Завдання: Реалізація та Аналіз Дерев Рішень для Класифікації

Датасет: https://www.kaggle.com/datasets/uom190346a/global-coffee-health-dataset/data
що містить 10,000 синтетичних записів про споживання кави, сон та здоров'я. Для задачі класифікації оберіть цільову змінну `Health_Issues` (категоріальна: None, Mild, Moderate, Severe) як клас для передбачення, використовуючи інші ознаки як предиктори.
Обробіть датасет: видаліть ID, закодуйте категоріальні ознаки (наприклад, Gender, Country, Sleep_Quality, Stress_Level, Occupation, Smoking, Alcohol_Consumption) за допомогою one-hot або label encoding. Поділіть дані на тренувальну та тестову вибірки (наприклад, 80/20). Реалізуйте все на Python. Оцініть моделі за метриками точності (accuracy), F1-score, confusion matrix, MCC. Результат повинен включати код, результати, графіки (наприклад, важливість ознак) та аналіз.

а) Реалізація Дерева Рішень з Використанням Gini Index

Дерево рішень є алгоритмом класифікації, що будує ієрархічну структуру для розділення даних на основі ознак. Gini index вимірює неоднорідність вузла: нижче значення вказує на кращий розділ. Додатково реалізуйте обчислення feature importance як сумарне зважене зменшення Gini по вузлах, де ознака використовується для розділу.

Формули:
- Gini impurity для вузла з класами c:

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

де p_i — частка класу i).
- Зважена Gini для розділу:
$Gini_{weighted} = \frac{n_{left}}{n} \cdot Gini_{left} + \frac{n_{right}}{n} \cdot Gini_{right}$, де n_{left}, n_{right} — кількість зразків у лівому/правому піддереві.
- Feature importance для ознаки f):

$$importance(f) = \sum_{nodes\ where\ f\ splits} \frac{n_{node}}{n_{total}} \cdot (Gini_{parent} - Gini_{weighted})$$

Нормалізуйте за сумою по всіх ознаках:

$$importance(f) = \frac{importance(f)}{\sum importance}.$$

Алгоритм:

1. Обчислити Gini для поточного вузла. Якщо Gini = 0 або досягнуто критерій зупинки (макс. глибина, мін. зразків), вузол стає листовим (majority class).
2. Для кожної ознаки та можливого порогу обчислити $Gini_{weighted}$ після розділу.
3. Обрати ознаку/поріг з мінімальною $Gini_{weighted}$ та оновити importance для цієї ознаки.
4. Рекурсивно застосувати до піддерев.
5. Для прогнозу: пройти від кореня до листа, повернути majority class.
6. Після побудови обчислити та повернути вектор importance для всіх ознак.

Прототип коду:

import numpy as np

```
class MyDecisionTree:
    def __init__(self, max_depth=5, min_samples=2):
       self.max_depth = max_depth
       self.min_samples = min_samples
       self.feature_importance = None
       self.tree = None

    def gini(self, y):
        # розрахунок джіні-індексу

    def best_split(self, X, y):
        best_gini = float('inf')
        best_feature, best_threshold = None, None
        for feature in range(X.shape[1]):
            # доповність цей код
     # ……..
       return best_feature, best_threshold, best_gini  # Повернути
для importance

    def build_tree(self, X, y, depth=0, n_total=None):
        # доповність цей код
        return {'leaf': False, 'feature': feature, 'threshold':
threshold,
                'left': left_subtree, 'right': right_subtree}
```

```
   def fit(self, X, y):
       """Навчає дерево та обчислює нормалізовану важливість
ознак."""
       self.n_total = len(y)
       self.feature_importance = np.zeros(X.shape[1])
       self.tree = self.build_tree(X, y)
       # Нормалізація важливості
       if np.sum(self.feature_importance) > 0:
           self.feature_importance /= np.sum(self.feature_importance)
       return self

def predict_one(self, x, node):
     """Прогнозує клас для одного зразка."""
     …
def predict(self, X):
    """Прогнозує класи для всіх зразків."""
```

б) Реалізація Механізму Уникнення Перенавчання (Прунінг)

Перенавчання виникає, коли дерево надто глибоке та адаптується до шуму. Прунінг (обрізка) зменшує складність, покращуючи узагальнення. Після прунінгу переобчисліть feature importance на pruned tree.

Формули:

- Cost-complexity pruning: $R_\alpha(T) = R(T) + \alpha \cdot |T|$, де $R(T)$ — помилка на валідації, $|T|$ — кількість листів, α — параметр (підбирається via cross-validation).

Алгоритм:

1. Побудувати повне дерево без обмежень.
2. Обчислити R_α для кожного піддерева.
3. Рекурсивно зливати вузли, якщо злиття зменшує R_α.
4. Підібрати оптимальне α за валідаційними даними.
5. Застосувати прунінг, переобчислити importance та переоцінити метрики.

Прототип коду:

```
def prune_tree(self, node, alpha, validation_X, validation_y):
    if 'leaf' in node: return self.calculate_error(validation_X,
validation_y, node)
    left_error = self.prune_tree(node['left'], alpha, validation_X,
validation_y)
    right_error = self.prune_tree(node['right'], alpha, validation_X,
validation_y)
    subtree_error = left_error + right_error
    leaf_error = self.calculate_error(validation_X, validation_y,
{'leaf': True, 'class': node['class']})
    if leaf_error + alpha <= subtree_error + alpha *
(self.count_leaves(node) - 1):
        node['leaf'] = True
        del node['left'], node['right'], node['feature'],
node['threshold']
        return leaf_error + alpha
    return subtree_error + alpha * self.count_leaves(node)
```

в) Порівняння Результатів з sklearn DecisionTree та RandomForest

Порівняння власної реалізації з бібліотечними моделями дозволяє оцінити ефективність. RandomForest — ансамбль дерев, що зменшує варіативність via bootstrapping та random feature selection. Порівняйте також feature importance з sklearn (наприклад, rf.feature_importances_).

Формули:

- Accuracy: $accuracy = \frac{TP + TN}{TP + TN + FP + FN}$.

- F1-score: $F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

Алгоритм:

1. Навчити sklearn.DecisionTreeClassifier(criterion='gini', max_depth=..., min_samples_leaf=...).
2. Навчити sklearn.RandomForestClassifier(n_estimators=100, criterion='gini').
3. Обчислити метрики (accuracy, F1, confusion matrix) для всіх моделей на тестових даних.

4. Порівняти час навчання/прогнозування, feature importance (візуалізуйте бар-графіки) та провести крос-валідацію (5-fold).
5. Аналізувати: RF часто кращий через averaging; порівняйте importance з вашим MyDecisionTree.

Прототип коду:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt

dt_sk = DecisionTreeClassifier(criterion='gini', max_depth=5)
dt_sk.fit(X_train, y_train)
y_pred_sk = dt_sk.predict(X_test)
print("sklearn DT Accuracy:", accuracy_score(y_test, y_pred_sk))

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("RF Accuracy:", accuracy_score(y_test, y_pred_rf))

# Візуалізація importance
plt.bar(range(len(rf.feature_importances_)), rf.feature_importances_)
plt.show()
```

д) Інженерія Ознак та Feature Selection

Інженерія ознак покращує якість даних, створюючи інформативні змінні. Feature selection обирає релевантні ознаки на основі importance, зменшуючи розмірність та шум.
Застосуйте після інженерії, використовуючи importance з MyDecisionTree.

Формули:

- Кореляція Пірсона:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}.$$

- Feature importance (з а)):

$$importance(f) = \sum_{nodes\ where\ f\ splits} \frac{n_{node}}{n_{total}} \cdot (Gini_{parent} - Gini_{weighted}).$$

Алгоритм:

1. Інженерія: Обчислити кореляції; видалити ознаки з $|r| > 0.8$. Створити нові: поліноміальні (x^2), взаємодії ($x_1 \cdot x_2$), напр. Coffee_Intake * Sleep_Hours), бінінг (розбиття на категорії, напр. Age bins), нормалізацію (z-score). Закодувати категоріальні (one-hot для Country, Occupation).
2. Feature selection: Навчити MyDecisionTree на оновленому датасеті, отримати importance. Обрати топ-k ознак (наприклад, топ-10 за importance > threshold, напр. 0.01) або використовувати recursive feature elimination.
3. Повторно навчити моделі на відібраних ознаках та порівняти метрики (очікуваний приріст accuracy через зменшення шуму). Оцінити importance via permutation (sklearn.inspection.permutation_importance).

Прототип коду:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.inspection import permutation_importance

# Завантаження та обробка
df = pd.read_csv('global_coffee_health.csv')  # Припустимо файл
завантажено
X = df.drop(['ID', 'Health_Issues'], axis=1)
y = df['Health_Issues']

# Інженерія
X['Coffee_Sleep_Interaction'] = X['Coffee_Intake'] * X['Sleep_Hours']
X['Age_Binned'] = pd.cut(X['Age'], bins=[18,30,50,80],
labels=['Young', 'Middle', 'Old'])

# Кодування
cat_cols = ['Gender', 'Country', 'Sleep_Quality', 'Stress_Level',
'Occupation', 'Age_Binned']
X = pd.get_dummies(X, columns=cat_cols)
```

```
# Нормалізація
scaler = StandardScaler()
X[['Age', 'Coffee_Intake', 'Caffeine_mg', 'Sleep_Hours', 'BMI',
'Heart_Rate', 'Physical_Activity_Hours']] =
scaler.fit_transform(X[['Age', 'Coffee_Intake', 'Caffeine_mg',
'Sleep_Hours', 'BMI', 'Heart_Rate', 'Physical_Activity_Hours']])

# Feature selection за importance
tree = MyDecisionTree()
tree.fit(X.values, y.values)
importances = tree.feature_importance
selected_features = np.argsort(importances)[-10:]  # Топ-10
X_selected = X.iloc[:, selected_features]

# Permutation importance
perm_importance = permutation_importance(tree, X.values, y.values)
```

Детальна Формула Обчислення Важливості Ознак (Feature Importance) у Деревах Рішень з Використанням Gini Impurity

Важливість ознак у деревах рішень оцінює внесок кожної ознаки у зменшення неоднорідності (impurity) даних під час побудови дерева. Для критерію Gini, важливість базується на сумарному зменшенні Gini impurity, зваженому на кількість зразків у вузлах, де відбувається розділ за даною ознакою. Обчислення відбувається у два етапи: (1) розрахунок важливості для кожного вузла, (2) агрегація та нормалізація для кожної ознаки.

Крок 1: Розрахунок Важливості для Кожного Вузла
Для вузла k) важливість ($\mathrm{Importance}_{\mathrm{Node}_k}$) вимірює зменшення impurity після розділу, зважене на пропорцію зразків, що досягають цього вузла. Формула:

$$\mathrm{Importance}_{\mathrm{Node}_k} = \frac{n_k}{n_{\mathrm{total}}} \cdot \left(\mathrm{Gini}_k - \frac{n_{\mathrm{left}}}{n_k} \cdot \mathrm{Gini}_{\mathrm{left}} - \frac{n_{\mathrm{right}}}{n_k} \cdot \mathrm{Gini}_{\mathrm{right}} \right)$$

де:
- n_k — кількість зразків у вузлі k,

- n_{total} — загальна кількість зразків у датасеті,
- n_{left}, n_{right} — кількість зразків у лівому та правому піддеревах,
- Gini_k — Gini impurity вузла k:

$$\text{Gini}_k = 1 - \sum_{i=1}^{c} p_i^2,$$

де p_i — частка класу i у вузлі, c — кількість класів,
- $\text{Gini}_{\text{left}}$, $\text{Gini}_{\text{right}}$ — Gini impurity лівого та правого піддерев.

Ця формула еквівалентна зменшенню Gini (Gini gain), зваженому на $\frac{n_k}{n_{\text{total}}}$). У деяких реалізаціях (наприклад, scikit-learn) вона нормалізована на 100 для відсотків, але сутність та сама.

Крок 2: Розрахунок Важливості для Ознаки
Для ознаки f сумуються важливості всіх вузлів, де розділ відбувається за f, після чого нормалізується на сумарну важливість усіх вузлів у дереві:

$$\text{Importance}(f) = \frac{\sum_{\text{nodes where } f \text{ splits}} \text{Importance}_{\text{Node}_k}}{\sum_{\text{all nodes}} \text{Importance}_{\text{Node}_k}}$$

Це забезпечує, що сума важливостей усіх ознак дорівнює 1 (або 100%). У scikit-learn це доступно як атрибут `feature_importances_`, де важливість — нормалізоване сумарне зменшення критерію (Gini) для даної ознаки.

Приклад Обчислення
Розглянемо бінарну класифікацію з 2000 зразками (по 1000 на клас). Кореневий вузол (розділ за ознакою "Total Impressions", Gini=0.5, $n_k = 2000$):

- Ліве піддерево: $n_{\text{left}} = 1047$, Gini=0.086,
- Праве піддерево: $n_{\text{right}} = 953$, Gini=0.

Важливість вузла:

$$\text{Importance}_{\text{Node}} = \frac{2000}{2000} \cdot \left(0.5 - \frac{1047}{2000} \cdot 0.086 - \frac{953}{2000} \cdot 0\right) = 1 \cdot (0.5 - 0.045) = 0.455$$

Якщо це єдиний розділ за ознакою f, і сумарна важливість усіх вузлів = 0.5, то:

$$\text{Importance}(f) = \frac{0.455}{0.5} = 0.91$$