Department of Computer Engineering
University of Peradeniya

CO 544 Machine Learning and Data Mining
Lab 05

March 31, 2022

1. Objectives

To demonstrate the decision tree algorithm, attribute selection measures, optimize descision tree performance, evaluate performance, visualize decision trees and provide students hands on experience in building a classfier using `sklearn`.

2. Preliminaries

In this lab, you will use a set of predefined Python functions to build and manipulate decision trees. In order to run this, you need to have `sklearn` and `Pandas` libraries installed. We also use the `graphviz` and `pydotplus` libraries for plotting trees.

3. Decision tree algorithm

A decision tree is a flowchart-like tree structure where an internal node represents a 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 and partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision making. The visualization of a decision tree is similar to a flowchart diagram, which easily mimics human-level thinking.

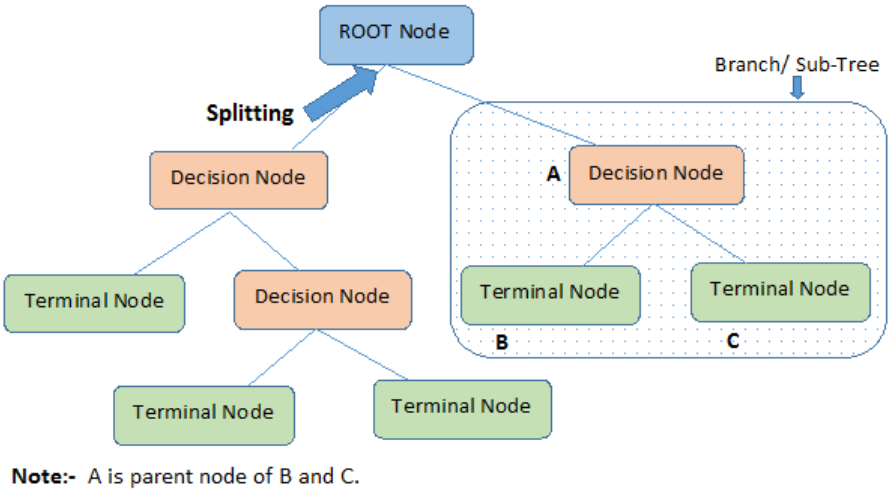

Figure 1. A typical decision tree

(Source: Hands-On Machine Learning on Google Cloud Platform by Giuseppe Ciaburro, V Kishore Ayyadevara, and Alexis Perrier)

The preceding diagram is explained as follows:

- **ROOT Node**: this represents the entire population or a sample, and it is further divided into two or more further nodes.

- **Splitting**: a process of dividing a node into two or more subnodes based on a certain rule.
- **Decision Node**: when a subnode splits into further subnodes, it is called decision node.
- **Leaf/Terminal Node**: the final node in a decision ...

The decision tree is a white-box type of machine learning algorithm. It shares the logic of internal decision-making, which is not available in black-box algorithms, such as neural networks. Its training time is relatively fast compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and the number of attributes in a given data. The decision tree is a distribution-free or non-parametric method, which does not rely on probability distribution assumptions. Decision trees can handle higher dimensional data with better accuracy.

The intuition behind any decision tree algorithm is as follows:

1. Select the best attribute using attribute selection measures (ASM) to split records.
2. Turn that attribute into a decision node and split the dataset into smaller subsets.
3. Start building the tree by repeating this process recursively to each child until one of the conditions is met:
    - All tuples belong to the same attribute value.
    - No more remaining attributes.
    - No more instances.

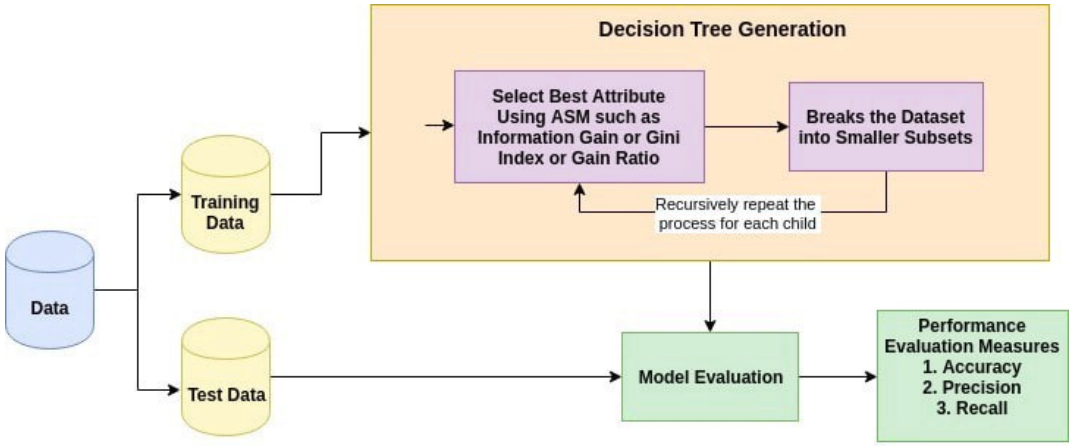

Figure 2. A complete pipeline of a typical decision tree algorithm-based ML model

4. Attribute selection measures (ASM)

ASM is a heuristic for selecting data splitting criteria in the best possible way. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a ranking for each feature (or attribute) by explaining the given dataset. The best score attribute is selected as a splitting attribute (Devi and Nirmala, 2013). In the case of a continuous-valued attribute, split points for branches should also be defined. The most popular selection measures are information gain, gain ratio, and Gini index.

Information gain

Shannon (1948) invented the concept of entropy, which measures the impurity of the input set. In general, entropy is referred to as the randomness or the impurity in a system. In information theory, it refers to the impurity in a group of examples. Information gain means decreasing entropy. It computes the difference between the entropy before splitting and the average entropy after splitting the dataset

based on given attribute values. The iterative dichotomiser (ID3) decision tree algorithm uses entropy to calculate information gain.

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

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} XInfo(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

where p_i is the probability that an arbitrary tuple in D belongs to class C_i, $Info(D)$ is the average amount of information needed to identify the class label of a tuple in D, $\frac{|D_j|}{|D|}$ acts as the weight of the j^{th} partition, and $Info_A(D)$ 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 [i.e., $Gain(A)$] is selected as the splitting attribute at node $N()$.

Gain ratio

Information gain is biased for an attribute with many outcomes. This means that it prefers an attribute with a large number of distinct values. Information gain maximizes the information retrieval and creates useless partitions.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right)$$

where v is the number of discrete values in attribute A.

The gain ratio can be defined as:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The attribute with the highest gain ratio is selected as the splitting attribute (Nair et al., 2010).

Gini index

Another attribute selection measure used by the categorical and regression trees (CART) to create split points is the Gini index.

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

The Gini Index considers a binary split for each attribute. Here a weighted sum of the impurity of each partition can be calculated. If a binary split on attribute A partitions data D into D_1 and D_2, the Gini index of D can be written as:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

In the case of a discrete-valued attribute, the subset that gives the minimum Gini index is selected as a splitting attribute, and 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 a smaller Gini index as the splitting point.

5. Dataset description

In this lab work, the Pima Indians Diabetes dataset is used to build the decision tree. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The dataset can be downloaded from Kaggle using the following url: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.

6. Building decision trees

i. Importing libraries

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Decision tree classifier
from sklearn.model_selection import train_test_split
from sklearn import metrics # scikit-learn metrics module for computing accuracy

import warnings
warnings.filterwarnings('ignore') # Ignore warning messages
```

ii. Dataset loading and exploratory data analysis

Loading dataset

```
diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head() # Preview the dataset
diabetes_df.shape # Number of instances and variables
```

Renaming columns

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age',
             'label'] # Define new column names
diabetes_df.columns = col_names # Rename column names
```

Summary of dataset

```
diabetes_df.info()
```

Frequency distributions of values in variables

```
for col in col_names:
    print(diabetes_df[col].value_counts())
```

Exploring target variable

```
diabetes_df['label'].value_counts()
```

Checking missing values in variables

```
diabetes_df.isnull().sum()
```

iii. Defining feature vector and target variable

```
X = diabetes_df.drop(['label'], axis=1) # Drop the target variable
y = diabetes_df['label']
```

iv. Splitting data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                    random_state=1) # 75% training and 25% test
X_train.shape, X_test.shape # Shapes of X_train and X_test
```

v. Feature engineering: encoding categorical variables

This is the process of transforming raw data into useful features that help us better understand our model better and increase its predictive power.

```
X_train.dtypes # Check data types in X_train

import category_encoders as ce # Import the relevant library
encoder = ce.OrdinalEncoder(cols=['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
                                    'pedigree', 'age'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

vi. Building decision tree classifier with the Gini index criterion

```
clf_gini = DecisionTreeClassifier(criterion='gini',
                                  max_depth=4,
                                  random_state=0) # Create decision tree
                                                  classifier object
clf_gini.fit(X_train, y_train) # Train the classifier
```

vii. Predicting results for the test set

```
y_pred = clf_gini.predict(X_test)
```

viii. Evaluating model

```
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
```

ix. Confusion matrix

A confusion matrix is a matrix that can be used to measure the performance of an machine learning algorithm, usually a supervised learning one. In general, each row of the confusion matrix represents the instances of an actual class and each column represents the instances of a predicted class, but it can be the other way around as well.

Four types of outcomes are possible while evaluating a classification model performance:

- **True Positives (TP)** – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.
- **True Negatives (TN)** – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.
- **False Positives (FP)** – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error**.
- **False Negatives (FN)** – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error**.

```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, y_pred)
```

7. Optimizing decision tree performance

In scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In addition to the pre-pruning parameters, other attribute selection measures such as entropy can be used.

- **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. (Source: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

8. Visualizing decision trees

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

dot_data = StringIO()
export_graphviz(clf_gini,
                out_file=dot_data,
                filled=True,
                rounded=True,
                special_characters=True,
                feature_names=X.columns,
                class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

The output visualization for the decision tree classifier discussed in Section 6 is represented in Figure 3.

9. Classification report

Classification report is another way to evaluate the classification model performance. It displays the **precision**, **recall**, **f1**, and other support scores for the model.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

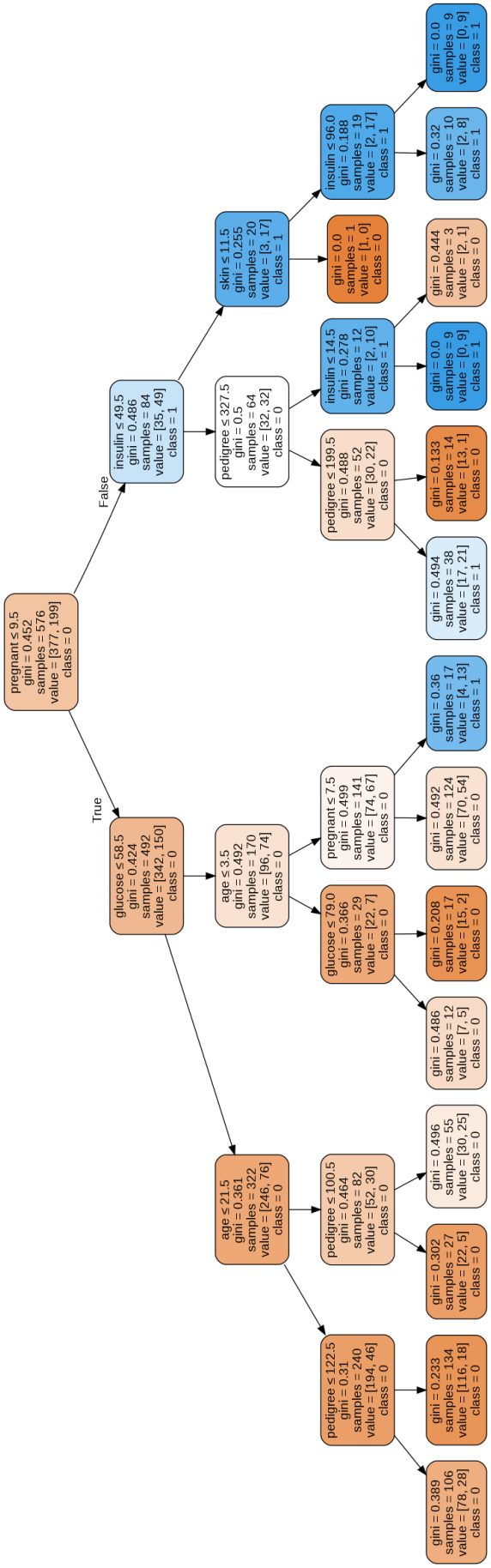

Figure 3. Visualization of the decision tree classifier with the Gini index criterion

Report

Write a short report based on the following tasks. The report should be submitted as a PDF file and should not longer than four pages. Rename it as XXYYYlab05.pdf, where XX indicates your batch and YYY indicates your registration number.

Task 1: Build two decision tree classifiers with **Gini index** and **entropy** criteria for the given **Wine.csv** data set. More information on the dataset is available on `UCI Machine Learning Repository` (source: `https://archive.ics.uci.edu/ml/datasets/Wine`).

Task 2: Demonstrate how decision trees deal with missing values.

Task 3: Evaluate the classifiers with suitable performance metrices.

Task 4: Demonstrate how pruning can be applied to overcome overfitting of decision tree classifiers.

Task 5: Visualize decision trees.

***** End of Labsheet *****