به نام خدا

پروژه اول درس هوش مصنوعی – درخت تصمیم

بهاره کاوسی نژاد – 99431217

```python
import numpy as np
import pandas as pd
import random
```

ابتدا کتابخانه های لازم را import می‌کنیم.

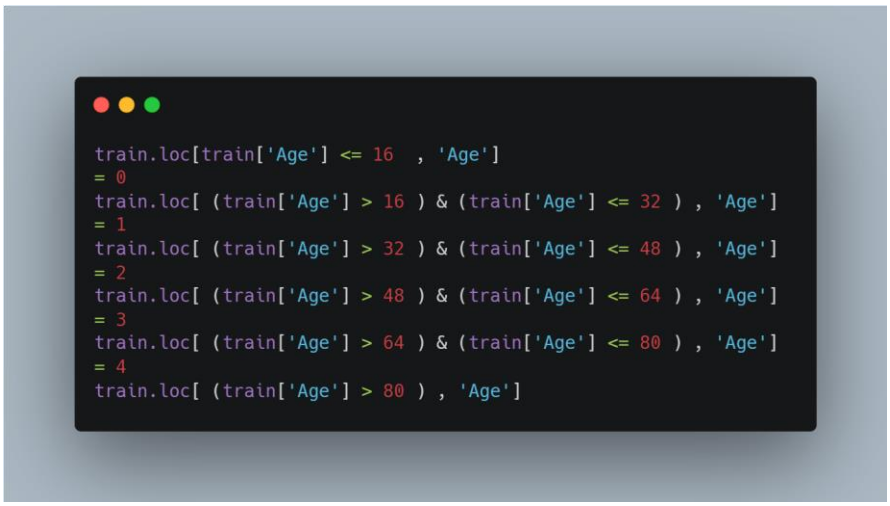
```python
train = pd.read_csv('Airplane.csv')

train = train.drop(train.columns[0], axis=1)
train = train.drop("id", axis='columns')

print (train.info())
```

با دستور pd.read_csv فایل csv را می‌خوانیم و دو ستون اول آن را حذف می‌کنیم زیرا شماره سطر و id هستند که attribute نیستند و با دستور info اطلاعات attribute ها را چاپ می‌کنیم.

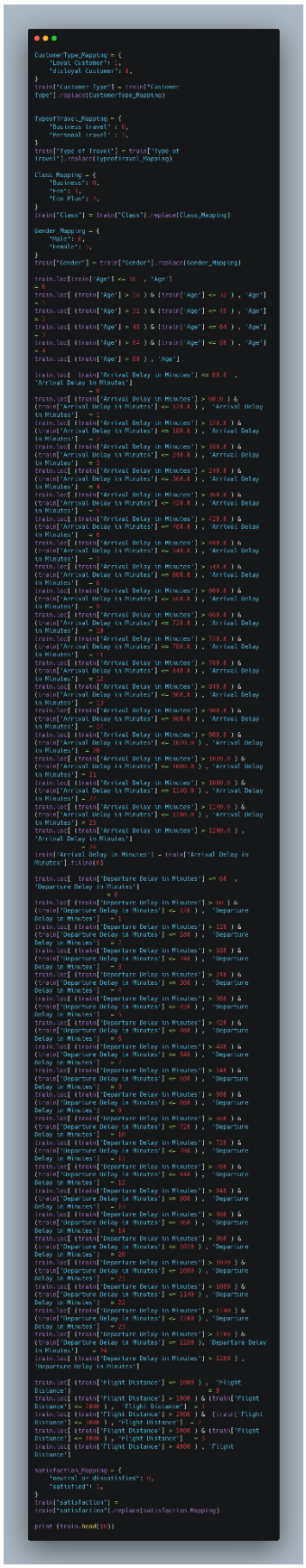
در مرحله بعد باید attribute ها را map کنیم و به آنها مقادیر عددی نسبت دهیم. به عنوان مثال mapping ویژگی Customer Type به صورت زیر خواهد بود:

```
CustomerType_Mapping = {
    "Loyal Customer": 1,
    "disloyal Customer": 0,
}
train["Customer Type"] = train["Customer
Type"].replace(CustomerType_Mapping)
```

همچنین attributeهایی مانند Age و Arrival Delay in Minutes را به صورت بازه map می‌کنیم:



```
train.loc[train['Age'] <= 16  , 'Age']
= 0
train.loc[ (train['Age'] > 16 ) & (train['Age'] <= 32 ) , 'Age']
= 1
train.loc[ (train['Age'] > 32 ) & (train['Age'] <= 48 ) , 'Age']
= 2
train.loc[ (train['Age'] > 48 ) & (train['Age'] <= 64 ) , 'Age']
= 3
train.loc[ (train['Age'] > 64 ) & (train['Age'] <= 80 ) , 'Age']
= 4
train.loc[ (train['Age'] > 80 ) , 'Age']
```



```python
class Node:
    def __init__(self, attribute=None, value=None, label=None):
        self.attribute = attribute
        self.value = value
        self.label = label
        self.children = {}

    def add_child(self, value, child_node):
        self.children[value] = child_node
```

کلاس Node را تعریف می‌کنیم:

این کلاس دارای چهار variable است:

- attribute: ویژگی در این node
- value: مقدار ویژگی (برای node های غیر برگ)
- label: مقدار label (برای node های برگ)
- children: دیکشنری برای ذخیره کردن node های child

همچنین این کلاس یک method دارد که برای اضافه کردن child استفاده می‌شود.

```python
def PLURALITY_VALUE(examples):
    return np.argmax(np.bincount(examples.iloc[:, -1]))
```

از تابع PLURALITY_VALUE برای پیدا کردن value ای که بیشترین تکرار را در examples دارد استفاده می‌شود.

```python
def find_attribute_index(attribute, attributes):
    index = attributes.index(attribute)
    return index
```

از تابع find_attribute_index برای پیدا کردن index یک ویژگی در لیست همه ویژگی‌ها استفاده می‌شود.

```python
def select_best_attribute_entropy(attributes, examples):
    best_attribute = None
    best_gain = -1
    for attribute in attributes:
        entropy = entropy_help(examples[attribute])
        if best_attribute is None or entropy < best_gain:
            best_attribute = attribute
            best_gain = entropy
    return best_attribute

def entropy_help(examples):
    unique_values, counts = np.unique(examples,
return_counts=True)
    probabilities = counts / len(examples)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy
```

تابع select_best_attribute_entropy به کمک تابع entropy_help بهترین ویژگی را بر اساس entropy مشخص می‌کند.

```
def select_best_attribute_GiniIndex(attributes, examples):
    best_attribute = None
    best_gini_index = float('inf')

    for attribute in attributes:
        attribute_values = np.unique(examples.iloc[:,
find_attribute_index(attribute, attributes)])
        attribute_gini_index = 0

        for value in attribute_values:
            value_examples = examples[examples.iloc[:,
find_attribute_index(attribute, attributes)] == value]
            value_prob = len(value_examples) / len(examples)
            value_gini_index =
GiniIndex_help(value_examples.iloc[:, -1])

            attribute_gini_index += value_prob *
value_gini_index

        if attribute_gini_index < best_gini_index:
            best_gini_index = attribute_gini_index
            best_attribute = attribute

    return best_attribute

def GiniIndex_help(labels):
    labels = labels.astype(int)
    label_counts = np.bincount(labels)
    label_probs = label_counts / len(labels)
    gini_index = 1 - np.sum(label_probs ** 2)
    return gini_index
```

تابع select_best_attribute_GiniIndex به کمک تابع GiniIndex_help بهترین ویژگی را بر اساس Gini Index مشخص می‌کند.

```
attributes = train.columns
```

ویژگی‌ها را در متغیر attributes ذخیره می‌کنیم.

```
def LEARN_DECISION_TREE(examples, attributes, parent_examples):
    if len(examples) == 0:
        return Node(label=PLURALITY_VALUE(parent_examples))

    elif np.all(examples.iloc[:, -1] == examples.iloc[0, -1]):
        return Node(label=examples.iloc[0, -1])

    elif len(attributes) == 0:
        return Node(label=PLURALITY_VALUE(examples))

    else:
        A = select_best_attribute_GiniIndex(attributes,
examples)
        # A = select_best_attribute_entropy(attributes,
examples)
        A_index = find_attribute_index(A, attributes)
        A_values = examples.iloc[:, A_index].unique()
        # A_index = examples.columns.get_loc(attribute)
        tree = Node(attribute=A)

        for value in A_values:
            exs = examples[examples.iloc[:, A_index] == value]
            subtree = LEARN_DECISION_TREE(exs,
attributes[:A_index] + attributes[A_index+1:], examples)
            tree.add_child(value, subtree)

        return tree
```

تابع LEARN_DECISION_TREE را پیاده سازی می‌کنیم. این تابع یک الگوریتم بازگشتی است که یک درخت تصمیم بر اساس نمونه‌ها و ویژگی‌ها تشکیل می‌دهد. این تابع 4 شرط اصلی دارد:

- اگر هیچ exampleای نباشد، به این معنا است که به برگ رسیده‌ایم. در این شرایط، تابع یک node را برمی‌گرداند که label آن توسط مقدار plurality در parent example محاسبه می‌شود.
- اگر همه exampleها مقدار target یکسانی داشته باشند، بدین معنا است که به برگی رسیده‌ایم که همه exampleها دارای یک کلاس هستند. در این شرایط، تابع یک node را برمی‌گرداند که label آن توسط مقدار target مشخص می‌شود.
- اگر ویژگی دیگری باقی نمانده باشد، یعنی ما همه ویژگی‌ها را برای درخت استفاده کرده‌ایم. در این شرایط، تابع یک node را برمی‌گرداند که label آن مقدار plurality در current example است.
- اگر هیچ یک از شرایط بالا برقرار نبود، تابع به صورت بازگشتی به ساختن درخت تصمیم می‌پردازد:
  - در ابتدا، بهترین ویژگی را بر اساس Gini Index یا Entropy انتخاب می‌کند.
  - سپس یک Node (درخت) با ویژگی‌های A می‌سازد.
  - مقادیر یکتای A را پیمایش می‌کند و برای هر مقدار، یک subtree می‌سازد (به صورت بازگشتی)
  - Subtree ساخته شده به عنوان child به current node با مقدار A اضافه می‌شود.
- در انتها درخت ساخته شده برگردانده می‌شود.

```python
def print_tree(node, indent='', file=None):
    if node.label is not None:
        output = indent + "Label: " + str(node.label)
        if file:
            print(output, file=file)
        else:
            print(output)
    else:
        output = indent + "Attribute: " + str(node.attribute)
        if file:
            print(output, file=file)
        else:
            print(output)
        for value, child_node in node.children.items():
            output = indent + "Value: " + str(value)
            if file:
                print(output, file=file)
            else:
                print(output)
            print_tree(child_node, indent + "  ", file)
```

از این تابع برای چاپ کردن درخت تصمیم و ذخیره آن در یک فایل استفاده می‌شود.

```python
def predict(node, instance):
    if node.label is not None:
        return node.label
    attribute_value = instance[attributes.index(node.attribute)]
    if attribute_value in node.children:
        child_node = node.children[attribute_value]
        return predict(child_node, instance)
    else:
        return None
```

از تابع predict برای تعیین مقادیر یک نمونه جدید با استفاده از decision tree استفاده می‌شود. این تابع دو ورودی node و instance دارد که node نمایانگر current node در درخت تصمیم است و instance نمایانگر نمونه جدید است.

این تابع از سه قسمت اصلی تشکیل شده است:

- اگر current node دارای label باشد، بدین معنا است که به برگ رسیده‌ایم. در این شرایط، تابع label مربوط به node که همان predicted class برای نمونه داده شده است را برمی‌گرداند.
- اگر current node دارای label نباشد، بدین معنا است که باید درخت را بیشتر پیمایش کنیم:
  - تابع مقدار attribute را پیدا می‌کند.
  - بررسی می‌کند که آیا این مقدار attribute یکی از childهای current node است یا خیر. اگر باشد، تابع به صورت بازگشتی صدا زده می‌شود و اگر نباشد، بدین معنا است که مقدار attribute وجود ندارد و تابع None را برمی‌گرداند.
- در انتها، اگر تابع به برگ رسیده باشد، label را برمی‌گرداند و در غیر این صورت None را برمی‌گرداند.

```python
def test_decision_tree(tree, test_data):
    predictions = []
    for instance in test_data:
        prediction = predict(tree, instance)
        predictions.append(prediction)
    return predictions
```

از این تابع برای ذخیره کردن پیش بینی‌ها استفاده می‌شود.

```python
def calculate_accuracy(predictions, actual_labels):
    correct_predictions = 0
    total_predictions = len(predictions)

    for i in range(total_predictions):
        if predictions[i] == actual_labels[i]:
            correct_predictions += 1

    accuracy = correct_predictions / total_predictions
    return accuracy
```

این تابع مقادیر پیش بینی شده را با مقادیر واقعی مقایسه کرده و دقت درخت را محاسبه می‌کند.

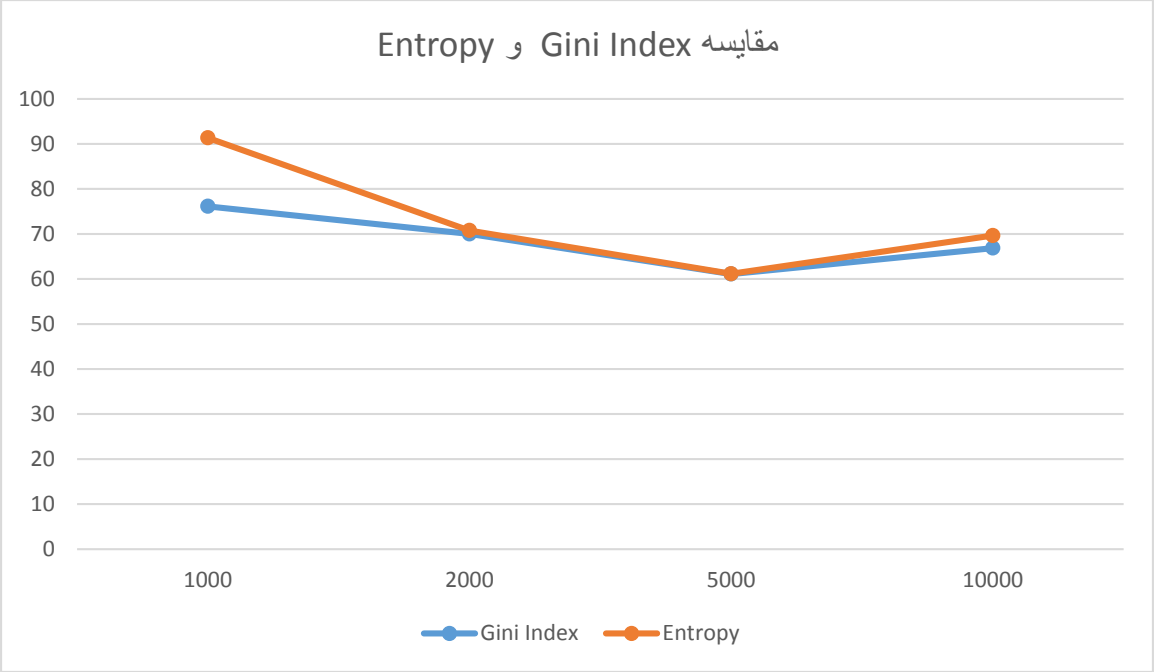
```python
def random_rows_after_2000(data_frame, num_rows):
    total_rows = data_frame.shape[0]
    eligible_indices = list(range(2001, total_rows))

    selected_indices = random.sample(eligible_indices, num_rows)
    selected_rows = data_frame.iloc[selected_indices]

    return selected_rows
```

از این تابع برای انتخاب نمونه‌های رندوم پس از نمونه 2000ام (زیرا 2000 نمونه اول برای تست هستند) برای train کردن درخت استفاده می‌شود.

```python
train_data = random_rows_after_2000(train, 5000)
target = train_data.iloc[:, -1]
attributes = list(attributes)
attributes = attributes[:-1]
DT = LEARN_DECISION_TREE(train_data, attributes, target)

print("Decision Tree:")
print_tree(DT)

with open("tree_output.txt", "w") as file:
    print_tree(DT, file=file)

NumberOfTestData = 2000
test_data_results = np.array(train.head(NumberOfTestData))[:,
-1]
test_data = np.array(train.head(NumberOfTestData))[:, :-1]
predictions = test_decision_tree(DT, test_data)
print("Predictions:")
print(predictions)

accuracy = calculate_accuracy(predictions, test_data_results)
print("Accuracy:", accuracy * 100)
```

نمونه‌هایی که برای train استفاده می‌شوند را در train_data و ستون آخر که همان هدف ماست را در target ذخیره می‌کنیم. همچنین ویژگی‌ها را به لیست تبدیل کرده و آخرین مقدار آن که همان target است را حذف می‌کنیم. سپس درخت تصمیم را می‌سازیم و آن را چاپ کرده و در فایلی به نام tree_output.txt ذخیره می‌کنیم.

تعداد 2000 تا از ابتدای نمونه‌ها به عنوان نمونه تست انتخاب کرده و درخت تصمیم را تست می‌کنیم.


مقایسه Gini Index و Entropy
100
90
80
70
60
50
40
30
20
10
0
1000
2000
5000
10000
Gini Index
Entropy