Université Mouloud Mammeri de Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique
Département d'Informatique

Implementer l'arbre de décision C4.5

MODULE : Data maining 2.

Réalisé par :

- **KHELF Ferhat**
- **AIT MOHAMED Saadi**
- **IKERMOUD Amayes**
- **HAMICHE Melissa**
- **OUALI Massinissa**
- **TOUZI Mahrez**

2022/2023

SOMMAIRE

• Introduction :

Les entreprises évoluent dans un environnement de plus en plus complexe, avec le succès de la numérisation et l'avènement d'Internet.

Le web représente un réservoir de données GIGANTESQUES qui continue de croître chaque jour, et avec l'émergence des objets connectés, les entreprises sont entrées dans l'air du Big Data, elles sont inondées d'informations en tous genres, et le volume des données stockées est gigantesque, sans parler de leur variété (textes, images, sons, etc...) qui ne cesse de s'accroître toujours en corrélation avec les données qui circulent en ligne.

Donc le problème actuel des entreprises n'est plus de récupérer de la donnée, au contraire elles en récupèrent dans le sens où elles n'arrivent plus à les traiter dans le temps voulu, par conséquent, les moyens traditionnels d'analyse ne sont plus en mesure de faire face à ces énormes quantités de données, et le data Manning qui est apparu à la fin des années 80 a prouvé son efficacité dans l'analyse de ces grandes quantités de données.

Donc, La data Manning permet à la machine de traiter ces données de façon plus rapide, et ce gain de temps ne peut être qu'un avantage des plus important pour le développement des entreprise.

• Arbre de décision :

Un **arbre de décision** est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches (les « feuilles » de l'arbre), et sont atteintes en fonction de décisions prises à chaque étape.

Il existe plusieurs algorithmes d'arbres de décision :

• ID3 :

L'**algorithme ID3** a été développé à l'origine par Ross Quinlan .C'est un algorithme de classification supervisé.

Pour séparer les données, sélectionne un attribut non déjà utilisé qui a la plus faible entropie.

• CART

L'algorithme CART permet de construire un arbre de décision lorsque les attributs sont binaires.

Il se base pour la séparation sur la mesure de GINI.

- **C4.5 :** (extension d'ID3)

C 4.5 est un algorithme qui a été développé par Ross Quinlan. Il est utilisé pour construire des arbres de décision à partir des données d'apprentissage et pour prédire la classe d'un nouvel enregistrement en se basent sur les caractéristiques de cet enregistrement.

L'algorithme C4.5 utilise l'entropie et le gain d'information pour sélectionner les caractéristiques les plus pertinentes à utiliser à chaque nœud de l'arbre.

➢ Entropie :

L'entropie est une mesure de la qualité d'un nœud de décision dans un arbre de décision. Plus l'entropie est faible, mieux c'est, car cela signifie que les exemples de données dans le nœud sont plus similaires entre eux.

Pour calculer l'entropie d'un nœud dans un arbre décision, vous devez d'abord compter le nombre de chaque classe dans le nœud. Vous pouvez calculer l'entropie du nœud en utilisant la formule suivante :

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

➢ Gain d'information :

Le gain d'information est une mesure utilisée par l'algorithme C4.5 pour déterminer quelles caractéristiques utiliser pour séparer les exemples de données lors de la construction de l'arbre de décision. Plus le gain d'information est élevé, mieux c'est, car cela signifie que la caractéristique sélectionnée permet de mieux séparer les exemples de données et de construire un arbre de décision plus précis.

Vous pouvez calculer le gain d'information en utilisant la formule suivante :

$$IG(X, A) = H(X) - \sum P(A=a_i) * H(X|A=a_i)$$

➢ Les attributs a valeur continues :

L'algorithme **C4.5** propose la solution suivante pour les attributs a valeur continu :

- Supprimer les valeurs redondantes.
- Ordonner la liste (croissant ou décroissant).
- Calculer le gain d'information a chaque point de séparation.
- Séparer par la valeur qui a la valeur du gain d'information maximale.

➢ Les valeurs manquantes :

Durant la construction de l'arbre de décision il est possible de gérer les données pour lesquels certains attributs ont une valeur inconnue. Une solutions consiste a attribuer a l'attribut manquant une valeur:

- la valeur la plus répondu dans la collection.
- moyenne de toutes les valeurs de l'attribut.
- La valeur maximale de l'attribut.

• Problématique :

- Comment implémenter l'algorithme C4.5 par le langage python on utilisant uniquement les bibliothèques **numpy**, **matplotlib** et **pandas ?**
- Comment savoir si le model est correct ou pas?

• Prétraitements sur les données :

L'objectif du data set qu'on a utilisé est de prédire la chance d'un étudiant pour être admis a l'université (admis ou non-admis)

Pour pouvoir utilisé les données du data set on a effectué les prétraitements suivant :

1. Chargement des données : en utilisant la bibliothèque pandas.

2. Convertir la classe de sortie : remplacer les valeurs par 1 si l'étudiant est admis et par 0 sinon.

```python
def convert_classes(s):
    if s >= 0.50:
        return 1
    else:
        return 0

def convert_data(df: pd.DataFrame):
    l = []
    for _, r in df.iterrows():
        t = list(r[1:])
        t[-1] = convert_classes(t[-1])
        l.append(t)
    return l
```

Boucler sur chaque ligne du dataframe

Supprimer la première colonne (id).

3. Transformé les données a une liste python

4. Transformé la liste en un array numpy.

5. mélanger les données aléatoirement.

```python
L = np.array(L)
np.random.shuffle(L)
X = L[:, :7]
Y = L[:, 7]
Y = list(Y)
l2 = []
a = int(len(X) * 0.7)
x_train = X[0: a]
y_train = Y[0: a]
x_test = X[a:]
y_test = Y[a:]
```

5. Séparer la liste en 2 parties (x et y) : les 7 première valeurs de la liste sont les données (x) et la dernière valeur c'est la sortie (y)

6. Diviser les 2 parties en deux : 70% pour l'entrainement et 30% pour le test.

• Implémentation de l'algorithme :

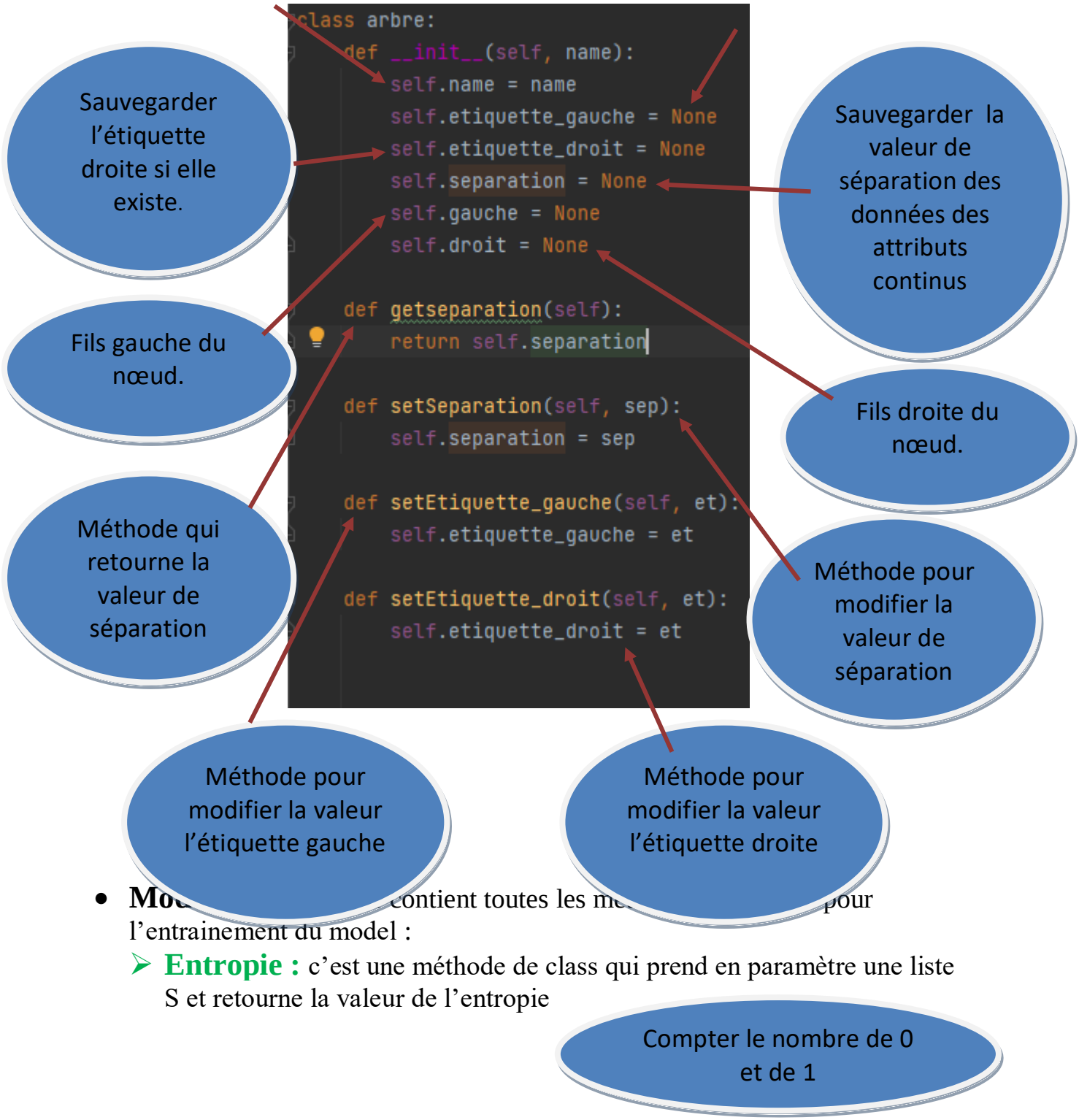
Pour implémenter l'algorithme on a créé 3 fichiers python :

- **Main.py : -** pour faire le chargement et les prétraitements des données.
  - pour exécuter le programme.

- **Nœud.py :** contient une class arbre qui représente les nœuds de l'arbre.

Sauvegarder le nom du nœud.

Sauvegarder l'étiquette gauche si elle existe.



- **Mod** contient toutes les m pour l'entrainement du model :
  - **Entropie :** c'est une méthode de class qui prend en paramètre une liste S et retourne la valeur de l'entropie

Compter le nombre de 0 et de 1

```python
def entropie(self, s):
    H = 0
    nbr1 = s.count(1)
    nbr0 = s.count(0)
    s1 = 0
    s2 = 0
    if nbr1 != 0:
        s1 = (nbr1 / len(s)) * math.log(nbr1 / len(s), 2)
    if nbr0 != 0:
        s2 = (nbr0 / len(s)) * math.log(nbr0 / len(s), 2)
    H = -(s1 + s2)
    return round(H, 3)
```

Appliquer la formule de l'entropie

- **Gain_info :** c'est une méthode de class qui prend en paramètre une liste **L** et retourne la valeur du gain d'information.

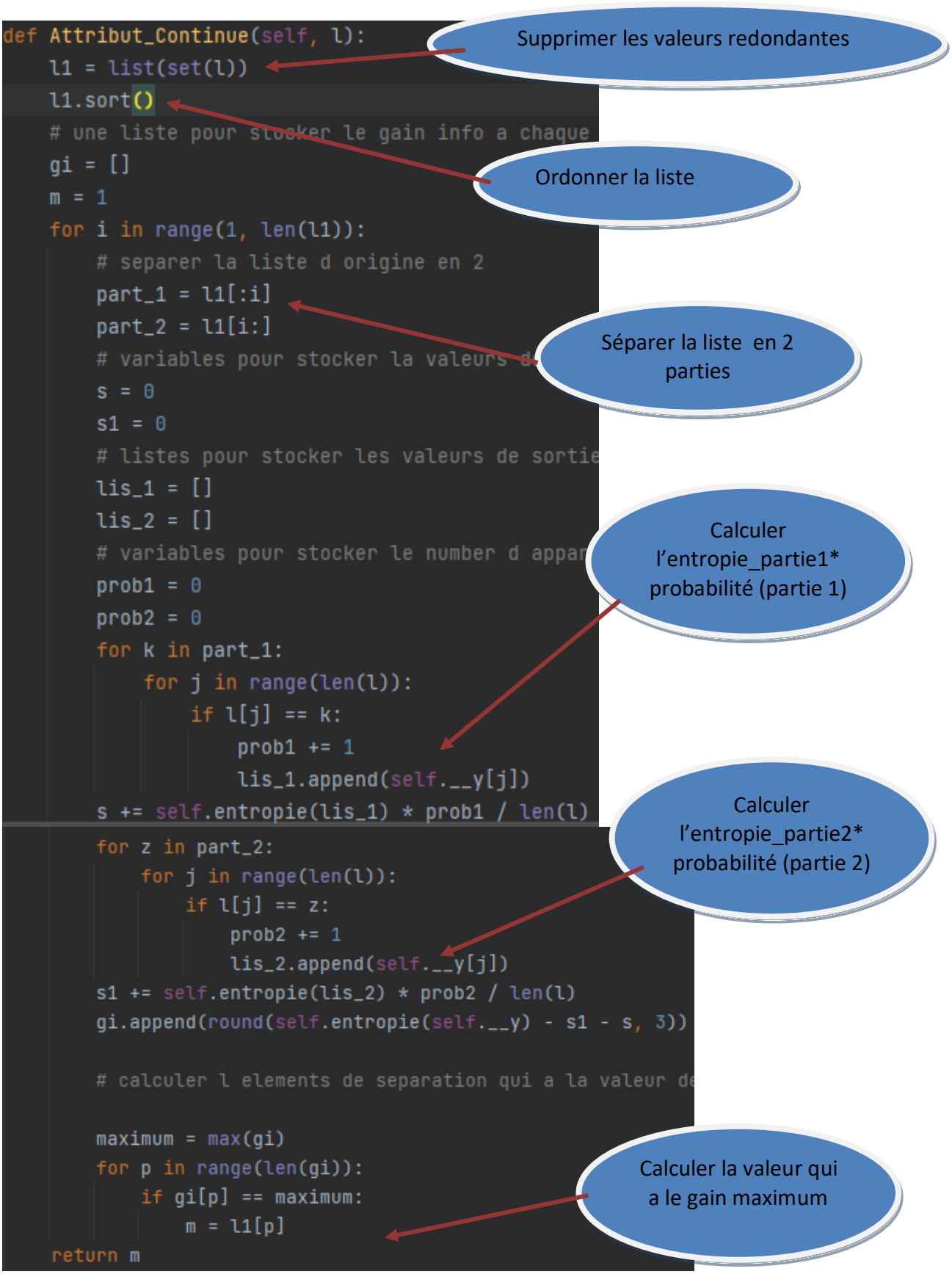
```python
def gain_info(self, l):
    l1 = list(set(l))
    s = 0
    for i in l1:
        list_lg = []
        for j in range(len(l)):
            if l[j] == i:
                list_lg.append(self.__y[j])
        s += self.entropie(list_lg) * (l.count(i) / len(l))

    return round(self.entropie(self.__y) - s, 3)
```

Récupérer toutes les valeurs possibles de la liste

Calculer l'entropie conditionnelle pour chaque valeur

Calculer le gain

- **Attribut_continu :** c'est une méthode de class qui prend en paramètre une liste **L** et calcul le gain d'information a chaque point de séparation, puis retourne la valeur qui a le gain maximum.




```
def Attribut_Continue(self, l):
    l1 = list(set(l))
    l1.sort()
    # une liste pour stocker le gain info a chaque
    gi = []
    m = 1
    for i in range(1, len(l1)):
        # separer la liste d origine en 2
        part_1 = l1[:i]
        part_2 = l1[i:]
        # variables pour stocker la valeurs d
        s = 0
        s1 = 0
        # listes pour stocker les valeurs de sortie
        lis_1 = []
        lis_2 = []
        # variables pour stocker le number d appar
        prob1 = 0
        prob2 = 0
        for k in part_1:
            for j in range(len(l)):
                if l[j] == k:
                    prob1 += 1
                    lis_1.append(self.__y[j])
        s += self.entropie(lis_1) * prob1 / len(l)
        for z in part_2:
            for j in range(len(l)):
                if l[j] == z:
                    prob2 += 1
                    lis_2.append(self.__y[j])
        s1 += self.entropie(lis_2) * prob2 / len(l)
        gi.append(round(self.entropie(self.__y) - s1 - s, 3))

        # calculer l elements de separation qui a la valeur de

        maximum = max(gi)
        for p in range(len(gi)):
            if gi[p] == maximum:
                m = l1[p]
    return m
```

- **Indice_gain_maximum :** c'est une méthode de class qui prend en paramètre les données d'entrainement et qui retourne l'indice de l'attribut qui a le gain d'information maximum (pour choisir l'attribut de séparation).

```
def indice_gain_maximum(self, data):
    max = 0
    indice = 0
    for i in range(len(data)):
        g = self.gain_info(data[i])
        if max < g:
            max = g
            indice = i
    return indice
```

- **Fit :** c'est une méthode de class qui prend en paramètre la liste des données et la liste des sortie, puis elle fais l'entrainement du model et retourne l'arbre construit.

```
def fit(self, x: [], y: []):
    self.__x = x
    self.__y = y

    att = ["GRE_Score", "TOEFL_Score", "University_Rating",
           "SOP", "LOR", "CGPA", "Research"]
    data = [[], [], [], [], [], [], []]

    # transformer les donnee en une liste
    for i in range(len(x)):
        for j in range(len(data)):
            data[j].append(x[i, j])

    # cree la racine
    indice = self.indice_gain_maximum(data)
    Noeud = arbre(att[indice])
    Noeud.setSeparation(self.Attribut_Continue(data[indice]))

    R = self.creation_arbre(Noeud, indice, data, att)
    return R
```

Transformé la data en une liste d'attribut (chaque liste représente toute les valeurs possibles de l'attribut

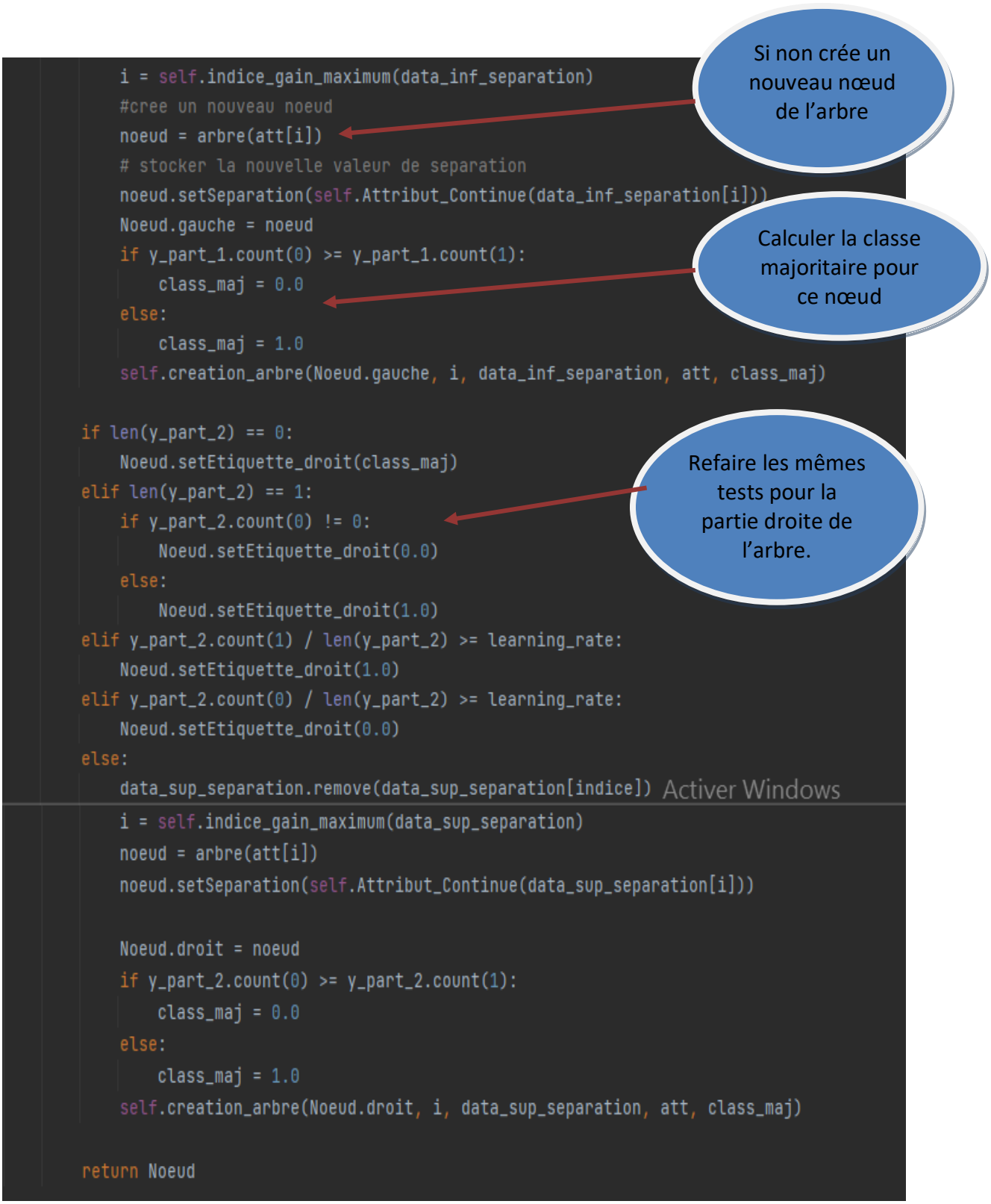
Calculer l'indice de l'attribut qui a le gain maximum

Cree la racine de l'arbre.

Calculer la valeur qui sépare l'attribut continue en 2 parties

- **Creation_arbre :** méthode de classe récursive qui prend en paramètre :
  - **Nœud** : la racine de l'arbre.
  - **Indice** : l'indice de l'attribut de séparation.
  - **Data** : les données.
  - **Att** : liste des attributs possibles.
  - **Class_maj** : la class majoritaire du père.
  - **Learning_rate** : le taux d'erreur tolérer lors de l'entrainement.

Elle retourne un nœud de l'arbre.



- **Prédict :** méthode de class qui prend en paramètre une racine d'un arbre, un échantillon, la listes des attributs de l'arbre, puis elle retourne la classe prédite par l'algorithme pour l'échantillon.

```python
def predict(self, racine, data, att):
    for i in range(len(att)):
        if att[i] == racine.name:
            condidat = i
    if data[condidat] <= racine.separation:
        if racine.etiquette_gauche is not None:
            return racine.etiquette_gauche

        elif racine.gauche is not None:
            self.predict(racine.gauche, data, att)

    else:
        if racine.etiquette_droit is not None:
            return racine.etiquette_droit
        elif racine.droit is not None:
            self.predict(racine.droit, data, att)
```

Récupérer l'indice de l'attribut de séparation

On fait le test, Si l'étiquette existe ➔ return l'étiquette Sinon passer dans le nœud suivant

➢ **Prédict_all :** méthode de class qui prend en paramètre une racine d'un arbre, liste de données à prédire, la listes des attributs et retourne une liste de class prédite pour chaque échantillon dans la liste.

```python
def predict_all(self, racine, data, att):
    predictions = []
    for d in data:
        predictions.append(self.predict(racine, d, att))

    return predictions
```

- # Evaluation du model :
  - ## Validation croisée :

```python
def validation_croise(m, X, Y, volets, att):
    acc = 0
    for i in range(volets):
        y_test = []
        r_val = range(int(i * len(X) / volets), int((i + 1) * len(X) / volets))
        r_train = list(set(range(len(X))).difference(set(r_val)))
        x_train = X[r_train]
        for j in r_train:
            y_train.append(Y[j])
        x_test = X[r_val]
        for j in r_val:
            y_test.append(Y[j])

        r = m.fit(x_train, y_train)

        y_pred = m.predict_all(r, x_test, att)
        a = accuracy(y_pred, y_test)

        y_pred.clear()
        acc = acc + a
    return acc / volets
```

- **Accuracy, précision, error_rate, rappel, f_score :**

```python
def evaluation(prediction, trues):
    s = 0;tp = 0;tn = 0;fp = 0;fn = 0
    for i in range(len(prediction)):
        if trues[i] == prediction[i]:
            if trues[i] == 1:
                tp += 1
            else:
                tn += 1
        else:
            if trues[i] == 1:
                fn += 1
            else:
                fp += 1
    pre_class_1 = tp / (tp + fp)
    rap_class_1 = tp / (tp + fn)
    pre_class_2 = tn / (tn + fn)
    rap_class_2 = tn / (tn + fp)
    print('accuracy =', (tp + tn) / (tp + tn + fp + fn))
    print('error_rate =', (fp + fn) / (tp + tn + fp + fn))
    print('precision pour la class forte =', pre_class_1)
    print('rappel pour la class forte =', rap_class_1)
    print('f_score pour la class forte = ', 2 * pre_class_1 * rap_class_1 / (pre_class_1 + rap_class_1))
    print('precision pour la class faible =', pre_class_2)
    print('rappel pour la class faible =', rap_class_2)
    print('f_score pour la class faible = ', 2 * pre_class_2 * rap_class_2 / (pre_class_2 + rap_class_2))
```

Compter le nombre de prédiction correctement classé comme forte(1)

Compter le nombre de prédiction correctement classé comme faible(0)

Compter le nombre de prédiction mal classé comme forte(1)

Compter le nombre de prédiction mal classé comme faible(0)

- **Résultat :**

```
accuracy = 0.95
error_rate = 0.05
precision pour la class forte = 0.9
rappel pour la class forte = 0.95
f_score pour la class forte =  0.92
precision pour la class faible = 0.98
rappel pour la class faible = 0.95
f_score pour la class faible =  0.96

Process finished with exit code 0
```

Accuracy : le pourcentage des éléments bien placés : **95%**

Error rate : le pourcentage des éléments mal placés : **5%**

Précision de la classe 1 : proportion des éléments pertinents prédits parmi les éléments prédits par l'algorithme (classe 1) : **90%.**

Précision de la classe 2 : proportion des éléments pertinents prédits parmi les éléments prédits par l'algorithme (classe 2) :**98%**

Rappel de la classe 1 : proportion des éléments pertinents prédit parmi l'ensemble des éléments pertinents (classe 1) :**95%**

Rappel de la classe 2 : proportion des éléments pertinents prédit parmi l'ensemble des éléments pertinents (classe 2) :**95%**