Decision tree Api write up

Om shinde

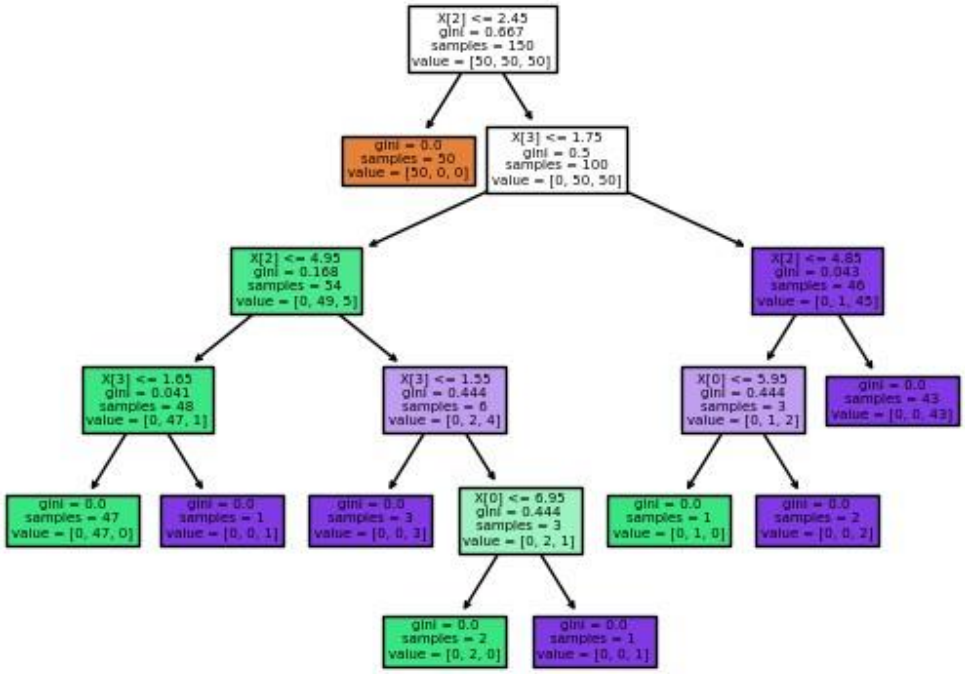
J066

1) Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Code:-

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
```



Important Parameters are:-

- Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to. Use `max_depth` to control the size of the tree to prevent overfitting.
- Use `min_samples_split` or `min_samples_leaf` to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered. A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data. Try `min_samples_leaf=5` as an initial value. If the sample size varies greatly, a float number can be used as percentage in these two parameters. While `min_samples_split` can create arbitrarily small leaves, `min_samples_leaf` guarantees that each leaf has a minimum size, avoiding low-variance, over-fit leaf nodes in regression problems. For classification with few classes, `min_samples_leaf=1` is often the best choice.

  Note that `min_samples_split` considers samples directly and independent of `sample_weight`, if provided (e.g. a node with m weighted samples is still treated as having exactly m samples).

Consider `min_weight_fraction_leaf` or `min_impurity_decrease` if accounting for sample weights is required at splits.

- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (`sample_weight`) for each class to the same value. Also note that weight-based pre-pruning criteria, such as `min_weight_fraction_leaf`, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like `min_samples_leaf`.
- If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as `min_weight_fraction_leaf`, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.

Classification Criterion:

If a target is a classification outcome taking on values 0,1,...,K-1, for node m, let

$$p_{mk} = 1/N_m \sum_{y \in Q_m} I(y = k)$$

be the proportion of class k observations in node m. If m is a terminal node, `predict_proba` for this region is set to p_{mk}. Common measures of impurity are the following.

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

Entropy:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

Misclassification:

$$H(Q_m) = 1 - \max(p_{mk})$$

Important Methods : -

Fit(X, y)- fit the linear model.

Predict(X)-predict using linear model.

Score(X,y)-returns the coefficient of determination R^2 of the prediction.

Advantages :

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Disadvantages:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.