**Decision tree**

1. **What are some common hyperparameters of decision tree models, and how do they affect the model's performance ?**

Decision trees are a versatile and interpretable machine learning algorithm, but they have several hyperparameters that can significantly influence their performance. Here’s a breakdown of some common hyperparameters and how they affect the model:

**1. max\_depth**

* **Description**: The maximum depth of the tree. This limits the number of levels in the tree.
* **Effect on Performance**:
  + **Shallow Trees**: May underfit the data, leading to poor performance on both training and test sets.
  + **Deep Trees**: Can capture more complex patterns but are prone to overfitting, which may lead to excellent training performance but poor generalization to unseen data.

**2. min\_samples\_split**

* **Description**: The minimum number of samples required to split an internal node.
* **Effect on Performance**:
  + **Low Values**: The tree might grow too complex and overfit the training data.
  + **High Values**: Can lead to a more generalized model but may miss finer patterns in the data.

**3. min\_samples\_leaf**

* **Description**: The minimum number of samples required to be at a leaf node.
* **Effect on Performance**:
  + **Low Values**: Can result in a very detailed tree that overfits the training data.
  + **High Values**: Results in a more generalized tree with potentially smoother decision boundaries, but might miss important patterns.

**4. max\_features**

* **Description**: The number of features to consider when looking for the best split.
* **Effect on Performance**:
  + **Low Values**: Can reduce the variance of the model and help prevent overfitting, but may also reduce the model’s ability to find the best split.
  + **High Values**: Can make the model more complex and potentially overfit the training data.

**5. max\_leaf\_nodes**

* **Description**: The maximum number of leaf nodes in the tree.
* **Effect on Performance**:
  + **Low Values**: Can prevent the model from overfitting by limiting its complexity.
  + **High Values**: Allows the model to capture more intricate patterns but can lead to overfitting.

**6. min\_impurity\_decrease**

* **Description**: A node will be split if the impurity decrease is greater than or equal to this value.
* **Effect on Performance**:
  + **Low Values**: Encourages more splits and potentially a more complex model.
  + **High Values**: Restricts splitting and can result in a simpler model with potentially higher bias.

**7. criterion**

* **Description**: The function to measure the quality of a split. Common criteria are "gini" (Gini impurity) and "entropy" (information gain).
* **Effect on Performance**:
  + Different criteria can slightly affect the structure of the decision tree, but the choice often has a minimal impact compared to other hyperparameters.

**8. splitter**

* **Description**: The strategy used to choose the split at each node. Common options are "best" (chooses the best split) and "random" (chooses the best random split).
* **Effect on Performance**:
  + **"Best"**: Ensures that the splits are optimal at each node, usually leading to better performance but can be more computationally expensive.
  + **"Random"**: Can lead to faster training times but might result in a less optimal tree structure.

**Tuning Hyperparameters**

To achieve optimal performance, you typically need to perform hyperparameter tuning. Techniques such as grid search, random search, or more advanced methods like Bayesian optimization can help find the best combination of hyperparameters for your specific dataset.

**2. What is the difference between the Label encoding and One-hot encoding?**

**Label Encoding**

**Definition**: Label encoding converts each category into a unique integer. For example, if you have a categorical feature like Color with values ["Red", "Green", "Blue"], label encoding would convert these to [0, 1, 2].

**How It Works**:

* Assign an integer to each category.
* Categories are replaced with these integers in the dataset.

**Example**: For a feature Color with values ["Red", "Green", "Blue"], label encoding might transform it into [0, 1, 2].

**Pros**:

* **Simple**: Easy to implement and understand.
* **Compact**: Requires less memory compared to one-hot encoding because it uses a single integer per category.

**Cons**:

* **Implied Ordinality**: The method introduces an ordinal relationship where none may exist. For instance, it might imply that Green (1) is somehow between Red (0) and Blue (2), which might not be true.
* **Not Suitable for All Algorithms**: Some algorithms (like tree-based models) can handle label encoding well, but others (like linear models) might interpret the numeric values as ordinal and thus may perform poorly.

**One-Hot Encoding**

**Definition**: One-hot encoding converts each category into a binary vector. Each category is represented by a vector of length equal to the number of categories, with a single 1 indicating the presence of the category and 0s elsewhere.

**How It Works**:

* For a categorical feature with k distinct categories, each category is represented by a vector of length k.
* For instance, with categories ["Red", "Green", "Blue"], one-hot encoding would convert these into:
  + Red → [1, 0, 0]
  + Green → [0, 1, 0]
  + Blue → [0, 0, 1]

**Pros**:

* **No Implied Ordinality**: One-hot encoding does not imply any ordinal relationship between categories, which is useful for categorical features without intrinsic order.
* **Widely Compatible**: Many machine learning algorithms, especially those that rely on numerical input, can handle one-hot encoded features effectively.

**Cons**:

* **Increased Dimensionality**: Can lead to a significant increase in the number of features, especially if the categorical feature has many distinct values (high cardinality). This can result in higher memory usage and longer computation times.
* **Sparsity**: One-hot encoded vectors are sparse (mostly zeros), which can be inefficient for algorithms that do not handle sparse data well.

**When to Use Each Method**

* **Label Encoding**: Useful when the categorical variable is ordinal (i.e., there is a meaningful order or ranking among the categories) and when you are using models that can handle categorical variables effectively or where the algorithm can handle ordinal relationships.
* **One-Hot Encoding**: Preferred for nominal categorical variables (i.e., categories with no inherent order) and when using algorithms that may misinterpret numeric values as ordinal or when the relationships between categories are not meaningful.