

REPUBLIQUE DU CAMEROUN
Paix-Travail-Patrie
UNIVERSITE DE YAOUNDE 1
DEPARTEMENT
D'INFORMATIQUE
BP/P.O.Box 812
Yaounde-Cameroun



REPUBLIC OF CAMEROON
Peace-Work-Fatherland
UNIVERSITY OF YAOUNDE 1
COMPUTER SCIENCES
DEPARTMENT
BP/P.O.Box 812
Yaounde-Cameroun

Devoir 9 - INFO5099 :
Algorithme d'Arbre de Decision

Présenté par :

Prénom	Nom	Matricule	Option
VICTOR NICO	DJIEMBOU TIENTCHEU	17T2051	DS

Enseignant : Dr. MESSI Thomas

Table des matières

1	Généralité	4
1.1	Versions existantes et comparaison	5
1.2	Fonctionnement d'ID3	7
2	Algorithmes	9
2.1	Version récursive	9
2.2	Version itérative	10
3	Implementation python From scratch	11
3.1	Module de gestion de fichier et données	11
3.2	Module de métriques	14
3.3	Module de gestion d'arbre de décision	16
4	Résultats expérimentaux (données, métriques d'évaluation, résultats)	18
4.1	Données	18
4.2	Comparaisons des résultats	19
5	Conclusion	19

1 Généralité

La classification est une tâche qui consiste à associer un objet à une catégorie.

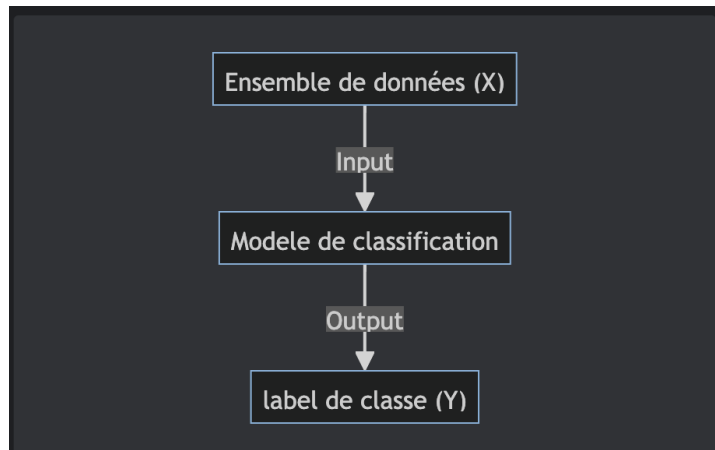


FIGURE 1 – Schéma d'une Classification

Un arbre de décision est un algorithme qui permet de former un modèle qui se base sur les arbres de décision pour associer une classe ou une catégorie à une entrée.

Explicitement, un arbre classique comporte une racine, des branches et des feuilles. Les arbres de décision en comportent de même. Les conditions qui permettent de séparer au mieux l'ensemble de données est contenu dans les nœuds, les sorties possibles de la condition de séparation sont portées par les branches ou arêtes ; et enfin les catégories ou classes sont définies par les feuilles.

Le nœud racine est le parent de tous les nœuds, c'est le nœud le plus haut dans l'arbre. L'arbre de décision permet donc de représenter chaque attribut dans les nœuds ; les branches définissent les règles de séparations et les feuilles définissent les sorties qui peuvent être numériques ou catégorielles.

Nous remarquons donc qu'il s'agit là d'une reproduction du processus de décision de la vie réelle. Si nous voulons par exemple représenter un processus de décision dépendant des conditions météorologiques, nous pouvons construire l'arbre suivant

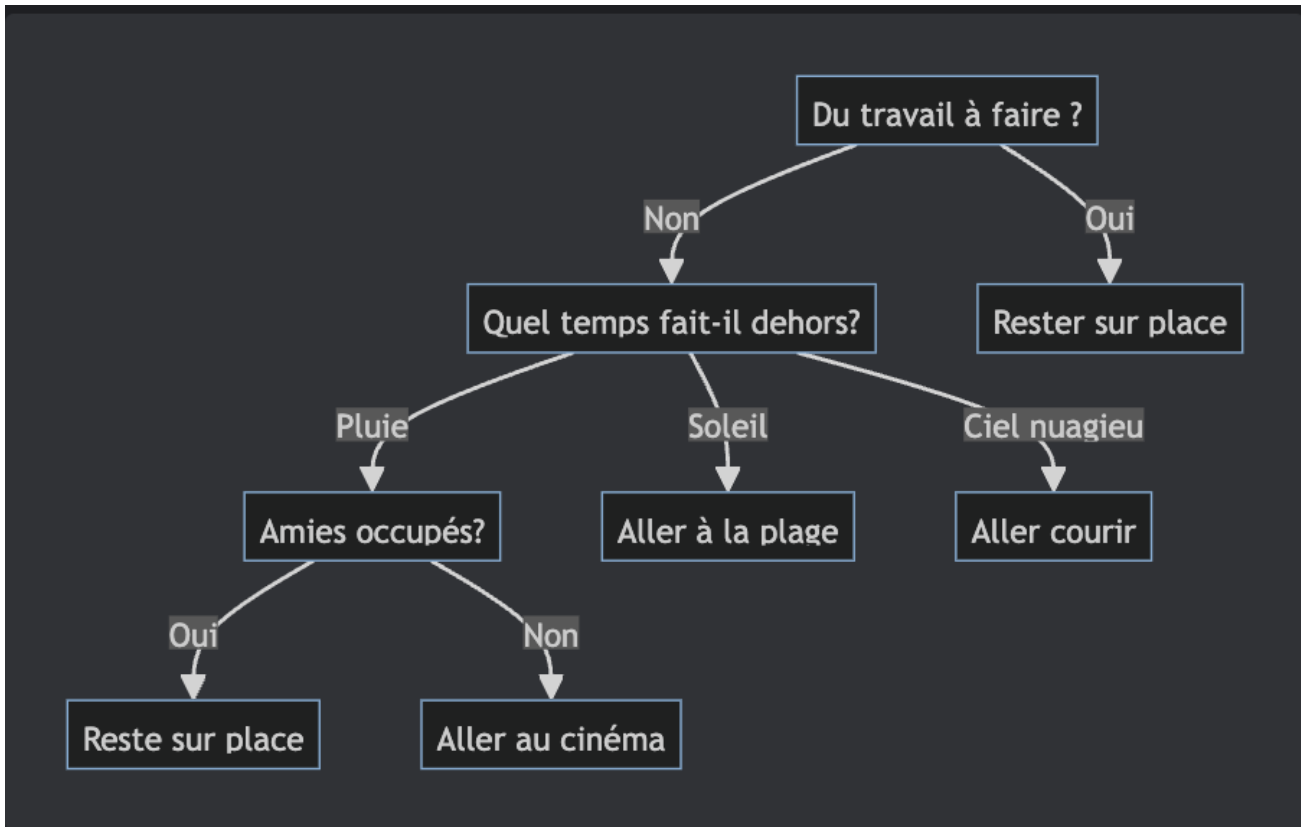


FIGURE 2 – Exemple de processus de décision

Nous remarquons cette situation de décision dans la vie réelle peut bien se résumer ou se reproduire en faisant usage d'un arbre de décision.

Lorsque les données n'offrent pas d'avantages lors du fractionnement, l'exécution est directement interrompue. Essayez de trouver un test à la fois plutôt que d'optimiser l'ensemble de l'arbre.

1.1 Versions existantes et comparaison

Il existe plusieurs variétés d'algorithmes d'arbre de décision, les tableaux ci contre vous détailles les spécificités de chacune d'elles.

L'algorithme *ID3* est considéré comme un algorithme d'arbre de décision très simple (Quinlan, 1986). *ID3* utilise le gain d'information comme critère de division. La croissance s'arrête lorsque toutes les instances appartiennent à une seule valeur de la caractéristique cible ou lorsque le meilleur gain d'information n'est pas supérieur à zéro. *ID3* n'applique aucune procédure d'élagage et ne traite pas les attributs numériques ou les valeurs manquantes.

C4.5 est une évolution de *ID3*, présentée par le même auteur (Quinlan, 1993), qui utilise le rapport de gain comme critère de division. Le découpage cesse lorsque le nombre d'instances à découper est inférieur à un certain seuil. L'élagage basé sur les erreurs est effectué après la phase de croissance. *C4.5* peut traiter des attributs numériques. Il peut produire à partir d'un ensemble d'apprentissage qui incorpore des valeurs

manquantes en utilisant des critères de rapport de gain corrigés comme présenté ci-dessus.

CART est l'acronyme de Classification and Regression Trees (Breiman et al., 1984) et se caractérise par le fait qu'il construit des arbres binaires, c'est-à-dire que chaque nœud interne a exactement deux arêtes sortantes. Les scissions sont sélectionnées à l'aide des deux critères de séparation et l'arbre obtenu est élagué par l'élagage coût-complexité. Lorsque cela est possible, CART peut prendre en compte les coûts de mauvaise classification dans l'induction de l'arbre. Une caractéristique importante de CART est sa capacité à générer des arbres de régression. Les arbres de régression sont des arbres dont les feuilles prédisent un nombre réel et non une classe. Dans le cas de la régression, CART recherche les divisions qui minimisent l'erreur quadratique des prédictions (l'écart le moins élevé). La prédiction dans chaque feuille est basée sur la moyenne pondérée pour le nœud.

Dès le début des années soixante-dix, des chercheurs en statistiques appliquées ont développé des procédures de génération d'arbres de décision, telles que : AID (Sonquist et al., 1971), MAID (Gillo, 1972), THAID (Morgan et Messenger, 1973) et CHAID (Kass, 1980). CHAID (Chi-square-Automatic-Interaction-Detection) a été conçu à l'origine pour traiter uniquement des attributs nominaux. Pour chaque attribut d'entrée a_i , CHAID recherche la paire de valeurs dans V_{a_i} qui est la moins significativement différente par rapport à l'attribut cible. La différence significative est mesurée par la valeur p obtenue à partir d'un test statistique. Le test statistique utilisé dépend du type d'attribut cible. Si l'attribut cible est continu, un F test est utilisé. S'il est nominal, un test chi-carré de Pearson est utilisé. S'il est ordinal, un test de rapport de vraisemblance est utilisé.

CHAID vérifie si la valeur p obtenue est supérieure à un certain seuil de fusion. Si la réponse est positive, il fusionne les valeurs et recherche une autre paire potentielle à fusionner. Le processus est répété jusqu'à ce qu'aucune paire significative ne soit trouvée. Le meilleur attribut d'entrée à utiliser pour scinder le nœud actuel est alors sélectionné, de sorte que chaque nœud enfant soit constitué d'un groupe de valeurs homogènes de l'attribut sélectionné. Il convient de noter qu'aucune scission n'est effectuée si la valeur p ajustée du meilleur attribut d'entrée n'est pas inférieure à un certain seuil de scission. Cette procédure s'arrête également lorsque l'une des conditions suivantes est remplie :

1. La profondeur maximale de l'arbre est atteinte.
2. le nombre minimum de cas dans le nœud pour être un parent est atteint, de sorte qu'il ne peut plus être scindé.
3. le nombre minimum de cas dans le nœud pour être un nœud enfant est atteint.

CHAID traite les valeurs manquantes en les considérant toutes comme une seule catégorie valide. CHAID n'effectue pas d'élagage.

Algorithme d'arbre de décision	Type de données	Méthode de fractionnement des données numériques	Outils possibles
CHAID (CHI-square Automatic Interaction Detector)	Catégorielle	Indéfini	SPSS answer tree
ID3 (Iterative Dichotomiser 3)	Catégorielle	Pas de restriction	WEKA
C4.5	Catégorielle et Numérique	Pas de restriction	WEKA
CART (Classification and Regression Tree)	Catégorielle et Numérique	Séparation binaire	CART 5.0

TABLE 1 – Tableau caractérisation des arbres de décision

Nom de l'algorithme	Classification	Description
CHAID (CHi-square Automatic Interaction Detector)	Antérieur à l'implémentation originale de l'ID3	Ce type d'arbre de décision est utilisé pour une variable nominale à échelle. La technique détecte la variable dépendante à partir des variables catégorisées d'un ensemble de données
ID3 (Iterative Dichotomiser 3)	Utilise la fonction d'entropie et le gain d'information comme mesures	La seule préoccupation concerne les valeurs discrètes. Par conséquent, l'ensemble de données continues doit être classé dans l'ensemble de données discrètes
C4.5	La version améliorée de l'ID 3	Traite à la fois des données discrètes et continues. Il peut également gérer les données incomplètes
CART (Classification and Regression Tree)	Utilise l'indice de Gini comme mesure	En appliquant la division numérique, nous pouvons construire l'arbre basé sur CART

TABLE 2 – Tableau de description des arbres de décision

1.2 Fonctionnement d'ID3

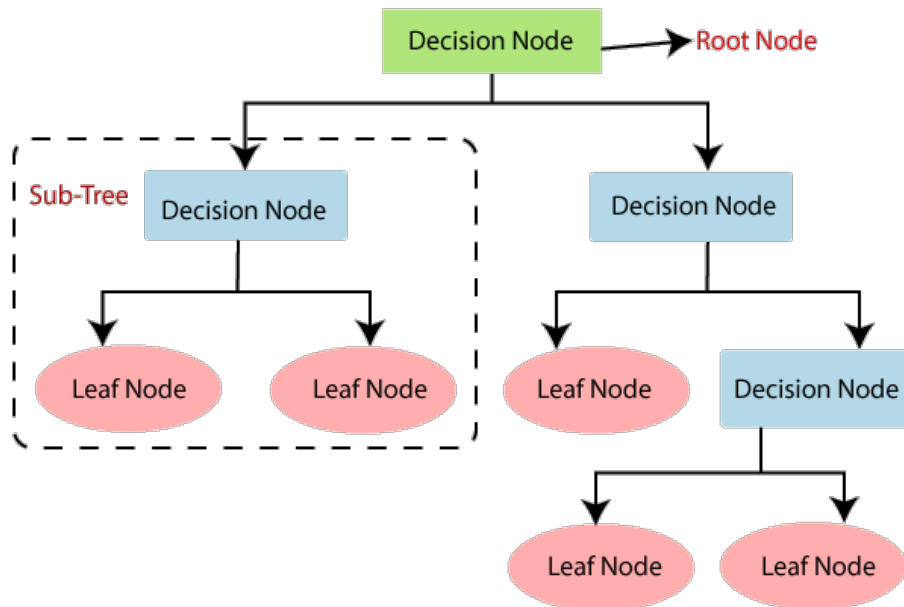


FIGURE 3 – Squelette d'un arbre de décision

L'algorithme d'ID3 se base essentiellement sur un certain nombre d'opération :

1. *Le calcul de l'impureté dans l'ensemble de données.* Ceci est possible en estimant soit **l'indice de Gini** ou **l'entropie** :

— **Entropie** : L'entropie est une mesure de l'incertitude dans un ensemble de données. Plus l'entropie est élevée, plus les données sont dispersées et moins elles sont prévisibles. Dans le contexte de l'algorithme ID3, l'entropie est utilisée pour quantifier l'impureté d'un ensemble d'exemples en termes de distribution de classes.

Une entropie de 0 signifie que tous les exemples de l'ensemble de données appartiennent à la même classe, donc l'ensemble est pur.

Une entropie plus proche de 1 signifie que les exemples sont répartis de manière équitable entre plusieurs classes, donc l'ensemble est impur. Son equation est fournie par

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

— **indice de Gini** : L'indice de Gini est une mesure alternative utilisée pour évaluer l'impureté d'un ensemble de données. Il quantifie la probabilité qu'un exemple choisi au hasard soit incorrectement classé en fonction de la distribution des classes.

Un indice de Gini de 0 indique que l'ensemble de données est pur, c'est-à-dire que tous les exemples appartiennent à la même classe.

Un indice de Gini plus proche de 1 indique une plus grande impureté, avec une distribution équitable ou presque équitable des exemples entre les classes.

$$Gini = 1 - \sum_{i=1}^n (P_i)^2$$

2. Lorsque nous avons, ceci nous intéresse plus à comment identifier la meilleurs condition de séparation sachant une variable et encore ; quelle variable qui permet de purifier au mieux l'ensemble de données :

$$Information\ Gain_{Classification} = E(d) \sum \frac{|s|}{|d|} E(s)$$

$$Information\ Gain_{Regression} = Variance(d) \sum \frac{|s|}{|d|} Variance(s)$$

3. la construction d'un arbre de décision peut donc se définir comme une succession de recherche du meilleur attribut qui permet de purifier au mieux l'ensemble de données jusqu'à atteindre un seuil où tous les exemples d'un sous ensemble de données appartiennent à la même classe.

Notons que l'objectif de ID3 est de réduire l'entropie, c'est-à-dire de diviser les données de manière à obtenir des sous-ensembles plus purs lorsque nous utilisons l'entropie comme indicateur. D'autre part, de minimiser l'indice de Gini, c'est-à-dire de trouver la division des données qui minimise l'incertitude de classification pour une version d'ID3 basé sur Gini.

Concernant la sensibilité aux valeurs extrêmes, entropie l'emporte haut la main sur Gini et en terme d'utilisation, Entropie est plus propre à ID3 et Gini à CART.

2 Algorithmes

2.1 Version récursive

Algorithm 1 Algorithme de l'arbre de décision recursive (DTR)

Require: *dataset* : Ensemble d'entraînement $D \leftarrow \{(x_1, y_1), \dots, (x_n, y_n)\}$, *depth* la profondeur de l'arbre et le critère d'arrêt

Require: *attributes* : Liste des attributs de l'ensemble de données

Require: *target_attribute* : Attribut cible (classe)

Ensure: *tree* : Arbre de décision résultant

```
1: if  $D, depth$  satisfont le critère d'arrêt then
2:   return un noeud avec comme etiquette la classe majoritaire du jeu
3: else
4:   Trouver le meilleur candidat diviseur de  $D \leftarrow \{col, val, ig, type\}$ 
5:   if  $ig$  satisfait le critère d'arrêt then
6:     return un noeud avec comme etiquette la classe majoritaire du jeu
7:   else
8:      $gauche, droite \leftarrow$  Diviser  $D$  sur la base du meilleur candidat
9:     if Candidat numérique then
10:       $cond \leftarrow col + ' <=' + val$ 
11:    else
12:       $cond \leftarrow col + ' in' + val$ 
13:    end if
14:     $sous\_arbre \leftarrow \{cond : []\}$ 
15:     $réponse_{gauche} \leftarrow DTR(gauche, depth + 1, \text{critère d'arrêt})$ 
16:     $réponse_{droite} \leftarrow DTR(droite, depth + 1, \text{critère d'arrêt})$ 
17:    if  $rponse_{gauche} == rponse_{droite}$  then
18:       $sous\_arbre \leftarrow rponse_{droite}$ 
19:    else
20:      ajouter  $réponse_{gauche}$  dans  $sous\_arbre[cond]$ 
21:      ajouter  $réponse_{droite}$  dans  $sous\_arbre[cond]$ 
22:    end if
23:  end if
24: end if
25: return  $node$  comme nœud de l'arbre de décision
```

Il faut noter que la version séquentielle traite les différentes branches gauches et droites de façons recursive. De plus, le nombre de branche n'est pas figé pour tous les niveaux du graphes.

2.2 Version itérative

Algorithm 2 Algorithme de l'arbre de décision itératif

Require: Ensemble d'entraînement $D \leftarrow \{(x_1, y_1), \dots, (x_n, y_n)\}$ et le critère d'arrêt

Require: *attributes* : Liste des attributs de l'ensemble de données

Require: *target_attribute* : Attribut cible (classe)

Ensure: *tree* : Arbre de décision résultant

```

1: Créer un noeud racine définit Root  $\{col, val, ig, cond, left, right\}$ 
2: Initialiser une pile  $T$  vide
3: Empiler  $\{depth = 0, node = Root, data = D\}$  dans  $T$ 
4: while il y a des noeuds non étiquetées dans  $T$  do
5:    $depth, node, data \leftarrow$  Depiller  $T$ 
6:   if  $depth, data$  satisfont le critère d'arrêt then
7:     Étiqueter  $node.cond$  avec l'étiquette la plus fréquente parmi les échantillons dans  $data$ .
8:   else
9:     Trouver le meilleur candidat diviseur de  $data \leftarrow \{col, val, ig, type\}$ 
10:    if  $ig$  satisfait le critère d'arrêt then
11:      Étiqueter  $node.cond$  avec l'étiquette la plus fréquente parmi les échantillons dans  $data$ .
12:    else
13:       $left, right \leftarrow$  Diviser  $data$  sur la base du meilleur candidat
14:      if Candidat numérique then
15:         $node.cond \leftarrow col + ' \leq ' + val$ 
16:      else
17:         $node.cond \leftarrow col + ' in ' + val$ 
18:      end if
19:      Mettre à jour  $\{col, val, ig\}$  par celui de la meilleure division
20:      Empiler  $\{depth = depth + 1, node = node.left, data = left\}$  dans  $T$ 
21:      Empiler  $\{depth = depth + 1, node = node.right, data = right\}$  dans  $T$ 
22:    end if
23:  end if
24: end while

```

3 Implementation python From scratch

3.1 Module de gestion de fichier et données

```
def process_file(file_path, details_separator, header_flag):  
    """  
    :param file_path: path to file  
    :param details_separator: which separator to use for details scission  
    :param header_flag: does file content headers' informations  
    :return: descriptions  
    """  
  
    # Read the text file  
    with open(file_path, 'r') as file:  
        lines = file.readlines()  
  
    # Determine the start index based on the presence of a header line  
    start_index = 1 if header_flag else 0  
  
    # Retrieve the header line if it exists  
    header = None  
    if header_flag:  
        header = [elt.strip() for elt in lines[0].split(details_separator)]  
  
    # Process each line and extract the descriptions  
    descriptions = {}  
    for line in lines[start_index:]:  
        if line:  
            fields = line.split(details_separator)  
  
            for i, field in enumerate(fields):  
                # Trim the field and convert to the appropriate type  
                field = field.strip()  
                if header:  
                    if sum([header[i] in key for key in descriptions.keys()]) == 0:  
                        descriptions[header[i]] = []  
                        field_name = header[i]  
                        descriptions[field_name].append(cast_to_appropriate_type(field))  
                    else:  
                        if sum([i in key for key in descriptions.keys()]) == 0:  
                            descriptions[i] = []  
                            descriptions[i].append(cast_to_appropriate_type(field))  
                descriptions['Index'] = list(range(len(descriptions[list(descriptions.keys())[0]])))  
    return descriptions
```

FIGURE 4 – Fonction from scratch pour charger les fichiers

Cette fonction lit un fichier et charge les données comme un dictionnaire de vecteurs qui contient les informations de chaque ligne du fichier associé à la dimension qui fait office de clef.

```
def cast_to_appropriate_type(data):  
    if data.isdigit():  
        return int(data)  
    try:  
        return float(data)  
    except ValueError:  
        if data.lower() == 'true':  
            return True  
        elif data.lower() == 'false':  
            return False  
        else:  
            return data
```

FIGURE 5 – Fonction from scratch pour convertir des types des différents informations

Cette fonction se charge d'assurer que les données du fichier sont caster vers les types appropriés.

```

def train_test_split(data, labels, test_size=0.2, random_state=None):
    :param labels: labels for train and test sets
    :param test_size: size of test set
    :param random_state: random state
    :return:
    """
    class_samples = defaultdict(list)
    for sample, label in zip(data['Index'], labels):
        class_samples[label].append(sample)
    # print(class_samples)
    test_data = {"Index": []}
    train_data = {"Index": []}
    y_test = []
    y_train = []

    if random_state is not None:
        random.seed(random_state)

    for classe, class_samples_list in class_samples.items():
        random.shuffle(class_samples_list)
        # print(class_samples_list, classe, len(class_samples_list))
        i = 0
        num_test_samples = int(test_size * len(class_samples_list))
        # print(num_test_samples)
        index_te = class_samples_list[:num_test_samples]
        index_tr = class_samples_list[num_test_samples:]
        # print(f"""
        # -----
        # {index_tr} @@@
        # {index_te} @@@
        # """)
        for ele in list(data.keys()):
            if sum([(ele in k) for k in list(train_data.keys())]) == 0:
                train_data[ele] = []
                test_data[ele] = []
            if "Index" not in ele:
                test_data[ele].extend([data[ele][i] for i in index_te])
                train_data[ele].extend([data[ele][i] for i in index_tr])

        y_test.extend([classe] * num_test_samples)
        y_train.extend([classe] * (len(class_samples_list) - num_test_samples))
        train_data["Index"].extend(index_tr)
        test_data["Index"].extend(index_te)
    return train_data, test_data, y_train, y_test

```

FIGURE 6 – fig :train-test

Lors du chargement, un champ d'indexage est ajouté pour faciliter la manipulation de lignes du jeu de données. Dans fonction, nous faisons usage de ce détail pour garantir une séparation conforme du jeu de données basé sur la variable cible.

3.2 Module de métriques

```
def compute_confusion_matrix(true_labels, predicted_labels, labels):  
    """  
    Computes the confusion matrix.  
    :param true_labels: true labels  
    :param predicted_labels: predicted labels  
    :param labels:  
    :return:  
    """  
  
    num_classes = len(labels)  
    confusion_matrix = [[0] * num_classes for _ in range(num_classes)]  
  
    for true, predicted in zip(true_labels, predicted_labels):  
        true_index = labels.index(true)  
        predicted_index = labels.index(predicted)  
        confusion_matrix[true_index][predicted_index] += 1  
  
    return confusion_matrix
```

FIGURE 7 – Fonction from scratch pour construire la matrice de confusion

```
def accuracy(confusion_matrix):  
    """  
    Computes the accuracy metric.  
    :param confusion_matrix: the confusion matrix  
    :return:  
    """  
  
    accuracy_v = None  
    if confusion_matrix is not None and (len(confusion_matrix) == 2 and len(confusion_matrix[0]) == 2):  
        accuracy_v = ((  
            confusion_matrix[0][0] + confusion_matrix[1][1]) /  
            (confusion_matrix[0][1] + confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matrix[1][0]))  
    return accuracy_v
```

FIGURE 8 – Fonction from scratch pour évaluer l'accuracy

```
def precision(confusion_matrix):  
    """  
    Computes the precision metric.  
    :param confusion_matrix: the confusion matrix  
    :return:  
    """  
  
    precision_v = None  
    if confusion_matrix is not None and (len(confusion_matrix) == 2 and len(confusion_matrix[0]) == 2):  
        precision_v = confusion_matrix[0][0] / (confusion_matrix[0][1] + confusion_matrix[0][0])  
    return precision_v
```

FIGURE 9 – Fonction from scratch pour évaluer la precision

```
def recall(confusion_matrix):
    """
    Computes the recall metric.
    :param confusion_matrix: the confusion matrix
    :return: recall_v - the recall value
    """
    recall_v = None
    if confusion_matrix is not None and (len(confusion_matrix) == 2 and len(confusion_matrix[0]) == 2):
        recall_v = confusion_matrix[0][0] / (confusion_matrix[0][0] + confusion_matrix[1][0])
    return recall_v
```

FIGURE 10 – Fonction from scratch pour évaluer le rappel

```
def f1_score(confusion_matrix):
    """
    Computes the f1 score metric.
    :param confusion_matrix:
    :return: f1 - the f1 score
    """
    precision_val = precision(confusion_matrix)
    recall_val = recall(confusion_matrix)
    f1 = 2 * ((precision_val * recall_val) / (precision_val + recall_val))
    return f1
```

FIGURE 11 – Fonction from scratch pour évaluer la f1-score

3.3 Module de gestion d'arbre de décision

```
def entropy(y):  
    """  
    Given a Python List, it calculates the Shannon Entropy  
    :param y: list of labels  
    :return: entrop  
    """  
  
    if isinstance(y, list):  
        # Initialize an empty dictionary to store the counts  
        count_dict = {}  
  
        # Iterate over each value in the list  
        for value in y:  
            # Check if the value is already present in the dictionary  
            if value in count_dict:  
                # If present, increment the count by 1  
                count_dict[value] += 1  
            else:  
                # If not present, add the value to the dictionary with a count of 1  
                count_dict[value] = 1  
  
        p = [val / len(y) for (key, val) in count_dict.items()]  
        entrop = 0.0  
        epsilon = 1e-9  
  
        for value in p:  
            value = max(value, epsilon) # Ensure value is not zero to avoid log(0) issue  
            entrop += -value * math.log2(value)  
  
        return entrop  
    else:  
        raise 'Object must be a Python list.'
```

FIGURE 12 – Fonction from scratch pour évaluer l'entropie

```
def gini_impurity(y):
    """
    Calculates the Gini impurity of a binary classification problem
    :param y: list of labels
    :return: gini
    """

    if isinstance(y, list):
        # Initialize an empty dictionary to store the counts
        count_dict = {}

        # Iterate over each value in the list
        for value in y:
            # Check if the value is already present in the dictionary
            if value in count_dict:
                # If present, increment the count by 1
                count_dict[value] += 1
            else:
                # If not present, add the value to the dictionary with a count of 1
                count_dict[value] = 1

        p = [val / len(y) for (key, val) in count_dict.items()]
        gini = 1 - sum([p1 ** 2 for p1 in p])
        return gini

    else:
        raise 'Object must be a Python list.'
```

FIGURE 13 – Fonction from scratch pour évaluer l'indice de gini

```
def information_gain(y, mask, func=entropy):
    """
    It returns the Information Gain of a variable given a loss function.
    y: target variable.
    mask: split choice.
    func: function to be used to calculate Information Gain in case of classification.
    """

    a = sum(mask)
    b = len(mask) - a

    if a == 0 or b == 0:
        ig = 0
    else:
        if isinstance(y[0], (int, float)):
            ig = variance(y) - (a / (a + b) * variance([y[i] for i in range(len(y)) if mask[i]])) - (
                b / (a + b) * variance([y[i] for i in range(len(y)) if not mask[i]]))
        else:
            ig = func(y) - a / (a + b) * func([y[i] for i in range(len(y)) if mask[i]]) - b / (a + b) * func(
                [y[i] for i in range(len(y)) if not mask[i]])

    return ig
```

FIGURE 14 – Fonction from scratch pour évaluer le gain


```
def variance(y):
    """
    Given a Python List, it calculates the Variance
    :param y: list of labels
    :return: variance
    """

    if len(y) == 1:
        return 0
    else:
        mean = sum(y) / len(y)
        variance = sum((xi - mean) ** 2 for xi in y) / (len(y) - 1)
        return variance
```

FIGURE 15 – Fonction from scratch pour évaluer la variance

4 Résultats expérimentaux (données, métriques d'évaluation, résultats)

4.1 Données

Pour évaluer le travail fournir, nous avons éprouvé sur le jeu de données *GERMAN*. Il s'agit d'un jeu de données est utilisé dans le cadre de mon projet de mémoire de Master deux et conserve des historiques bancaire anonymes.

```
Attribute_1,Attribute_2,Attribute_3,Attribute_4,Attribute_5,Attribute_6,Attribute
A11,6,A34,A43,1169,A65,A75,4,A93,A101,4,A121,67,A143,A152,2,A173,1,A192,A201,1
A12,48,A32,A43,5951,A61,A73,2,A92,A101,2,A121,22,A143,A152,1,A173,1,A191,A201,2
A14,12,A34,A46,2096,A61,A74,2,A93,A101,3,A121,49,A143,A152,1,A172,2,A191,A201,1
A11,42,A32,A42,7882,A61,A74,2,A93,A103,4,A122,45,A143,A153,1,A173,2,A191,A201,1
A11,24,A33,A40,4870,A61,A73,3,A93,A101,4,A124,53,A143,A153,2,A173,2,A191,A201,2
A14,36,A32,A46,9055,A65,A73,2,A93,A101,4,A124,35,A143,A153,1,A172,2,A192,A201,1
A14,24,A32,A42,2835,A63,A75,3,A93,A101,4,A122,53,A143,A152,1,A173,1,A191,A201,1
A12,36,A32,A41,6948,A61,A73,2,A93,A101,2,A123,35,A143,A151,1,A174,1,A192,A201,1
A14,12,A32,A43,3059,A64,A74,2,A91,A101,4,A121,61,A143,A152,1,A172,1,A191,A201,1
A12,30,A34,A40,5234,A61,A71,4,A94,A101,2,A123,28,A143,A152,2,A174,1,A191,A201,2
A12,12,A32,A40,1295,A61,A72,3,A92,A101,1,A123,25,A143,A151,1,A173,1,A191,A201,2
A11,48,A32,A49,4308,A61,A72,3,A92,A101,4,A122,24,A143,A151,1,A173,1,A191,A201,2
A12,12,A32,A43,1567,A61,A73,1,A92,A101,1,A123,22,A143,A152,1,A173,1,A192,A201,1
A11,24,A34,A40,1199,A61,A75,4,A93,A101,4,A123,60,A143,A152,2,A172,1,A191,A201,2
A11,15,A32,A40,1403,A61,A73,2,A92,A101,4,A123,28,A143,A151,1,A173,1,A191,A201,1
A11,24,A32,A43,1282,A62,A73,4,A92,A101,2,A123,32,A143,A152,1,A172,1,A191,A201,2
A14,24,A34,A43,2424,A65,A75,4,A93,A101,4,A122,53,A143,A152,2,A173,1,A191,A201,1
A11,30,A30,A49,8072,A65,A72,2,A93,A101,3,A123,25,A141,A152,3,A173,1,A191,A201,1
```

FIGURE 16 – extrait du jeu de données

4.2 Comparaisons des résultats

```
Training labels:
{1: 560, 2: 240}
Test labels:
{1: 140, 2: 60}
[[112, 28], [40, 20]]

Accuracy: 0.56,
precision: 0.8,
Recall: 0.7368421052631579,
F1-score: 0.7671232876712328
```

FIGURE 17 – Métriques du modèle construit from scratch

	precision	accuracy	recall	f1-score
LDA	0.62745098	0.75	0.68462519	0.69328917
LR	0.66666667	0.765	0.69557409	0.70697341
SVM	0.67346939	0.77	0.7035106	0.71478175
DT	0.42622951	0.64	0.57861198	0.57924264
RF	0.69230769	0.735	0.61366006	0.61703819
XGB	0.47826087	0.675	0.58701193	0.59015101

FIGURE 18 – Métriques du modèle construit avec sklearn dans mon environnement expérimentale

5 Conclusion

Parvenu au terme de ce rapport, il était question pour nous d'étudier un algorithme de notre sujet de recherche, bien la comprendre et proposer implanter une version sequence existante sous python sans usage de modules. Il a été vraiment éprouvant pour nous de ne pas tomber une version données sequence des algorithmes d'arbre de décision suite à nos recherches. Il a donc été important pour nous de proposer une version séquentielle aux algorithmes de croissances des arbres de décision et puis proposer une imple-mentation par la suite. Chose qui a été fait et éprouvé sur un jeu de données que nous manipulons dans

notre mémoire. Si comme nous recherche nous l'ont monté, l'enseignant approuve l'absence d'un dit algorithme, il sera important pour nous de confronter l'effectivité de notre algorithmes aux version recursive existante afin de pouvoir proposer un article.

Nous avons appris bien des choses en faisant cette exercice de code de zéro ; notamment le fonctionnement de nombreux librairies python, qui nous donne l'idée de transcrire ce savoir dans bien des langages tels que le c et c++ qui ont l'avantage de proposer une communication directe avec l'ordinateur et plus rapide de sur quoi. Nous continuerons donc par la suite en reproduisant l'environnement sous c ou c++ et parallélisant.