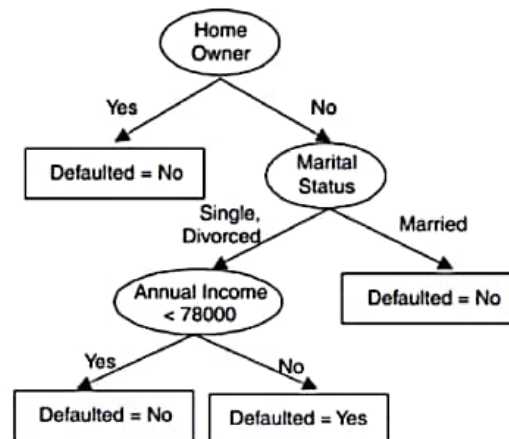


ID	Home Owner	Marital Status	Annual Income	Defaulted?
1	Yes	Single	125000	No
2	No	Married	100000	No
3	No	Single	70000	No
4	Yes	Married	120000	No
5	No	Divorced	95000	Yes
6	No	Single	60000	No
7	Yes	Divorced	220000	No
8	No	Single	85000	Yes
9	No	Married	75000	No
10	No	Single	90000	Yes



either $p_0(t)$ or $p_1(t)$ equals to 1). The following examples illustrate how the values of the impurity measures vary as we alter the class distribution.

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Entropy} = -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Entropy} = -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$$

Node N_3	Count
Class=0	3
Class=1	3

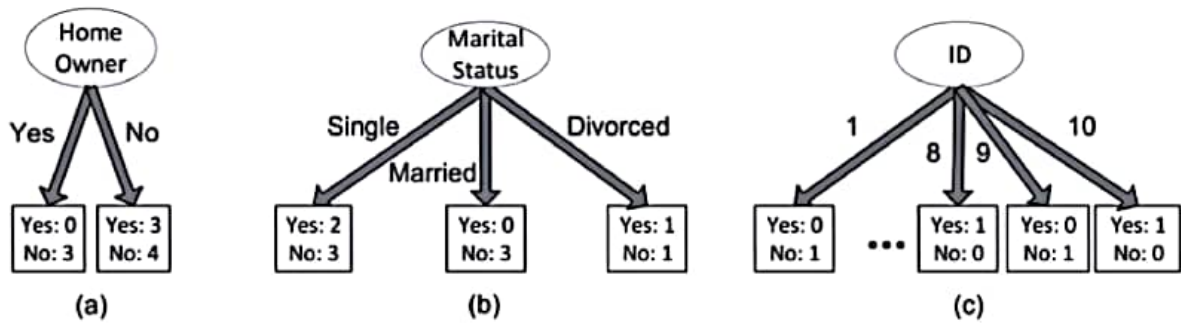
$$\text{Entropy} = -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$$

Collective Impurity of Child Nodes

Consider an attribute test condition that splits a node containing N training instances into k children, $\{v_1, v_2, \dots, v_k\}$, where every child node represents a partition of the data resulting from one of the k outcomes of the attribute test condition. Let $N(v_j)$ be the number of training instances associated with a child node v_j , whose impurity value is $I(v_j)$. Since a training instance in the parent node reaches node v_j for a fraction of $N(v_j)/N$ times, the collective impurity of the child nodes can be computed by taking a weighted sum of the impurities of the child nodes, as follows:

$$I(\text{children}) = \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j), \quad (3.7)$$

Example 3.3. [Weighted Entropy] Consider the candidate attribute test condition shown in Figures 3.12(a) and (b) for the loan borrower classification problem. Splitting on the **Home Owner** attribute will generate two child nodes



whose **weighted entropy** can be calculated as follows:

$$I(\text{Home Owner} = \text{yes}) = -\frac{0}{3} \log_2 \frac{0}{3} - \frac{3}{3} \log_2 \frac{3}{3} = 0$$

$$I(\text{Home Owner} = \text{no}) = -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0.985$$

$$I(\text{Home Owner}) = \frac{3}{10} \times 0 + \frac{7}{10} \times 0.985 = 0.690$$

Splitting on **Marital Status**, on the other hand, leads to three child nodes with a weighted entropy given by

$$\begin{aligned}
 I(\text{Marital Status} = \text{Single}) &= -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971 \\
 I(\text{Marital Status} = \text{Married}) &= -\frac{0}{3} \log_2 \frac{0}{3} - \frac{3}{3} \log_2 \frac{3}{3} = 0 \\
 I(\text{Marital Status} = \text{Divorced}) &= -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1.000 \\
 I(\text{Marital Status}) &= \frac{5}{10} \times 0.971 + \frac{3}{10} \times 0 + \frac{2}{10} \times 1 = 0.686
 \end{aligned}$$

Thus, **Marital Status** has a lower weighted entropy than **Home Owner**. ■

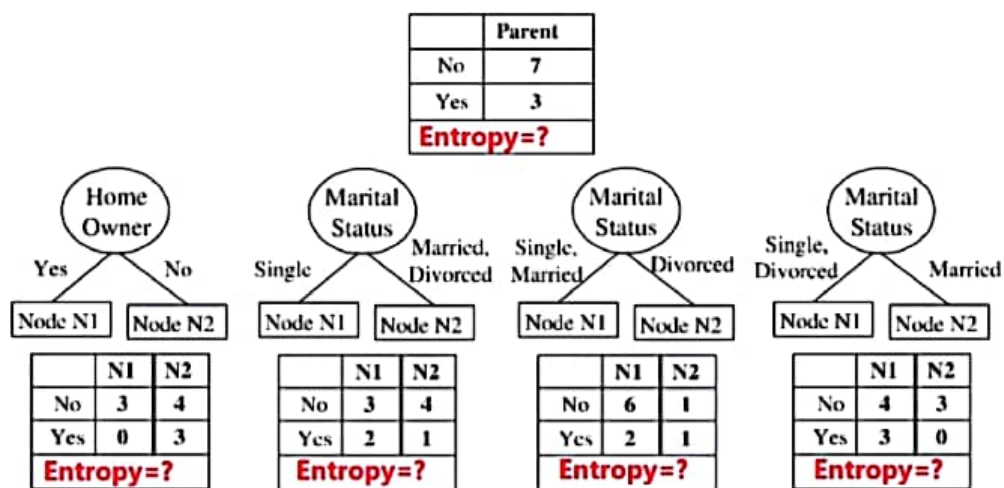
Identifying the best attribute test condition

To determine the goodness of an attribute test condition, we need to compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. This difference, Δ , also termed as the **gain** in purity of an attribute test condition, can be defined as follows:

$$\Delta = I(\text{parent}) - I(\text{children}), \quad (3.8)$$

where $I(\text{parent})$ is the impurity of a node before splitting and $I(\text{children})$ is the weighted impurity measure after splitting. It can be shown that the gain is non-negative since $I(\text{parent}) \geq I(\text{children})$ for any reasonable measure such as those presented above. The higher the gain, the purer are the classes in the child nodes relative to the parent node. The splitting criterion in the decision tree learning algorithm selects the attribute test condition that shows the maximum gain. Note that maximizing the gain at a given node is equivalent to minimizing the weighted impurity measure of its children since $I(\text{parent})$ is the same for all candidate attribute test conditions. Finally, when entropy is used as the impurity measure, the difference in entropy is commonly known as **information gain**, Δ_{info} .

In the following, we present illustrative approaches for identifying the best attribute test condition given qualitative or quantitative attributes.



Binary Splitting of Quantitative Attributes

Consider the problem of identifying the best binary split $\text{Annual Income} \leq \tau$ for the preceding loan approval classification problem. As discussed previously,

		Class		No	No	No	Yes	Yes	Yes	No	No	No	No										
		Annual Income (in '000s)																					
Sorted Values	→	60		70		75		85		90		95		100		120		125		220			
	Split Positions	55		65		72.5		80		87.5		92.5		97.5		110		122.5		172.5		230	
		<= >		<= >		<= >		<= >		<= >		<= >		<= >		<= >		<= >		<= >		<= >	
	Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
	No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Entropy		?		?		?		?		?		?		?		?		?		?		?	

Figure 3.14. Splitting continuous attributes.

Example 8: The hyper-rectangular regions in Fig. 3.6(A), which partitions the space, are produced by the decision tree in Fig. 3.6(B). There are two classes in the data, represented by empty circles and filled rectangles. ■

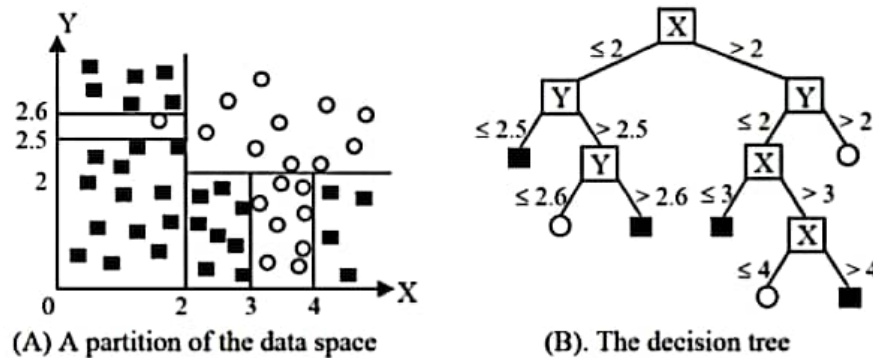


Fig. 3.6. A partitioning of the data space and its corresponding decision tree

Handling of continuous (numeric) attributes has an impact on the efficiency of the decision tree algorithm. With only discrete attributes the algorithm grows linearly with the size of the data set D . However, sorting of a continuous attribute takes $|D|\log|D|$ time, which can dominate the tree learning process. Sorting is important as it ensures that gain or gainRatio can be computed in one pass of the data.

We now discuss several other issues in decision tree learning.

Tree Pruning and Overfitting: A decision tree algorithm recursively partitions the data until there is no impurity or there is no attribute left. This process may result in trees that are very deep and many tree leaves may cover very few training examples. If we use such a tree to predict the training set, the accuracy will be very high. However, when it is used to classify unseen test set, the accuracy may be very low. The learning is thus not effective, i.e., the decision tree does not generalize the data well. This

phenomenon is called **overfitting**. More specifically, we say that a classifier f_1 **overfits** the data if there is another classifier f_2 such that f_1 achieves a higher accuracy on the training data than f_2 , but a lower accuracy on the unseen test data than f_2 [45].

Overfitting is usually caused by noise in the data, i.e., wrong class values/labels and/or wrong values of attributes, but it may also be due to the complexity and randomness of the application domain. These problems cause the decision tree algorithm to refine the tree by extending it to very deep using many attributes.

To reduce overfitting in the context of decision tree learning, we perform pruning of the tree, i.e., to delete some branches or sub-trees and replace them with leaves of majority classes. There are two main methods to do this, **stopping early** in tree building (which is also called **pre-pruning**) and **pruning** the tree after it is built (which is called **post-pruning**). Post-pruning has been shown more effective. Early-stopping can be dangerous because it is not clear what will happen if the tree is extended further (without stopping). **Post-pruning is more effective because after we have extended the tree to the fullest, it becomes clearer which branches/sub-trees may not be useful (overfit the data).** The general idea of post-pruning is to estimate the error of each tree node. If the estimated error for a node is less than the estimated error of its extended sub-tree, then the sub-tree is pruned. Most existing tree learning algorithms take this approach. See [49] for a technique called the pessimistic error based pruning.

Example 9: In Fig. 3.6(B), the sub-tree representing the rectangular region

$$X \leq 2, Y > 2.5, Y \leq 2.6$$

in Fig. 3.6(A) is very likely to be overfitting. The region is very small and contains only a single data point, which may be an error (or noise) in the data collection. If it is pruned, we obtain Fig. 3.7(A) and (B). ■

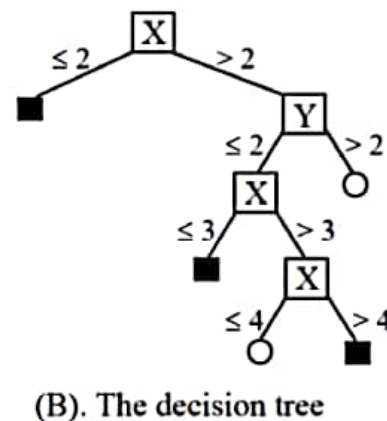
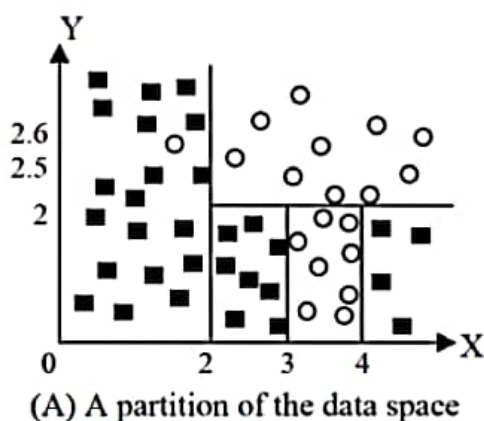


Fig. 3.7. The data space partition and the decision tree after pruning

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

It is often convenient to combine precision and recall into a single metric called the F_1 score, in particular if you need a simple way to compare two classifiers. The F_1 score is the *harmonic mean* of precision and recall (Equation 3-3). Whereas the regular mean

Equation 3-3. F_1

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2TP}{2TP + FN + FP}$$

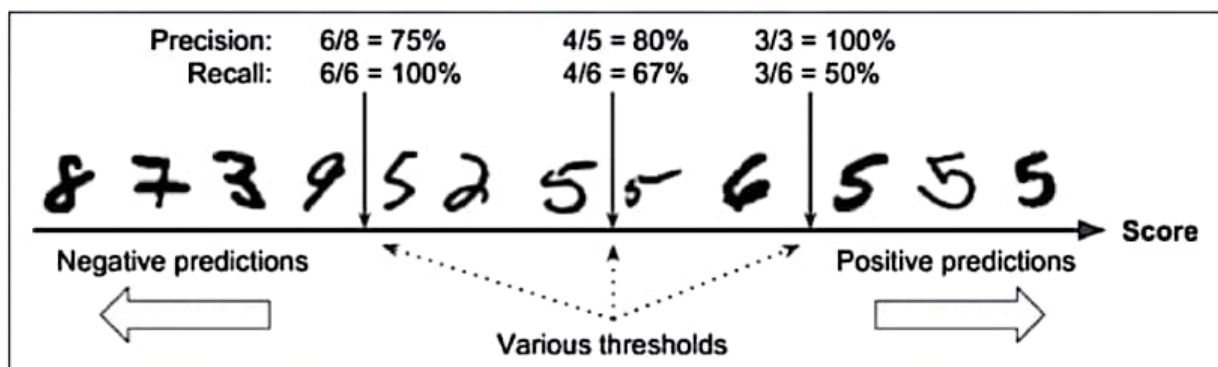
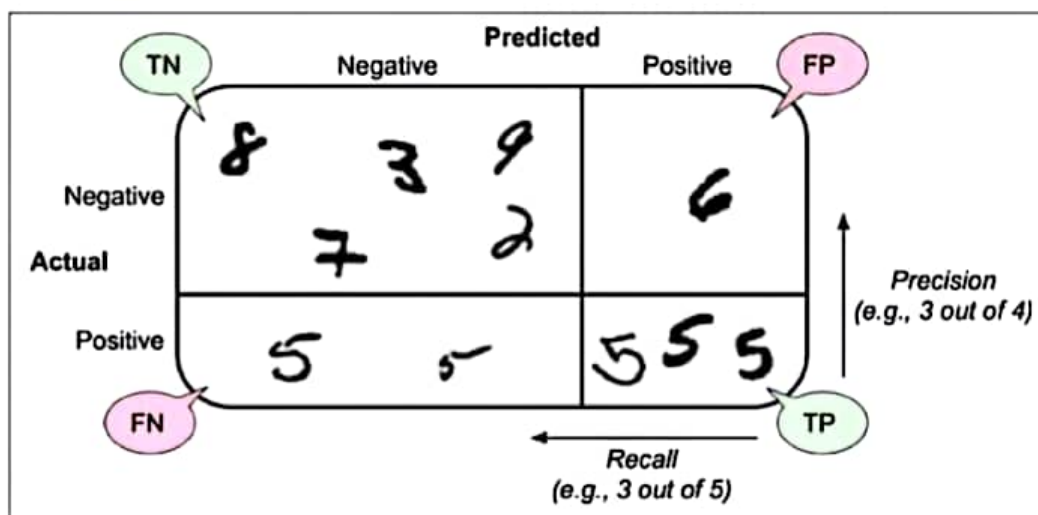


Figure 3-3. Decision threshold and precision/recall tradeoff

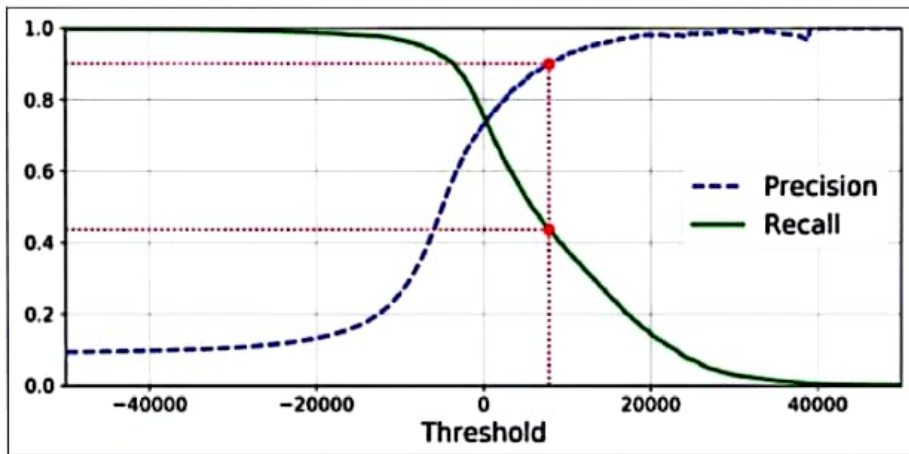
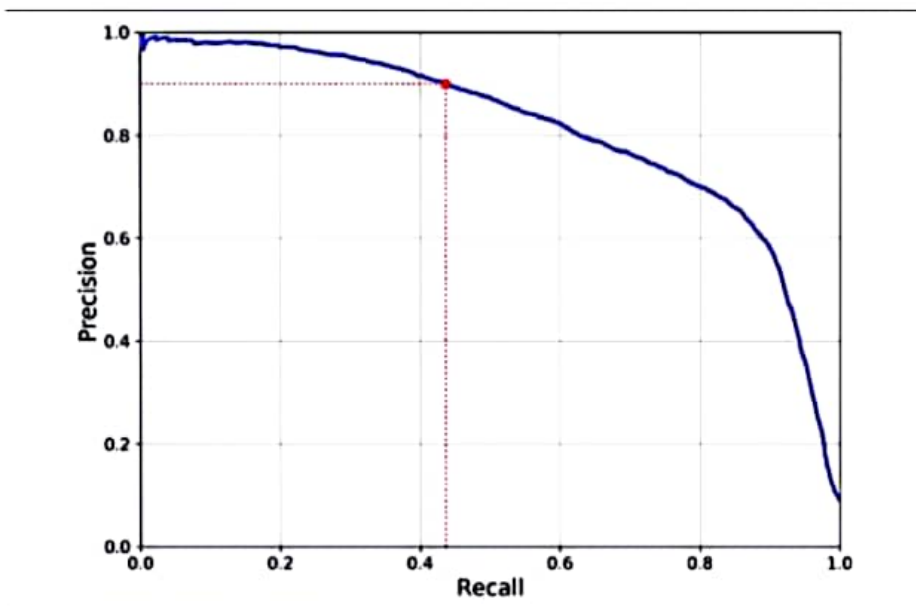


Figure 3-4. Precision and recall versus the decision threshold



3-5. Precision versus recall

1) What is a confusion matrix?

A confusion matrix is a tool for assessing the quality of a classification model in machine learning. It checks how many of the predictions were correctly or incorrectly assigned to a class.

2) What are the components (elements) of the Confusion Matrix, and explain them?

- True Positives (TP) – It is the case when both actual class & predicted class of data point is 1.
- True Negatives (TN) – It is the case when both actual class & predicted class of data point is 0.
- False Positives (FP) – It is the case when actual class of data point is 0 & predicted class of data point is 1.
- False Negatives (FN) – It is the case when actual class of data point is 1 & predicted class of data point is 0.

3) For the given confusion matrix

		Predicted class	
		NO	YES
Actual class	NO	55	15
	YES	10	105

Calculate :

- A) Accuracy
- B) Precision
- C) Recall
- D) Error rate
- E) F_1 score

- Accuracy = (TP + TN) / Total = (55+105)/185 = 0.86
- Precision = TP / Predicted Yes = 105 / 120 = 0.87
- Recall = TP / Actual Yes = 105 / 115 = 0.91
- Error Rate = 1 - Accuracy = (FN + FP) / Total = 1 - 0.86 = (15 +10) /185 = 0.14

$$\frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- $F_1 \text{ score} =$ = 0.8895

4) How to Calculate Confusion Matrix for a 2-class classification problem, If your threshold = .6 ?

y	y pred
0	0.5
1	0.9
0	0.7
1	0.7
1	0.3
0	0.4
1	0.5

Answer

	y	y pred	output for threshold 0.6	
Actual	0	0.5	0	Predicted
	1	0.9	1	
	0	0.7	1	
	1	0.7	1	
	1	0.3	0	
	0	0.4	0	
	1	0.5	0	

		Predicted class	
		NO	YES
Actual class	NO	2	1
	YES	2	2

5) From question 4, calculate precision, recall, Accuracy

- Accuracy = $(TP + TN) / \text{Total} = (2+2)/7 = 0.571$
- Precision = $TP / \text{Predicted Yes} = 2 / 3 = 0.67$
- Recall = $TP / \text{Actual Yes} = 2 / 4 = 0.5$

6) If you have a classifier to predict whether a person has diabetes or not. Say you have trained your model for 200K samples with 180K samples as a negative class (no diabetes), 20K samples as a positive class (diabetes), and you have achieved accuracy greater than 95%.

- Does your classifier performs well according to its accuracy?, Explain?

Answer

Diabetes detection and similar problems are mostly imbalanced classification tasks where most data points represent a negative class, and a positive class greatly outnumbered with a negative class. This is a fairly common problem in most of the classification tasks. In such a case the only accuracy metric is not a correct measure to check the performance of the model. Here Precision and Recall come in pictures when we want to get a clear insight about the performance of each class.

7) What is **precision/recall tradeoff**?

Unfortunately, you can't have both precision and recall high. If you increase precision, it will reduce recall and vice versa. This is called the **precision/recall tradeoff**.

8) The importance of precision or recall depends on the application itself, please mention some kinds of those applications

- **Example of High Precision**, If we have multiple platform for video streaming like well known YouTube, you have restricted mode to restrict the violent and adult videos for the kids. So model focus on high precision $\{TP/(TP+FP)\}$ by reducing the false positive. Means If model has classified the video is good for kids it must be safe to watch by kids. So, this can be done by reducing the false positive. Which will make higher Precision.
- **Example of High Recall**, Let's take an example, you are creating a model to detect a patient is having disease or not. In this case the aim of the model is to have high recall $\{TP/(TP+FN)\}$ means a smaller number of false negative. If model predict a patient is not having a disease so, he must not have the disease. Think about the vice-verse, if it predicts you do not have the disease and you enjoy your life and later you come to know that you that disease at the last stage.
- A model which detects a Loan Applicant is not a defaulter. Again, aim of the model is **high recall** $\{TP/(TP+FN)\}$. If model detects that applicant is not a defaulter so, applicant must not be a defaulter. So, model should reduce the false negative, which will increase the recall.

9) For the given confusion matrix, Calculate: (Accuracy, Misclassification Rate, True Positive Rate, False Positive Rate, True Negative Rate, Precision)

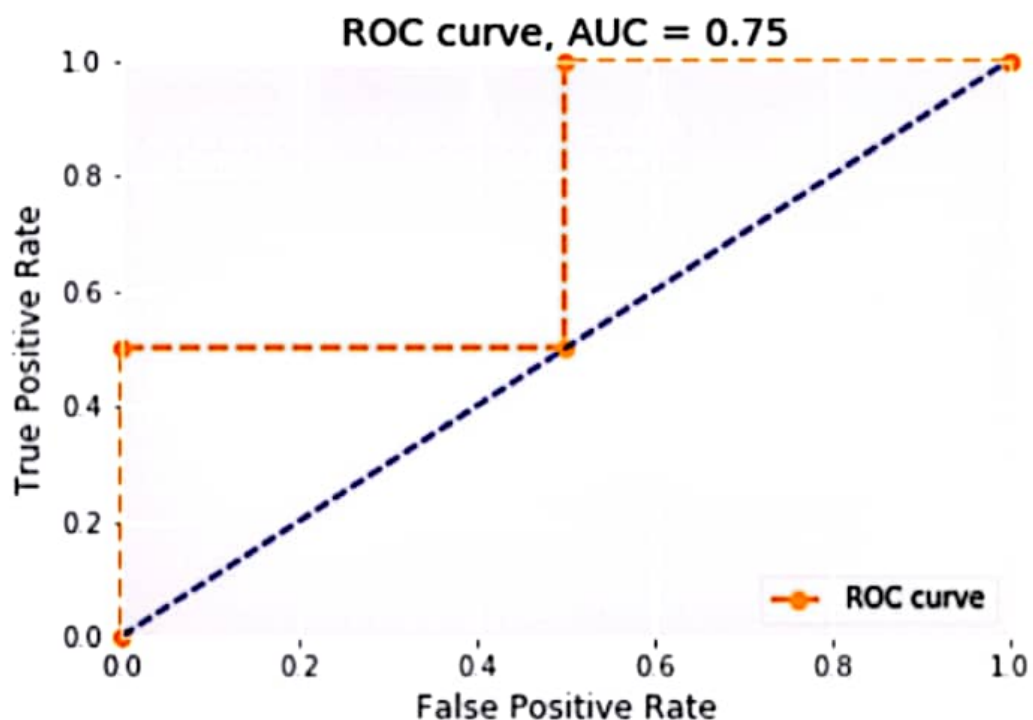
n=165		Predicted: NO	Predicted: YES	
Actual: NO		TN = 50	FP = 10	60
Actual: YES		FN = 5	TP = 100	105
		55	110	

- **Accuracy:** Overall, how often is the classifier correct?
 - $(TP+TN)/total = (100+50)/165 = 0.91$
- **Misclassification Rate:** Overall, how often is it wrong?
 - $(FP+FN)/total = (10+5)/165 = 0.09$
 - equivalent to 1 minus Accuracy
 - also known as "Error Rate"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
 - $TP/actual\ yes = 100/105 = 0.95$
 - also known as "Sensitivity" or "Recall"
- **False Positive Rate:** When it's actually no, how often does it predict yes?
 - $FP/actual\ no = 10/60 = 0.17$
- **True Negative Rate:** When it's actually no, how often does it predict no?
 - $TN/actual\ no = 50/60 = 0.83$
 - equivalent to 1 minus False Positive Rate
 - also known as "Specificity"
- **Precision:** When it predicts yes, how often is it correct?
 - $TP/predicted\ yes = 100/110 = 0.91$

10) **Draw the ROC curve.** If we have a binary classification problem with 4 observations. We know true class and predicted probabilities obtained by the algorithm. All we need to do, based on different threshold values, is to compute True Positive Rate (TPR) and False Positive Rate (FPR) values for each of the thresholds and then plot TPR against FPR.

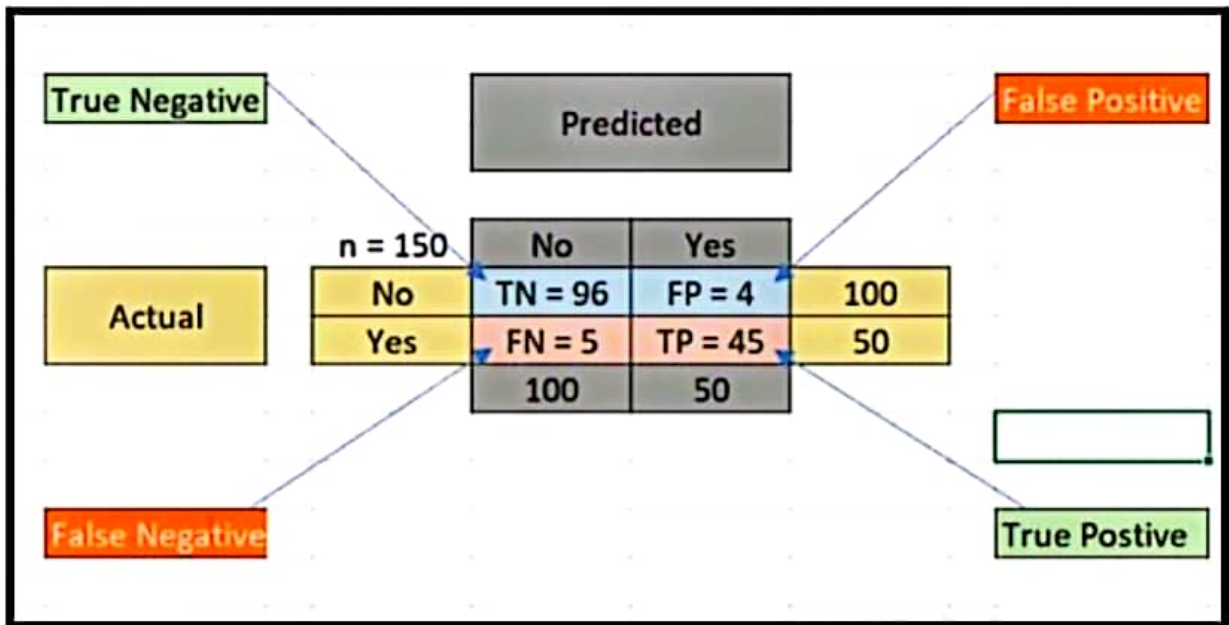
Actual Class (y)	Predicted Probabilities (\hat{y})	Threshold					
		(\hat{y}_0)	($\hat{y}_{0.2}$)	($\hat{y}_{0.4}$)	($\hat{y}_{0.6}$)	($\hat{y}_{0.8}$)	(\hat{y}_1)
1	0.8	1	1	1	1	1	0
0	0.6	1	1	1	1	0	0
1	0.4	1	1	1	0	0	0
0	0.2	1	1	0	0	0	0

TPR	1	1	1	0.5	0.5	0
FPR	1	1	0.5	0.5	0	0



Try to solve By yourself

- 1) For the given confusion matrix, Calculate: (Accuracy, Misclassification Rate, True Positive Rate, False Positive Rate, True Negative Rate, Precision)



- 2) How to Calculate Confusion Matrix for a 2-class classification problem, If your threshold = .5 ?

Y_original	Y_prob	Y_pred(threshold = 0.5)
1	0.6	1
1	0.4	0
0	0.2	0
1	0.7	1
0	0.6	1
0	0.3	0
1	0.4	0



The Decision Tree Algorithm: Information Gain

[Leave a Comment](#) / [Algorithms](#) / By [mehran@mldawn.com](#)

Which attribute to choose? (Part-2)

Today we are going to touch on something quite exciting. In our [previous post](#) we talked about Entropy of a set, $E(S)$, and told you that entropy is nothing but a quantitative measure of how mixed up our set is! I also showed you that regardless of how the decision tree (DT) chooses the attributes for splitting, the general trend in Entropy is a continuous reduction to the point that when we get to the leaves of the DT, the sets are absolutely pure and the Entropy is reduced to 0. Today, we will talk about this interesting idea called the **Information Gain (IG)**, which tells the DT about the quality of a potential split on an Attribute (A) in a set (S). Let's start 😊

Information Gain: Split with Fastest Descent in Entropy

The entropy of a given subset, S, can give us some information regarding the

chaos within S before we do any splitting. The decision tree (DT) however wants to know how it can reduce the entropy by choosing a smart splitting policy. More specifically, DT has an internal dialogue like this:

If I choose attribute A and split my subset S , how much will my current entropy, $E(S)$, decrease? Which attribute will give me the largest reduction in entropy $E(S)$?

So, clearly DT needs to measure the entropy before and after a split for all the attributes in the training set and choose the attribute with the highest reduction in entropy. Remember the golden rule:

*Highest Reduction in Entropy \equiv Highest Gain in Certainty \equiv
Highest Information Gain (IG)*

Below we can see the formal definition of Information Gain (IG):



Information Gain

- Measure the reduction in Entropy before and after a split on a subset **S** using the attribute **A**

$$IG(S, A) = E(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} E(S_v)$$

- The more the IG the better then!
- IG can help the Decision tree grow smarter!

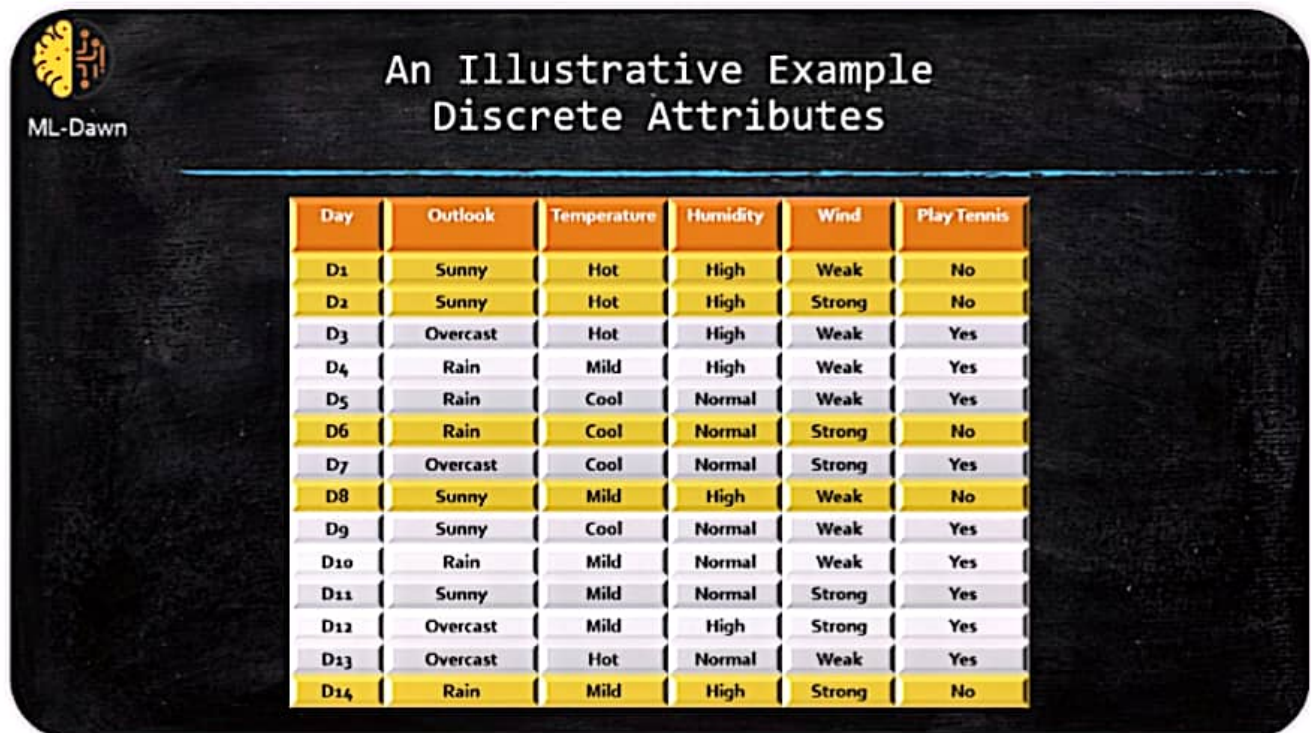
So, let me introduce you to all these little bits and pieces in the formula:

1. $E(S)$: The current entropy on our subset S , before any split
2. $|S|$: The size or the number of instances in S
3. A : An attribute in S that has a given set of values (Let's say it is a discrete attribute)
4. v : Stands for value and represents each value of the attribute A
5. S_v : After splitting S using A , S_v refers to each of the resulted subsets from S , that share the same value in A
6. $E(S_v)$: The entropy of a subset S_v . This should be computed for each value of A (assuming it is a discrete attribute)

Could you tell me why we have that bloody \sum over there? Well it is because the attribute A (let's consider it a discrete attribute with N number of possible values) has N number of values and when we split using A , we need to compute the entropy for every branch corresponding to every value of A . The \sum tries to sort of aggregate these individual entropy values after the split on A across all branches. Finally when we subtract the second term from

$E(S)$, we get the expected reduction in entropy if we choose attribute A for splitting. Cool ha? But we need a nice example to really understand it yes?

Consider the following nice dataset:



An Illustrative Example
Discrete Attributes

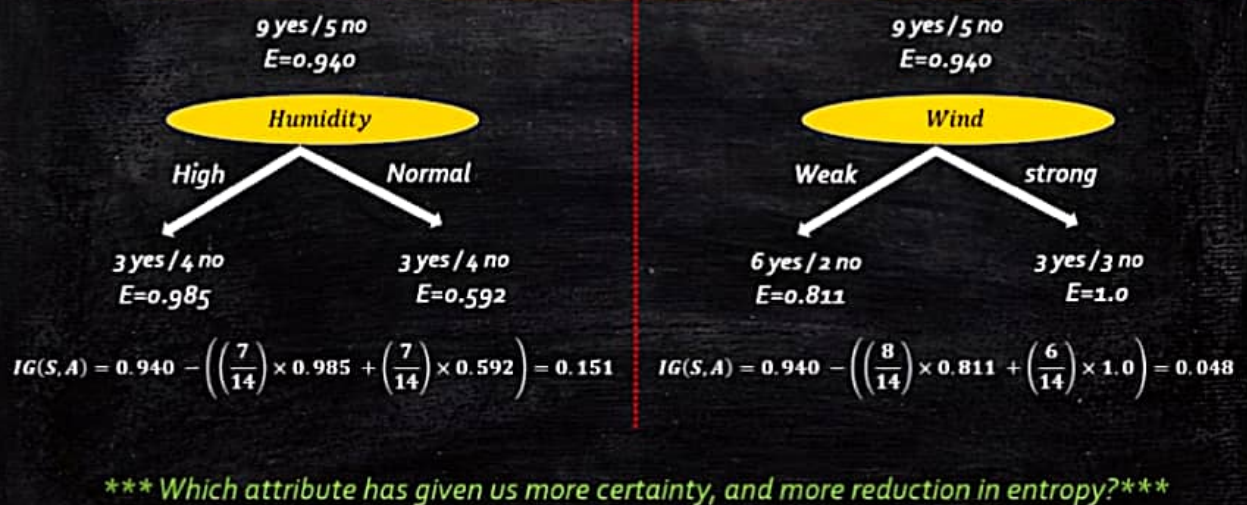
ML-Dawn

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

So we have a few attributes here, and the job is to learn when is Joe going to play tennis depending on the values these attributes can take. You can see that all of them these attributes are discrete and the target feature, Play Tennis, is actually both discrete and binary. The big question is, which attribute should we choose to learn whether Joe would play tennis on a given day? Outlook, Temperature, Humidity, or Wind?

Notice that all of these attributes are discrete but don't worry, I will show you the **big picture** on how to compute IG for the continuous attributes as well. For now, let's compute the IG for 2 of the attributes: **Humidity** and **Wind** and decide which one is a better candidate for splitting:

An Illustrative Example Discrete Attributes

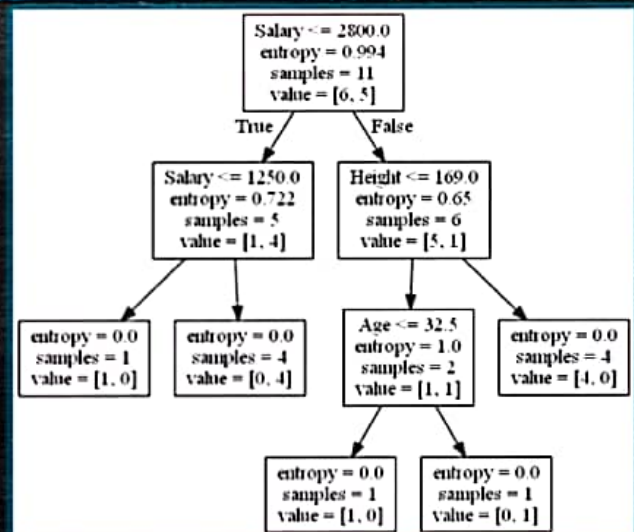


Beautiful! At the start, before the splitting started, we have an entropy of 0.94 which is quite high. Why? Because as I showed you before, in the case of binary classification the maximum of entropy for a given set S is always 1. So, 0.94 is pretty high! In the case of Humidity, the values are: High and Normal. On the left, for the case of value being equal to High, the $|S_v|=7$ (i.e., 3 yes / 4 no). Also, $|S|$ is 14, as that is the number of samples in our subset before any splitting starts. So the question is, which attribute has given us more reduction in the initial entropy $E(S) = 0.940$. We can see that $IG(S, \text{Humidity}) = 0.151$ and it is larger than $IG(S, \text{Wind}) = 0.048$. As a result, the decision tree will choose Humidity over Wind for splitting. Simple, isn't it?

So far all we have done has been on discrete attributes. Let's consider our old dataset that has nothing but continuous attributes. Below, you can see the dataset and the corresponding decision tree, produced by **Scikitlearn** using **Python**.

An Illustrative Example Continuous Attributes

Age	Height	Salary	Single
25	180	2000	Yes
40	170	2500	Yes
18	200	3000	No
30	190	1000	No
50	170	5000	No
40	168	4000	Yes
33	190	3000	No
17	195	1500	Yes
37	160	2600	Yes
25	157	3000	No
27	189	3700	No

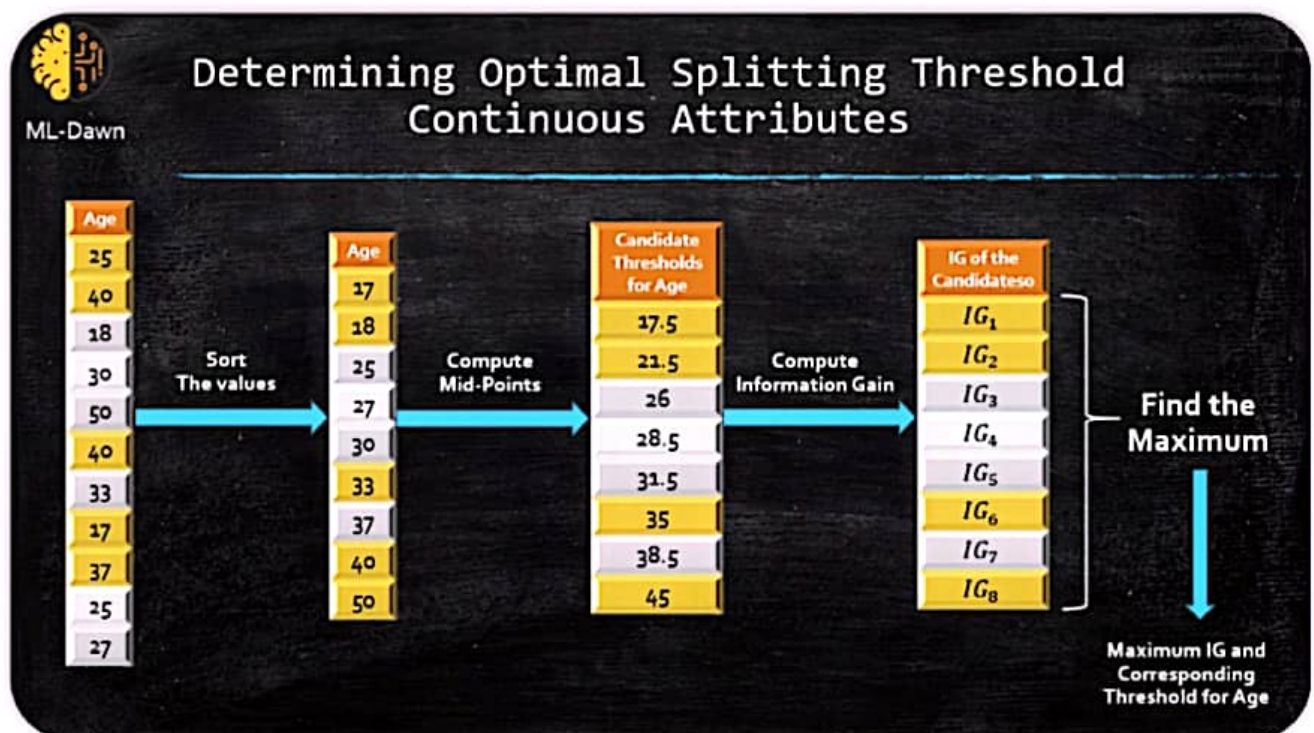


This is all nice and fun but there is a big question here:

How does the DT decide on the threshold for the attributes to measure the reduction in entropy in the first place? For example, in the root of the tree we have a threshold of 2800.0 for Salary. But why?

It is actually quite simple. Let's consider the Age attribute (the DT will do this for all the attributes). The question is: **Which threshold for Age will give me the highest IG, that is, what is the best that Age can do in reducing the current entropy ($E(S) = 0.994$ which is too high)?** Now, the DT first sorts the values of Age, and then considers the mid point for each pair of values. This becomes a vector of candidate threshold for the attribute Age. Now, the DT will measure the IG while splitting on Age using each and every value from the candidate threshold vector, and considers the highest IG and the corresponding candidate

threshold. This will happen for all attributes, and the one with the highest IG with its corresponding threshold will be chosen for splitting. Below you can see the whole story for the attribute Age:



Weighted vs. Unweighted IG!

One of the most important features of Information Gain is how it aggregates the entropy values of multiple children nodes to report the final value for after the split on a given attribute. If you notice, what it does is actually a weighted averaging of the individual entropy values. It is weighted by the size of the subset per branch (S_v/S). This can be contrasted with a regular averaging that we all have learned when in high school, just so we can see why we have that weighted averaging.

For example, in our previous example where we split on Humidity, we got to branches with entropy values of 0.985 and 0.592. The way IG has mixed these two values is by weighting each one by $|S_v|$ over the original size of

the subset before the split $|S| = 14$, that is $7/14=0.5$. Now, why doesn't IG just do a simple averaging and say $(0.985 + 0.592)/2$?



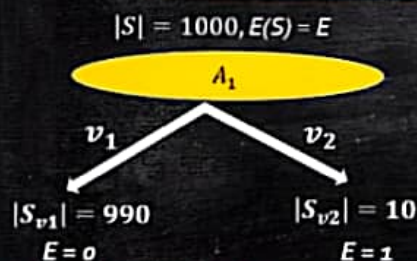
Why Weighted Average in $IG(S, A)$?

- Let's say we have 2 classes in our training set
- Say we are torn between 2 attributes A_1 and A_2
- To make the example easier let's say both attributes are binary and categorical
- The Question is which attribute's split would give us more certainty (more reduction in entropy) \rightarrow More Information Gain?

Now, take a look at 2 different splits and how a weighted and unweighted version of IG will report the best attribute for splitting.



Why Weighted Average in $IG(S, A)$?

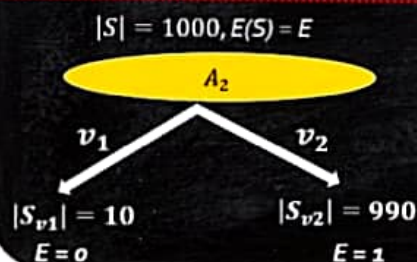


Unweighted Average of Entropies:

$$IG(S, A_1) = E - \left(\frac{0 + 1}{2} \right) = E - 0.50$$

$$IG(S, A_2) = E - \left(\frac{0 + 1}{2} \right) = E - 0.50$$

Suggests A_1 and A_2 are Worth the Same!!!



Weighted Average of Entropies:

$$IG(S, A_1) = E - \left(\left(\frac{990}{1000} \right) \times 0 + \left(\frac{10}{1000} \right) \times 1.0 \right) = E - 0.01$$

$$IG(S, A_2) = E - \left(\left(\frac{10}{1000} \right) \times 0 + \left(\frac{990}{1000} \right) \times 1.0 \right) = E - 0.99$$

Suggests A_1 is worth more!!!

This is just really interesting! You can see that the unweighted version suggests that A1 and A2 are equally valuable, whereas the Weighted version, suggests that the split using A1 is almost 100 times better in terms of how much it reduces the entropy. This makes sense right? We can see this visually that:

The split on A1 has resulted in a HUGE chunk of pure set and a very small subset of non-pure.

However:

The split on A2 gives a very small pure subset and a HUGE impure subset.

It is clear that A1 must be rated better, which is determined correctly using the Weighted averaging that is done in IG. This is why they have bothered with the weighted averaging for IG and didn't just do a simple averaging. Perfect!

So the morals of the story as to why the weighted averaging is used can be summarized in the following:



ML-Dawn

Why Weighted Average in $IG(S, A)$?

Weighting by the
Size of the Sub-Sets
Suggests that



Absolutely perfect! Now you know how a decision tree works. You know what entropy is and how Information Gain utilizes it to decide on the best attribute for splitting, for both discrete and continuous attributes. In the next post, we will talk about pruning and the techniques to avoid over-fitting to the training data.

Until then,

Take care of yourself,

MLDawn

[← Previous Post](#)

[Next Post →](#)

Leave a Comment

Your email address will not be published.