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

La classification est un tache qui consiste a associer un object à une catégorie.

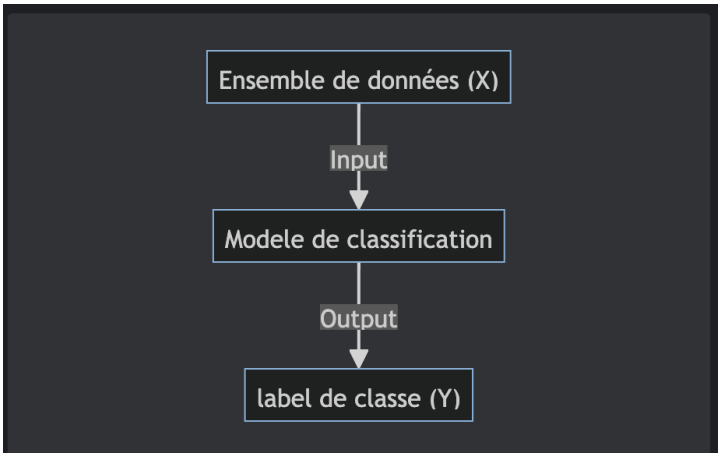

FIGURE 1 – Schéma d'une Classification

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 un catégorie a une entrée.

Explicitement, un arbre classique comporte une racine, des branches et des feuilles. Les arbres de décision en comporte de même. Les conditions qui permettent de séparer au mieux l'ensemble de données est contenu dans les noeuds, 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 noeud racine est le parent de tous les noeuds, c'est le noeud le plus haut dans l'arbre. L'arbre de décision permet donc de représenter chaque attribut dans les noeuds ; 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'un reproduction du processus de décision de la vie réelle. Si nous voulons par exemple représenter un processus de décision dépendanment des conditions météorologiques, nous pouvons construire l'arbre suivant

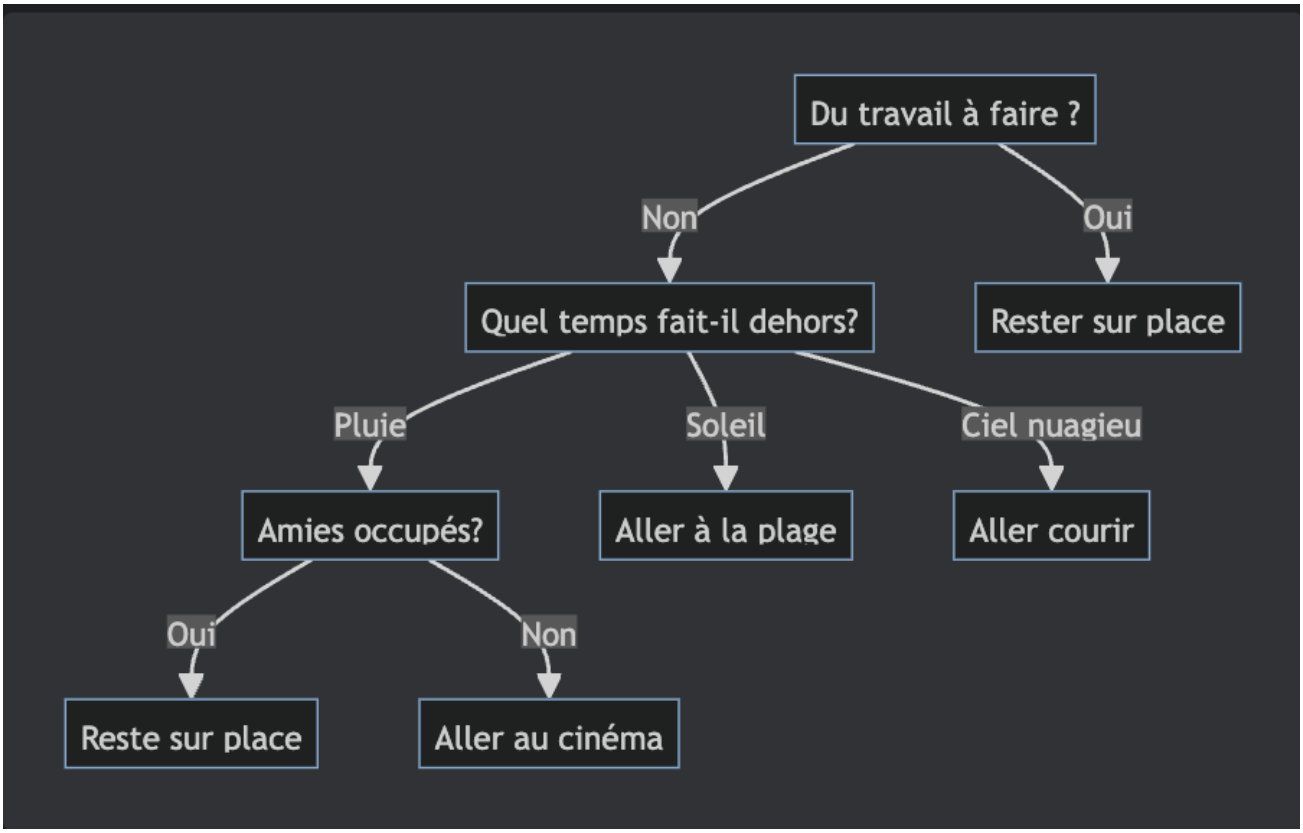

FIGURE 2 – Exemple de processus de décision

Nous remarquons cette situation de décision dans la vie réelle peut bien se résumé 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 si contre vous détails 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'induc-tion 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 (Chisquare-Automatic-Interaction-Detection) a été conçu à l'origine pour traiter uniquement des attributs nominaux. Pour chaque attribut d'entrée ai, CHAID recherche la paire de valeurs dans Vith 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 decription des arbres de décision

1.2 Fonctionnement d'ID3

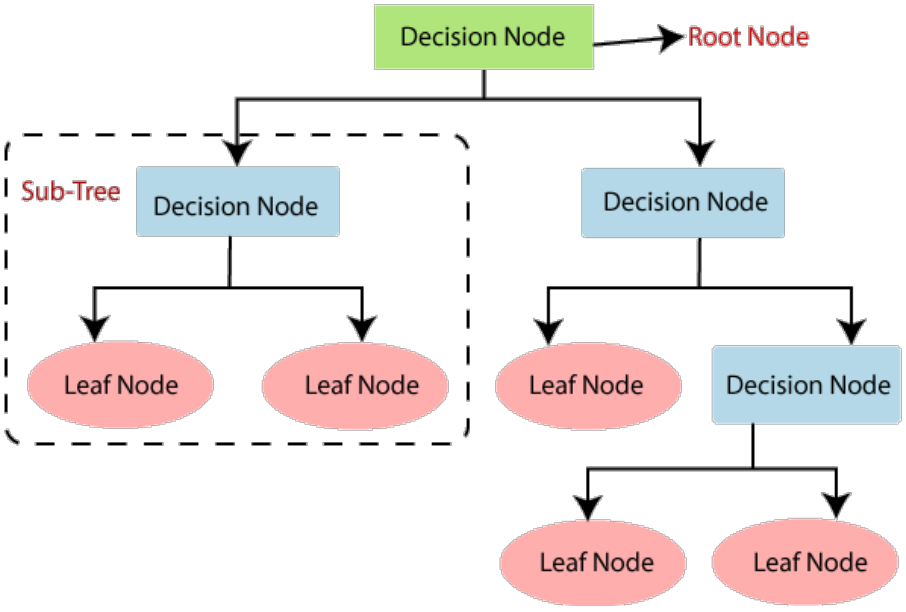

FIGURE 3 – Squelette d'un arbre de décision

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 intéressons 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 :

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

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

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 $D \leftarrow \{(x_1, y_1), \ldots, (x_n, y_n)\}$, *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

if $D, depth$ satisfont le critère d'arrêt **then**
  return un noeud avec comme etiquette la classe majoritaire du jeu
else
  Trouver le meilleur candidat diviseur de $D \leftarrow \{col, val, ig, type\}$
  if ig satisfait le critère d'arrêt **then**
    return un noeud avec comme etiquette la classe majoritaire du jeu
  else
    $gauche, droite \leftarrow$ Diviser D sur la base du meilleur candidat
    if Candidat numérique **then**
      $cond \leftarrow col +' <=' + val$
    else
      $cond \leftarrow col +' in' + val$
    end if
    $sous\_arbre \leftarrow \{cond : []\}$
    réponse$_{gauche} \leftarrow DTR(gauche,\ depth + 1,$ critère d'arrêt$)$
    réponse$_{droite} \leftarrow DTR(droite,\ depth + 1,$ critère d'arrêt$)$
    if $rponse_{gauche} == rponse_{droite}$ **then**
      $sous\_arbre \leftarrow rponse_{droite}$
    else
      ajouter réponse$_{gauche}$ dans $sous\_arbre[cond]$
      ajouter réponse$_{droite}$ dans $sous\_arbre[cond]$
    end if
  end if
end if
return *node* comme nœud de l'arbre de décision

Il faut noter que la version séquentielle traites 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 $D \leftarrow \{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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

Créer un noeud racine définit Root $\{col, val, ig, cond, left, right\}$
Initialiser une pile T vide
Empiler $\{depth = 0, node = Root, data = D\}$ dans T
while il y a des noeuds non étiquetées dans T **do**
  $depth, node, data \leftarrow$Depiller T
  if $depth, data$ satisfont le critère d'arrêt **then**
    Étiqueter *node.cond* avec l'étiquette la plus fréquente parmi les échantillons dans *data*.
  else
    Trouver le meilleur candidat diviseur de $data \leftarrow \{col, val, ig, type\}$
    if ig satisfait le critère d'arrêt **then**
      Étiqueter *node.cond* avec l'étiquette la plus fréquente parmi les échantillons dans *data*.
    else
      $left, right \leftarrow$ Diviser *data* sur la base du meilleur candidat
      if Candidat numérique **then**
        $node.cond \leftarrow col +' <=' + val$
      else
        $node.cond \leftarrow col +' in' + val$
      end if
      Mettre a jour $\{col, val, ig\}$ par celui de la meilleur division
      Empiler $\{depth = depth + 1, node = node.left, data = left\}$ dans T
      Empiler $\{depth = depth + 1, node = node.right, data = right\}$ dans T
    end if
  end if
end while

3 Implementation python From scratch

3.1 Module de gestion de fichier et données

```python
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.

```python
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.

```python
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 un separation conforme du jeu de données basé sur la variable cible.

3.2 Module de métriques

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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 Implémentation cpp et parallélisation

L'implémentation cpp de notre algorithme a été juste une traduction ligne par ligne de la version python from scratch. Suite à ce processus, nombreux idées de parallélisation ont été pensé.

4.1 Parallélisation du calcul du mask

Lors de la croissance de l'arbre de décision, il existe une étapes qu'on appel maskage. Sachant un attribut A et une valeur v de celui, elle vise a retourner un vecteur de binaire où une cellule i possède 1 si la ligne i possède une valeur inférieur v ($Ligne[i, A] < v$).

Bien que l'action n"étant pas complexe, il pourrait être intéressant qu'il soit effectué en parallèle surtout pour des grand jeu de données.

4.2 Parallélisation du calcul du l'information de gain

Après avoir identifier le masque associé à la valeur v de l'attribut A, l'impureté doit être évaluer sur les deux ensembles formés par les lignes ayant 0 et 1. La parallélisation du procédé pourrait apporté un gain en temps.

4.3 Parallélisation du calcul du meilleur candidat séparateur

A chaque niveau durant la croissance d'un arbre de décision, la question mère est de savoir quel est l'attribut qui permet de séparer au mieux l'ensemble de données. Pour cela, nous devoir déjà parcourir toutes les valeurs possibles d'une variables afin d'identifier celle qui permet de faire cette séparation et faire ceci

pour tous les attributs. Intuitivement, l'affectation de ses taches à des threads en parallèle doit améliorer le temps de croissance, tout en conservant les métriques.

4.4 Parallélisation du processus complet de croissance de l'arbre

Une autre idée serait de pouvoir à des moments, lancer plusieurs calcul parallèle de la croissance de l'arbre. Ceci revient a faire la recherche du meilleur candidat séparateurs, séparer les données et créer les noeuds en parallèle. Pour cela, il faut controller à chaque itération le nombre de noeud dans la pille et si supérieur à 1. lancer des threads parallèles pour les traiter indépendamment.

4.5 Parallélisation de deux options précédentes simultannement

Une autre approche serai d'hybrider les intuitions 1 et 2, puis 3 et 4.

4.6 Quelques captures de codes parallèles cpp

```cpp
// Fonction de thread pour calculer une partie du masque
void process_thread(SharedData0& data, int thread_id) {
    for (size_t i = thread_id; i < data.X.size(); i += data.completed_threads) {
        // data.mutex.lock();
        (*data.mask)[i] = data.X[i] < data.val;
        // data.mutex.unlock();
        // std::cout << thread_id << "("<<data.X[i] << "," << data.val << ") ...> " << (*data.mask)[i] << " i: " << std::to_string(i) << std::endl;
    }
}

// Fonction parallèle pour obtenir le masque
std::vector<bool> getMaskParallel(const std::vector<float>& X, const float& val) {
    std::vector<bool> mask(X.size()); // Pré-allouer le vecteur de masque
    SharedData0 data { X, val, &mask ,std::mutex(), std::condition_variable(), num_threads_ };

    // Créer les threads
    std::vector<std::thread> threads;
    for (int i = 0; i < num_threads_; ++i) {
        threads.emplace_back(process_thread, std::ref(data), i);
    }
    for (auto& thread : threads) {
        thread.join();
    }

    return mask;
}
```

FIGURE 16 – Masquage parallèle

```cpp
// Fonction de thread pour compter les 'true' dans le masque
void process_thread_count(SharedData& data, int thread_id) {
    size_t l = 0; // Variable locale pour compter les "true"
    for (size_t i = thread_id; i < data.mask.size(); i += data.completed_threads) {
        if (data.mask[i]) {
            ++l;
            // std::cout << "<--->" << std::endl;
        }
    }
    std::atomic_size_t& a = data.a;
    a += l;
}


// Fonction parallèle pour compter les 'true' dans le masque
size_t count_true_parallel(const std::vector<bool>& mask, int num_threads) {
    SharedData data { mask, 0,std::mutex(), std::condition_variable(), num_threads_ };

    // Créer les threads
    std::vector<std::thread> threads;
    for (int i = 0; i < num_threads_; ++i) {
        threads.emplace_back(process_thread_count, std::ref(data), i);
    }
    for (auto& thread : threads) {
        thread.join();
    }
    // std::cout << data.a << s td::endl;
    return data.a;
}
```

FIGURE 17 – Comptage Parallèle

```cpp
// Fonction de thread pour séparer les données
void process_thread_separate(SharedData2& data, int thread_id) {
    std::vector<bool> d1, d2;
    d1.reserve(data.y.size() / data.completed_threads); // Allouer de la mémoire pour d1
    d2.reserve(data.y.size() / data.completed_threads); // Allouer de la mémoire pour d2
    for (size_t i = thread_id; i < data.y.size(); i += data.completed_threads) {
        if (data.mask[i]) {
            data.mutex.lock();
            (*data.pos_data).push_back(data.y[i]);
            data.mutex.unlock();
        } else {
            data.mutex.lock();
            (*data.neg_data).push_back(data.y[i]);
            data.mutex.unlock();
        }
    }
    // data.mutex.lock();
    // (*data.pos_data).insert((*data.pos_data).end(), std::make_move_iterator(d1.begin()), std::make_move_iterator(d1.end()));
    // (*data.neg_data).insert((*data.neg_data).end(), std::make_move_iterator(d2.begin()), std::make_move_iterator(d2.end()));
    // data.mutex.unlock();
}

// Fonction parallèle pour séparer les données
void separate_data_parallel(const std::vector<float>& y, const std::vector<bool>& mask, std::vector<float>& pos_data, std::vector<float>& neg_data) {
    SharedData2 data { y, mask, &pos_data, &neg_data,std::mutex(), std::condition_variable(), num_threads_ };

    // Créer les threads
    std::vector<std::thread> threads;
    for (int i = 0; i < num_threads_; ++i) {
        threads.emplace_back(process_thread_separate, std::ref(data), i);
    }
    for (auto& thread : threads) {
        thread.join();
    }

}

// Fonction parallèle pour calculer le gain d'information
float information_gain_parallel(const std::vector<float>& y,
                                const std::vector<bool>& mask,
                                std::function<float(const std::vector<float>&)> func) {
    size_t a = count_true_parallel(mask, num_threads_);
    size_t b = mask.size() - a;

    float ig = 0.0;
    if (a == 0 || b == 0) {
        ig = 0.0;
    } else {
        std::vector<float> pos_data, neg_data;
        separate_data_parallel(y, mask, pos_data, neg_data);

        ig = func(y) - (static_cast<float>(a) / (a + b) * func(pos_data)) -
             (static_cast<float>(b) / (a + b) * func(neg_data));
    }

    return ig;
}
```

FIGURE 18 – calcul de gain en parallèle

```
TreeNode* interativeTrainTree_Parallel(
        std::unordered_map<std::string, std::vector<float>> data,
        std::string y,
        bool target_factor,
        int max_depth,
        int min_samples_split,
        float min_information_gain,
        int counter,
        int max_categories) {

    std::vector<NodeData> stack;
    TreeNode* root = new TreeNode;
    init_tree(data, stack, root);
    clock_t start, end;
    double duration;
    while (!stack.empty()) {
       stack_trace = std::move(stack);
       int nb_threads = stack_trace.size() < num_threads_ ? stack_trace.size() : num_threads_;
       avalaible_ = num_threads_ - nb_threads;
       stack.resize(0);
       // std::cout << nb_threads << " inse " << stack_trace.size() << std::endl;
       GrowthShareData GrowthShareData_ {stack,
                    nb_threads,
                    y,
                    target_factor,
                    max_depth,
                    min_samples_split,
                    min_information_gain,
                    counter,
                    max_categories};

        if (nb_threads > 1){
           // std::cout << "inse" << std::endl;
            usedThreads = nb_threads;
            std::vector<std::thread> threads;
            for (int i = 0; i < nb_threads; i++){
                threads.emplace_back(
                    DecisionTreeGrowth,
                    std::ref(GrowthShareData_),
                    i
                    );
            }
            for (auto& thread : threads) {
                thread.join();
            }
        }else{
            usedThreads = 0;
            DecisionTreeGrowth(std::ref(GrowthShareData_),0);
        }



    }
//    std::cout << "end"<< std::endl;
    return std::move(root);
}
```

FIGURE 19 – Gestion de la croissance parallèle

```
void DecisionTreeGrowth(
    GrowthShareData& GrowthShareData_,
    int thread_id
    ){
    for (size_t iter = thread_id; iter < stack_trace.size(); iter+=GrowthShareData_.nb_threads){
        NodeData current = stack_trace[iter];
        std::unordered_map<std::string, std::vector<float>> xy_current = current.data;
        int depthg = current.depth;
        TreeNode* current_node = std::move(current.node);
        std::pair<bool, bool> conditions = checkConditionsDTree(depthg, xy_current, GrowthShareData_.max_categories, GrowthShareData_.max_depth, 0, GrowthShareData_.min_samples_split);
        if (conditions.first && conditions.second) {
            auto [var, val, ig, is_num] = get_best_split(GrowthShareData_.y, xy_current);

            if (ig != std::numeric_limits<float>::infinity() && ig >= GrowthShareData_.min_information_gain) {
                auto [left, right] = make_split(var, val, xy_current, is_num);

                std::string split_type = is_num ? "<=" : "in";
                std::string question;

                question = var + " " + split_type + " " + std::to_string(val);

                sub_tree(current_node, var, ig, question, depthg, val, GrowthShareData_.stack, left, right);
            } else {
                leaf_tree(xy_current, GrowthShareData_.y, GrowthShareData_.target_factor, var, ig, val, current_node);
            }
        } else {
            leaf_tree(xy_current, GrowthShareData_.y, GrowthShareData_.target_factor, std::string(), std::numeric_limits<float>::infinity(), float(), current_node);
        }
        xy_current.clear();
        current.data.clear();
    }
}
```

FIGURE 20 – processus de croissance parallèle

5 Expérimentation

5.1 Données et Ressources

Pour évaluer le travail fournir, nous avons éprouvé sur les jeux de données $GERMAN$ et $CREDITRISK$. Il s'agit de jeux de données est utilisés dans le cadre de mon projet de mémoire de Master II et conserve des historiques bancaire anonymes. Pour ce travaillons sur la version déjà prétraitée avec uniquement des attributs numériques pour avoir une bonne base de comparaisons avec la version python from scratch et avec module.

TABLE 3 – Description des données

	lignes de trainning	lignes de test	nombre de colonnes	nombre d'exemples positif	nombre d'exemples négatif
CREDIT RISK	26064	6517	20	25473	7108
GERMAN	800	200	62	700	300

TABLE 4 – Description des ressources

	Description
nombre de coeur	16
Fréquence d'exécution	2.4GHz
RAM	32
OS	MAC OS Sonama
Version cpp	20
Version python	3.9
outil de supervision	htop

5.2 Métriques

Pour évaluer la qualité de notre parallélisation, nous allons déjà évaluer le gain en temps : $SpeedUp \leftarrow \frac{T_{seq}}{T_{par}}$

Et pour la qualité de généralisation : $Accuracy \leftarrow \frac{TN+TN}{TP+FP+TN+FN}$; $F1 - score \leftarrow \frac{2 \times Precision \times Recall}{Precision \times Recall}$

5.3 Résultats

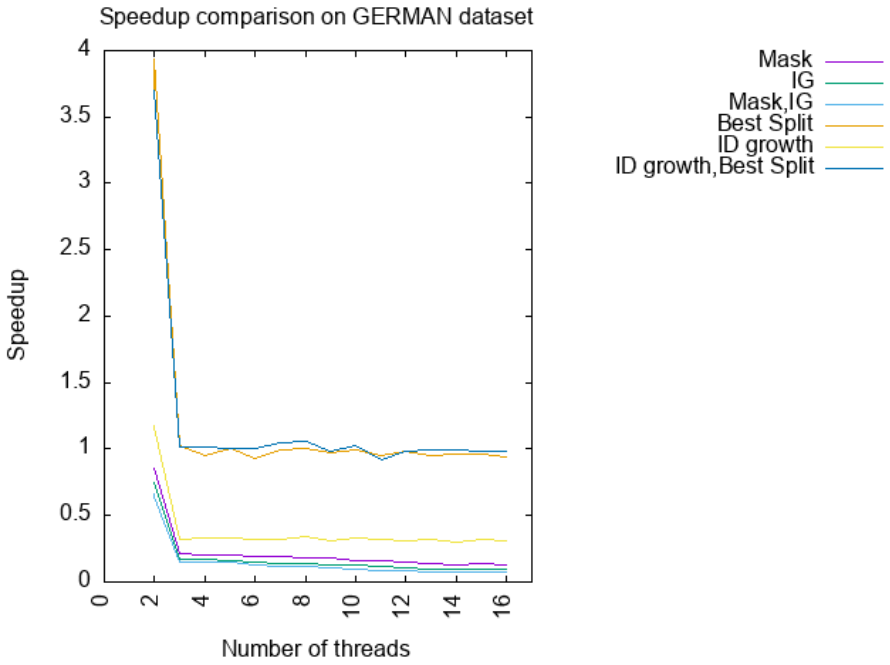

FIGURE 21 – Speedup sur CREDIT RISK

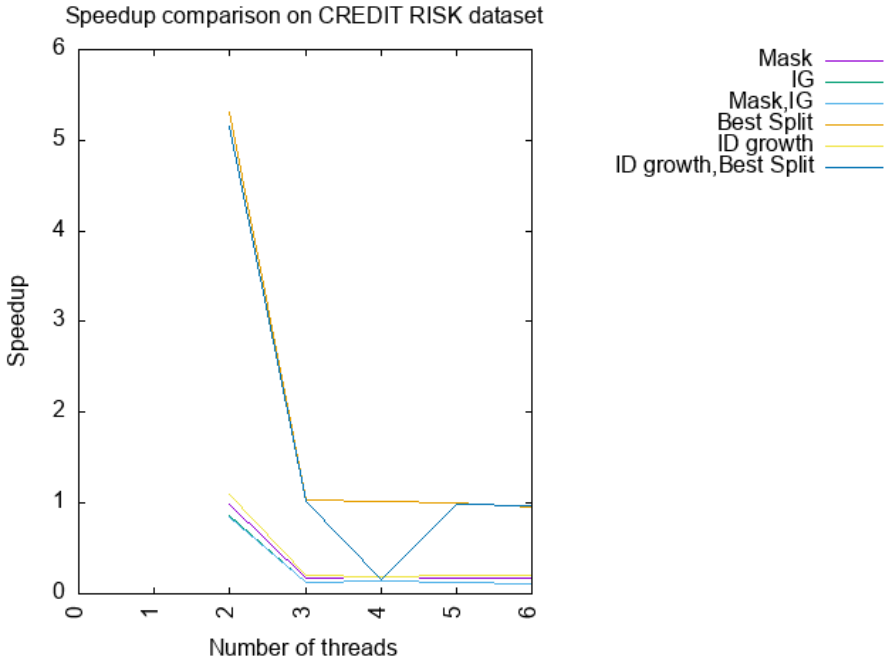

FIGURE 22 – Speedup sur GERMAN

Table 5 – German DT

	Temps (ms)	Accuracy	F1-score	Profondeur maximal
Python sklearn	388	0.68	0.76296	indéfinie
Python Sequentiel	1158	0.64	0.75	indéfinie
cpp Sequentiel	732	0.655	0.76125	indéfinie
cpp Parallèl Best Split	186	0.655	0.76125	indéfinie

5.4 Discusion

L'échec dans les idées de la parallélisation du masquage, du calcul du gain, la croissances en parallèle sont explicable. Le nombre considérable de création de threads pour les deux premiers est la cause de leur échec. En effet, le coût de création est proportionnel au nombre de création. Dans notre cas, sachant n threads, m variables à k valeurs chacune, nous aurons $n \times m \times k$ pour le traitement d'un seul noeud.

Aussi, en ce qui concerne, l'échec de la croissance. la charge des noeuds, n'étant pas identique, l'attente devient un goulot étrangleur à la stratégie.

Nous ramerons aussi de part les résultats, que nos meilleurs performances, sont pour chaque jeu de données obtenu lorsque nous faisons usage de **2 threads**. ceci n'est qu'une conséquence des explications précédente. La croissance de l'arbre de décision n'est pas vraiment un processus qui ce prête à la parallélisation, alors la forte création de threads n'est pas vraiment avantageux pour le temps du processus, vu l'attente que ça peut provoquer.

6 Conclusion

Parvenu au terme de ce travail, il était question pour nous paralléliser un algorithme de notre travail de recherche. Nous avons porté notre intérêt sur l'algorithme Arbre de décision. Cet dernière est reconnu pour algorithme qui représente fidèlement le processus de décision humain mais aussi pour ne pas être évident à paralléliser. Nous avons donc durant de travail, proposé un version itérative de l'algorithme pour la rendre plus accessible à un processus de parallélisation, ensuite nous avons implémenté une version python sans utiliser de module, une version c++ fidèle à la version python. Nous avons aussi, fait usage de la version sklearn pour comparer nos résultats à des standards. Suite à quoi, nous avons éprouvé 5 idées de parallélisation dans une environnement **UMA**. Nous avons constaté d'une seule stratégie était vraiment viable et que notre processus obtient ses meilleurs résultats lorsque nous utilisons uniquement deux threads pour nous offrir un speedup de 4 et 5 pour les jeu de données GERMAN et CREDIT RISK respectivement.