この課題の目的

- コーディングを知識とする
- 実データを扱うことを理解する
- 機械学習のプロセスを理解する

必要最低限の技術

実務経験者向けの内容。上記の目的はあくまで経験できたと思います。さらにはさすがに忙しくて出来ない、また、機械学習を知識とする事にしたい。ちょうどちょうどこの今年度はいろいろ実験で使えるかもしれません。とりあえず、可能な範囲の内容を実務経験者向けに用意しました。ちょうどこの今年度はいろいろ実験で使えるかもしれません。

この課題は機械学習のプロセスを理解することを目的としています。実務（活用データ一体型プロセス）をしっかりとこなすためには、まずはこれくらいの経験は必要かなと思います。実際（活用データ一体型プロセス）をしっかりとこなすためには、まずはこれくらいの経験は必要かなと思います。

課題として使用するライブラリ等の一覧

- データ整形ライブラリ系
- 数値計算ライブラリ系
- 作図ライブラリ系
- 機械学習ライブラリ系
- Scikit-learn の上位版

データセットの用意

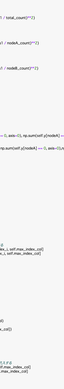
In [1]:
```
import numpy as np
import pandas as pd
X = np.array([[ 0.54895, -1.20072, 1.2, 1.4021 , -2.4308 ],
[ 0.71157, 0.28093, 1.2, 0.9021 , 0.5692 ],
[ 1.5988 , -4.3214 , 1.2, 1.4021 , -2.3908 ],
[ 2.03421, -0.72588, 1.2, 0.40210 , 0.40392 ],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ],
[ 2.60847, -2.2125 , 1.2, 0.90210 , -1.3584 ],
[ 3.1126 , 0.15188, 1.2, 0.4021 , -2.5492 ],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ],
[ 0.58619, -0.37114, 1.2, 0.40210 , -0.4308 ],
[ 0.54895, -1.20072, 1.2, 1.4021 , -2.4308 ],
[ 2.9857 , 0.59837, 1.2, 3.51084 , -0.03184],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ],
[ 0.99761, 0.53302, 1.2, 2.1763 , 1.2547 ],
[ 2.234 , 2.5323 , 1.2, 0.9412 , -1.8019 ],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ],
[ 0.62026, 5.3444 , 1.2, 0.39912 , 3.1324 ],
[ 3.50214, -0.31181, 1.2, 0.3565 , -3.5461 ],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ],
[ 1.1719 , 2.4693 , 1.2, 2.4021 , 0.6192 ],
[ 0.71157, 0.28093, 1.2, 0.90210 , 0.56092 ]])
y = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
print(X.shape, y.shape)
```

In [2]:
```
X.shape, y.shape
```

Out[2]:
```
((20, 5), (20,))
```

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='rainbow')
plt.show()
```



テストデータの分割

学習用（train）と検証用（test）に分けます。シャッフルは必要ない。

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5, test_size=0.2, stratify=y)
```

ScratchDecesionTreeClassifier

In [5]:
```
import numpy as np
import pandas as pd
class ScratchDecesionTreeClassifier:

    # クロスエントロピー
    def __init__(self):
        pass

    # 情報利得のindex(parent, gain)の作成関数
    def _info_gain(self, l, total_l, r, l_B, node0_class0, node0_class1):

        # total_class0 -> np.sum(y[mask == 0], axis=0)
        # total_class1 -> np.sum(y[mask == 1], axis=0) クラス0の個数
        # クラス1の個数

        node1_count = node0_class0 - node0_class0
        node1_count = node0_class1 - node0_class1

        # 分母の計算
        info_gain = l.shape[0] / total_l * (1 - (node0_class0 / l.shape[0])**2 - (node0_class1 / l.shape[0])**2)

        return info_gain

    # ジニ不純度の計算式のための関数
    def _gini_impurity(self, node0_class0, node0_class1):
        # total_class0 -> np.sum(y[mask == 0], axis=0)
        # total_class1 -> np.sum(y[mask == 1], axis=0) クラス0の個数
        # クラス1の個数

        # メソッドのインポート
        node1_count = node0_class0 - node0_class0
        node1_count = node0_class1 - node0_class1

        # 分母のクラス数 class0個数 class1個数
        node0_class0 = total_class0 - node0_class0
        node0_class1 = total_class1 - node0_class1
        node1_class0 = total_class0 - node1_class0
        node1_class1 = total_class1 - node1_class1

        # メソッドの出力
        L_total = 1 - (total_class0 / total_class0)**2 - (total_class1 / total_class0)**2

        # メソッドの出力
        if node0_count == 0:
            L_A = 0
        else:
            L_A = 1 - (node0_class0 / node0_count)**2 - (node0_class1 / node0_count)**2

        # メソッドの出力
        if node1_count == 0:
            L_B = 0
        else:
            L_B = 1 - (node1_class0 / node1_count)**2 - (node1_class1 / node1_count)**2

        return L_total, L_A, L_B

    # メソッドの子ノードの分割
    def _divide_node(self, x, node0_feat, val):
        lg_box = np.array([])

        for i in range(x.shape[0]):
            # 全体の行のindexを取ってきてmaskをかける
            if x[i, node0_feat] < val:
                mask = self.append_lg(i, mask, lg_box)

        return lg_box

    # 子ノードの情報からL,A,Bのindexを返却する関数
    def fit(self, X, y):
        # X, y = X, y
        self.y = y
        lg_box = self._divide_node(X, np.shape(X)[0], 0, 0)
        lg_box = self._divide_node(X, np.shape(X)[0], 0, 1)

        # 初期のL,A,Bのindexを作成
        lg_box_all = np.concatenate([lg_box], lg_box-1])

        # 最初の情報利得を出力するためのパラメータを保存する
        info_gain = self.info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0))

        # 情報利得がマックスな
        lg_box = np.appendlg_box, self._info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0))

        return lg_box

    # 子ノードの子ノードに対するindexを返却する関数
    def fit(self, X, y):
        # X, y = X, y
        self.y = y
        lg_box = self._divide_node(X, self._info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0)))

        # 子ノードの情報利得を計算する
        lg_box_1 = self._divide_node(X, np.shape(X)[0], lg_box_all[0], lg_box_all[1])
        lg_box_2 = self._divide_node(X, np.shape(X)[0], lg_box_all[2], lg_box_all[3])

        # 初期のindexの保存
        lg_box = np.concatenate([lg_box], lg_box_1])
        lg_box = np.concatenate([lg_box], lg_box_2])

        # 最初の情報利得を出力するためのパラメータを保存する
        info_gain = self.info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0))

        # 情報利得がマックスな
        lg_box = np.appendlg_box, self._info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0))

        return lg_box

    # 学習データが第1ノードに対するindexを返却する関数
    def fit(self, X, y):
        # X, y = X, y
        self.y = y
        lg_box = self._divide_node(X, self._info_gain(np.shape(X)[0], np.shape(X)[0], lg_box_all, np.sum(y[lg_box_all == 0], axis=0), np.sum(y[lg_box_all == 1], axis=0)))

        # 学習データが第1ノードに対するindexを返却する
        lg_box_1 = self._divide_node(X, np.shape(X)[0], lg_box_all[0], lg_box_all[1])
        lg_box_2 = self._divide_node(X, np.shape(X)[0], lg_box_all[2], lg_box_all[3])

        # 学習データが第2ノードに対するindexを返却する
        lg_box = np.concatenate([lg_box], lg_box_1])
        lg_box = np.concatenate([lg_box], lg_box_2])

        # 学習データが第3ノードに対するindexを返却する
        lg_box = np.concatenate([lg_box], lg_box_1])
        lg_box = np.concatenate([lg_box], lg_box_2])

        # 学習データが第4ノードに対するindexを返却する
        lg_box = np.concatenate([lg_box], lg_box_1])
        lg_box = np.concatenate([lg_box], lg_box_2])

        # 学習データのindexからノードのクラスにインデックスを当てはめる
        node_count = self.fit(X, self.fit(X, self._divide_node(X, self.fit(X, self._divide_node(X, self.fit(X, self._divide_node
```