به نام خدا

گزارش پروژه اول – درخت تصمیم **نیایش خانی ۹۹۵۲۱۲۳۵**

پروژه‌ای که در دست انجام داشتم با جستجو درباره‌ی درخت تصمیم و روش‌های Entropy و Gini آغاز شد. پس از مطالعه‌ی این موضوعات، به بررسی نمونه‌هایی از پروژه‌های مشابه پرداختم؛ با این حال، متوجه شدم که در همه‌ی آن‌ها از کتابخانه‌ی درخت تصمیم SKlearn استفاده شده است. به همین دلیل، بخش چالش‌برانگیز پروژه برای من، پیاده‌سازی درخت تصمیم‌گیری و سپس، روش‌های چاپ و نمایش آن بود.

در ادامه، به گسسته‌سازی داده‌ها پرداختم.

```
data = pd.read_csv('onlinefraud.csv')

# Convert categorical variables into numerical
Type_Mapping = {
    "PAYMENT": 0,
    "TRANSFER": 1,
    "CASH_OUT": 2,
    "DEBIT": 3,
    "CASH_IN" : 4
}
data["type"] = data["type"].replace(Type_Mapping)

# Discretize continuous
bins = [0, 1000, 10000, 100000, 1000000, np.inf]
data["amount"] = np.digitize(data["amount"], bins) - 1
data["oldbalanceOrg"] = np.digitize(data["oldbalanceOrg"], bins) - 1
data["newbalanceOrig"] = np.digitize(data["newbalanceOrig"], bins) - 1
data["oldbalanceDest"] = np.digitize(data["oldbalanceDest"], bins) - 1
data["newbalanceDest"] = np.digitize(data["newbalanceDest"], bins) - 1

data = data.drop('nameOrig', axis=1)
data = data.drop('nameDest', axis=1)
```

برای تبدیل داده "تایپ" به مقادیر عددی، هرکدام از تایپ ها را به یک عدد مپ کرده و با داده قبلی جا به جا می کنیم.

سپس متغیرهای پیوسته را به دسته‌هایی تقسیم می‌کنیم (یا به عبارت دیگر، آن‌ها را گسسته می‌کنیم). برای مثال، مقادیر متغیر ”amount“ را بر اساس بازه های مشخص شده در bins به یکی از دسته‌های ۰ تا ٤ تقسیم می‌کنیم. بنابراین مقادیری که کمتر از ۱۰۰۰ هستند در دسته صفر، مقادیر بین ۱۰۰۰ تا ۱۰۰۰۰ در دسته یک و به همین ترتیب سایر مقادیر دسته بندی می شوند.

در آخر ستون‌های ’nameOrig‘ و ’nameDest‘ را حذف می‌کنیم زیرا این اطلاعات برای مدل پیش‌بینی مورد نیاز نیست.

```python
class DecisionTreeNode:
    def __init__(self, attribute=None, threshold=None, label=None):
        self.split_attribute = attribute  # Attribute used for splitting the node
        self.split_threshold = threshold  # Value used for splitting the node
        self.label = label  # Class label if the node is a leaf, None otherwise
        self.child_nodes = {}  # Dictionary to hold the child nodes

    def is_leaf_node(self):
        """Check if the node is a leaf node (i.e., doesn't have any children)"""
        return not bool(self.child_nodes)

    def add_child(self, attribute_value, node):
        """Add a child node for a specific attribute value"""
        self.child_nodes[attribute_value] = node
```

این کلاس یک گره در درخت تصمیم را نمایش می‌دهد که شامل متدها و خصوصیات زیر است:

- Attribute : این خصوصیت برای نگهداری نام ویژگی (یا ستون داده) است که برای تقسیم گره استفاده می‌شود.
- Threshold : این خصوصیت برای نگهداری مقدار ویژگی است که برای تقسیم گره استفاده می‌شود.
- Label : اگر گره یک گره برگ باشد، این خصوصیت لیبل کلاس را نگه می‌دارد.
- Child nodes : یک دیکشنری است که گره‌های فرزند را نگه می‌دارد. کلیدهای دیکشنری مقادیر ویژگی هستند و مقادیر دیکشنری گره‌های فرزند هستند.

- is_leaf_node(self) : این متد بررسی می‌کند که آیا گره فعلی یک گره برگ است یا خیر. اگر گره فرزندی نداشته باشد، برگ است و Trueبرمی‌گرداند. در غیر این صورت، False برمی‌گرداند.

- add_child(self, attribute_value, node) : این متد یک گره فرزند جدید به گره فعلی اضافه می‌کند

- attribute_value مقدار ویژگی است که به گره فرزند منتسب شده و nodeخود گره فرزند است.

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

این تابع لیبلی که بیشترین تکرار را در داده‌های ورودی دارد را برمی‌گرداند.

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

از این تابع نیز برای پیدا کردن index یك ویژگی در لیست همه ویژگی ها استفاده می شود.

در ادامه از دو تابع find_best_entropy و find_best_gini برای پیدا کردن بهترین ویژگی بر اساس entropy و gini استفاده می کنیم. پیاده سازی این دو تابع به کمک اسلاید های درس و منابع موجود در اینترنت صورت گرفته است.

سپس به قسمت چالش برانگیز پروژه که پیاده سازی درخت تصمیم است می رسیم.

```
def decision_tree(data_samples, attribute_list, parent_samples, criterion):
    if len(data_samples) == 0:
        return
DecisionTreeNode(label=determine_most_common_value(parent_samples))

    elif len(data_samples['isFraud'].unique()) == 1:
        return DecisionTreeNode(label=data_samples['isFraud'].values[0])

    elif len(attribute_list) == 0:
        return DecisionTreeNode(label=determine_most_common_value(data_samples))

    else:
        if criterion == 'entropy':
            best_attribute = find_best_entropy(attribute_list, data_samples)
        else:  # Default to Gini index if no valid criterion is specified
            best_attribute = find_best_gini(attribute_list, data_samples)

        best_attribute_index = find_attribute_index(best_attribute,
attribute_list)
        best_attribute_values = data_samples.iloc[:,
best_attribute_index].unique()

        tree = DecisionTreeNode(attribute=best_attribute)

        for value in best_attribute_values:
            subset_data_samples = data_samples[data_samples.iloc[:,
best_attribute_index] == value]

            subtree = decision_tree(subset_data_samples,
attribute_list[:best_attribute_index] + attribute_list[best_attribute_index+1:],
data_samples, criterion)

            tree.add_child(value, subtree)

        return tree
```

این تابع یک درخت تصمیم را با استفاده از الگوریتم بازگشتی می‌سازد.

- if len(data_samples) == 0 : اگر هیچ داده‌ای برای تجزیه وجود نداشت، یک گره برگ با لیبلی که در نمونه‌های والد بیشترین تکرار را دارد ایجاد می‌کند.
- elif len(data_samples['isFraud'].unique()) == 1: اگر تمام داده‌ها به یک کلاس تعلق داشته باشند، یک گره برگ با آن لیبل ایجاد می‌کند.
- elif len(attribute_list) == 0 : اگر هیچ ویژگی دیگری برای بررسی وجود نداشت، یک گره برگ با لیبلی که در داده‌های فعلی بیشترین تکرار را دارد ایجاد می‌کند.
- else : در غیر این صورت، یک گره داخلی بر اساس بهترین ویژگی که بر اساس Entropy یا Gini انتخاب شده است را برای تقسیم ایجاد می‌کند. برای هر مقدار از بهترین ویژگی، یک زیردرخت را به صورت بازگشتی می‌سازد و به شاخه مربوطه اضافه می‌کند.

این تابع در نهایت درخت تصمیم را برمی‌گرداند.

در ادامه، با استفاده از تابع print_tree، درخت حاصل را چاپ می‌کنیم. متأسفانه، امکان استفاده از ابزارهای پیشرفته‌تر برای نمایش گرافیکی بهتر درخت وجود نداشت، زیرا این کتابخانه‌ها تنها با تابع درخت مخصوص به خود قابل استفاده هستند.

تابع predict یک پیش‌بینی را بر اساس یک درخت تصمیم و یک نمونه داده انجام می‌دهد.

```
def predict(node, instance, attribute_list):
    # If the node is a leaf node, return its label
    if node.is_leaf_node():
        return node.label
    # Get the value of the splitting attribute for the instance
    attribute_value = instance[attribute_list.index(node.split_attribute)]
    # If the attribute value is in the child nodes of the current node
    if attribute_value in node.child_nodes:
        # Get the child node corresponding to the attribute value
        child_node = node.child_nodes[attribute_value]
        # Recursively predict the class label for the instance
        return predict(child_node, instance, attribute_list)
    else:
        # If the attribute value is not in the child nodes, return None
        return None
```

اگر گره فعلی یک گره برگ باشد (یعنی هیچ فرزندی نداشته باشد)، لیبل آن گره برگ به عنوان پیش‌بینی برمی‌گردد.

سپس تابع مقدار attribute را پیدا می کند. اگر مقدار ویژگی در گره‌های فرزند گره فعلی وجود داشته باشد، گره فرزند مربوط به آن مقدار ویژگی انتخاب می‌شود. سپس تابع predict به صورت بازگشتی بر روی گره فرزند فراخوانی می‌شود تا لیبل کلاس را برای نمونه داده پیش‌بینی کند. اگر مقدار ویژگی در گره‌های فرزند گره فعلی وجود نداشته باشد، تابع None برمی‌گرداند به این معنا که درخت تصمیم قادر به پیش‌بینی لیبل کلاس برای نمونه داده نیست.

```
def test_decision_tree(decision_tree, test_data, attribute_list):
    predictions = []
    for instance in test_data:
        predicted_label = predict(decision_tree, instance, attribute_list)
        predictions.append(predicted_label)

    return predictions
```

تابع test_decision_tree لیبل کلاس را برای هر نمونه در داده‌های آزمون پیش‌بینی می‌کند. برای هر نمونه در داده‌های تست، لیبل کلاس را با استفاده از تابع predict و درخت تصمیم پیش‌بینی می‌کند. سپس برچسب پیش‌بینی شده را به لیست پیش‌بینی‌ها اضافه می‌کند و آن را برمی‌گرداند.

سپس برای محاسبه دقت درخت در تابع calculate_accuracy مقادیر پیش بینی شده را با مقادیر واقعی مقایسه می کنیم.

در ادامه کد زیر را داریم.

```
num_train_samples = 2000
train_data = data.iloc[:num_train_samples]
test_data = data.iloc[num_train_samples:]
train_target = train_data.iloc[:, -1]
train_predictors = train_data.iloc[:, :-1]

attribute_list = list(train_data.columns)[:-1]

decision_tree_model = decision_tree(data_samples=train_data,
                                    attribute_list=attribute_list,
                                    parent_samples=None,
                                    criterion="gini")

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

with open("tree.txt", "w") as file:
    print_tree(decision_tree_model, file=file)

test_data_results = test_data.iloc[:, -1].values
test_data_predictors = test_data.iloc[:, :-1].values

predictions = test_decision_tree(decision_tree_model, test_data_predictors,
attribute_list)
print("Predictions:")
print(predictions)
accuracy = calculate_accuracy(predictions, test_data_results)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

در این کد ابتدا تعداد داده‌های train را ۲۰۰۰ داده و بقیه داده‌ها را به عنوان داده‌های تست در نظر گرفته ایم. سپس متغیر هدف را از پیش‌بینی‌کننده‌ها جدا می‌کنیم. لیست ویژگی‌ها را نیز به استثنای ویژگی هدف تعریف می کنیم. در ادامه با استفاده از داده‌های train، یک درخت تصمیم ساخته و ضمن چاپ کردن، آن را در یک فایل تکست ذخیره می کنیم. سپس داده های تست را با استفاده از درخت تصمیم پیش بینی کرده و دقت پیشبینی را محاسبه می کنیم.

مقدار دقت بدست آمده در دو متود Entropy و Gini index با هم متفاوت است. معیارهای Gini Index و Entropyدر الگوریتم‌های درخت تصمیم برای انتخاب بهترین ویژگی برای تقسیم داده‌ها استفاده می‌شوند. هر دوی این معیارها نشان‌دهنده درجه ناخالصی یا ناهمگونی داده‌ها در یک گره خاص هستند. با این حال، روش محاسبه آن‌ها متفاوت است:

- **Gini Index** : این معیار نشان‌دهنده احتمال اشتباه بودن یک لیبل که به صورت تصادفی به یک نمونه اختصاص داده شده است. اگر تمام نمونه‌ها در یک گره به یک کلاس تعلق داشته باشند (یعنی گره کاملا خالص باشد)، Gini Indexبرابر با صفر خواهد بود .اگر نمونه‌ها به طور مساوی بین کلاس‌ها تقسیم شوند (یعنی گره کاملا ناخالص باشد)، Gini Indexبرابر با ۰,۵ خواهد بود.
- **Entropy** : این معیار نشان‌دهنده درجه ناهمگونی یا ناخالصی داده‌ها در یک گره است. اگر تمام نمونه‌ها در یک گره به یک کلاس تعلق داشته باشند (یعنی گره کاملا خالص باشد)، Entropyبرابر با صفر خواهد بود .اگر نمونه‌ها به طور مساوی بین کلاس‌ها تقسیم شوند (یعنی گره کاملا ناخالص باشد)، Entropyبرابر با ۱ خواهد بود.

به طور کلی، هر دو معیار معمولا نتایج مشابهی در الگوریتم‌های درخت تصمیم ارائه می‌دهند .با این حال، محاسبه Entropy نسبت به Gini Index بیشتر هزینه محاسباتی دارد.

برای افزایش دقت راه های متفاوتی داریم. مثلا می تونیم با هرس کردن از بیش‌برازش (overfitting) جلوگیری کنیم. همچنین می توانیم با روش های random forest یا boosting چندین درخت تصمیم را ترکیب کنیم تا دقت را افزایش دهیم. راه حل دیگر پیش پردازش داده ها است. مثلا با انتخاب ویژگی، کاهش ابعاد یا نرمال سازی می توانیم به بهبود عملکرد درخت تصمیم کمک کنیم.

همچنین برای افزایش دقت تبدیل بازه ها هم می توانیم ابتدا دیتاها را سورت کنیم و سپس آنهایی که پشت سر هم مقادیر خروجی یکسانی داشتند را در یک گروه قرار دهیم و گروه ها را بر اساس فاکتورهای مختلف هرس کنیم.

پ.ن : کامنت های موجود در کد برای تمیزی بیشتر و کوتاهی گزارش حذف شده اند. 😊