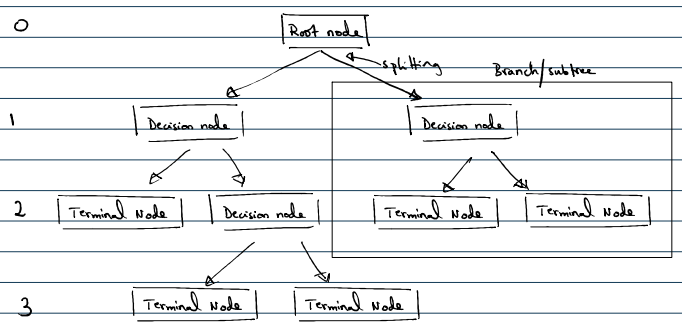
Algorithm Working

Thursday 2 May 2024 10:02 PM

→ Non-Parametric model

→ gaint Nested if-else structure → DT

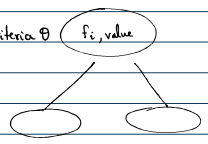
Terminology



CART (classification and Regression Trees) (Binary trees)

→ Algorithm to grow decision tree

→ Each node has a splitting criteria θ (fi, value)



→ Which feature & value to perform splitting?

Algorithm

Given training vectors $x_i \in R^n$, i=1,..., l and a label vector $y \in R^l$, a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node m be represented by Q_m with n_m samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ subsets

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

(feature, value)

The quality of a candidate split of node m is then computed using an impurity function or loss function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

(overall impurity; impurity function)

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_\theta G(Q_m, \theta)$$

Recurse for subsets $Q_m^{left}(\theta^*)$ and $Q_m^{right}(\theta^*)$ until the maximum allowable depth is reached, $n_m < \min_{samples}$ or $n_m = 1$.

1. **Splitting Criteria**: At each step of building the tree, the algorithm looks for the best way to split the data. It does this by choosing a feature and a value to split on. The goal is to find the split that maximizes the homogeneity of the target variable within each subset created by the split.
2. **Impurity Measures**: Sklearn uses impurity measures like Gini impurity or entropy to quantify the homogeneity within a set of data points. Gini impurity measures the probability of incorrectly classifying a randomly chosen element if it was randomly labeled according to the distribution of labels in the subset. Entropy, on the other hand, measures the average amount of information needed to classify a data point.
3. **Optimal Split Selection**: The algorithm evaluates different splits by calculating the impurity of the subsets created by each split. It then chooses the split that minimizes the impurity or maximizes the information gain (the difference between the impurity before and after the split).
4. **Recursive Partitioning**: This process of selecting the best split and partitioning the data is repeated recursively for each subset until certain stopping criteria are met, such as reaching a maximum tree depth, having too few data points in a node, or no further improvement in impurity reduction.
5. **Prediction**: Once the tree is built, to make predictions for new data points, the algorithm navigates through the tree based on the feature values of the data point until it reaches a leaf node. The majority class or the average value of the target variable in that leaf node is then assigned as the predicted value for the new data point.

Recursive Partitioning Algorithm :-

Suppose we have a response variable Y and a set of P predictor variables X_j for $j = 1, \cdots, P$. For a partition A of records, recursive partitioning will find the best way to partition A into two subpartitions:

1. For each predictor variable X_j:
   a. For each value s_j of X_j:
      i. Split the records in A with X_j values $< s_j$ as one partition, and the remaining records where $X_j \geq s_j$ as another partition.
      ii. Measure the homogeneity of classes within each subpartition of A.
   b. Select the value of s_j that produces maximum within-partition homogeneity of class.
2. Select the variable X_j and the split value s_j that produces maximum within-partition homogeneity of class.

Now comes the recursive part:

1. Initialize A with the entire data set.
2. Apply the partitioning algorithm to split A into two subpartitions, A_1 and A_2.
3. Repeat step 2 on subpartitions A_1 and A_2.
4. The algorithm terminates when no further partition can be made that sufficiently improves the homogeneity of the partitions.

Classification problem

(Multi class; Numerical; Binary class)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	85	85	Weak	No
2	Sunny	80	90	Strong	No
3	Overcast	83	78	Weak	Yes
4	Rain	70	96	Weak	Yes
5	Rain	68	80	Weak	Yes
6	Rain	65	70	Strong	No
7	Overcast	64	65	Strong	Yes
8	Sunny	72	95	Weak	No
9	Sunny	69	70	Weak	Yes
10	Rain	75	80	Weak	Yes
11	Sunny	75	70	Strong	Yes
12	Overcast	72	90	Strong	Yes
13	Overcast	81	75	Weak	Yes
14	Rain	71	80	Strong	No

gini impurity $H() = 1 - \sum_{i=1}^{K} P_k^2$

P_k = prob of each class

1)

Wind = Weak n=9

Yes No

n=6 n=3

PlayTennis: YES (4) NO (2) PlayTennis: YES (1) NO (2)

$P_Y = \frac{4}{6}$ $P_N = \frac{2}{6}$ $P_Y = \frac{1}{3}$ $P_N = \frac{2}{3}$

$H(Q_m^{left}(\theta)) = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2$ $H(Q_m^{right}(\theta)) = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2$

$G(Q_m, \theta) = \frac{6}{9} H(Q_m^{left}(\theta)) + \frac{3}{9} H(Q_m^{right}(\theta)) = G^1$

G^1

2)

Outlook = Sunny	Outlook = Rain	Outlook = Overcast
n=9	n=9	n=9
Yes / No	Yes / No	Yes / No
n=3 / n=6	n=3 / n=6	n=3 / n=6
PlayTennis: YES (0) NO (3) / PlayTennis: YES (5) NO (1)	PlayTennis: YES (2) NO (1) / PlayTennis: YES (3) NO (3)	PlayTennis: YES (3) NO (0) / PlayTennis: YES (1) NO (5)
$P_Y = 0$ $P_N = 1$ / $P_Y = \frac{5}{6}$ $P_N = \frac{1}{6}$	$P_Y = \frac{2}{3}$ $P_N = \frac{1}{3}$ / $P_Y = \frac{1}{2}$ $P_N = \frac{1}{2}$	$P_Y = 1$ $P_N = 0$ / $P_Y = \frac{2}{6}$ $P_N = \frac{4}{6}$
$H(Q_m^{left}(\theta)) = 1 - (0)^2 - (1)^2$ / $H(Q_m^{right}(\theta)) = 1 - \left(\frac{5}{6}\right)^2 - \left(\frac{1}{6}\right)^2$	$H(Q_m^{left}(\theta)) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2$ / $H(Q_m^{right}(\theta)) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2$	$H(Q_m^{left}(\theta)) = 1 - (1)^2 - (0)^2$ / $H(Q_m^{right}(\theta)) = 1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2$
$G(Q_m, \theta) = \frac{3}{9} H(Q_m^{left}(\theta)) + \frac{6}{9} H(Q_m^{right}(\theta)) = G_1^2$	$G(Q_m, \theta) = \frac{3}{9} H(Q_m^{left}(\theta)) + \frac{6}{9} H(Q_m^{right}(\theta)) = G_2^2$	$G(Q_m, \theta) = \frac{3}{9} H(Q_m^{left}(\theta)) + \frac{6}{9} H(Q_m^{right}(\theta)) = G_3^2$

(Pure node)

$\min(G_1^2, G_2^2, G_3^2)$

3)

Humidity	Yes/No			
65	Yes	Humidity < 67.5	Humidity < 79	Humidity < 92.5
67.5		G_1^3	G_4^3	G_7^3
70	NO			
72.5		Humidity < 72.5	Humidity < 82.5	Humidity < 95.5
75	Yes	G_2^3	G_5^3	G_8^3
76.5				
78	Yes			
79				
80	Yes			

$\min(G_1^3, G_2^3, G_3^3, G_4^3, G_5^3, G_6^3, G_7^3, G_8^3)$

72.5	75	Yes
76.5	78	Yes
79	80	Yes
82.5	85	NO
87.5	90	NO
92.5	95	NO
95.5	96	Yes

Humidity < 72.5 G^3_2 | Humidity < 82.5 G^3_5 | Humidity < 95.5 G^3_8

Humidity < 76.5 G^3_3 | Humidity < 87.5 G^3_4

$\min(G^1_1, G^1_2, G^1_3, G^1_4, G^1_5, G^1_6, G^1_7, G^1_8)$

$$\text{Root node decision} = \text{argmin}\left(|G^1| \quad |\min(G^2_1, G^2_2, G^2_3)| \quad |\min(G^3_1, G^3_2, G^3_3, G^3_4, G^3_5, G^3_6, G^3_7, G^3_8)| \right)$$

4) Repeat until we have pure nodes

Gini impurity

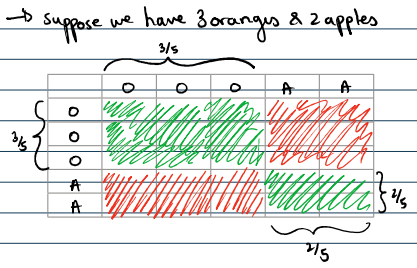
$$H(\cdot) = 1 - \sum_{i=1}^{k} P_k^2$$

Gini impurity measures the probability of incorrectly classifying a randomly chosen element if it was randomly labelled according to the distribution of labels in the subset.

Yes, that's a good way to think about it. Gini impurity measures the degree of impurity or uncertainty in a dataset. In a classification problem, if you were to randomly guess the class label for a data point based on the class distribution in the dataset, the Gini impurity represents the probability of guessing incorrectly.

So, when the Gini impurity is low (close to 0), it indicates that the dataset is predominantly pure, and therefore the probability of guessing wrong is low. Conversely, when the Gini impurity is high (close to 1), it suggests that the dataset is more mixed or impure, and the probability of guessing wrong is higher.

→ Suppose we have 3 oranges & 2 apples



Probability of guessing wrong = H(·) = Gini Impurity = $1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2$

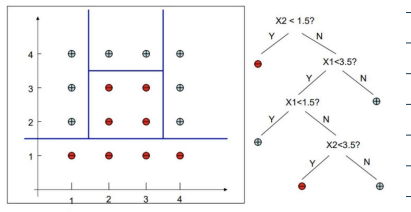
↑ Variety ↑ Impurity

$0 \leq H(\cdot) < (1)$ → Categories → ∞

Gini Impurity Range: $\left[0, 1 - \frac{1}{n}\right]$ ← # of classes

→ At each split we aim to attain max purity i.e. min impurity

Geometric perspective



Notes

The default values for the parameters controlling the size of the trees (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those parameter values.

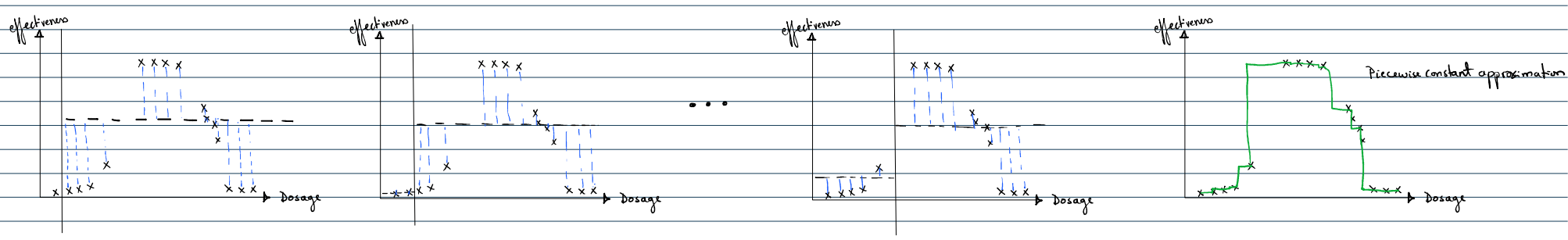
The :meth:`predict` method operates using the :func:`numpy.argmax` function on the outputs of :meth:`predict_proba`. This means that in case the highest predicted probabilities are tied, the classifier will predict the tied class with the lowest index in :term:`classes_`.

Regression

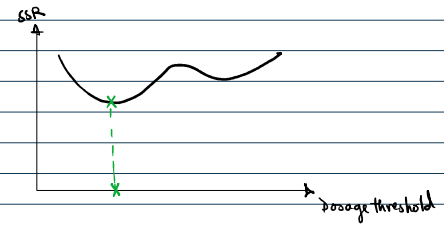
→ steps are similar to classification problem, but instead of gini impurity we use Mean Squared error.

$mse = H(\cdot) = \frac{1}{n}\sum (x_i - \bar{x})^2$ → think of this as variance

Geometric perspective



SSR



→ choose threshold value which results smallest sum of squares residuals.

Pros & Cons

→ Simple to understand & interpret. Trees can be visualized

→ Requires little data preparation

→ The cost of using the tree is logarithmic in the number of data points used to train the tree.

→ Able to handle both numerical & categorical data.

→ Can work on non-linear datasets

→ Can give you feature importance

→ DT learners can create over-complex trees that don't generalize the data well. (overfitting)

→ DT can be unstable b/c small variations in data might result in a completely different tree being generated.

→ predictions of DT's are neither smooth nor continuous, but piecewise constant approximations. ∴ they are not good at extrapolation.

→ low bias high variance models