

```
In [1]:  
import multiprocessing  
import os  
from time import time  
from typing import Iterable, List  
  
import numpy as np  
import pandas as pd  
from joblib import Parallel, delayed
```

```
In [2]:  
if not os.path.isfile("train.csv") or not os.path.isfile("test.csv"):  
    print("Downloading data...")  
    train = pd.read_csv("https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw6/hw6_train.csv")  
    test = pd.read_csv("https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw6/hw6_test.csv")  
  
    train.to_csv("train.csv", index=False)  
    test.to_csv("test.csv", index=False)  
    print("Completed")  
else:  
    train = pd.read_csv("train.csv")  
    test = pd.read_csv("test.csv")  
  
  
class TreeNode:  
    def __init__(self, threshold, feature, value=None):  
        self.left = None  
        self.right = None  
        self.value = value  
        self.feature = feature  
        self.threshold = threshold  
  
  
def tree_predict(tree: TreeNode, data: List[float]) -> float:  
    while (not tree.value):  
        if data[int(tree.feature)] >= tree.threshold:  
            tree = tree.right  
        else:  
            tree = tree.left  
  
    return tree.value  
  
  
def learn_tree(data: pd.DataFrame) -> List[List]:  
    if is_pure(data):  
        return TreeNode(None, None, data["10"].unique()[0])  
    else:  
        featureNum, threshold = find_best_feature(data)  
        rightBranch = data[(data[featureNum] >= threshold)]  
        leftBranch = data[(data[featureNum] < threshold)]  
        DecisionTreeClassifier = TreeNode(threshold, featureNum, None)  
        DecisionTreeClassifier.right = learn_tree(rightBranch)  
        DecisionTreeClassifier.left = learn_tree(leftBranch)  
  
    return DecisionTreeClassifier  
  
  
def is_pure(data: pd.Series) -> bool:  
    return len(pd.unique(data["10"])) == 1
```

```

def find_best_feature(data: pd.DataFrame) -> List:
    features = data.drop(["10"], axis=1).columns
    purityList = [find_best_threshold(data, feature) for feature in features]
    _, threshold, featureNumber = sorted(purityList, key=lambda x: (x[0], x[1], int(x[2])))
    return [str(featureNumber), threshold]

def find_best_threshold(data: pd.DataFrame, feature: str) -> List[float]:
    featureList = []
    for threshold in generate_threshold(data[feature]):
        rightBran = data["10"][(data[feature] >= threshold)]
        leftBran = data["10"][(data[feature] < threshold)]
        weight = rightBran.count() / (rightBran.count() + leftBran.count())
        impurity = weight * gini_coef(rightBran) + (1 - weight) * gini_coef(leftBran)
        featureList.append([impurity, threshold, feature])

    return sorted(featureList, key=lambda x: (x[0], x[1], x[2]), reverse=False)[0]

def generate_threshold(data: pd.Series) -> List[float]:
    tmp = sorted(data.to_list())
    thresholdList = []
    for i in range(len(tmp)):
        if i == (len(tmp) - 1):
            break
        thresholdList.append((tmp[i] + tmp[i + 1]) / 2)

    return list(set(thresholdList))

def gini_coef(data: pd.Series) -> float:
    probability = (data.value_counts() / len(data)).values
    return 1 - np.sum(np.square(probability))

def calculate_eout(classifier, testData: pd.DataFrame) -> float:
    score = 0
    for index, row in testData.iterrows():
        predict = tree_predict(classifier, row[:10])
        score += int(predict != row[10])

    return score / testData.shape[0]

```

In [3]:

```

tree = learn_tree(train)
eout = calculate_eout(tree, test)
print(f"Eout of a decsiob tree is {eout:.2f}")

```

Eout of a decsiob tree is 0.20

In [4]:

```

cpu_nums = multiprocessing.cpu_count()

```

In [5]:

```

def random_forest_with_bagging_eout(n: int, data: pd.DataFrame, test: pd.DataFrame) ->
    def decision_tree(data: pd.DataFrame, test: pd.DataFrame) -> float:
        tree = learn_tree(bagging(data))
        return calculate_eout(tree, test)

```

```

errorList = Parallel(n_jobs=cpu_nums)(delayed(decision_tree)(data, test) for _ in range(n))

return sum(errorList) / len(errorList)

def bagging(data: pd.DataFrame) -> pd.DataFrame:
    indexes = np.random.choice(data.index, size=int(0.5 * data.shape[0]), replace=True)
    return data.iloc[indexes]

```

In [6]:

```

eout = random_forest_with_bagging_eout(2000, train, test)
print(f"Eout of 2000 trees is {eout:.2f}")

```

Eout of 2000 trees is 0.24

In [7]:

```

def random_forest_with_bagging_and_vote(n: int, data: pd.DataFrame, test: pd.DataFrame):
    def decision_tree(data: pd.DataFrame) -> float:
        return learn_tree(bagging(data))

    def random_tree_error(treeList: List[TreeNode], data: List[int]) -> int:
        error = 0
        for index, row in data.iterrows():
            predictList = []
            for tree in treeList:
                while (not tree.value):
                    if row[int(tree.feature)] >= tree.threshold:
                        tree = tree.right
                    else:
                        tree = tree.left
                predictList.append(tree.value)

            if sum(predictList) / len(predictList) >= 0:
                error += (row[10] != 1)
            else:
                error += (row[10] != -1)

        return error / data.shape[0]

    treeList = Parallel(n_jobs=cpu_nums)(delayed(decision_tree)(data) for _ in range(n))
    if error == "ein":
        return random_tree_error(treeList, data)
    elif error == "eout":
        return random_tree_error(treeList, test)

```

In [8]:

```

ein = random_forest_with_bagging_and_vote(2000, train, test, "ein")
print(f"Ein of 2000 trees is {ein:.2f}")

```

Ein of 2000 trees is 0.01

In [9]:

```

eout = random_forest_with_bagging_and_vote(2000, train, test, "eout")
print(f"Eout of 2000 trees is {eout:.2f}")

```

Eout of 2000 trees is 0.16

## Problem 19

Answer: (b)

會最喜歡Matrix Factorization這個單元的原因是，之前有接觸過NN覺得NN可以某種程度上，做出feature extraction的效果，但是卻沒有想過NN居然可以將抽象的feature轉換成有意義的特徵，我想這堂課是讓我最驚訝的，更讓我知道推薦系統最初階的模型大概像這個樣子。

## Problem 20

Answer: (d)

其實技法每個單元我都很喜歡，真的很難做抉擇，硬要選的話我會選Adaboost跟Gradient Boosting的模型，因為老師講蘋果的故事，太過精彩了且常常讓我印象深刻，作夢都在想XDD，讓我難以入睡。