人工智慧導論 HW4

資訊113 黃佳倫 E64096245

Section 1. 了解資料

1. 先載入csv檔案: 用pandas去載入

```
import pandas as pd
data = pd.read_csv("train.csv")
```

2. 知資料有什麼屬性

- code: 把表頭全部輸出看有甚麼屬性

```
# all properties
  data_properties = data.columns.values.tolist()
  print(data_properties)
```

- output

```
['profile pic', 'nums/length username', 'fullname words', 'nums/length fullname', 'name==username', 'description length', 'externa
l URL', 'private', '#posts', '#followers', '#follows', 'fake']
```

3. 檢查是否有缺失資料: 沒有

- code: 用資料全部長度減掉真的有資料的數量檢查，如果有缺失就drop掉

```
# check missing data
missing_count = len(data) - data.count()
print("Missing values count:\n", missing_count)
# if missing, drop the data
data = data.dropna()
```

- output: 沒有資料缺失，不用做啥特別處理

```
Missing values count:
 profile pic             0
nums/length username    0
fullname words          0
nums/length fullname    0
name==username          0
description length      0
external URL            0
private                 0
#posts                  0
#followers              0
#follows                0
fake                    0
dtype: int64
```

Section 2. 資料前處理

```
# split train and test data
X = data.drop(['fake'], axis = 1).values
y = data['fake'].values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=50)
```

1. `X = data.drop(['fake'], axis = 1).values`
   - 要輸入給model判斷的features，所以drop掉fake那一行，取除了fake那一行的所有值
2. `y = data['fake'].values.reshape(-1,1)`
   - 目標是判斷是否是真的帳號，所以 `y` 只要最後一行，header為fake的所有的值
   - 只有0,1 兩個值: 0是fake, 1是real
3. split train and test data
   - `test_size` : 要當作test的比例
   - `random_state` : 可以打亂資料

Section 3. 建立模型

1. 決策樹的節點 `class Node()`
   - `__init__`

```
class Node():
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None, samples=None, class_=None):

        # 用哪個特徵做split
        self.feature_index = feature_index

        # 用來比較data中的值
        self.threshold = threshold

        # children: left, right
        self.left = left
        self.right = right

        # information gain
        self.info_gain = info_gain

        # 這個node底下總共有多少sample
        self.samples = samples

        # 0(fake)和1(real)的數量, Ex: [12,23]
        self.value = value

        # 預測結果, 是0還是1, 主要會在leaf呈現
        self.class_ = class_
```

2. 決策樹 `class DecisionTreeClassifier()`

```
class DecisionTreeClassifier():
  def __init__(self, min_samples_split=2, max_depth=2, features=None, classes_=None):
      ... # 初始化終止條件、特徵、類別
  def fit(self, X, y):
      ... # 把要訓練的資料丟入決策樹
  def grow_tree(self, dataset, curr_depth=0):
      ... # 訓練決策樹
  def find_best_split(self, dataset, n_samples, n_features):
      ... # 用於訓練, 找到最佳分裂資訊
  def split(self, dataset, feature_index, threshold):
      ... # 基於輸入的feature和threshold, 分裂dataset
  def information_gain(self, parent, child_left, child_right, mode="entropy"):
      ... # 找到目前的信息增益
  def entropy(self, y):
      ... # 算商
  def gini(self, y):
      ... # 算gini
  def print_tree(self, tree=None, indent=" "):
      ... # 把tree印在terminal
  def predict(self, X):
      ... # 輸入為dataset, 輸出預測dataset的所有值
  def make_prediction(self, x, tree):
      ... # 輸入為單一feature序列, 輸出預測值
  def plot_tree_dots(self, tree=None):
      ... # 準備這顆決策樹所有的dot資訊, 並用於pydotplus.graph_from_dot_data()畫圖
```

- 初始化

```
def __init__(self, min_samples_split=2, max_depth=2, features=None, classes_=None):

    # 初始化決策樹的root
    self.root = None

    # 設定生長終結的條件
    self.min_samples_split = min_samples_split
    self.max_depth = max_depth

    # 可以輸入features和class的值, 在plot_dots_tree()會加入到要繪製tree的圖中,
    self.features = features
    self.classes_ = classes_
```

- `fit` 把訓練資料fit到model中

```
def fit(self, X, y):
    dataset = np.concatenate((X, y), axis=1)
    self.root = self.grow_tree(dataset)
```

- 生長決策樹 `grow_tree`

```
def grow_tree(self, dataset, curr_depth=0):
    # X: 除了fake的所有資料, y: 只有fake那一行
    X, y = dataset[:,:-1], dataset[:,-1]

    # 計算sample和feature的數量
    n_samples, n_features = X.shape[0], X.shape[1]

    # 分裂樹枝直到: 剩下的sample小於最小可分裂的樹量 & 樹的高度= 最大高度
    if n_samples>=self.min_samples_split and curr_depth<=self.max_depth:
        # 找到最好的分裂點
        best_split = self.find_best_split(dataset, n_samples, n_features)
        # 檢查 information gain > 0
        if best_split["info_gain"]>0:
            # 遞迴生長左子樹
            subtree_left = self.grow_tree(best_split["dataset_left"], curr_depth+1)
            # 遞迴生長右子樹
            subtree_right = self.grow_tree(best_split["dataset_right"], curr_depth+1)
            # return最佳分裂的節點
            return Node(feature_index=best_split["feature_index"], threshold=best_split["threshold"],
                        left=subtree_left, right=subtree_right, info_gain=best_split["info_gain"],
                        value=best_split["value"],samples=best_split["samples"])

    # 計算leaf的值, 也就是預測的值: 看在y裡面的0和1誰的樹量最多
    leaf_class_ = max(list(y), key=list(y).count)
```

```
    # return 長好的leaf: 也就是預測的結果, 0或1
    return Node(class_ = leaf_class_)
```

- 尋找最佳分裂 `find_best_split`：看哪個feature是目前可以讓information gain最大的，並回傳最佳分裂的資訊

```
def find_best_split(self, dataset, n_samples, n_features):

  # 初始化最佳分裂和最大信息增益
  best_split = {}
  max_info_gain = -float("inf")

  # loop all features: 看哪個feature是目前可以讓information gain最大的
  for feature_index in range(n_features):
      # feature的所有值
      feature_values = dataset[:, feature_index]
      # features的所有相異的值
      thresholds = np.unique(feature_values)
      # loop all values of thresholds
      for threshold in thresholds:
          # 分裂左子樹和右子樹
          dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
          # 檢查小孩不是空的, 空的話不能分裂
          if len(dataset_left)>0 and len(dataset_right)>0:
              # 左邊和右邊的fake or not的數值們: dataset的最後一行就是
              y, y_left, y_right = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
              # get current information gain
              curr_info_gain = self.information_gain(y, y_left, y_right, "gini")
              # 如果找到比目前還大的information gain, 更新最佳分裂的資訊
              if curr_info_gain > max_info_gain:
                  best_split={
                      "feature_index": feature_index,
                      "threshold": threshold,
                      "dataset_left": dataset_left,
                      "dataset_right": dataset_right,
                      "info_gain": curr_info_gain,
                      "value": list(Counter(list(y)).values()),
                      "samples": n_samples
                  }
                  max_info_gain = curr_info_gain

  # return the best split
  return best_split
```

- 執行分裂 `split()`

```
def split(self, dataset, feature_index, threshold):
    # <=threshold: 左子樹
    # >threshold: 右子樹
    dataset_left = np.array([row for row in dataset if row[feature_index]<=threshold])
    dataset_right = np.array([row for row in dataset if row[feature_index]>threshold])
    return dataset_left, dataset_right
```

- `information gain` (信息增益): 父節點 - Σ (小孩佔父親的權重*子節點gini(or 熵))，要使information 最大化就是決策樹的生長過程，讓熵或gini不斷降低

```
def information_gain(self, parent, child_left, child_right, mode="gini"):

    # 計算權重
    weight_left = len(child_left) / len(parent)
    weight_right = len(child_right) / len(parent)

    # compute the information gain: 模式有gini和entropy
    if mode=="gini":
        gain = self.gini(parent) - (weight_left*self.gini(child_left) + weight_right*self.gini(child_right))
    if mode == "entropy":
        gain = self.entropy(parent) - (weight_left*self.entropy(child_left) + weight_right*self.entropy(child_right))

    return gain
```

- `entropy()` :計算熵，最大1，最小0， 越大不確定因素越大

```
def entropy(self, y):
  # class的種類, 在這個case裡只有 [0, 1]
  classes_ = np.unique(y)
  entropy = 0

  # loop所有class算熵
  for class_ in classes_:
      p_class_ = len(y[y == class_]) / len(y)
      entropy += -p_class_ * np.log2(p_class_)
  return entropy
```

- `gini()`: 計算gini，最大0.5，最小0，越大不確定因素越大

```
def gini(self, y):
    # class的種類, 在這個case裡只有 [0, 1]
    classes_ = np.unique(y)
    gini = 0

    # loop所有class算gini
    for class_ in classes_:
        p_class_ = len(y[y == class_]) / len(y)
        gini += p_class_**2
    return 1 - gini
```

- 印出tree在terminal

```
def print_tree(self, tree=None, indent=" "):
    # 沒東西 -> 沒東西
    if not tree:
        tree = self.root
    # 是leaf, 因為有值(預測的類別): 印出結果
    if tree.class_ is not None:
        print(tree.class_)
    else:# node, 印出分裂的資訊
        print(self.features[tree.feature_index]+ " <= ", tree.threshold, " ,info_gain=", tree.info_gain, ' ,value=', tree.value, ' ,samples=', tree.samples)
        print("%sleft:" % (indent), end="")
        self.print_tree(tree.left, indent + indent)
        print("%sright:" % (indent), end="")
        self.print_tree(tree.right, indent + indent)
```

- `predict(X)`:
  - input: dataset
  - output: predict series

```
def predict(self, X):
    # 預測每個在X裡的data
    preditions = [self.make_prediction(x, self.root) for x in X]
    return preditions
```

- `make_predict(x)` :
  - input: single data, a feature such as [1, 0.1, 2, 0, 0, 0, 0, 1, 13, 159, 98]
  - output: 0 or 1

```
def make_prediction(self, x, tree):
    # 如果只有一個點
    if tree.class_!=None: return tree.class_

    # 開始分類, 從決策樹的root開始, 用每個node的分裂資訊走下去, 直到走到leaf, 也就是結果
    feature_threshold = x[tree.feature_index]
    if feature_threshold <=tree.threshold:
        return self.make_prediction(x, tree.left)
    else:
        return self.make_prediction(x, tree.right)
```

- 準備畫決策樹的dot資訊 → 之後再用 `pydotplus` 畫出決策樹

```
def plot_tree_dots(self, tree=None):

  if not tree:
      tree = self.root

  # 初始化dot資訊
  dot_data = ['''digraph Tree{
  node [shape=box, style="rounded", color="black", fontname="helvetica"] ;
  edge [fontname="helvetica"] ;''']

  # 計算有幾個node
  i_node = 0

  # 用遞迴方式來找所有node, 然後把樹枝的分裂資訊和葉子的預測結果放到dot資訊中
  def generate_dot_data(tree, curr_node=0):
      nonlocal i_node

      # 如果是葉子的話: 把class放到dot_data中
      if tree.class_ is not None:
          dot_data.append('%d [label="class: %s"];'%(curr_node, self.classes_[round(tree.class_)]))
          return
      else: # 樹枝的話: 放分裂資訊
          dot_data.append('%d [label= "%s <= %f \n info_gain = %f \n value= %s \n samples= %d"];'
              %(curr_node, self.features[tree.feature_index], tree.threshold, tree.info_gain, str(tree.value), tree.samples))

          # 有左子樹: 畫箭頭並去遞迴左子樹
          if tree.left is not None:
              dot_data.append('%d -> %d;'%(curr_node, i_node+1))
              i_node += 1
          generate_dot_data(tree.left, i_node)

          # 有右子樹: 畫箭頭並去遞迴右子樹
          if tree.right is not None:
              dot_data.append('%d -> %d;'%(curr_node, i_node+1))
              i_node += 1
          generate_dot_data(tree.right, i_node)

  # 執行此function來找出所有node的dot資料
  generate_dot_data(tree)

  # 結尾要加這個
  dot_data.append('}')

  # 原本是list, 把它變成要換行的規定格式, 才可以給pydotplus畫
  dot_datas = '\n'.join(dot_data)

  # return 可畫圖的 dot 資料
  return dot_datas
```

3. 預測模型: 建立決策樹

```
# fit the model
features = data.drop(['fake'], axis = 1).columns.values.tolist()
```

```
decision_tree = DecisionTreeClassifier(min_samples_split=2, max_depth=11, features=features, classes_=['fake', 'real'])
decision_tree.fit(X_train,y_train)
```

4. 測試決策樹準確率:
   - 學習過的test資料(X_test): 0.905
   - 沒學習過的資料(test.csv): 0.908

```
# test model
y_predict = decision_tree.predict(X_test)
y_predict = decision_tree.predict(X_test)
print(accuracy_score(y_test, y_predict))


# test model using new datset
data_new = pd.read_csv('data/test.csv')
y_new = data_new['fake']
X = data_new.drop(['fake'], axis = 1).values
y_new_predict = decision_tree.predict(X)
print(accuracy_score(y_new, y_new_predict))
```

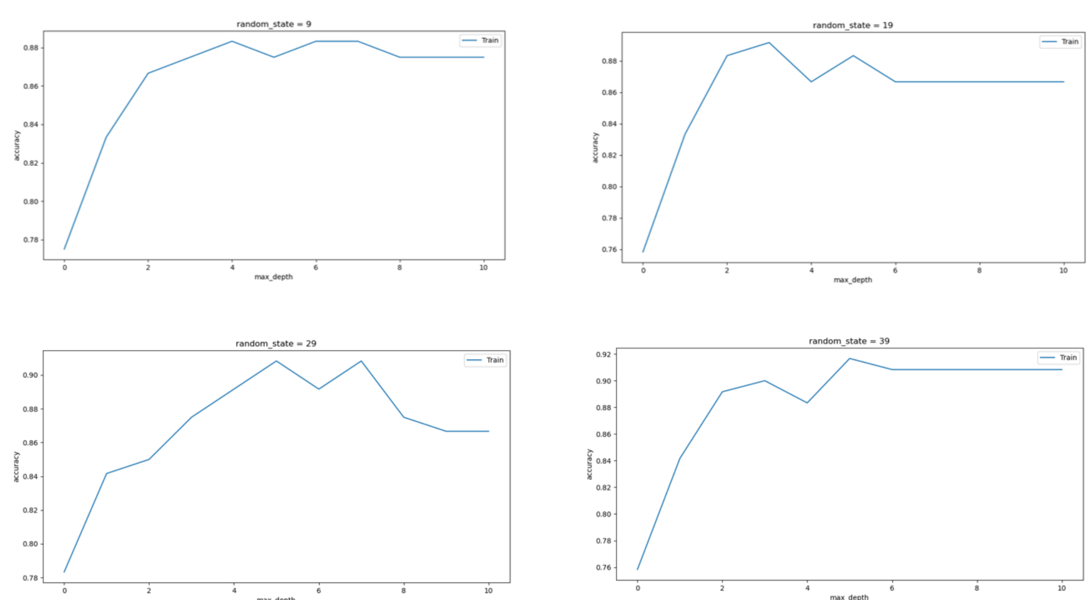
5. 畫出決策樹，把dot資訊放到 `pydotplus.graph_from_dot_data` 即可

```
decision_tree.print_tree()
dot_datas = decision_tree.plot_tree_dots()
graph = pydotplus.graph_from_dot_data(dot_datas)
graph.write_png("DT.png")
```

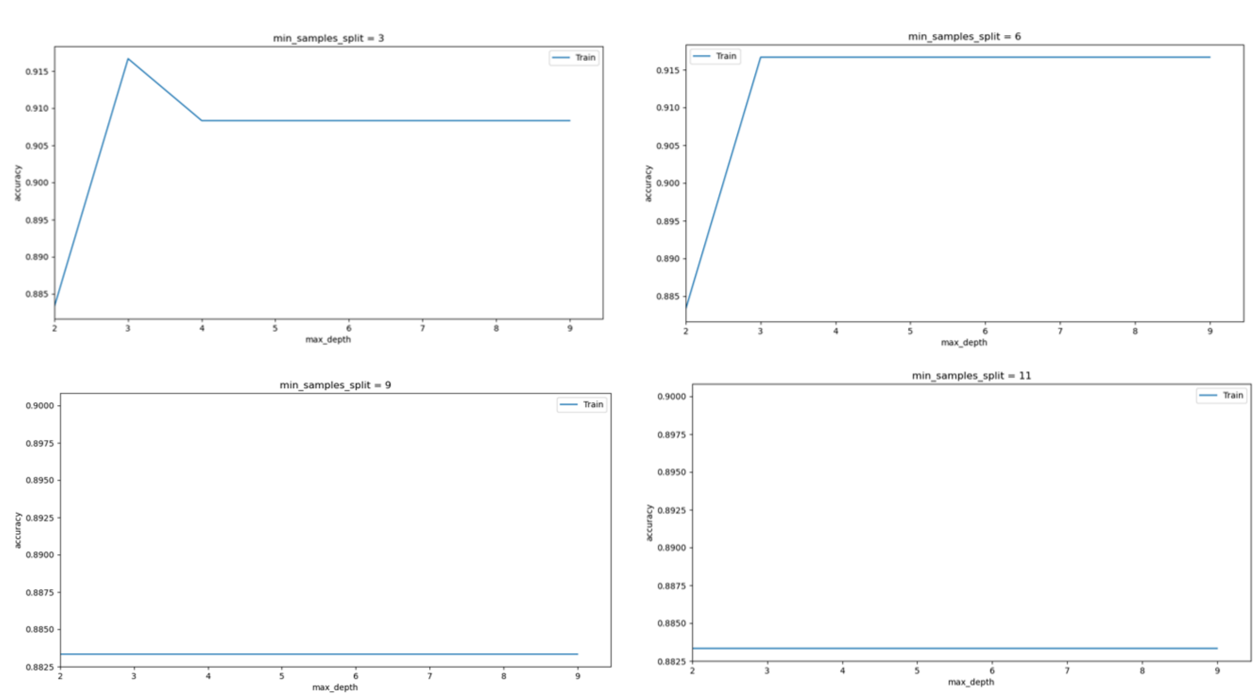
Section 4. 優化

選擇gini來算information gain時最佳

1. `random_state` 和 `max_depth`：在 `min_samples_split` =2的情況下，`random_state` 越大越準

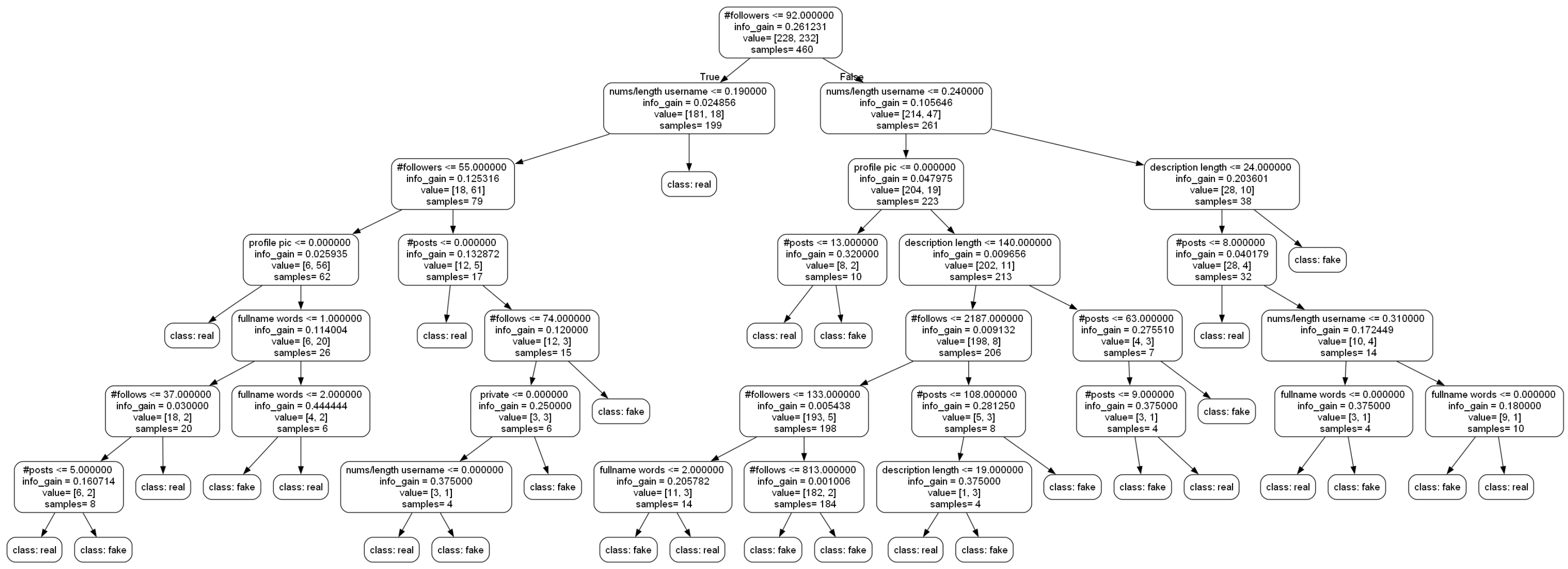


2. `min_samples_split` 和 `max_depth`：在 `random_state` = 39的情況下，`min_samples_split` 設置越小越準



3. 目前最佳:
   - `min_samples_split` = 2
   - `max_depth` = 6
   - `random_state` = 39
   - 學習過的test資料(X_test): 0.92

- 沒學習過的資料(test.csv): 0.908
- decision tree



Section 5. 解釋

1. 畫出決策樹 section 4
2. code: 在section 3
3. 解釋

   可以看出root是從#followers開始分裂的，所以可知一開始在找best split時，#follower 的gini或熵應該是最小的，讓information gain是最大，並且決策樹才會選擇從#follower開始分裂，從現實面來講可以知道，#follower如果超過92，大概就是假帳號，反之，如果≤92的可能是真帳號。經由這次功課，了解到在人工智慧的領域中，決策樹的建立不僅可以幫助預測結果，在訓練的過程中，也較容易解釋期訓練(分裂)的過程，不像深度學習的訓練過程猶如黑盒子，我們可以經由了解information gain的運算過程進而得知為何決策樹當前選擇哪個feature和threshold來進行分裂，讓訓練過程更清晰明瞭。