Báo cáo Project

Lớp TTNT-154016, Nhóm G07

1. Thông tin chung

Thành viên

- Đỗ Huy Đạt 20220024
- Đoàn Nguyễn Hải Nam 20220035
- Lê Minh Triết 20220045
- Đàm Hồng Thái 20183625

2. Đề xuất project (W2-3)

Bài toán

Phân loại Email Spam

Phương pháp

Sử dụng Decision Tree

Phân công

- ĐH Đạt: Xây dựng Decision Tree, đánh giá kết quả
- ĐNH Nam: Chuyển đổi dữ liệu thành vector, xây dựng Decision Tree
- LM Triết: Xây dựng Decision Tree, đánh giá kết quả
- ĐH Thái: Làm sạch, phân tích dữ liệu

3. Tiến độ giữa kỳ (W9)

Chương trình

Trước hết, nhóm xây dựng một Decision Tree đơn giản dựa trên chỉ số Gini để tạo cây. Mã nguồn lưu trong g7_decision_tree.py.

Mã nguồn cũng được triển khai trong cell dưới đây

In [1]:

```python
import numpy as np

# Class đại diện cho một nút trong cây quyết định
class TreeNode:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=N
        self.feature = feature      # Đặc trưng sử dụng để chia dữ liệu
        self.threshold = threshold  # Ngưỡng sử dụng để chia dữ liệu
        self.left = left            # Con trỏ tới nút con bên trái
        self.right = right          # Con trỏ tới nút con bên phải
        self.value = value          # Giá trị của nút nếu là nút lá

# Hàm tính chỉ số Gini để đo độ thuần nhất của nút
def gini(y):
    _, counts = np.unique(y, return_counts=True)  # Tìm các lớp và số lượng phần tử
    gini = 1.0 - sum((count / len(y)) ** 2 for count in counts)  # Tính chỉ số Gini
    return gini

class G07DecisionTree():
    def __init__(self, max_depth=10):
        self.max_depth = max_depth
        self.tree = TreeNode()

    # Hàm fit tree với datasets
    def fit(self, X, y):
        self.tree = self.build_tree_(X, y, depth=0, max_depth=self.max_depth)

    # Hàm dự đoán một tập dữ liệu
    def predict(self, X):
        return np.array([self.predict_tree_(self.tree, x) for x in X])

    # Hàm chia dữ liệu theo đặc trưng và ngưỡng
    def split_(self, X, y, feature, threshold):
        left_mask = X[:, feature] <= threshold  # Mặt nạ để lấy các phần tử nhỏ hơn
        right_mask = X[:, feature] > threshold  # Mặt nạ để lấy các phần tử lớn hơn
        return X[left_mask], X[right_mask], y[left_mask], y[right_mask]

    # Hàm tìm đặc trưng và ngưỡng tốt nhất để chia dữ liệu
    def best_split_(self, X, y):
        best_gini = 1.0
        best_feature = None
        best_threshold = None
        for feature in range(X.shape[1]):  # Duyệt qua từng đặc trưng
            thresholds = np.unique(X[:, feature])  # Tìm tất cả các ngưỡng duy nhất
            for threshold in thresholds:  # Duyệt qua từng ngưỡng
                X_left, X_right, y_left, y_right = self.split_(X, y, feature, thres
                if len(y_left) == 0 or len(y_right) == 0:  # Nếu một trong hai phần
                    continue
                gini_left = gini(y_left)  # Tính chỉ số Gini cho phần bên trái
                gini_right = gini(y_right)  # Tính chỉ số Gini cho phần bên phải
                gini_split = (len(y_left) * gini_left + len(y_right) * gini_right)
                if gini_split < best_gini:  # Nếu Gini nhỏ hơn, cập nhật đặc trưng
                    best_gini = gini_split
                    best_feature = feature
                    best_threshold = threshold
        return best_feature, best_threshold
```

```
    # Hàm xây dựng cây quyết định đệ quy
    def build_tree_(self, X, y, depth=0, max_depth=10):
        if len(np.unique(y)) == 1:  # Nếu tất cả các phần tử cùng một lớp, trả về n
            return TreeNode(value=y[0])
        if depth >= max_depth:  # Nếu độ sâu đạt giới hạn, trả về nút lá
            return TreeNode(value=np.bincount(y).argmax())  # Trả về lớp phổ biến n
        feature, threshold = self.best_split_(X, y)  # Tìm đặc trưng và ngưỡng tốt
        if feature is None:  # Nếu không tìm được đặc trưng tốt, trả về nút lá
            return TreeNode(value=np.bincount(y).argmax())  # Trả về lớp phổ biến n
        X_left, X_right, y_left, y_right = self.split_(X, y, feature, threshold)  #
        left_child = self.build_tree_(X_left, y_left, depth + 1, max_depth)  # Xây
        right_child = self.build_tree_(X_right, y_right, depth + 1, max_depth)  # X
        return TreeNode(feature=feature, threshold=threshold, left=left_child, righ

    # Hàm dự đoán giá trị dựa trên cây quyết định
    def predict_tree_(self, node, X):
        if node.value is not None:  # Nếu là nút lá, trả về giá trị của nút lá
            return node.value
        if X[node.feature] <= node.threshold:  # Nếu giá trị nhỏ hơn hoặc bằng ngưỡ
            return self.predict_tree_(node.left, X)
        else:  # Nếu giá trị lớn hơn ngưỡng, duyệt cây con bên phải
            return self.predict_tree_(node.right, X)
```

Sau đây là kết quả chạy thử với bộ dữ liệu đầu vào đã được mã hóa TF-IDF sang vector (chi tiết sẽ được báo cáo đầy đủ sau)

In [8]:
```
import pandas as pd
```

In [ ]:
```
X = pd.read_csv('tfidf.csv')
y = pd.read_csv('tfidf_y.csv')
```

In [20]:
```
X_train = X.head(200).to_numpy()
y_train = y.head(200).to_numpy()[:,0]
X_test = X.tail(200).to_numpy()
y_test = y.tail(200).to_numpy()[:,0]
```

In [21]:
```
%%time
dtree = G07DecisionTree(max_depth=12)
dtree.fit(X_train, y_train)
```

```
CPU times: total: 3min 27s
Wall time: 10min 38s
```

In [22]:
```
y_pred = dtree.predict(X_test)
y_pred
```

Out[22]:
```
array([1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [23]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[23]:
```
0.89
```

Kết quả, vấn đề gặp phải

Như vậy, nhóm G07 đã thử nghiệm Decision Tree đã xây dựng và đạt độ chính xác 89% khi mới chỉ huấn luyện trên 200 hàng đầu của datasets.

Tuy vậy, hạn chế vẫn còn khi huấn luyện 200 hàng đầu này đã mất khoảng 10 phút (với 3 phút 27 s CPU). Trong những tuần sau nhóm sẽ tập trung vào cải thiện thời gian fit cho datasets.