ML Term Project Report

109550178 黃昱翰

Decision Tree Algorithm:

根據 information gain 找到當前最好的 feature 去 split samples，分割完後再 recursively 的往左右的 child node 去找最好切割的 feature，並且設置停止條件，當深度達到 max_depth 或當前 node 只有一種類別或當前 node 沒有 instances。

Implementation:

```python
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, *, value=None)
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf(self):
        return self.value is not None
```

is_leaf 回傳 true or false 根據 node 的 value 是否為 None

```python
class DecisionTree:
    def __init__(self, max_depth=None, min_samples_split=5, split_criterion="entropy"):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.split_criterion  = split_criterion
        self.root = None
        self.classes = None
        self.deepest = 0
```

```python
    def _build_tree(self, X, y, depth=0, parent_samples=0):
        self.n_samples, self.n_features = X.shape
        self.classes = len(np.unique(y))
        if(depth > self.deepest): self.deepest = depth

         # randomly select features to consider for split
        rnd_feats = np.random.choice(self.n_features, self.n_features, replace=False)
         # find best split based on selected features
        score, best_feat, best_thresh = self._best_split(X, y, rnd_feats)

         # Check if reach stop criteria
        if self._is_finished(depth, parent_samples) or score == 0:
            most_common_Label = np.argmax(np.bincount(y))
            return Node(value=most_common_Label)

        # create children nodes and continue building tree recursively
        left_idx, right_idx = self._create_split(X[:, best_feat], best_thresh)
        left_child = self._build_tree(X[left_idx, :], y[left_idx], depth + 1, self.n_samples)
        right_child = self._build_tree(X[right_idx, :], y[right_idx], depth + 1, self.n_samples)
        return Node(best_feat, best_thresh, left_child, right_child)
```

呼叫_best_split 找最好的 feature, threshold，再判斷是否觸碰到停止條件，若是則把當前 node 中最多 instances 的類別當作這個 node 的 class，若非則切割 instance 並繼續往左右 recursively _build_tree。

```python
    def _best_split(self, X, y, features):
        split = {'score':- 1, 'feat': None, 'thresh': None}

        for feat in features:
            X_feat = X[:, feat]
            thresholds = np.unique(X_feat)
            for thresh in thresholds:
                score = self._information_gain(X_feat, y, thresh)

                if score > split['score']:
                    split['score'] = score
                    split['feat'] = feat
                    split['thresh'] = thresh

        return split['score'], split['feat'], split['thresh']
```

Loop 所有 features 和所有 features value 當作 threshold 去計算 information gain，紀錄最佳的 infromation gain 所用的 feature 和 threshold

```python
    def _is_finished(self, depth, parent_samples):
        """Check if stop to grow or not."""
        if (self.max_depth is not None and depth >= self.max_depth
            or self.classes == 1
            or self.n_samples < self.min_samples_split
            or self.n_samples == parent_samples
            or self.n_samples == 0):
            return True
        return False
```

若深度達到最大深度、當前 node 只剩一個類別、當前 node 沒有 instance，或當前 node instances 和 parent node instances 一樣多就達到停止條件。

```python
    def _create_split(self, X, thresh):
        """Create a split in the data based on a given threshold."""
        left_idx = np.argwhere(X <= thresh).flatten()
        right_idx = np.argwhere(X > thresh).flatten()
        return left_idx, right_idx

    def _split_criterion(self, y):
        proportions = np.bincount(y) / len(y)
        if(self.split_criterion == 'gini'):
            value = 1 - np.sum([p*p for p in proportions if p > 0])
        else:
            value = -np.sum([p * np.log2(p) for p in proportions if p > 0])
        return value

    def _information_gain(self, X, y, thresh):
        """Calculate the information gain from splitting on a given feature and threshold."""
        left_idx, right_idx = self._create_split(X, thresh)
        n, n_left, n_right = len(y), len(left_idx), len(right_idx)

        if n_left == 0 or n_right == 0:
                return 0
        parent_loss = self._split_criterion(y)
        child_loss = (n_left / n) * self._split_criterion(y[left_idx]) + (n_right / n) * self._split_criterion(y[right_idx])
        return parent_loss - child_loss
```

_create_split 根據 threshold 切割 instances

_split_criterion 根據 self.split_criterion 判斷計算 gini 或 entropy。

_information_gain 計算 parent node 和 child nodes 的 entropy 或 gini index 來計算 infromation gain

```python
    def fit(self, X, y):
        self.root = self._build_tree(X, y)

    def _predict_one(self, x, node):
        if node.is_leaf():
            return node.value
        if x[node.feature] <= node.threshold:
            return self._predict_one(x, node.left)
        return self._predict_one(x, node.right)

    def predict(self, X):
        return [self._predict_one(x, self.root) for x in X]

    def getdeepest(self):
        return self.deepest
```

fit: 開始 build tree 並 assign root node。

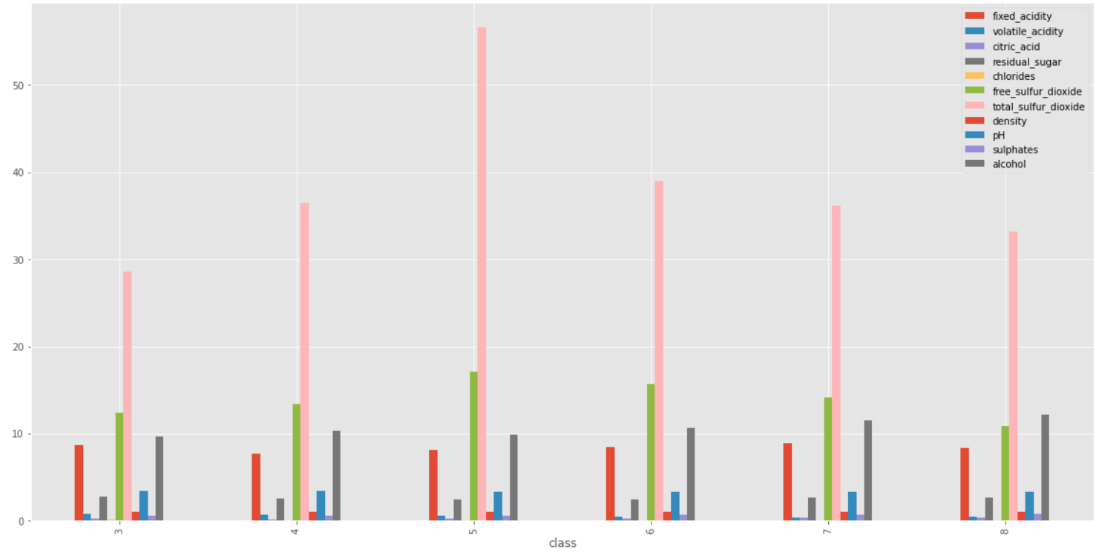
_predict_one, predict: 根據預測傳進來的 instance feature 去走一遍 path 得到 prediction class。

getdeepest: 回傳最深走到第幾層。

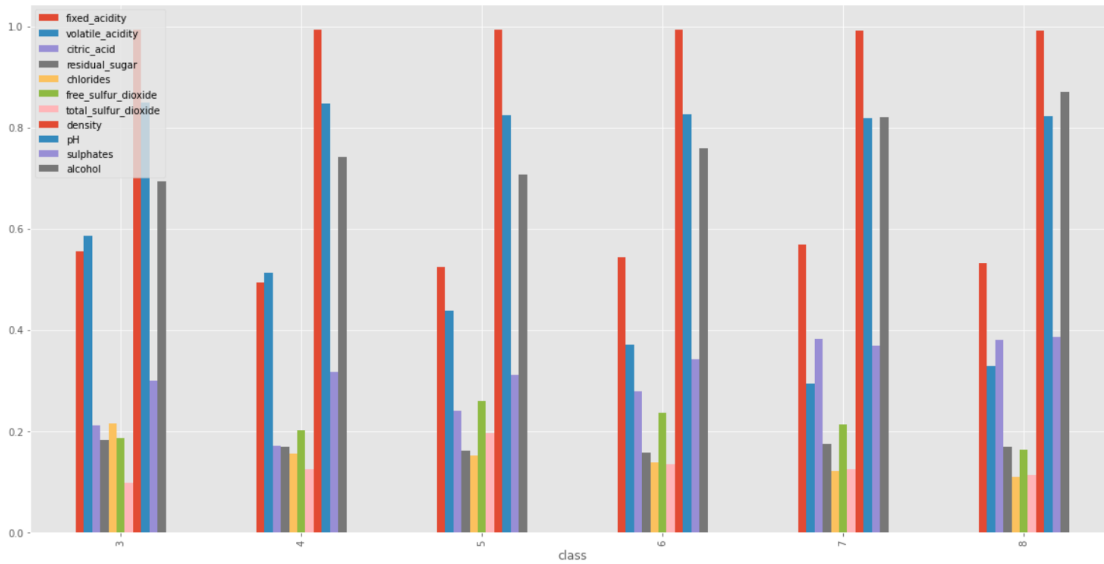
- Dataset 1

Data Preprocessing:

1. Normalize Feature & Remove Outlier
   使用 IQR，把落在 upper bound 和 lower bound 外的 instance remove，但因為資料本來就不多而且很不平衡，所以有額外設定 threshold 去決定是否要 remove
2. Impute Missing Value
   使用 KNN imputer 並根據距離去找最近的 5 個 neighbors 去補值



Unnormalized data



Normalized data

Experiment Result:

1. Without Data Preprocessing

```
train Accuracy: 0.7341153470185728
pred: [ 0  0  0  0  8 90 87 19  1]
label: [ 0  0  0  2  7 85 83 25  3]
Accuracy: 0.5414634146341464
Confusion Matrix:
[[ 0  0  2  0  0  0]
 [ 0  0  4  3  0  0]
 [ 0  6 53 25  1  0]
 [ 0  2 26 47  8  0]
 [ 0  0  5 10 10  0]
 [ 0  0  0  2  0  1]]
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00         7
           5       0.59      0.62      0.61        85
           6       0.54      0.57      0.55        83
           7       0.53      0.40      0.45        25
           8       1.00      0.33      0.50         3

    accuracy                           0.54       205
   macro avg       0.44      0.32      0.35       205
weighted avg       0.54      0.54      0.54       205
```

2. Add Data Preprocessing

```
train Accuracy: 0.8636363636363636
pred: [ 0  0  0  2  6 89 75 23  3]
label: [ 0  0  0  1  6 84 82 23  2]
Accuracy: 0.5353535353535354
Confusion Matrix:
[[ 0  0  1  0  0  0]
 [ 0  1  3  2  0  0]
 [ 1  2 55 22  3  1]
 [ 1  1 29 40 11  0]
 [ 0  2  1 10  9  1]
 [ 0  0  0  1  0  1]]
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.17      0.17      0.17         6
           5       0.62      0.65      0.64        84
           6       0.53      0.49      0.51        82
           7       0.39      0.39      0.39        23
           8       0.33      0.50      0.40         2

    accuracy                           0.54       198
   macro avg       0.34      0.37      0.35       198
weighted avg       0.54      0.54      0.54       198
```

實驗發現即使做了 data preprocessing，prediction 也沒有取得明顯的進步，並且也和沒做 data preprocessing 時一樣在 0.48~0.64 的 accuracy，推測可能是 Data Preprocessing 做的不夠完整，可能有 irrelevent feature 之類的影響。也有使用迴圈去看不同 hyper parameter 的 performance，但基本上也只是在差不多的區間有小小的浮動。

- Dataset 2

Data Preprocessing:

基於 demo 的 code，把 phrase 切成單字並移除 stop words(大量出現但不助於情感分析的單詞)，再轉換成 one hot encoded 的 list 並 padding 成最長的 phrase。有移除 missing value 的資料。

Experiment Result:

```
Train Accuracy: 0.837590235990268
Accuracy: 0.5376852222667201
Confusion Matrix:
[[ 409  390  210   89   34]
 [ 534 1767 1530  421  112]
 [ 354 1851 8763 1556  209]
 [ 145  494 2114 2074  441]
 [  43  130  290  597  413]]
Classification Report:
              precision    recall  f1-score   support

           0       0.28      0.36      0.31      1132
           1       0.38      0.40      0.39      4364
           2       0.68      0.69      0.68     12733
           3       0.44      0.39      0.41      5268
           4       0.34      0.28      0.31      1473

    accuracy                           0.54     24970
   macro avg       0.42      0.43      0.42     24970
weighted avg       0.54      0.54      0.54     24970

pred: [ 1485  4632 12907  4737  1209]
label: [ 1132  4364 12733  5268  1473]
```

這個 dataset 資料雖然比較充足，但也是不平衡，Accuracy 比上一個 dataset 浮動的要來的小很多，基本上都在 0.52~0.54 的區間。

Conclusion:

　　雖然 dataset1 有做了滿多資料前處理，但 Accuracy 還是這麼低，不太確定問題是出在哪裡，或是可能這就是 DT 的極限(?)。至於 dataset2 我有點不太知道該怎麼進行處理所以就只有處理 missing value，原本有想要試試 random forest，但一直遇到各種 error，就放棄了。