



Module-4

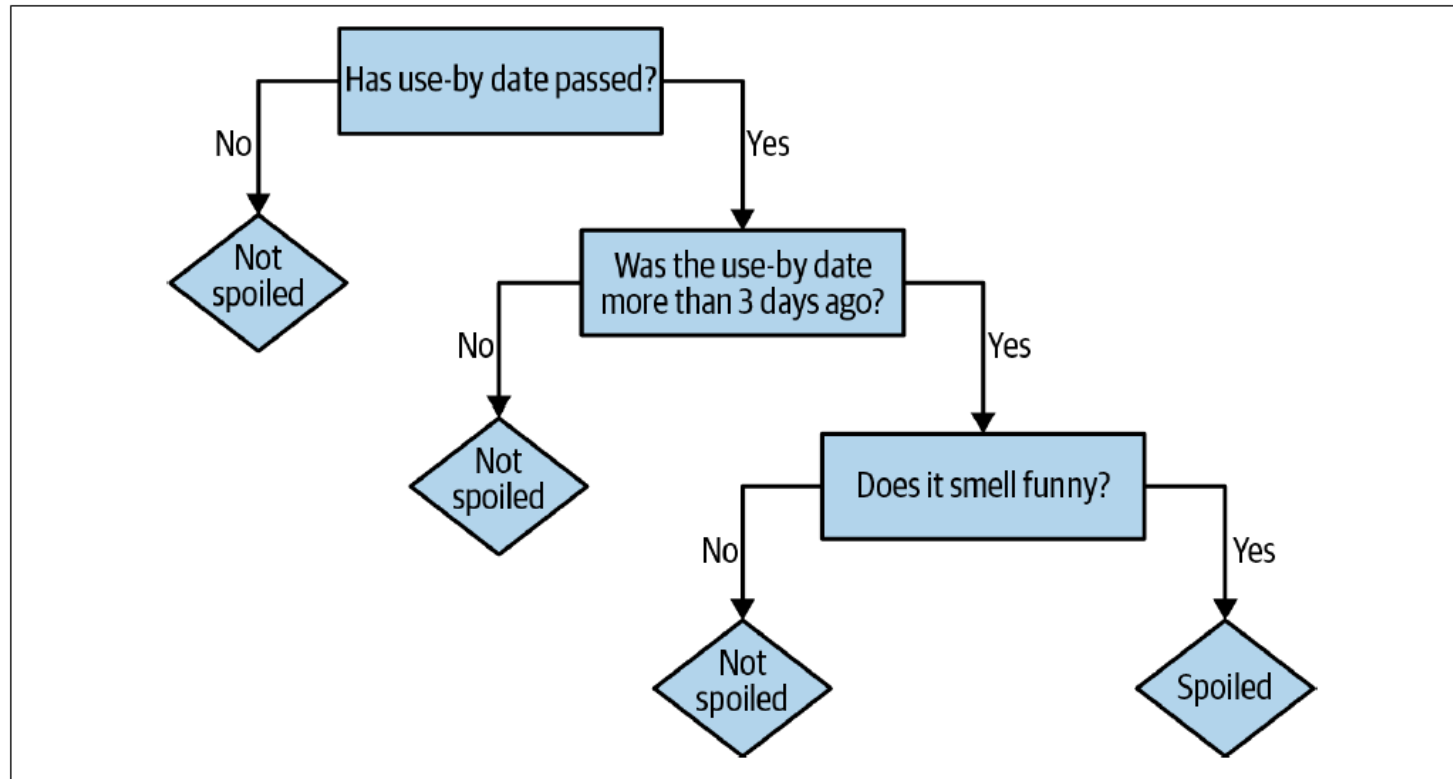
COMPILED BY,

DR. PRAKASH KALINGRAO AITHAL

Decision Trees and Forests

Decision trees are a family of algorithms that can naturally handle both categorical and numeric features. Building a single tree can be done using parallel computing, and many trees can be built in parallel at once. They are robust to outliers in the data, meaning that a few extreme and possibly erroneous data points might not affect predictions at all. They can consume data of different types and on different scales without the need for preprocessing or normalization.

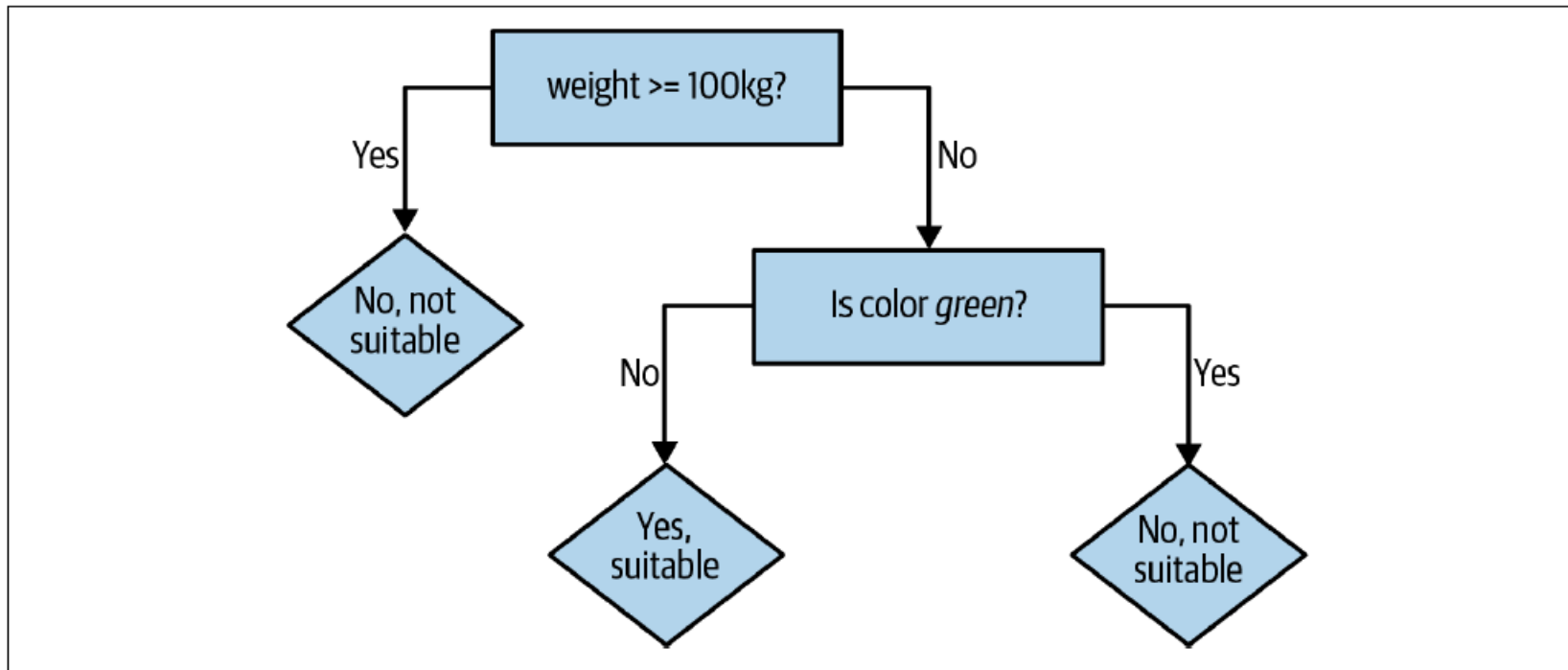
Binary Decision Tree



Binary Decision Tree

Name	Weight (kg)	# Legs	Color	Good pet?
Fido	20.5	4	Brown	Yes
Mr. Slither	3.1	0	Green	No
Nemo	0.2	0	Tan	Yes
Dumbo	1390.8	4	Gray	No
Kitty	12.1	4	Gray	Yes
Jim	150.9	2	Tan	No
Millie	0.1	100	Brown	No
McPigeon	1.0	2	Gray	No
Spot	10.0	4	Brown	Yes

Binary Decision Tree



Random Forests

Decision trees generalize into a more powerful algorithm, called *random forests*. Random forests combine many decision trees to reduce the risk of overfitting and train the decision trees separately. The algorithm injects randomness into the training process so that each decision tree is a bit different. Combining the predictions reduces the variance of the predictions, makes the resulting model more generalizable, and improves performance on test data.

Encoding Categorical Variable

Be careful when encoding a categorical feature as a single numeric feature. The original categorical values have no ordering, but **when encoded as a number, they appear to**. Treating the encoded feature as numeric leads to meaningless results because the algorithm is effectively pretending that rainy is somehow greater than, and two times larger than, cloudy. It's OK as long as the encoding's numeric value is not used as a number.

Decision Tree Implementation

```
from pyspark.ml.classification import DecisionTreeClassifier

classifier = DecisionTreeClassifier(seed = 1234, labelCol="Cover_Type",
featuresCol="featureVector",
predictionCol="prediction")

model = classifier.fit(assembled_train_data)

print(model.toDebugString)
```


Evaluating Decision Tree

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="Cover_Type",
predictionCol="prediction")

evaluator.setMetricName("accuracy").evaluate(predictions)

evaluator.setMetricName("f1").evaluate(predictions)
```

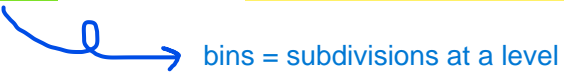
Confusion Matrix

```
confusion_matrix = predictions.groupBy("Cover_Type").\  
pivot("prediction", range(1,8)).count().\  
na.fill(0.0).\  
orderBy("Cover_Type")  
confusion_matrix.show()
```

Decision Tree Hyperparameters

- Maximum depth
- maximum bins
- impurity measure
- minimum information gain.

Decision Tree Hyperparameters

- **Maximum depth** simply limits the number of levels in the decision tree. It is the maximum number of chained decisions that the classifier will make to classify an example. It is useful to limit this to avoid overfitting the training data.
- A larger number of **bins** requires more processing time but might lead to finding a more optimal decision rule.
 bins = subdivisions at a level
- There are two commonly used **measures of impurity**: Gini impurity and entropy. criterion used to decide how good a split is at each node.
- **minimum information gain** is a hyperparameter that imposes a minimum information gain, or decrease in impurity, for candidate decision rules. Rules that do not improve the subsets' impurity enough are rejected. Like a lower maximum depth, this can help the model resist overfitting because decisions that barely help divide the training input may in fact not helpfully divide future data at all.

Decision Tree Hyperparameters

Gini impurity is directly related to the accuracy of the random guess classifier. Within a subset, it is the probability that a randomly chosen classification of a randomly chosen example (both according to the distribution of classes in the subset) is *incorrect*. To calculate this value, we first multiply each class with its respective proportion among all classes. Then we subtract the sum of all the values from 1. If a subset has N classes and p_i is the proportion of examples of class i , then its Gini impurity is given in the Gini impurity equation:

$$I_G(p) = 1 - \sum_{i=1}^N p_i^2$$

Decision Tree Hyperparameters

Entropy is another measure of impurity, borrowed from information theory. Its nature is more difficult to explain, but it captures how much uncertainty the collection of target values in the subset implies about predictions for data that falls in that subset. A subset containing one class suggests that the outcome for the subset is completely certain and has 0 entropy—no uncertainty. A subset containing one of each possible class, on the other hand, suggests a lot of uncertainty about predictions for that subset because data has been observed with all kinds of target values. This has high entropy. Hence, low entropy, like low Gini impurity, is a good thing. Entropy is defined by the entropy equation:

$$I_E(p) = \sum_{i=1}^N p_i \log \left(\frac{1}{p_i} \right) = - \sum_{i=1}^N p_i \log (p_i)$$

Tuning Hyperparameters

```
from pyspark.ml.tuning import ParamGridBuilder
```

```
paramGrid = ParamGridBuilder(). \
```

```
addGrid(classifier.impurity, ["gini", "entropy"]). \
```

```
addGrid(classifier.maxDepth, [1, 20]). \
```

```
addGrid(classifier.maxBins, [40, 300]). \
```

```
addGrid(classifier.minInfoGain, [0.0, 0.05]). \
```

```
build()
```

In PySpark, ParamGridBuilder is used in conjunction with the CrossValidator or TrainValidationSplit to perform hyperparameter tuning for machine learning models, such as decision trees.

It allows you to define a grid of hyperparameters to search over in order to find the best model configuration.

How it works:

ParamGridBuilder creates a grid of hyperparameters by specifying different values for the parameters you want to tune.

The grid is then passed to a cross-validation or train-validation routine to test all combinations of parameters and select the best model based on performance metrics.

Evaluating Decision Tree

```
multiclassEval = MulticlassClassificationEvaluator(). \
```

```
setLabelCol("Cover_Type"). \
```

```
setPredictionCol("prediction"). \
```

```
setMetricName("accuracy")
```

actual output column name

predicted o/p column

evaluation metric

Random Forest

```
from pyspark.ml.classification import RandomForestClassifier
```

```
classifier = RandomForestClassifier(seed=1234, labelCol="Cover_Type",
```

```
featuresCol="indexedVector",
```

```
predictionCol="prediction")
```

what column to predict

input features

which column to store the predicted data

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

Gini Index

Calculating the Gini Index for past trend

Since the past trend is positive 6 number of times out of 10 and negative 4 number of times, the calculation will be as follows:

P(Past Trend=Positive): 6/10

P(Past Trend=Negative): 4/10

- If (Past Trend = Positive & Return = Up), probability = 4/6
- If (Past Trend = Positive & Return = Down), probability = 2/6

$$\text{Gini Index} = 1 - ((4/6)^2 + (2/6)^2) = 0.45$$

Gini Index

- If (Past Trend = Negative & Return = Up), probability = 0
- If (Past Trend = Negative & Return = Down), probability = 4/4

$$\text{Gini Index} = 1 - ((0)^2 + (4/4)^2) = 0$$

- Weighted sum of the Gini Indices can be calculated as follows:

$$\text{Gini Index for Past Trend} = (6/10)0.45 + (4/10)0 = 0.27$$

Gini Index

Calculating the Gini Index for open interest

Coming to open interest, the open interest is high 4 times and low 6 times out of total 10 times and is calculated as follows:

$P(\text{Open Interest}=\text{High}): 4/10$

$P(\text{Open Interest}=\text{Low}): 6/10$

- If (Open Interest = High & Return = Up), probability = $2/4$
- If (Open Interest = High & Return = Down), probability = $2/4$

$$\text{Gini Index} = 1 - ((2/4)^2 + (2/4)^2) = 0.5$$

Gini Index

- If (Open Interest = Low & Return = Up), probability = $2/6$
- If (Open Interest = Low & Return = Down), probability = $4/6$

$$\text{Gini Index} = 1 - ((2/6)^2 + (4/6)^2) = 0.45$$

- Weighted sum of the Gini Indices can be calculated as follows:

$$\text{Gini Index for Open Interest} = (4/10)0.5 + (6/10)0.45 = 0.47$$

Gini Index

Calculating the Gini Index for trading volume

Trading volume is 7 times high and 3 times low and is calculated as follows:

$P(\text{Trading Volume}=\text{High}): 7/10$

$P(\text{Trading Volume}=\text{Low}): 3/10$

- If (Trading Volume = High & Return = Up), probability = $4/7$
- If (Trading Volume = High & Return = Down), probability = $3/7$

$$\text{Gini Index} = 1 - ((4/7)^2 + (3/7)^2) = 0.49$$

Gini Index

- If (Trading Volume = Low & Return = Up), probability = 0
- If (Trading Volume = Low & Return = Down), probability = 3/3

$$\text{Gini Index} = 1 - ((0)^2 + (1)^2) = 0$$

- Weighted sum of the Gini Indices can be calculated as follows:

$$\text{Gini Index for Trading Volume} = (7/10)0.49 + (3/10)0 = 0.34$$

Gini Index

Gini Index attributes or features

Attributes/Features	Gini Index
Past Trend	0.27
Open Interest	0.47
Trading Volume	0.34

Gini Index

From the above table, we observe that 'past trend' has the lowest Gini Index and hence, it will be chosen as the root node for how the decision tree works.

We will repeat the same procedure to determine the sub-nodes or branches of the decision tree.

We will calculate the Gini Index for the 'positive' branch of past trend as follows:

Gini Index

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Positive	Low	High	Up
Positive	High	High	Up
Positive	Low	Low	Down
Positive	Low	Low	Down
Positive	High	High	Up

Gini Index

Calculating Gini Index of open interest for positive past trend

Open interest for positive past trend is high 2 times out of 6 and low 4 times out of 6 and the Gini Index of open interest for positive past trend is calculated as follows:

$P(\text{Open Interest}=\text{High}): 2/6$

$P(\text{Open Interest}=\text{Low}): 4/6$

- If (Open Interest = High & Return = Up), probability = $2/2$
- If (Open Interest = High & Return = Down), probability = 0

$$\text{Gini Index} = 1 - (\text{sq}(2/2) + \text{sq}(0)) = 0$$

Gini Index

- If (Open Interest = Low & Return = Up), probability = 2/4
- If (Open Interest = Low & Return = Down), probability = 2/4

$$\text{Gini Index} = 1 - (\text{sq}(0) + \text{sq}(2/4)) = 0.50$$

- Weighted sum of the Gini Indices can be calculated as follows:

$$\text{Gini Index for Open Interest} = (2/6)0 + (4/6)0.50 = 0.33$$

Gini Index

Calculating Gini Index for trading volume

The trading volume is high 4 out of 6 times and low 2 out of 6 times and is calculated as follows:

$P(\text{Trading Volume}=\text{High}): 4/6$

$P(\text{Trading Volume}=\text{Low}): 2/6$

- If (Trading Volume = High & Return = Up), probability = 4/4
- If (Trading Volume = High & Return = Down), probability = 0

$$\text{Gini Index} = 1 - (\text{sq}(4/4) + \text{sq}(0)) = 0$$

Gini Index

- If (Trading Volume = Low & Return = Up), probability = 0
- If (Trading Volume = Low & Return = Down), probability = 2/2

$$\text{Gini Index} = 1 - (\text{sq}(0) + \text{sq}(2/2)) = 0$$

- Weighted sum of the Gini Indices can be calculated as follows:

$$\text{Gini Index for Trading Volume} = (4/6)0 + (2/6)0 = 0$$

Gini Index

Gini Index attributes or features

Attributes/Features	Gini Index
Open interest	0.33
Trading volume	0

References

Sandya Ryza, Uri Laserson, Sean Owen and Josh Wills, Advanced Analytics with Spark (2e), O'Reilly Media Inc, 2017

<https://blog.quantinsti.com/gini-index/>