Decision Trees

Entropy/ Information Gain

Gini index

How decision tree work for regression and classification use cases?

When to use decision trees?

Decision tree vs. Regression Model

Decision Trees

Decision trees are not mathematical models.

Subset of data are created based on if, else conditions and training happen by understanding these patterns. Root node for every subset of data is decided based on IG or Gini index. Splitting of the nodes continue till pure nodes are obtained or stopped further splitting.

Basic concepts before reading about decision trees

▼ Entropy

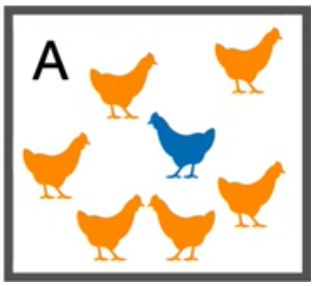
Entropy quantifies the amount of uncertainty(or surprise) involved in values of a random variable or the outcome of the random process. In decision tree it allows us to estimate impurity or heterogeneity of target variable.

Entropy is used for classification. Entropy is also the basis of something called Mutual information which quantifies the relationship between two things. Entropy is the basis of **Relative Entropy(**aka **The kullback-Leibler distance)** and **Cross Entropy** which show up all over the place, including fancy dimension reduction algorithms like t-SNE(Distributed Stochastic Neighborhood Estimate) and **UMAP**(Uniform Manifold Approximation and Projection).

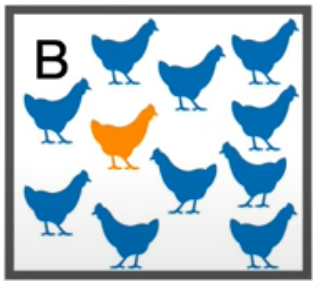
One thing common in above all three things is Entropy or something derived from it to quantify similarities and differences.

To understand more about entropy: Suppose we have three different areas:

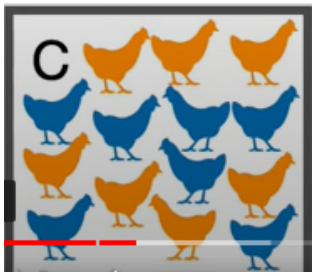
Area1: Has one blue and 6 orange chickens.



Area2: Has one orange and 10 blue chickens



Area3: Has 7 orange and 7 blue chickens



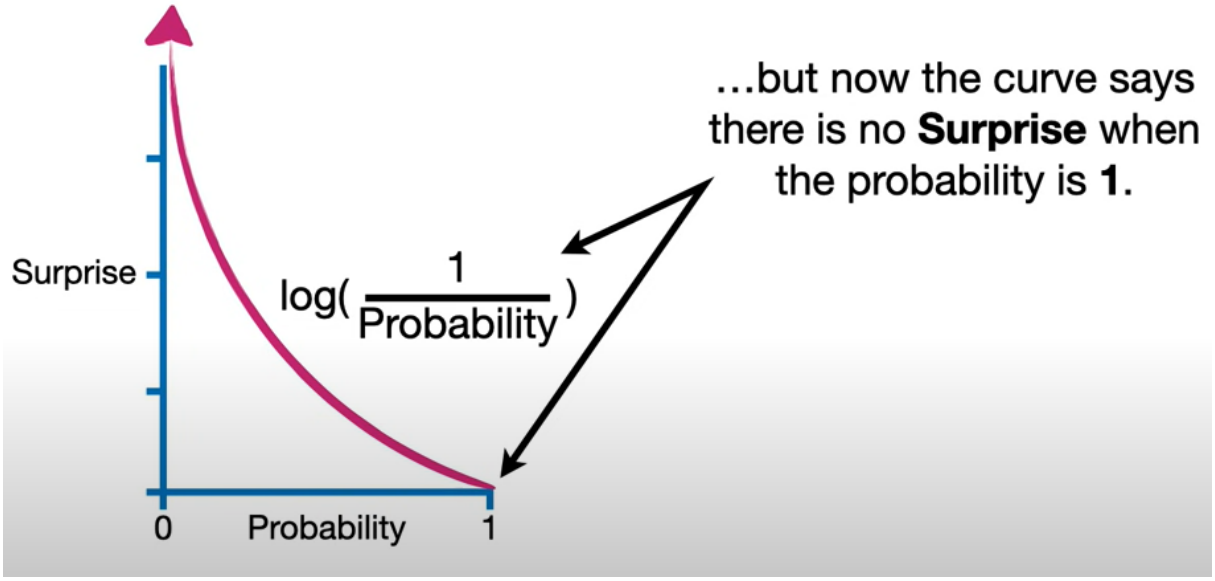
If someone picks up orange chicken in first attempt from Area1, it will not be surprising to get orange chicken as the probability of getting orange chicken is higher. Higher the probability less the surprise. Same with blue chicken from Area2. But for either of the chickens there will be moderate surprise from Area3. We can say surprise is inversely proportion to probability.

Can we write surprise = 1/Probability

No because of the edge cases like suppose every flip of a particular coin gets head so probability of getting head is 1. In this case in next flip getting again will not be surprising but with above formulae surprise = 1/1 = 1 instead of 0.

To avoid such cases we will take log of class <no of target variables>(if target classes are two then log of class 2. If target classes are more than 2 log with taken for that many classes like for three target classes it will be log3 and so on.

Surprise = log(1/P(heads)) = log(1) - log(P(heads)) = 0 (log(1) = 0 and P(heads) = 1, so log(P(heads)) = 0). In this case getting heads in next flips will never surprise us.



$Surprise = log_2(1/Probability)$ - log_2 if output classes are two otherwise it has to be number of output classes

$E(Surprise) = \sum p(x) * log(1/p(x))$

$E(surprise) = \sum p(x)[log(1) - log(p(x))]$

$E(surprise) = -\sum p(x)log(p(x))$, as log(1) = 0

Lets calculate Entropy of Area1:

Probability of picking orange chicken = 6/7

Probability of picking blue chicken = 1/7

E(surprise) = (6/7 * log(1/6/7)) + (1/7 * log(1/1/7))

E(surprise) = (0.86 * **0.22**) + (0.14 * 2.81)

E(surprise) = **0.59**

Entropy is more aligned with entropy of picking orange chicken in Area1

Similarly Entropy in Area2:

Probability of picking orange chicken = 1/11

Probability of picking blue chicken = 10/11

E(surprise) = (10/11 * log(1/10/11)) + (1/11 * log(1/1/11))

E(surprise) = (0.91 * **0.14**) + (0.09 * 3.46)

E(surprise) = **0.44**

Entropy is more aligned with entropy of picking blue chicken in Area2

Entropy in Area2 is less than Entropy in Area1 which makes sense because Area2 has higher probability with lower surprise(0.91 * 0.14) for picking up a chicken.

Entropy in Area3 is 1 because number of orange and blue chickens are same. It will never be outweighed by smaller value of surprise as we saw in earlier two Areas

As a result we can use Entropy to quantify similarities and difference in number of orange and blue chickens in an Area.

As Entropy decreases means the probability of a particular class is high with very less surprise and eventually the Entropy decreases. If it is one that means all classes have same number of values.

Entropy is highest when we have both types of chickens in equal numbers in Area3. As we increases the differences in number of orange and blue chickens we get lower Entropy.

Information gain

$Information\ gain = 1 - Entropy$

The more decrease in information gain, more increase in Entropy.

Entropy quantifies the amount of uncertainty (or surprise) involved in the value of a random variable or the outcome of a random process. Its significance in the decision tree is that it allows us to estimate the impurity or heterogeneity of the target variable

Column with more information gain should be taken as root.

Information gain more → entropy less → probability of a particular class is high → it should be root class.

Gini index

- Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
- It means an attribute with lower Gini index should be preferred.
- Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value.
- The Formula for the calculation of the of the Gini Index is given below

  $Gini\ index = 1 - \sum_i p_i^2$, where i is number of classes * 2

  Suppose in below example:

Index	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	1.2	positive
9	7	3.2	4.7	1.4	negative

16	4.9	2.4	3.3	1	negative

Lets just look at total data for A :

A ≥ 5 = 12 rows out of which positive rows = 5 and negative rows = 7

A < 5 = 4 rows out of which positive rows = 3 and negative rows = 1

Gini index for A ≥ 5 = $1 - [(5/12)^2 + (7/12)^2] = 0.4860$

Gini index for A < 5 = $1 - [(3/4)^2 + (1/4)^2] = 0.375$

By adding weight and sum each of the gini indices

$gini(Target, A) = 12/16 * 0.4860 + 4/16 * 0.375 = 0.45825$

Similarly we will calculate gini for other columns and **the column with lowest gini will be root node of the tree.**

How decision trees work?

Decision trees work by making subsets of data based on some condition. This condition is quantified by calculating Information gain or Gini index in classification decision trees. The column with highest information gain or lowest gini index is considered as root node.

After that again based taking the target value data is divided and again IG or GI is computed to get the next node condition.

In Regression decision trees after fitting regression line, SSME(Square of standard mean error/residual) is computed for all data points. The data point for which SSME is least is considered as root node.

Regression and classification based decision trees.

Regression Decision Tree

Regression decision trees root node is decided by calculating Sum of square residuals(SME). The value where SME is least should be taken as root of the tree.

When to use decision trees?

Decision trees does not need much data cleaning. Decision trees has little impact by outliers, the branch which contains outlier the further nodes will be impacted by the outlier????

Decision trees has high variance. High variance can be controlled by pruning. Two types of pruning:

Pre Pruning

Post Pruning

Pruning can be achieved by specifying:

Depth of tree - suppose tree should stop after 3 levels

Define maximum number of leaf nodes needed

Stop tree after attaining a certain accuracy (Stop criteria)

Decision tree vs Regression Models

Decision trees need not to know the relation between features and target variables. But, in regression model it is necessary to know the impact of variable over target variable. The relation between variable and target variable should be known.

Decision trees and not mathematical models, regression models are mathematical model(fit the mathematical equation to the model). In regression models we can add new features, calculate p value for analysis or to check impact of variable on target value, remove few features etc. But in decision trees no such thing are done.

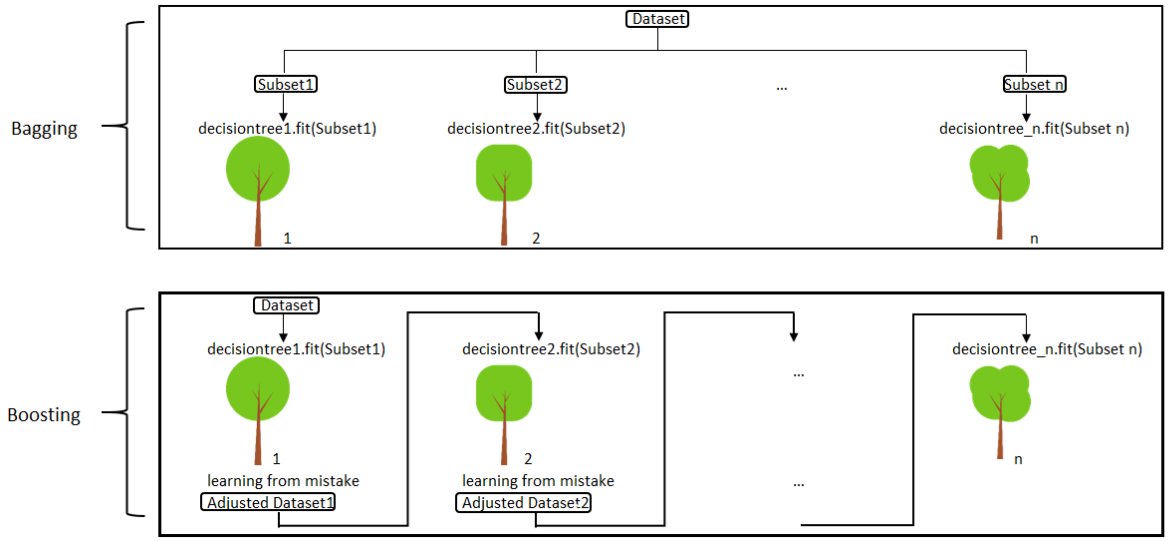
Decision trees are high variance models and regression trees are high bias models.

For controlling high variance we use ensembling as well.

Ensemble Learning

Ensemble learning is a model that makes predictions based on a number of different models. By a combining a number of different models, an ensemble learning tends to be more flexible (less bias) and less data sensitive (less variance). The two most popular ensemble learning methods are bagging and boosting.

1. **Bagging** : Training a bunch of models in parallel way. Each model learns from a random subset of the data.
2. **Boosting** : Training a bunch of models sequentially. Each model learns from the mistakes of the previous model.The stress is on developing new weak learners to handle the remaining difficult observations at each step. One of the very first boosting algorithms developed was Adaboost. Gradient boosting improvised upon some of the features of Adaboost to create a stronger and more efficient algorithm.



AdaBoost

Adaboost used decision stumps as weak learners. Decision stumps are decision trees with only a single split. It also attached weights to observations, **adding more weight to difficult to classify instances and less weight to easy to classify instances**. The aim was to put stress on the difficult to classify instances for every new weak learner. **The final result was average of weighted outputs from all individual learners.** The weights associated with outputs were proportional to their accuracy.

Gradient Boosting Decision Trees

Gradient Boosting Decision Trees are slightly different than Adaboost. Instead of using the weighted average of individual outputs as the final outputs, it uses a loss function to minimize loss and converge upon a final output value. The loss function optimization is done using gradient descent, and hence the name gradient boosting.

Gradient boosting uses short, less-complex decision trees instead of decision stumps. Existing trees in the model do not change when a new tree is added.

The weak learners are fit in such a way that each new learner fits into the residuals of the previous step so as the model improves. The final model aggregates the result of each step and thus a strong learner is achieved.

A loss function is used to detect the residuals. For instance, mean squared error (MSE) can be used for a regression task and logarithmic loss (log loss) can be used for classification tasks.

HYPERPARAMETERS:

Learning rate and n_estimators - Learning rate denoted by α, shows how fast or slow model learns. Each tree added modifies the overall model. The magnitude of the modification is controlled by learning rate. Lower learning rate means model will learn slow. The advantage of slower learning rate is that the model becomes more robust and efficient. In statistical learning, models that learn slowly perform better, but yes with extra cost of computing and time for training model.

More time in learning brings in another parameter n_estimator. n_estimator is the number of trees used in the model. If the learning rate is low, we need more trees to train the model. However, we need to be very careful at selecting the number of trees. It creates a high risk of overfitting to use too many trees.

Note: Gradient boosting decision trees can have overfitting if number of trees are more, but random forest does not have a risk of overfitting. RF will stop increasing the performance but will not overfit.

Improving performance of gradient boosted decision trees

Some pointers you can keep in mind to improve the performance of gradient boosting algorithm and avoid overfitting.

1. Stochastic Gradient Boosting

Stochastic gradient boosting involves sub sampling the training dataset and training individual learners on random samples created by this sub sampling. This reduces the correlation between results from individual learners and combining results with low correlation provides us with a better overall result.

A few variants of stochastic boosting that can be used:

* Sub sample rows before creating each tree.

* Sub sample columns before creating each tree.

* Sub sample columns before considering each split.

2. Shrinkage

The predictions of each tree are added together sequentially. Instead, the contribution of each tree to this sum can be weighted to slow down the learning by the algorithm. This weighting is called a shrinkage or a learning rate. Using a low learning rate can dramatically improve the performance of your gradient boosting model. Usually a learning rate in the range of 0.1 to 0.3 gives the best results. Keep in mind that a low learning rate can significantly drive up the training time, as your model will require more number of iterations to converge to a final loss value.

3. Regularization

L1 and L2 regularization penalties can be implemented on leaf weight values to slow down learning and prevent overfitting. Gradient tree boosting implementations often also use regularization by limiting the minimum number of observations in trees' terminal nodes

Tree Constraints

There are a number of ways in which a tree can be constrained to improve performance.

Number of trees : Adding excessive number of trees can lead to overfitting, so it is important to stop at the point where the loss value converges.

Tree depth : Shorter trees are preferred over more complex trees. Limit the number of levels in a tree to 4-8.

Minimum improvement in loss : If you have gone over the process of decision making in decision trees, you will know that there is a loss associated at each level of a decision tree. A minimum improvement in loss required to build a new level in a tree can be pre decided. This helps in pruning the tree to keep it short.

Number of observations per split :This imposes a minimum constraint on the amount of training data at a training node before a split can be considered

The most notable types of decision tree algorithms are:-

1. **Iterative Dichotomiser 3 (ID3):** This algorithm uses Information Gain to decide which attribute is to be used classify the current subset of the data. For each level of the tree, information gain is calculated for the remaining data recursively.

2. **C4.5:** This algorithm is the successor of the ID3 algorithm. This algorithm uses either Information gain or Gain ratio to decide upon the classifying attribute. It is a direct improvement from the ID3 algorithm as it can handle both continuous and missing attribute values.

3. **Classification and Regression Tree(CART):** It is a dynamic learning algorithm which can produce a regression tree as well as a classification tree depending upon the dependent variable.

Problem with decision trees is that they easily overfit.

References:

1. Entropy
2. Information gain and gini index
3. Classification Decision trees
4. Regression Decision Trees
5. Gradient Boosting Decision Trees
6. Gradient Tree Boosting – scikit-learn documentation (Read it again)
7. Leaf wise decision tree - LGBM Features