Information Gain

Before learning about Information gain, we need to know an important concept which is **"Entropy"**.

Entropy:

In simple terms, Entropy is the measure of impurity of an attribute when used as a decision node. This can easily be defined with the help of an example:
Let there are 10 friends going out and want to play a game. Now 4 of them want to play "Cricket" and 6 want to play "Football". This is a difficult situation to choose which game to play. Now on the other hand if 2 people want to play "Cricket" and 8 want to play "Football" then it can easily be decided. The entropy can be calculated by using the formula:

Entropy(s) = -[P(yes)*$\log_2$*P(yes)] - [P(no)*$\log_2$*P(no)]

The use of Entropy in decision trees can be described with the help of this example:
Entropy basically measures the impurity of a node. Impurity is the degree of randomness; it tells how random our data is. A pure sub-split means that either you should be getting "yes", or you should be getting "no".
Suppose a feature has 8 "yes" and 4 "no" initially, after the first split the left node gets 5 'yes' and 2 'no' whereas the right node gets 3 'yes' and 2 'no'.
We see here the split is not pure, why? Because we can still see some negative classes in both the nodes. In order to make a decision tree, we need to calculate the impurity of each split, and when the purity is 100%, we make it a leaf node.

Note: The higher the entropy, the lower the purity and higher the impurity.

Implementation of Entropy Calculation

```
import numpy as np
import math


def calc_entropy(column):
    """
    Calculate entropy given a pandas series, list, or numpy array.
    """
    # Compute the counts of each unique value in the column
```

```
    counts = np.bincount(column)
    # Divide by the total column length to get a probability
    probabilities = counts / len(column)

    # Initialize the entropy to 0
    entropy = 0
    # Loop through the probabilities, and add each one to the total
entropy
    for prob in probabilities:
        if prob > 0:
            # use a log from math and set base to 2
            entropy += prob * math.log(prob, 2)

    return -entropy
```

The goal of machine learning is to decrease the uncertainty or impurity in the dataset, hereby using the entropy we are getting the impurity of a particular node, we don't know if the parent entropy or the entropy of a particular node has decreased or not.

For this, we bring a new metric called "Information gain" which tells us how much the parent entropy has decreased after splitting it with some feature.

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

  Information Gain = Entropy(S) - [(Weighted Avg * Entropy(each feature)]

  Where,
  S = Total number of samples
  P(yes) = probability of yes
  P(no) = probability of no

Implementation of Information Gain Calculation

```
def calc_information_gain(data, split_name, target_name):
    """
    Calculate information gain given a data set, column to split
on, and target
    """
    # Calculate the original entropy
    original_entropy = calc_entropy(data[target_name])

    # Find the unique values in the column
    values = data[split_name].unique()

    # Make two subsets of the data, based on the unique values
    left_split = data[data[split_name] == values[0]]
    right_split = data[data[split_name] == values[1]]

    # Loop through the splits and calculate the subset entropies
    to_subtract = 0
    for subset in [left_split, right_split]:
        prob = (subset.shape[0] / data.shape[0])
        to_subtract += prob * calc_entropy(subset[target_name])

    # Return information gain
    return original_entropy - to_subtract
```

ID3 Algorithm

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.

Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.

Most generally ID3 is only used for classification problems with nominal features only. The ID3 algorithm selects the best feature at each step while building a Decision tree. Before you ask, the answer to the question: 'How does ID3 select the best feature?' is that ID3 uses **Information Gain** or just **Gain** to find the best feature.