Decision Tree classifier

Команда 20

- Стасишин Анна
- Слаблюк Назар

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
```
Ми вирішили обрати датасет про рак грудей.

In [ ]:
```
class Node:

    def __init__(self,X,y,gini,predicted_class=None):
        self.X=X
        self.y=y
        self.gini=gini
        self.feature_index=0
        self.threshold=0
        self.left=None
        self.right=None
        self.predicted_class=predicted_class
```

Як працює Tree Classiffier

Gini

В основі нашої моделі лежить функція активації, або ж cost function - Gini, яка обраховується за формулою

$$G = 1 - \sum_{k=1}^{n} p_k^2$$

Чим ближче Gini до 0, тим є чистішим наш розподіл.

Data split

Модель перебирає всі можливі варіанти розподілу класів та знаходить найкращий варіант з найменшим Gini. Повертає індекс цього розподілу та межу.

Build Tree

Рекурсивно будує дерево. Кожній вершині додає правого чи лівого сина. Якщо у вершини немає дітей - це листок.

Fit

'Обгортка' для нашої моделі.

Predict

Проходить по кожній вершині, допоки в неї є дитина. Якщо в неї нема дітей - обчислює передбачуваний клас.

Evaluate

Обраховує те, наскільки точно наша модель передбачила класи.

In [ ]:
```
class MyDecisionTreeClassifier:

    def __init__(self, max_depth):
        self.max_depth=max_depth

    @staticmethod
    def gini(classes):
        """
        A Gini score gives an idea of how good a split is by how mixed the
```

```
    classes are in the two groups created by the split.

    A perfect separation results in a Gini score of 0,
    whereas the worst case split that results in 50/50
    classes in each group result in a Gini score of 0.5
    (for a 2 class problem).
    """
    num=sum(classes)
    return 1-sum((i/num)**2 for i in classes)

def split_data(self,X,y):
    """
    test all the possible splits in O(N*F) where N in number of samples
    and F is number of features
    return index and threshold value
    """
    m=y.size
    if m<=1:
        return None, None
    num_parent=[np.sum(y==c) for c in range(self.n_classes)]

    best_gini=1-sum((n/m)**2 for n in num_parent)
    best_index,best_threshold=None,None
    for index in range(self.n_features):
        thresholds,classes=zip(*sorted(zip(X[:, index],y)))
        n_left=[0 for _ in range(self.n_classes)]
        n_right=num_parent.copy()
        for i in range(1,m):
            c=classes[i-1]
            n_left[c]+=1
            n_right[c]-=1
            gini_left=self.gini(n_left)
            gini_right=self.gini(n_right)
            gini=(i*gini_left+(m-i)*gini_right)/m
            if thresholds[i]==thresholds[i-1]:
                continue
            if gini<best_gini:
                best_gini=gini
                best_index=index
                best_threshold=(thresholds[i]+thresholds[i-1])/2
    return best_index,best_threshold

def build_tree(self,X,y,depth = 0):
    """
    create a root node
    recursively split until max depth is not exeeced
    """
    num_samples_per_class=[np.sum(y == i) for i in range(self.n_classes)]
    predicted_class = np.argmax(num_samples_per_class)
    node=Node(X,y,None,predicted_class)
    if depth<self.max_depth:
        index,threshold=self.split_data(X,y)
        if index is not None:
            node.gini=self.gini(y)
            indices_left=X[:,index]<threshold
            X_left,y_left=X[indices_left],y[indices_left]
            X_right,y_right=X[~indices_left],y[~indices_left]
            node.feature_index=index
            node.threshold=threshold
            node.left=self.build_tree(X_left,y_left,depth+1)
            node.right=self.build_tree(X_right,y_right,depth+1)
    return node

def fit(self, X, y):
    """
    basically wrapper for build tree / train
    """
    self.n_classes=len(set(y))
    self.n_features=X.shape[1]
    self.tree=self.build_tree(X,y)

def predict(self, X):
    """
```

```
        traverse the tree while there is a child
        and return the predicted class for it,
        note that X_test can be a single sample or a batch
        """
        predictions=[]
        for inputs in X:
            node=self.tree
            while node.left:
                if inputs[node.feature_index]<node.threshold:
                    node=node.left
                else:
                    node=node.right
            predictions.append(node.predicted_class)
        return predictions

    def evaluate(self, X_test, y_test):
        """
        return accuracy
        """
        predictions=self.predict(X_test)
        return sum(predictions==y_test)/len(y_test)
In [ ]:
cancer=load_breast_cancer()
X = cancer.data
y = cancer.target
X, X_test, y, y_test = train_test_split(X, y, test_size= 0.20)
clf=MyDecisionTreeClassifier(10)
clf.fit(X,y)
clf.evaluate(X_test, y_test)
Out[ ]:
0.9210526315789473
```

Висновок

Під час виконання лабораторної роботи, із застосування такого принципу дискретної математики, як дерево рішень, ми створили власну імплементацію Decision Tree Classifier, який є одним із підходів у контрольованому машинному навчанні. Загальна точність вийшла в проміжку (0.9;1), що свідчить про коректність роботи нашої імплементації.