**Decision Tree**

Introduction

Prerequisites: [Binary Search Tree](https://leetcode.com/explore/learn/card/introduction-to-data-structure-binary-search-tree/), [Machine Learning Basics](https://leetcode.com/explore/learn/card/machine-learning-101/)

This card is intended to illustrate a classic algorithm for the classification problems in the domain of machine learning.

By completing this card, you will be able to:

1. Understand the intuition behind decision tree;
2. Implement the algorithm of decision tree;
3. Understand the important metrics (gini impurity, entropy) that involved in the process of decision tree construction;
4. Know how to evaluate the performance of decision tree model and the importance of input features;

Before you start, bear in mind that should you have any questions or comments, you can always pose them in the [Discussion](https://leetcode.com/discuss/explore/decision-tree) forum that is located at the end of the card. We'll do our best to response to you as soon as we can.

Definition

In this chapter, we introduce one of the most popular Machine Learning algorithms, which is called Decision Tree.

By completing this chapter, you shall be able to:

1. Understand the definition of decision tree;
2. Know how to apply the decision tree model for a classification problem;
3. Understand the overall algorithm to construct a decision tree;

**Definition - Decision Tree**

Decision tree is a popular tool that is widely applied in various domains including operations research, data mining and machine learning. The definition of decision tree varies in different domains. In this card, we focus on the form of decision tree that is applied in the domain of machine learning. More specifically, decision tree can be used to solve the problems of [*classification* and *regression*](https://leetcode.com/explore/learn/card/machine-learning-101/287/what_is_ml/1633/) in the subdomain of [*supervised machine learning*](https://leetcode.com/explore/learn/card/machine-learning-101/287/what_is_ml/1620/).

*In this card, we elaborate on the decision tree for****classification****problems. Unless otherwise specified, in the rest of the card, we refer to the decision tree as the one used for classification problems.*

A decision tree for classification is a special form of binary tree, which is used as a classifier. There are two types of nodes in decision tree:

* leaf node: same as the ones in binary tree, *i.e.* the node that does not have any child node.
* decision node: the non-leaf node.

Each leaf node contains a label that is assigned to the objects which fall into this leaf node during the inference phase.

For each decision node in the tree, it contains a branching rule that can be expressed in the following form:

|  |
| --- |
| if (condition == true) {  object goes to the left child node;  } else {  object goes to the right child node;  } |

where the condition is a testing expression based on the value of certain attribute in the object.

* For numerical attributes, the condition takes the form of less-or-equal-than comparison, i.e.  object.attribute≤*C*. For example, object.height≤1.7.

* For categorical attributes, the condition is expressed as membership to a list of categorical values, i.e. object.attribute∈{*C*1​,*C*2​,*C*3​...}. For example, object.color∈{red, green, yellow}.

At each decision node, the branching would be performed according to the predictor, an object would then iteratively be attributed to the nodes along the tree, from top to bottom.

### ***Example***

We will show an example of decision tree for a classification problem. First of all, let us introduce the data set called [Iris](https://archive.ics.uci.edu/ml/datasets/iris), which is first published in the paper of "The use of multiple measurements in taxonomic problems" - Ronald. A. Fisher (1936) **[1]**. The Iris data set consists of measurement for 150 samples of iris flower. Each sample contains the measurement for the length and the width of its petal and sepal, and a 'class' attribute that indicates the category of iris flower, namely setosa, versicolor and virginica. Below we show a few samples from the Iris data set.

Table

Description automatically generated

With the Iris data set, the classification problem that we would like to tack with is to predict the category of iris flower, given a sample with attributes of petal and sepal, i.e. labelling the sample.

Therefore, the desired decision tree model can be defined as the following function F*F*:



This decision tree takes a vector of four real values, and gives a label as output. And here is what a decision tree might look like:

Diagram

Description automatically generated

In the above graph, each node in oval represents a decision node, while each node in box represents a leaf node. As we can see, each decision node in the tree contains a condition to further assign the samples that go through this node. The condition is designed to best split the samples, in a way that all the samples that are assigned to the same child node are more similar to each other, than the samples in the oppose child node. We will discuss more in detail about the criteria how the condition is chosen and calculated in the following articles.

### ***References***

- [1]. Fisher,R.A. "[*The use of multiple measurements in taxonomic problems*](https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x)" Annual Eugenics, 7, Part II, 179-188 (1936)

**Model Inference - Decision Tree**

Now one might wonder how the decision tree works, *i.e.* how we can apply the decision tree to classify a sample. This step is also called model inference in machine learning. Basically, to apply the decision tree, is to traverse the tree from top to down. At each decision node, we follow the corresponding branch with regards to the result of condition testing. When we reach the leaf node of the tree, we take the label of the leaf node as the classification result for the sample.

Let us take the first sample from the above table as an example *X*, as follows:

A picture containing text

Description automatically generated

We then can break down the steps to apply the decision tree that we gave as an example in the [definition article](https://leetcode.com/explore/learn/card/decision-tree/282/definition/1638/) for this particular example *X*, in the below graph where we highlight the path in red and denote each step with number.

Diagram

Description automatically generated

1. Starting from the root node with the condition {pedal\_length<=3.0}, by comparing the corresponding attribute in the sample *X* with the value on the right hand side of the condition, the test result of the condition turns out to be false. Therefore, we move on to its ***right*** child node.
2. Now we are at the decision node with the condition {pedal\_length<=4.8}, similarly by testing the condition, this time we would move on to its ***left*** child node.
3. Next, at the decision node with the condition {pedal\_width<=1.7}, we would then move on to its ***right*** child node which turns out to be a leaf node. As a result, by applying the decision tree, we obtain the classification result for the sample *X* as "versicolor", which is indeed the actual category of the sample.

As we can see, the conjunction of conditions along the path also forms a chain of rules, which explains how the label is chosen. This chain of rules serves naturally as the interpretability of the decision tree model, *i.e.* the reasoning that is comprehensive for humans to understand how the prediction (*i.e.* decision) is made by the machine. This interpretability is one of the advantages of decision tree model, comparing to the ***blackbox*** models such as those neural network based models.

**Algorithm - Decision Tree**

In this article, we explain the algorithm to build a decision tree for classification problems. Here is the overall intuition about the algorithm:

The algorithm to construct a decision tree follows the approach of *divide-and-conquer*, *i.e.* we recursivelly splitting the input samples into two subgroups with decision node, until we no longer need to split them. At the end, each of the samples is assigned to a leaf node, and we label the leaf node with the category of the majority samples within the leaf node.

As one can see, we can implement the decision tree construction algorithm by recursion. Given a list of samples with various labels, to construct a decision tree that could assign the sample with a proper label, here are the base cases and the recurrence relation of the recursive algorithm:

* base cases: If the samples are of the same labels, then we do not need to further split the samples. This is the fundamental base case. One can define more base cases in order to regulate the complexity of the final tree.
* recurrence relation: We find the*most distinguishable* feature of the samples and also the *best* value to split on, in order to obtain two subgroups of samples. We then construct subtrees out of the split subgroups. The criterion to split the samples is twofold: *1).* we should reduce the samples into smaller scales in a fast manner, so that we could reduce the occurrence of recursion, *i.e.* reduce the cost of the algorithm. *2).* we should also make sure the split subgroups are more uniform so that it becomes easier to classify the samples. We will discuss more about the criterion in the following articles.

Here is an example of pseudo code on how to construct a decision tree.

|  |
| --- |
| TreeNode build\_decision\_tree([samples]) {    // base cases:  - the target attributes of the samples are uniform  - the current depth of tree exceeds the max\_tree\_depth  - the number of samples is less than the minimal\_node\_size  // 1). we create a leaf node and return.  if (any of the above cases holds) {  leaf\_node = create\_leaf\_node([samples]);  return leaf\_node;  }    // 2). find the best attribute to split on, (also the best value to split)  feature\_to\_split, split\_value = find\_best\_split([samples]);  // 3). split the samples list into two sublists  left\_samples, right\_samples = split([samples], feature\_to\_split, split\_value);  // 4). create a decision node.  new\_node = create\_node(feature\_to\_split, split\_value);  // 5). for each sublist, recursively call the function to create the subtrees.  new\_node.left = build\_decision\_tree(left\_samples);  new\_node.right = build\_decision\_tree(right\_samples);  // 6). return the newly-created node  return new\_node;  } |

### ***Stopping Conditions***

A decision tree grows from top to bottom. When it stops growing, a leaf node is added. Here are a few conditions when we stop splitting samples :

1. All the examples that fall into the current node belong to the same category, i.e. no further classification is needed.
2. The tree reaches its predefined max\_depth.
3. The number of examples that fall into the current node is less than the predefined minimal\_number\_of\_examples.

The condition **(1)** is a natural and optimal case to stop adding more nodes, since we achieve our initial goal, i.e. there is no more ambiguity when the samples reach the node. While the conditions of **(2)** and **(3)** are more of an intervention to prevent the decision tree from overgrowing itself which leads to the scenario of **overfitting**. This intervention is also known as regularization in machine learning, which is a measure to prevent the model from overfitting.

### ***Further Readings***

- [1]. Breiman, L. (1984). [*CART: Classification and Regression Trees*](https://www.taylorfrancis.com/books/9781351460491). New York: Routledge.

Implementation

In this chapter, we will discuss some implementation details involved in the process to construct a decision tree.

After this chapter, you should be able to:

1. know the intuition of creating a decision node.
2. know how to implement the metrics that can be used to evaluate a split, namely gini impurity and entropy.

**Splitting Criterion**

As one might recall, in the algorithm to construct a decision tree, there is one important step, which is to split the given samples into two subgroups. This splitting of samples is a critical step that serves multiple purposes:

* The splitting reduces the problem into smaller ones, which eventually leads to the termination of the recursion process.
* The splitting criterion, i.e. the feature to split on and the value to split with, serves as the branching condition in the decision node.
* The splitted subgroups would be used as the input samples to further construct the left and right subtrees.

How can we choose the *splitting criterion* then ?

Since there could be numerous candidates, how can we determine which split is the best ?

In this article, we will walk you through the above questions, to give you at least the intuition of how we split the samples.

### ***Example***

We will stick with the [Iris](https://archive.ics.uci.edu/ml/datasets/iris) data set as we introduced at the beginning of this chapter. The original data set contains 150 records. For the purpose of illustration, here we just randomly pick up 4 samples for each specie of Iris flower, and obtain in total 12 samples as following:

Table

Description automatically generated

To see the distribution of the samples, we then plot a graph below with the above samples. On the horizontal dimension, we plot 4 sub-dimensions where each of them corresponds to a measurement (attribute, e.g. {sepal\_length) of a sample. The vertical dimension indicates the value of the corresponding measurement. As a result, each dot on the plot corresponds to a specific measurement of a specific sample. For example, we shall find exactly 12 dots along each sub-dimension. In addition, we also highlight the species of samples with different colors.

Chart, scatter chart

Description automatically generated

As one would recall from [the definition of Decision Tree](https://leetcode.com/explore/learn/card/decision-tree/282/decision-tree/1638/), the branching condition in the decision node for the above data set should take the form of less-or-equal-than comparison, i.e. object.attribute≤*C* e.g.  pedal\_width≤1.5.

If we draw the branching condition in the above plot, it would be a horizontal line that divides the space into two parts: the lower part would correspond to the samples that are assigned to the left child node of the decision node, while the upper part would correspond to the samples that are assigned to the right child node.

### ***Good Split***

Given the above examples, let's try to create our first split, i.e. creating a decision node. As we can observe from the above plot of the samples, the categories of samples are most ***distinguishable*** on the pedal\_length sub-dimension, i.e. there are clear boundaries among the samples of different categories.

Therefore, we can create a split either between the categories of **setosa** and ***versicolor*** or between the categories of ***versicolor*** and **virginica**, as shown in the plot below.

Chart, scatter chart

Description automatically generated

For example, the split line **(1)** corresponds to the branching condition {petal\_length≤2.5}. With this split, we have 4 samples of the same category (***setosa***) grouped in the left child node, which greatly facilitates our task of classification, i.e. no further split is required for these samples. On the other hand, we still have 8 samples assigned in the right child node, which belongs to two different categories (**versicolor** and ***virginica***).

Similarly, we then continue to split the 8 samples with the line **(2)** which corresponds to the branching condition {petal\_length≤5.0}. Again, this is a good split that clearly separate the two categories. With the split, on its left child node, we have the 4 samples of **virgicolor**; while on its right child node, we have 4 samples of **virginica**.

With the above two splits, we can construct a decision tree (shown below) that can precisely classify the given 12 samples.

Diagram

Description automatically generated

### ***Bad Split***

As a comparison to the above example, we will show an example of what a bad split looks like.

Chart, scatter chart

Description automatically generated

As shown in the above plot, this time we choose the attribute of sepal\_with to split on. For the first split, i.e. the line **(1)**, we choose the branching condtion  {*sepal*\_*width*≤2.4}. With this split, we have a single sample on the left child node and the rest 11 samples assigned to the right child node. Compared to the good split above, this time we only manage to classify one sample instead of 4 with the first split.

We then proceed with the 11 samples with a second split. This time, again, we choose deliberately a bad split, i.e. the line **(2)**, with the branching condition as {*sepal*\_*width*≤4.0}. With this second split, we have a single sample on the right child node and the rest 10 samples assigned to the left child node.

With the above two splits, we are still far from classifying all samples, i.e. we still have 10 samples with different categories remained in the same node. We could still proceed to add more decision nodes to the decision tree, as shown below.

Diagram

Description automatically generated

### ***Summary***

With the above examples on the good and bad splits, one should have a better idea of how the split is done. Given a data set with multiple numerical features, in order to find the best split that divides the data set into two subgroups, it boils down to two essential questions:

1. Which *feature* to split on ?
2. Which *value* of the chosen feature to split with ?

Chart

Description automatically generated

To answer the above question, one can enumerate all features and all values in the data set to find the best candidate. Given a candidate of splitting, essentially we need to answer the following question:

How can we evaluate the quality of the splits ?

How can we compare one split to another split? What are the criteria that we can use to quantify a split?

And these are the questions that we will answer in the following articles.

**Gini Impurity**

In the previous article, we conclude that in order to find the best split given a list of labeled samples, it all boils down to evaluate the quality of candidate splits, i.e. we should find certain criteria that allows us to compare different splits.

Chart

Description automatically generated

For example, we take a list of samples from Iris data set, as shown in the above graph where each dot represents a sample and the species (i.e. **setosa**, **versicolor** and **virginica**) of the samples are marked with different colors. In addition, we select one attribute (e.g. {*petal*\_*width*) and then we order the samples along with the selected attribute. As shown in the graph, all the samples are aligned in the coordinate of the selected attribute. So the question is where to split the list?

In this article, we will introduce one of the criteria called gini impurity, which we could apply to evaluate the quality of the splits.

### ***Gini Impurity***

The Gini impurity is ultimately a metric that is intended to measure the **impurity** for a group of values, which is on the contrary of **uniformity** where all values are identical.

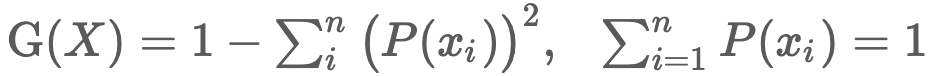
One can interpret the gini impurity as the **probability** for a scenario that one randomly draws two samples from a group and these two samples are of different values.

For example, for a group of uniform values, e.g. [1, 1, 1, 1], it is impossible to draw two samples of different values. As a result, the Gini impurity of the group is zero.

Given a list of samples *X* with {*n*} unique values, we could obtain the Gini impurity of the group with the following formula:



we can also expand the above formula and rewrite it as follows:



where *P*(*x*i​) is the probability of finding a sample with the value *x*i​ in a random sampling. The number of samples is greater and/or equal than *n*, where one can have multiple samples of the same value.

The intuition behind the concept of Gini impurity, as defined by the above formula, can be interpreted as the probability of finding two samples with different labels in a game of random selection with replacement. Here are the procedures that how we can derive the formula with the random selection game:

* Step 1): We randomly pick a sample from the group, then the probability that the sample is of value *x*i​ is *P*(*x*i​). We put the sample back to the group for the next selection, since it is the selection with replacement.
* Step 2): We randomly pick another sample, then the probability that the second picked sample is again of the value *x*i is *P*(*x*i​).
* Step 3): Since the above two steps are independent, the probability that both samples are of value A picture containing text

  Description automatically generated.
* Step 4): As there are {n}*n* different values in the group, the probability that the steps **(1)** and **(2)** appear for each of the values is then Icon

  Description automatically generated, i.e. the probability for the scenario where we randomly draw two samples and they are of the same value.
* Step 5): We then exclude the probability for the event in step **(4)**, from which we can obtain the probability for the desired case where the two randomly selected samples are of different values, i.e. the Gini impurity *G*(*L*) of the group.

For example, we have a group of 4 samples as [versicolor, setosa, setosa, setosa].If we randomly select two samples from the group with replacement, then the probability that the two samples are of different values would be Text

Description automatically generated with low confidence.To calculate the above probability, we can break it down into the following two cases:

* case 1: the first sample that we pick is of value versicolor, which is of 1/4​ chance, and the second sample that we pick is also of value versicolor. Therefore, the probability of this case is ¼ \* ¼​.
* case 2: the first sample that we pick is of value setosa, which is of 3/4​ chance, and the second sample that we pick is also of value setosa. Therefore, the probability of this case is ¾ \* ¾ ​.

By excluding the above two cases, we obtain the probability of the case where two samples are of different values, which is the gini impurity of this group.

### ***Gini Gain***

Now, given the definition and the intuition of Gini impurity, let us dial back to our splitting problem in decision tree to see how one can apply Gini impurity.

In the context of **Decision Tree**, the metric of gini impurity is used to evaluate the quality of a split, where we split a list of samples into two subgroups.

The more uniform for the samples in each group, the easier one can make a decision on how to classify a sample, i.e. the lower the gini impurity for a group, the easier we can assign the right label for the samples in the group.

Following the above example of [versicolor, setosa, setosa, setosa], if we are asked to guess the value for a randomly selected sample, to have a better chance, we could split this group into two subgroups so that the values in each subgroup are more uniform, i.e. to reduce the overall Gini impurity after the splitting. As a result, we reduce the uncertainty of guessing the value.

By enumerating all possible splits, one can find that the optimal split would be the subgroups of [versicolor] and [setosa, setosa, setosa], where each subgroup contains uniform values, i.e. the Gini impurity of each subgroup is then reduced to zero. And we would have 100% certainty to guess the label for samples within each subgroup.

The reduction of the gini impurity is also called "***gini gain***". The quality of the split is measured by gini gain. The higher the gini gain, the better the split.

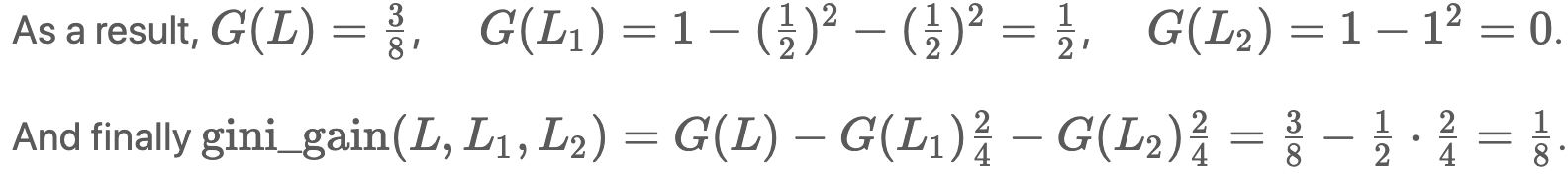
For a group *L*, we divide the group into two subgroups *L*1​,*L*2​, the gini gain of this split is defined as following:



The overall Gini impurity of split subgroups {*L*1​,*L*2​}, is the sum of Gini impurity for each subgroup weighted by its proportion with regards to the original group.

For example, let us apply the Gini gain to measure the quality of two splitting candidates for the group *L* = [versicolor, setosa, setosa, setosa]:

**Candidate 1):**  *L*1​ = [versicolor, setosa],  *L*2​ = [setosa, setosa].



**Candidate 2):** *L*1​ = [versicolor],  *L*2​ = [setosa, setosa, setosa].

Diagram, text

Description automatically generated with medium confidence

As we know that the candidate (2) is a better split, which is indeed confirmed by the Gini gain as shown above.

***Note***: one should not confuse the "Gini impurity" with another coefficient called ["Gini index"](https://en.wikipedia.org/wiki/Gini_coefficient) which is used to measure the **inequality** of values within a group. Some authors use the terms "Gini index" and "Gini impurity" interchangeably. In this Explore card, however, we prefer to highlight their differences.

**Entropy**

In the previous article, we discuss about the metric of Gini impurity to evaluate the quality of a split. In this article, we introduce another concept called entropy from the information theory, which can also be used as the metric for decision tree.

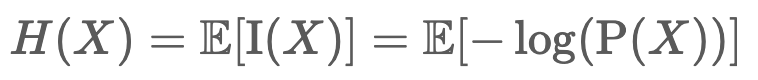
*Entropy*

The basic idea of information theory is that the more one knows about a topic, the less information that one is apt to get about it. If an event is very probable, it is no surprise when it happens and thus provides little new information **[1]**. In information theory,the concept of entropy is first proposed by Shannon **[2]** to establish the limit of data compression.

Entropy can be defined as the ***unpredictability*** of the state. The more probable a state is, the less entropy that it has, *i.e.* the less ***information*** this state contains.

For a variable that has multiple states, its entropy is defined as the average of ***information content*** that each state contains.

For a discrete random variable *X* with the possible values of {*x*1​,...,*xn*​}, Shannon **[2]** defined the entropy of *X* as *H*(*X*), as follows:



where  is the average function, *I*(*X*) is the information content for each state of *X*.

Literally, we can interpret the entropy for the random *X* is the average amount of information that it contains.

The above formula can be further expanded as follows:

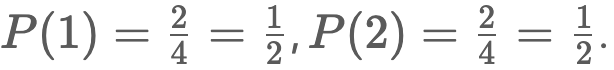
Text

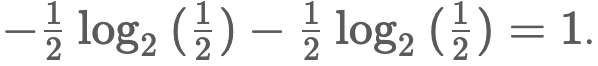
Description automatically generated

Chart, line chart

Description automatically generated

Since the logarithm function is monotone increasing, as one can see from the above graph [[image reference](https://en.wikipedia.org/wiki/Binary_logarithm)], the more likely a state *xi*​ (*i.e.* the bigger *P*(*x*i​)), the less information it contains (*i.e.* the smaller *I*(*x*i​)). For example, if a variable has a constant value, then we can say that it does not contain any information.

Let us look at a concrete example to see how one can calculate the entropy. Given a group of values {1,1,2,2}, we first calculate the probability for each unique vaue as: 

Then, we can further obtain the entropy as: 

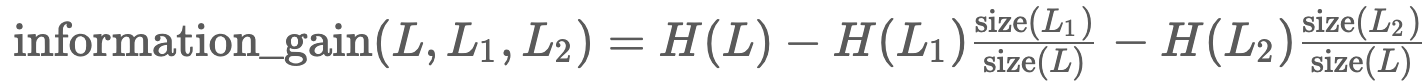
One can also interpret the entropy as the minimal number of bits that are required in order to encode the information contained in a group. In the above example, we can say that we only need one bit to encode all the possible values in the group, *e.g.* we could use the bit value 0 to represent the value 1 in the group, and use the bit value 1 to represent the value 2 in the group.

*Information Gain*

Entropy is a measure of disorder. The higher the entropy, the more disordered a group.

On the other hand, we can say that the more disorder a group is, the more entropy it has, *i.e* the more information it contains.

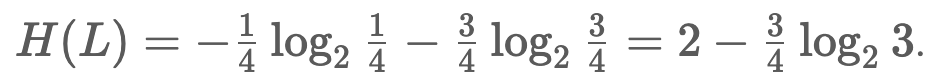
Based on the above insight, we can use the entropy as the splitting criterion for the decision tree. The quality of the split can be measured by the entropy reduction, *i.e.* the reduced entropy between the original group and the splitted subgroups. The entropy reduction is also known as information gain. For a group *L*, we split it into two subgroups {*L*1​,*L*2​}, the information gain of the split is defined as follows:



The overall entropy of the splitted subgroups {*L*1​,*L*2​}, is the sum of the entropy for each subgroup weighted by its proportion with regards to the original group.

For example, let us apply the information gain to measure the quality of two splitting candidates for the group *L* = [versicolor, setosa, setosa, setosa].

First of all, based on the formula of entropy, let us calculate the entropy of the group *L* as follows:



**Candidate 1):**  *L*1​ = [versicolor, setosa],  *L*2​ = [setosa, setosa].

Text

Description automatically generated with low confidence

**Candidate 2):** *L*1​ = [versicolor],  *L*2​ = [setosa, setosa, setosa].

Text

Description automatically generated with medium confidence

As we know from the previous article of Gini impurity, the candidate (2) is a better split, which is again confirmed by the value of information gain as we illustrated above.

*References*

- [1] [Entropy of Information Theory](https://en.wikipedia.org/wiki/Entropy_(information_theory)). Wikipedia.

- [2] [C.E. Shannon](https://en.wikipedia.org/wiki/C.E._Shannon), "[A Mathematical Theory of Communication](http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf)", [*Bell System Technical Journal*](https://en.wikipedia.org/wiki/Bell_System_Technical_Journal), vol. 27, pp. 379–423, 623-656, July, October, 1948

**Entropy VS. Gini Impurity**

As we learn from the previous articles, both [Gini impurity](https://leetcode.com/explore/learn/card/decision-tree/285/construction/2634/) and [entropy](https://leetcode.com/explore/learn/card/decision-tree/285/construction/2635/) can provide a measurement of disorder on a group of values. Indeed, they are often offerred as the options of splitting criterion, for many libraries that implement the algorithm of decision tree, *e.g.* [scikit-learn](https://scikit-learn.org/stable/modules/tree.html). One then might wonder what are the differences between these two metrics. In this article, we will further look into that.

First of all, as a reminder, we list the formulas for the Gini impurity *G*(*X*) and entropy *H*(*X*) as following:

Text

Description automatically generated

One can see the resemblance between the two metrics from the above formulas.

To see the differences, here we can take a group with two unique values as an example, we plot how each of the metrics evolves in the following *Figure (1)*, when we change the probabilities of the two values in the group.

Venn diagram

Description automatically generated

*Figure (1). gini impurity and entropy of a group of two values*

In the above chart, the X axis represents the probability *P* for one of the two values in the group. Accordingly, the probability of the other value would be (1−*P*). And the Y axis represents the entropy / Gini impurity of the group, given the probabilities of two values respectively.

* As we can see, they are both of bell shape, *e.g.* both metrics reach their maximum when the two values are equally likely to appear (*P*=50%), in which case we can say the group is the most chaotic, since we have the least certainty to determine the value of a random sample.
* The entropy metric has a **larger** **scale** than the Gini impurity, *i.e.* at each point of probability, the entropy value of the group is bigger than its Gini impurity.
* The entropy metric provides a **sharper** contrast between a chaotic group and a less chaotic one, as we can see that the curve of entropy is steeper than the Gini impurity one.
* Both metrics are capable to measure the quality of splits. Yet, entropy is a bit more **expensive** to calculate, in terms of computing.

To summarize, in general, there is no fundamental difference between entropy and Gini impurity. They are both suitable to measure the quality of split during the process of decision tree construction.

At the end of this chapter, one can find some exercises to calculate the entropy and the information gain. Should you have any questions or comments, feel free to post it in the [Discussion](https://leetcode.com/discuss/explore/decision-tree?currentPage=1&orderBy=recent_activity&query=) forum of this card. We will try our best to respond to you.

**Calculate Entropy**

Given a group of values, the entropy of the group is defined as the formula as following:



where P(x) is the probability of appearance for the value x.

The exercise is to calculate the entropy of a group. Here is one example.

the input group:  [1, 1, 2, 2]

the probability of value 1 is  2/4 = 1/2  
the probability of value 2 is  2/4 = 1/2

As a result, its entropy can be obtained by:  - (1/2) \* log2(1/2) - (1/2) \* log2(1/2) = 1/2 + 1/2 = 1

**Note**: the precision of result would remain within 1e-6.

**Calculate Maximum Information Gain**

### ***Definitions***

Given a group of values, the entropy of the group is defined as the formula as following:



where P(x) is the probability of appearance for the value x.

e.g.

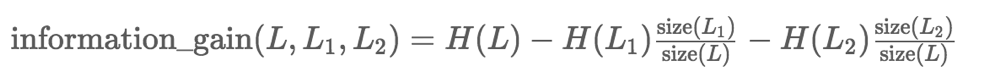
the input group:  [1, 1, 2, 2]

the probability of value 1 is:  2/4 = 1/2  
the probability of value 2 is:  2/4 = 1/2

Therefore, its entropy can be obtained by:  - (1/2) \* log2(1/2) - (1/2) \* log2(1/2) = 1/2 + 1/2 = 1

This exercise, however, is aimed to calculate the maximum information gain that one can obtain by splitting a group into two subgroups. The information gain is the difference of entropy before and after the splitting.

For a group of L, we divide it into subgroups of {L1, L2}, then the information gain is calculated as following:



The overall entropy of the splitted subgroups {L1, L2} is the sum of entropy for each subgroup weighted by its proportion with regards to the original group.

### ***Problem Description***

In this exercise, one can expect a list of samples on Iris flowers. Each sample is represented with a tuple of two values: <petal\_length, species>, where the first attribute is the measurement on the length of the petal for the sample, and the second attribute indicates the species of sample. Here is an example.

Table

Description automatically generated

The task is to split the sample list into two sublists, while complying with the following two conditions:

1. The petal\_length of sample in one sublist is less or equal than that of any sample in the other sublist.
2. The information gain on the species attribute of the sublists is **maximal** among all possible splits.

As output, one should just return the information gain.

In addition, one can expect that each value of petal\_length is unique.

In the above example, the optimal split would be L1 = [(0.5, 'setosa'), (1.0, 'setosa')] and L2 = [(1.5, 'versicolor'), (2.3, 'versicolor')]. According the above formulas, the maximum information gain for this example would be 1.0.

**Note:**  For certain languages (e.g. Java), there is no build-in type of tuple. As a reuslt, to make the input more general, we decompose the input into two lists: [petal\_length] [species] respectively, where the petal\_length would be of float value and the species is of string. The elements in the petal\_length list and species list are associated to each other one by one by order.

## Evaluation

In this chapter, we will talk about how to evaluate the performance of a machine learning model, in particular the decision tree model.

Also, we will discuss the concept of feature importance, which is to evaluate the contribution of a feature in the prediction results of a model.

**Precision VS. Recall**

In this article, we will talk about two important metrics namely Precision and Recall, which are used to measure the performance of classification models (i.e. classifiers). In particular, we will discuss how to evaluate the decision tree model with these two metrics.

In general, the metric of Precision addresses to the question of "How many selected items are ***relevant***?" and the metric of Recall to the question of "How many ***relevant*** items are selected?"

We will walk you through the definitions and some examples so that you would make sense of the above statement at the end of this article.

### ***Definitions***

Before we define Precision and Recall, first we need to clarify a few notions.

Suppose we have a classifier to tell if a picture contains cat or not, the target label (class) has two values: [cat, non-cat]. The classifier would output two possible values as well. For example, given a group of labeled pictures, we apply the classifier to predict a label for each picture. As shown in the table below, there would be 4 possible cases, according to the actual label of the picture and the predicted label. The table is also knowns as **Confusion Matrix [1]** in many literatures.

Table

Description automatically generated

Since the goal of the classifier is to predict whether the picture contains a cat or not, when the classifier gives the prediction as 'cat', we then call the result as **Positive** and we call the prediction result of 'non-cat' as **Negative**. We elaborate the 4 cases in the above table as follows:

***True Positive (Tp​)***

- For a picture, if the predicted class is positive (i.e. cat) and the actual class of the picture happens to be positive as well, we then call this case as True Positive.

***True Negative (Tn​)***

- For a picture, if the predicated class is negative (i.e. non-cat) and the actual class happens to be negative as well, we then call this case as True Negative.

***False Positive ({Fp​)***

- For a picture, if the predicted class is positive (i.e. cat), but the actual class of the picture is negative (i.e. non-cat), we then call this case as False Positive.

***False Negative ({Fn​)***

- For a picture, if the predicated class is negative (i.e. non-cat), but the actual class of the picture is positive (i.e. cat), we then call this case as False Negative.

Given the above definitions, we now can define the metrics of Precision and Recall.

Precision (*P*) is defined as the ratio between the number of true positives (*Tp*​) and the number of all positive prediction (*Tp*​+*Fp*​), i.e. the number of true positives plus the number of false positives.

A picture containing text, clock

Description automatically generated

We can interpret the Precision metric as the **certainty** when the classifier claims the sample as positive. For instance, for a classifier, if *Tp*​=*Fp*​=50, then its precision *P*=50/(50+50)​=0.5, i.e. we can say that whenever the classifier claims that the result is positive, there is only 50% of chance that the classifier is actually right.

If we consider the actual positive items (samples) as "**relevant**" and the claimed positive items as "**selected**", then the precision metric answers the question that how many selected items are relevant, as stated at the beginning of the article.

Recall (*R*) is defined as the ratio between the number of true positives (*Tp*​) and the number of all positive samples ({*Tp*​+*Fn*​), i.e. the number of true positives plus the number of false negatives.

A picture containing text, clock, gauge

Description automatically generated

We can interpret the Recall metric as the **percentage** of how many of those actual positive cases are identified by the classifier. For instance, for a classifier, if *Tp*​=*Fn*​=50, then its recall R = 50/(50 + 50) = 0.5, i.e. we can say that the classifier only captures 50% of the actual positive cases while misclassifying the other 50% of the actual positive cases.

### ***Example***

To illustrate the calculation of precision and recall, we will walk you through a detailed example in the following. First of all, here is a list of samples, where each sample has a **target** label and accordingly a ***prediction*** given by a decision tree model.

Table

Description automatically generated

By applying the above formulas, we can obtain the precision and recall for each label, as following:

Table

Description automatically generated

We take the label "setosa" as an example of a detailed breakdown. There are in total 4 actual positive samples (from row 0 to row 3) for the label of "setosa", and the model gives 3 positive predictions (i.e. at row 2, 5, 9). There is only one ***True Positives*** for the label of "setosa", which is located at the row 2. The **False Positives** for setosa are located in row 5 and 9. Finally, the False Negatives of setosa are 3 cases which are located at row 0, 1, 3.

### ***Accuracy***

Other than Precision-Recall, there is another well-known metric called accuracy which is used to measure the performance of classification models.

The accuracy (*A*) is defined as the proportion of true results (including true positives (*Tp*​) and true negatives (*Tn*​) with regards to all the predictions (*Tp*​+*Tn*​+*Fp*​+*Fn*​).

A picture containing text, object, clock

Description automatically generated

Comparing to Precision-Recall, the accuracy seems to be a more balanced metric, since it takes into account both True Positives and True Negatives. However, it turns out that accuracy is actually a misleading metric, especially for the imbalanced datasets. For example, for a data set with 5 spam emails (i.e. positive samples) and 95 normal emails (i.e. negative samples), a naive spam classifier that simply predicts all samples as negatives (non-spam) would score a high accuracy of 95%. While measuring with Precision-Recall metrics, the spam classifier has zero precision and recall, which tells more accurately the actual predicting power of the classifier. As a result, in practice people prefer the Precision-Recall metric over the accuracy as the measurement to benchmark their classifiers.

***References.*** [1]. [Confusion Matrix](https://en.wikipedia.org/wiki/Confusion_matrix). Wikipedia.

**Feature Importance - Decision Tree**

In this article, we introduce the concept of feature importance and in particular we explain the method to calculate the feature importance for decision tree model.

Decision tree model is one of the white-box models, whose prediction results can be interpreted by humans. We call this property of machine learning model as interpretability, which is not necessarily possessed by all machine learning models.

As part of the interpretability property, feature importance is a metric that one uses to measure the contribution of each input feature to the prediction results of model, i.e. how a minor change on certain feature could alter the prediction result.

### ***Intuition***

Unlike Gini impurity or entropy, there is no universal mathematical formula that defines the feature importance, which varies from model to model.

For instance, for the Linear Regression models, if we assume that all input features are of the same scale (e.g. [0,1]), then the feature importance of each feature is **the absolute value of the weight** that is associated with the feature. As one can see from the formula of the Linear Regression model A picture containing text

Description automatically generated, the outcome of the model is linearly proportional to each of the components (*wi*​⋅*xi*​) which is determined by the weight (*wi*​) of the component.

For decision tree, to measure the feature importance, we need to look into the model to see how each feature plays a role in the final "decision" of the model. As we have learned from the previous chapters, within the decision tree model, at each decision node, we choose the **best feature** to split on, in order to further distinguish the samples that reach this decision node. At each split, we are one step closer to the final decision (i.e. the leaf node). Therefore, we can say that at each decision node, the chosen splitting feature contributes to the final prediction result. Intuitively, we can also say that those chosen features are more ***important*** than those non-chosen features which actually play no role in the decision process. Now, the remaining question is how we can quantitively measure this importance.

As one might recall, we use the metrics of [information gain](https://leetcode.com/explore/learn/card/decision-tree/285/construction/2635/) or [Gini gain](https://leetcode.com/explore/learn/card/decision-tree/285/construction/2634/) to measure the quality of a split. Naturally, one can also associate the gain with the chosen feature, and use the gain to quantify the contribution of the feature at this particular occurrence of splitting. Furthermore, we can accumulate the gain for each feature that appears in the decision tree.

At the end, this accumulated gain for each feature can serve as the feature importance for the decision tree model.

On the other hand, as one might notice, the decision nodes are not equally important, since the decision node at the root of the tree helps to filter all of the input samples, while the decision nodes at the bottom of the tree helps to distinguish only a few of total samples. Therefore, the gain that a feature obtains at each decision node does not carry the same weight, i.e. the gain that a feature obtains at a decision node should be weighted by the proportion of samples that the decision node helps to distinguish.

Given the above intuition, one can derive the following formula that calculates the importance *I* for each feature in decision tree:



***Note:*** one can replace the Gini gain metric with the information gain in the above formula, as long as we stick with the same metric for all features.

With the above formula, one can obtain a value to measure the feature importance for each feature involved in a decision tree. Sometimes, one might want to normalize the values so that one can compare the values more intuitively, i.e. scaling the all values into the range of (0,1). For example, if there are two features that score the same value (i.e. 0.5) after the normalization, we can say that they are evenly important in the decision tree.

### ***Example***

Let us look at a concrete example to see how we can apply the above formula to calculate the feature importance in decision tree. First of all, we show a sample decision tree in the following figure.

Diagram

Description automatically generated

As we can see from the figure, there are in total 3 decision nodes in the tree. Within each decision node, we indicate three pieces of information:

1. the chosen feature to split on.
2. the Gini gain that the feature obtains.
3. the number of samples that are assigned to the left and right child node respectively.

In addition, we can tell that the decision tree is trained on 100 samples in total.

As a result, we can calculate the feature importance of two features involved in the tree as following:

Graphical user interface, text

Description automatically generated with medium confidence