Faculty of Computer Engineering

First Project (decision tree)

Artificial Intelligence Lesson

Pooria Rahimi - 99521289

1402-1403

- At first, to do this project, we need a series of libraries that we have to import first, which is according to the picture below:

```
import numpy as np
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
import matplotlib.image as mpimg
from tqdm import tqdm
import nltk
import spacy
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
```

Step1 : Then we form the node class.

- The node class represents each node in the tree, which can take different values depending on the location of that tree (leaf or middle node).

```
class Node:
    def __init__(self, feature = None, threshold = None, data_left = None , data_right = None, gain = None, value = None , gini = None):
        self.feature = feature
        self.threshold = threshold
        self.data_left = data_left
        self.data_right = data_right
        self.gain = gain
        self.value = value
        self.gini = gini
```

Step2 : Then it is time to implement the decision tree class.

- The decision tree class includes the general features of the tree and its different functions.

```
class DecisionTree:
    def __init__(self, min_samples_split=25, max_depth=100):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.root = None
```

Step3 : Then it is time to implement the Entropy function.

- Entropy is a method of decision tree class which is implemented according to the following formula:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

- The information gain method also calculates the gain of each node

```
    def _information_gain(self, parent, left_child, right_child):
        num_left = len(left_child) / len(parent)
        num_right = len(right_child) / len(parent)
        return self._entropy(parent) - (num_left * self._entropy(left_child) + num_right * self._entropy(right_child))
```

Step4 : Now, in this section, we will implement the main part of our code.

- In the main part of our program, we implement the best splite method.
- In this method, we intend to do the best division among the children.
- The working process of this function is that we separate children according to different thresholds defined for each feature and calculate information gain and gini index for each separation in order to make the best choice according to these parameters. Let's get in isolation.
- It should be noted that this function simulates the importance function in slides, which is as follows:

```python
def _best_split(self, X, y):
    best_split = {}
    best_info_gain = -1
    best_gini = 1.0
    n_rows, n_cols = X.shape
    for f_idx in range(n_cols):
        X_curr = X[:, f_idx]
        thresholds = np.unique(X_curr)
        for threshold in thresholds:
            df_left = np.concatenate((X[X_curr <= threshold], y[X_curr <= threshold].reshape(-1, 1)), axis=1)
            df_right = np.concatenate((X[X_curr > threshold], y[X_curr > threshold].reshape(-1, 1)), axis=1)
            if len(df_left) > 0 and len(df_right) > 0:
                y = np.concatenate((X, y.reshape(-1, 1)), axis=1)[:, -1]
                y_left = df_left[:, -1]
                y_right = df_right[:, -1]
                gain = self._information_gain(y, y_left, y_right)
                gini = self._gini_index(X, y, f_idx, threshold)
                if gini < best_gini:
                    best_gini = gini
                #     best_feature = f_idx
                #     best_threshold = threshold

                # if gain > best_info_gain:
                    best_split = {
                        'feature_index': f_idx,
                        'threshold': threshold,
                        'df_left': df_left,
                        'df_right': df_right,
                        'gain': gain,
                        'gini': gini
                    }
                    # best_info_gain = gain
    return best_split
```

- The gini impurity function calculates the gini value according to the following formula:

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

```
Codiumate: Options | Test this method
def _gini_impurity(self, y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    gini = 1 - np.sum(probabilities ** 2)
    return gini
```

- The gini_index method multiplies the probability of data occurrence in gini obtained in gini_impurity, which is as follows:

Codiumate: Options | Test this method

```python
def _gini_index(self, X, y, feature_index, threshold):
    left_indices = X[:, feature_index] <= threshold
    right_indices = X[:, feature_index] > threshold

    left_labels = y[left_indices]
    right_labels = y[right_indices]

    left_gini = self._gini_impurity(left_labels)
    right_gini = self._gini_impurity(right_labels)

    num_left = len(left_labels)
    num_right = len(right_labels)
    total_samples = num_left + num_right

    gini_index = (num_left / total_samples) * left_gini + (num_right / total_samples) * right_gini
    return gini_index
```

- Then, in the function implemented below, we try to normalize the data:
- For example, in this part, we label the dataset data:

```python
columns_to_convert = ['type','nameOrig','nameDest']
label_encoder = LabelEncoder()

for column in columns_to_convert:
    data[column] = label_encoder.fit_transform(data[column])
```

- Now, we may have none data or empty data. In this section, to solve this problem, we put the average amount of available data for this feature, which is as follows:

```python
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

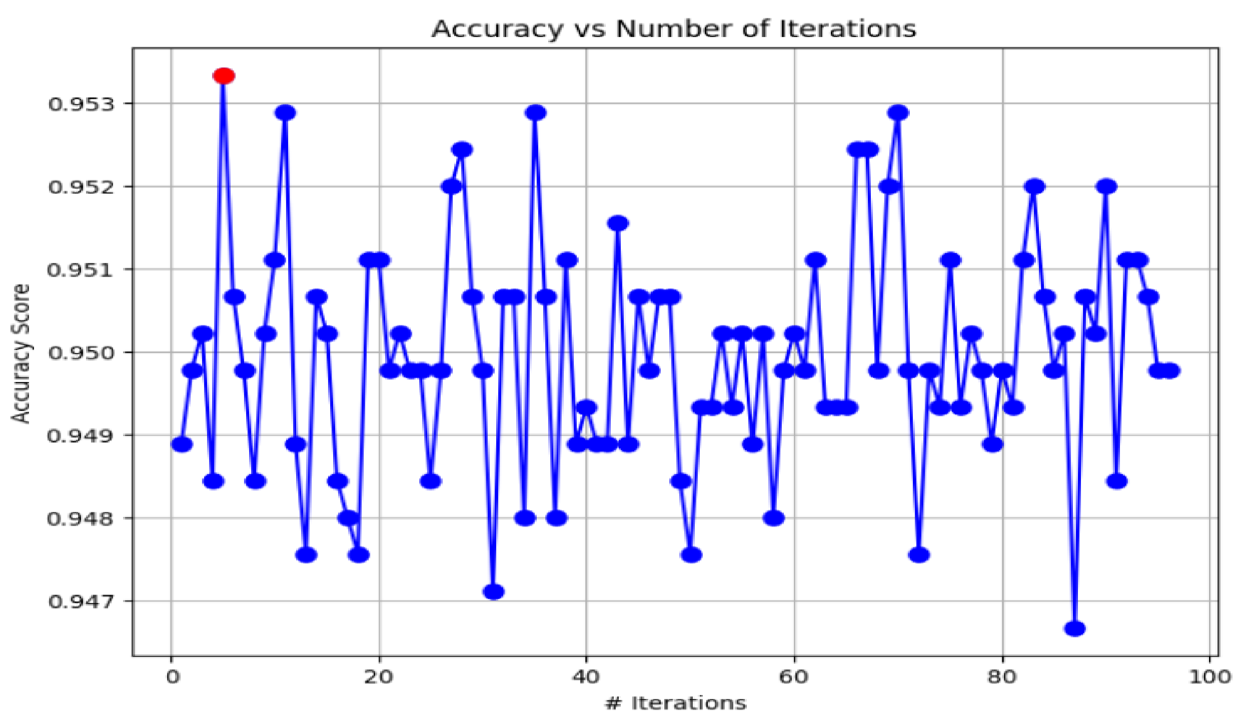
- And finally, to test the designed model, we compare it with the model implemented in sikit learn, and the accuracy percentage of our function and the ready decision tree is as follows, and we also used the accuracy_score function to test the accuracy, and the accuracy of the model You can see the designed and ready model below:

```
max_index = np.argmax(accuracy_scores)
max_acc = accuracy_scores[max_index]

print("\nOptimal iteration:", max_index + 1)
print("Maximum Accuracy:", max_acc*100)
```

[59]

```
...
Optimal iteration: 5
Maximum Accuracy: 95.33333333333334
```



```
model = DecisionTree()
model.fit(X_train, y_train)
preds = model.predict(X_test)
```

[65]

```
accuracy_score(y_test, preds)
```

[66]

```
0.9736842105263158
```

The End.