree that inputs and outputs integers. The `DTL` algorithm is running with the following parameters:

- `examples = {2, 3, 1, 7, 2, 9, 4}`
- `attributes = {}`
- `default = MODE(examples)`

What will the algorithm return?

☐ 1

☐ 2

☐ 3

☐ 4

Answer: 2. The default return value when there are no attributes is the mode of the examples. In this case, the most occurring value is 2. 1 is `min(examples)`, 3 is `median(examples)`, and 4 is `mean(examples)`.

Information Theory

When answering a question, there is a set of possible answers. Observations that narrow the set of possible answers have information. We quantify information with a unit called bits.

If an observation halves set of possible answers is halved, we say it has 1 bit of information. If the observation reduces the set of possible answers by a factor of four, we say it has 2 bits of information. If the observation reduces the set of possible answers by a factor of eight, we say it has 3 bits of information, so on and so forth. With this knowledge, we can calculate a formula for bits:

$$\left(\frac{1}{2}\right)^I = p$$
$$2^I = \frac{1}{p}$$
$$I = \log_2\left(\frac{1}{p}\right)$$
$$I = -\log_2(p)$$

p is the size of the reduced set of answers divided by the size of the original set of answers (ex. If an observation reduced 12 answers to 6 answers, $p = \frac{1}{2}$). I is the number of bits of information the observation has.

The **entropy** of a random variable (in our case, attribute) is a measurement of its uncertainty: the more information, the less entropy. the expected value of information the random variable has. We calculate the entropy of a random variable X with the following process:

1. Iterate through all $X_i \epsilon X$ (all assignments of the random variable), where X has n assignments.
2. Multiply the p (set reduction ratio) of X_i ($P(X_i)$) with the information (I) of X_i.
3. Sum all values.

Following this process, the Entropy of X is expressed through the following summation:

$$H(X) = \sum_{i=1}^{n} P(X_i) * (-\log_2 P(X_i)) = -\sum_{i=1}^{n} P(X_i) \log_2 P(X_i)$$

To demonstrate these concepts, we'll use our Decision Tree restaurant example. The entropy of the **prior** is the entropy considering no attributes. In the restaurant example, $WillWait$ is either $true$ or $false$. We will label the number of examples where $WillWait = true$ as p. We will label the number of examples where $WillWait = false$ as n. There are 12 examples total, 6 where $WillWait = true$, so $p = 6$, and 6 where $WillWait = false$, so $n = 6$. Each splits the examples in half, so $P(X_i)$ for the assignments are $P(WillWait = true) = P(WillWait = false) = \frac{p}{p+n} = \frac{n}{p+n} = \frac{6}{12} = \frac{1}{2}$.

The entropy of the prior ($H(Prior)$) is:

$$= -\big(P(WillWait = true)\log_2 P(WillWait = true) + P(WillWait = false)\log_2 P(WillWait = false)\big)$$

$$= -\left(\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$$

$$= -\left(\left(-\tfrac{1}{2}\right) + \left(-\tfrac{1}{2}\right)\right) = 1 \text{ bit.}$$

Now consider an attribute A with d distinct values. We'll use $Patrons?$, which has $d = 3$ (None, Some, and Full). For each example, A is assigned to one of these three values. This means that each value has a set of examples in which it is true, and a set of examples in which it is false. We will iterate through all distinct values of A with the iterator i. The number of examples where $A_i = true = A_{i_p}$, the number of examples where $A_i = false = A_{i_n}$, and the total number of examples = t.

The entropy (uncertainty) remaining after factoring in A is:

$$Remainder(A) = \sum_{i=1}^{d} \frac{A_{i_p} + A_{i_n}}{t} H\left(< \frac{A_{i_p}}{A_{i_p} + A_{i_n}}, \frac{A_{i_n}}{A_{i_p} + A_{i_n}} >\right)$$

$$= \frac{A_{i_p} + A_{i_n}}{t}\left(-\left(\frac{A_{i_p}}{A_{i_p} + A_{i_n}} \log_2 \frac{A_{i_p}}{A_{i_p} + A_{i_n}} + \frac{A_{i_n}}{A_{i_p} + A_{i_n}} \log_2 \frac{A_{i_n}}{A_{i_p} + A_{i_n}}\right)\right)$$

Information gain is expected reduction in entropy from factoring in an attribute/random variable. Therefore, the information gain of an attribute A is the entropy of the prior minus the entropy remaining after factoring in A:

$$Gain(A) = H(Prior) - Remainder(A)$$

Recommended watching: Solving Wordle using information theory by 3Blue1Brown - https://www.youtube.com/watch?v=v68zYyaEmEA

Example

We will calculate the information gain of $Patrons?$:

$$H(Prior) = 1$$

$$A_{None_p} = 0, A_{None_n} = 2, H(A_{None}) = -\left(\frac{0}{2}\log_2\left(\frac{0}{2}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right) = 0$$

$$A_{Some_p} = 4, A_{Some_n} = 0, H(A_{Some}) = -\left(\frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\left(\frac{0}{4}\right)\right) = 0$$

$$A_{Full_p} = 2, A_{Full_n} = 4, H(A_{Full}) = -\left(\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right) \approx -(-0.918) = 0.918$$

$$Remainder(A) = \frac{0+2}{12}A_{None_p} + \frac{4+0}{12}A_{Some_p} + \frac{2+4}{12}A_{Full_p} = 0 + 0 + 0.459 = 0.459$$

$$Gain(A) = 1 - 0.459 \approx 0.541\ bits$$

Therefore, when factoring in $Patrons?$, we gain 0.541 bits of information.

Problem 3

Compute the Information Gain of Outlook from the following table:

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
		9	5	**14**

Answer:

First, we must get the entropy of the prior.

$$H(Prior) = -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) \approx 0.94$$

Next, we need to compute the remaining entropy of Outlook.

$$H(Sunny) = -\left(\frac{3}{5}\log_2\left(\frac{3}{5}\right) + \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right) \approx 0.971$$
$$H(Overcast) = -\left(\frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\frac{4}{4}\right) = 0$$
$$H(Rainy) = H(Sunny) \approx 0.971$$
$$Remainder(Outlook) = \frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971) \approx 0.694$$

Now that we have these numbers, we can compute the information gain for Outlook in bits.

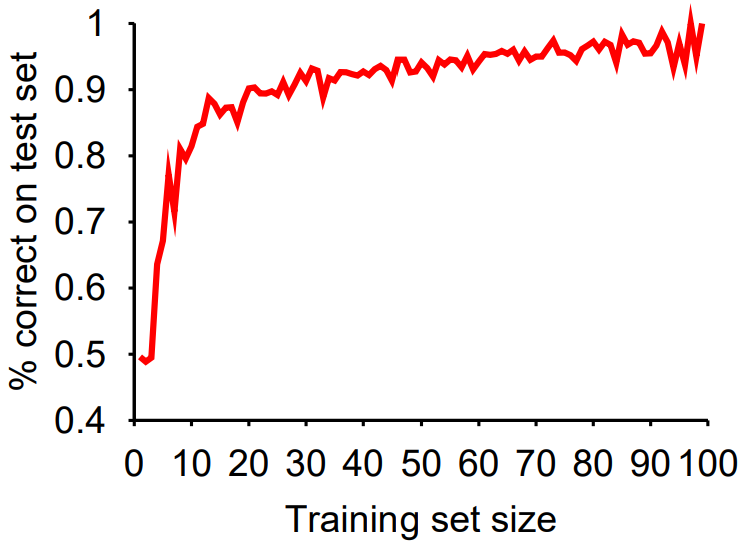
$$Gain(Outlook) = H(Prior) - Remainder(Outlook)$$
$$Gain(Outlook) = 0.94 - 0.694 = 0.246$$

When factoring in Outlook, we gain 0.246 bits of information.

Performance Measurement

So, we've got some working strategies to learn h, how do we know that $h \approx f$? One way is to use theorems of computational/statistical learning theory. However, another way is to try h on a new **test set** of examples. The practical implementation of this is splitting your total data set into training data and test data when you begin creating your model. Training data is used to learn h, and the test set is to see how it performs on data it hasn't seen before.

To quantify model performance, we introduce the **learning curve**, which is the % of correct answers from the model on the test set as a function of training set size:



This learning curve depends on whether the target function is even **realizable** or not. Target function may be **non-realizable** due to missing attributes. Additionally, the learning curve could be negatively impacted by redundant expressiveness, where there are many irrelevant attributes being passed into the model.