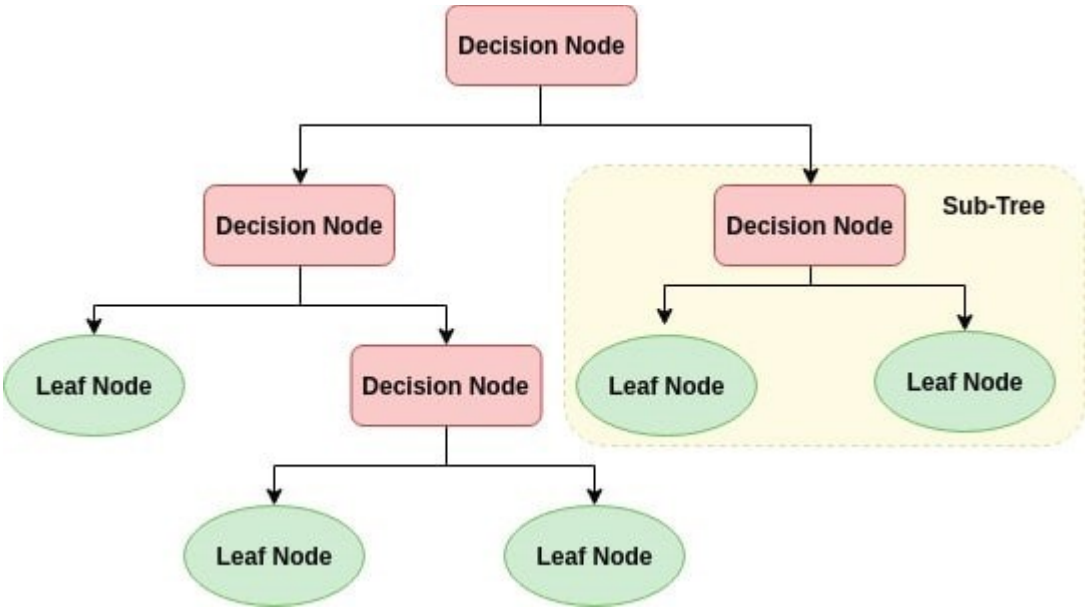
Decision Tree

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps us in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



Decision Tree algorithm

The basic idea behind any decision tree algorithm is as follows:

1.Select the best attribute using Attribute Selection Measures(ASM) to split the records.
2.Make that attribute a decision node and breaks the dataset into smaller subsets.
3.Starts tree building by repeating this process recursively for each child until one of the condition will match:
◦All the tuples belong to the same attribute value.
◦There are no more remaining attributes.
◦There are no more instances.

Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature(or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute. In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Information gain is the decrease in entropy(randomness or the impurity in the system). Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

Where,
•Info(D) is the average amount of information needed to identify the class label of a tuple in D.
•|Dj|/|D| acts as the weight of the jth partition.
•InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N().

Gain Ratio

Information gain is biased for the attribute with many outcomes. It means it prefers the attribute with a large number of distinct values. For instance, consider an attribute with a unique identifier such as customer_ID has zero info(D) because of pure partition. This maximizes the information gain and creates useless partitioning.

C4.5, an improvement of ID3, uses an extension to information gain known as the gain ratio. Gain ratio handles the issue of bias by normalizing the information gain using Split Info.

Gini index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points.

$$\text{Gini(D)}=1-\sum_{i=1}^{m}\text{Pi}^2$$

Where, pi is the probability that a tuple in D belongs to class Ci.

The Gini Index considers a binary split for each attribute. We can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

$$\text{Gini}_A\text{(D)}=\frac{|D1|}{|D|}\text{Gini}(\text{D}_1)+\frac{|D2|}{|D|}\text{Gini}(\text{D}_2)$$

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with minimum Gini index is chosen as the splitting attribute.

CHAID and CART

Classification and Regression Trees (CART)

Regression Tree : The outcome (dependent) variable is a continuous variable and predictor (independent) variables can be continuous or categorical variables (binary). It creates binary split.
Classification Tree : The outcome (dependent) variable is a categorical variable (binary) and predictor (independent) variables can be continuous or categorical variables (binary). It creates binary split.
The impurity of a node is measured by the Least-Squared Deviation (LSD), which is simply the within variance for the node.
Algorithm of Classification Tree: Gini Index
Gini Index measures impurity in node. It varies between 0 and (1-1/n) where n is the number of categories in a dependent variable.
Process :
1.Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable
2.Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure)
3.Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. (Alternatively, the data are split as much as possible and then the tree is later pruned.)

Chi-square Automated Interaction Detection(CHAID)

The outcome (dependent) variable can be continuous and categorical. But, predictor (independent) variables are categorical variables only (can be more than 2 categories). It can create multiple splits (more than 2).

When independent variables are continuous, they need to be transformed into categorical variables (bins/groups) before using CHAID.
Algorithm :
If dependent variable is categorical, Chi-Square test determines the best next split at each step.
If dependent variable is continuous, F test determines the best next split at each step.

Decision Tree Classifier Building in Scikit-learn

In [49]:
```
# Load libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier , plot_tree    # Import D
from sklearn.model_selection import train_test_split           # Import t
from sklearn import metrics                                    # Import s
```

Let's first load the Pima Indian Diabetes dataset using pandas' read CSV function.

In [50]:
```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedi
# Load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_na
pima.head()
```

Out[50]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

We need to divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

In [51]:
```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedig
X = pima[feature_cols]     # Features
y = pima.label             # Target variable
```

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.
Let's split the dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size.

In [52]:
```
# Split dataset into training set and test set
# 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

Let's create a Decision Tree Model using Scikit-learn.

In [65]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

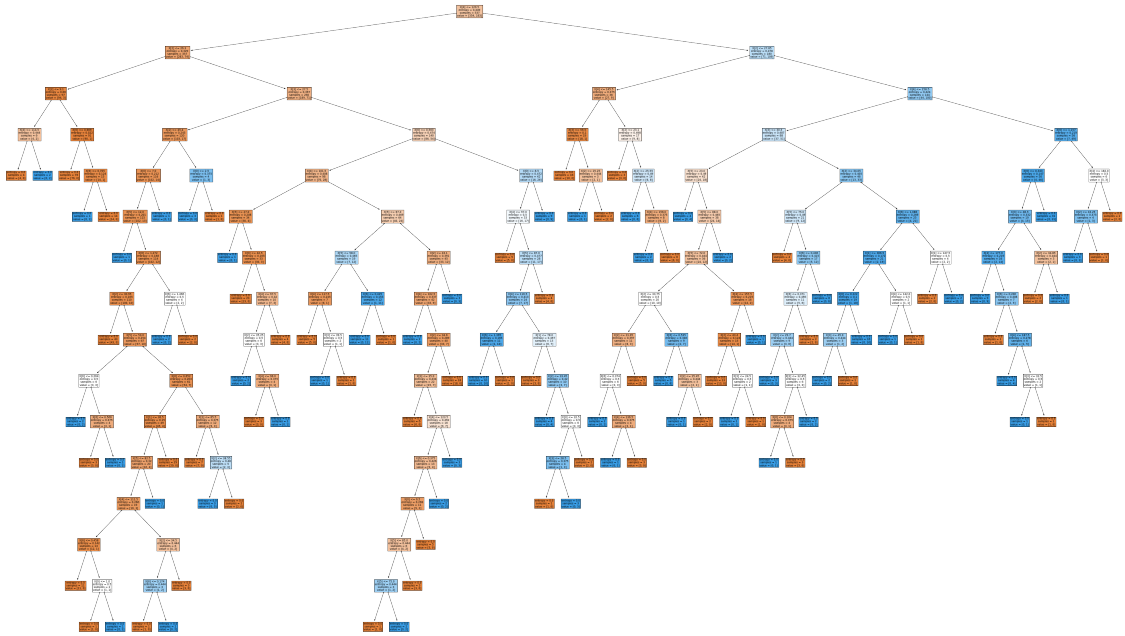
Let's estimate, how accurately the classifier or model can predict the type of cultivars.
Accuracy can be computed by comparing actual test set values and predicted values.

In [66]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6623376623376623
```

Well, we got a classification rate of 67.53%, considered as good accuracy. We can improve this accuracy by tuning the parameters in the Decision Tree Algorithm.

In [75]:
```
plt.figure(figsize = [50 , 30])
plot_tree(clf, filled=True)
plt.show()
```



In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred as Gini ratio, which measures the impurity of the node. We can say a node is pure when all of its records belong to the same class, such nodes known as the leaf node.

Here, the resultant tree is unpruned. This unpruned tree is unexplainable and not easy to understand. Let's optimize it by pruning.

Optimizing Decision Tree Performance

•**criterion : optional (default="gini") or Choose attribute selection measure**: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

•**splitter : string, optional (default="best") or Split Strategy**: This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

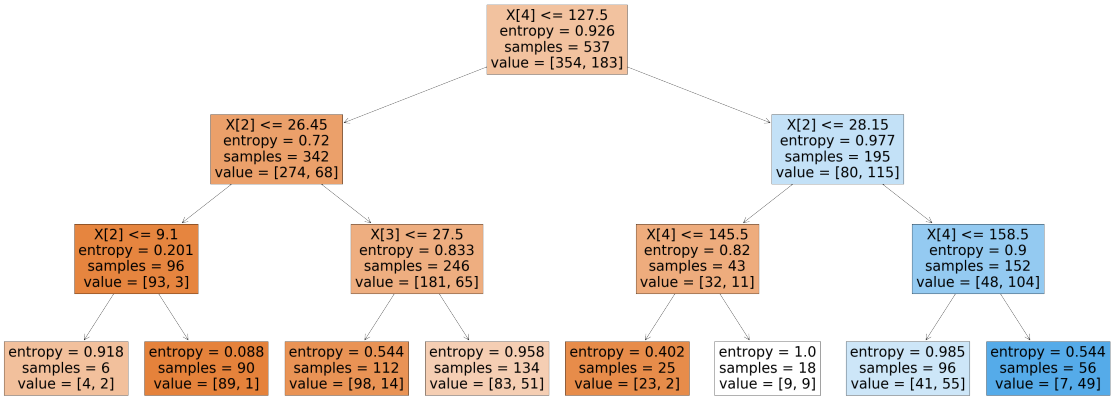
•**max_depth : int or None, optional (default=None) or Maximum Depth of a Tree**: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting.

In [72]:
```
# Create Decision Tree classifer object
clf_1 = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf_1 = clf_1.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf_1.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7705627705627706
```

Well, the classification rate increased to 77.05%, which is better accuracy than the previous model.

In [73]:
```
plt.figure(figsize = [50 , 20])
plot_tree(clf_1, filled=True)
plt.show()
```

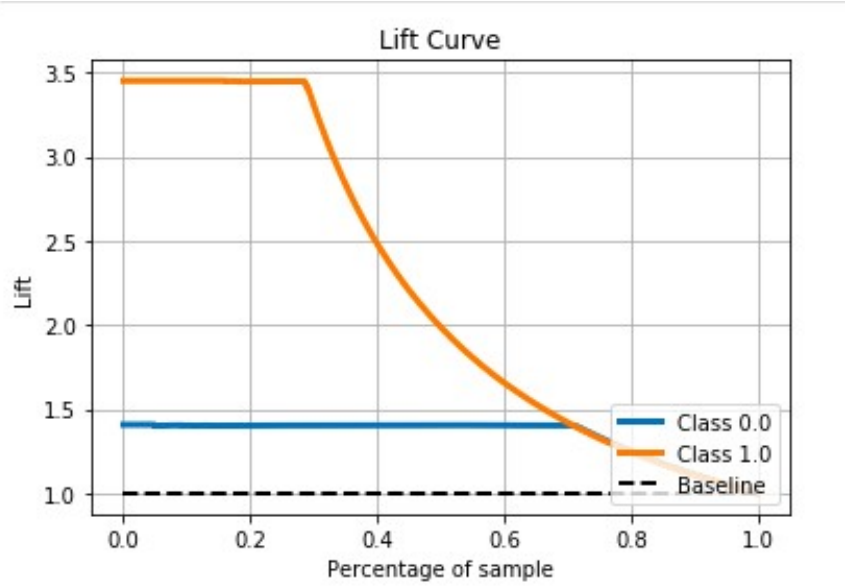


This pruned model is less complex, explainable, and easy to understand than the previous decision tree model plot.

Model Lift

Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model. A targeting model is doing a good job if the response within the target is much better than the average for the population as a whole. Lift is simply the ratio of these values: target response divided by average response.

import matplotlib.pyplot as plt import scikitplot as skplt skplt.metrics.plot_lift_curve(y_test, y_pred) plt.show()



Advantages:

•Decision trees are easy to interpret and visualize.

•It can easily capture Non-linear patterns.

•It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.

•It can be used for feature engineering such as predicting missing values, suitable for variable selection.

•The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.

Disadvantages:

•Sensitive to noisy data. It can overfit noisy data.

•The small variation(or variance) in data can result in the different decision tree. This can be reduced by bagging and boosting algorithms.

•Decision trees are biased with imbalance dataset, so it is recommended that balance out the dataset before creating the decision tree.

Questionnaire

What is difference between Pre-pruning and Post-pruning of decision tree?

What is over fitting in decision tree?

Are tree based models better than linear models?

Solution (https://github.com/ebi-byte/kt/blob/master/trees/Trees%20Questionnaire.ipynb)