



Lannion

SOUTENANCE SAÉ 2.02

- EXPLORATION ALGORITHMIQUE D'UN PROBLÈME

Présenté par

Arthur CHAUVEL, Marceau LESECH, Yannis LECHEVERE, Timéo LEMOING



INTRODUCTION

Contexte :

- Projet réalisé dans le cadre de la SAÉ 2.02.
- Étude de deux algorithmes d'apprentissage automatique : KNN (classification supervisée) et KMeans (clustering non supervisé).
- Analyse de leur fonctionnement, de leurs cas d'usage et de leurs performances.

Objectif :

- Comprendre les principes et mécanismes de ces algorithmes.
- Explorer leurs applications pratiques et leurs limites.
- Implémenter et tester ces méthodes sur des jeux de données réels.
- Analyser les résultats et surmonter les difficultés rencontrées.

PRÉSENTATION DES ALGORITHMES

AlgoDélimiteur

Segmente un texte selon des délimiteurs. Simple, rapide et adapté aux grands volumes.

KNN (K NEAREST NEIGHBORS)

Classe et prédit en fonction des K plus proches voisins. Simple et efficace sur petits jeux de données.

KMeans

Regroupe les données en K clusters selon leur similarité. Rapide et utile pour l'exploration.

FONCTIONNEMENT DE L'ALGO DÉLIMITEUR

1

Chargement des données

Le texte à segmenter est lu en entrée.

2

Identification des délimiteurs

Consultation d'une table de délimiteurs connus servant à diviser le texte en unités distinctes.

3

Segmentation du texte

Le texte est découpé en fonction des délimiteurs détectés.

4

Sortie des segments

Sort une liste contenant les segments du texte extraits

STRUCTURE DES DONNÉES DE L'ALGO

DÉLIMITEUR

Paramètre de l'algo :

L'algo délimiteur traite principalement des données textuelles sous forme de chaînes de caractères mais aussi des données semi-structurées (csv, JSON, ...)

Organisation :

- chaîne de caractère (entrée)
- listes (stocker segment)
- ensemble ou liste (pour les délimiteurs)

Utilisation :

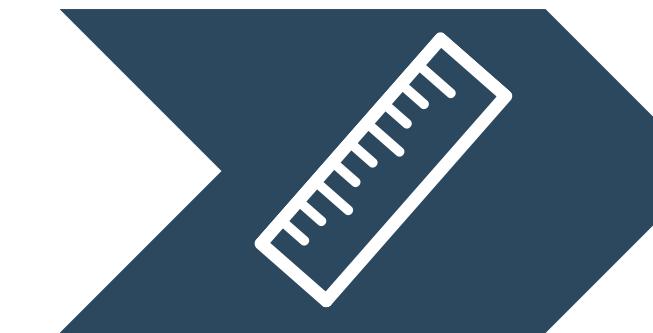
Traitement de texte, nettoyage et structuration de la chaîne de caractère passé en paramètre

```
1 from collections import deque
2
3 def testCorrespondanceDelimiteurs(S):
4     L = []
5     T = deque()
6     c = 1
7     n = len(S)
8
9     delimitateursGauche = "([{"
10    delimitateursDroit = "])}"
11    correspondance = {')': '(', ']': '[', '}': '{'}
12
13    for i in range(n):
14        char = S[i]
15
16        if char in delimitateursGauche:
17            L.append(c)
18            T.append((char, c))
19            c += 1
20
21        elif char in delimitateursDroit:
22            if not T:
23                return ""
24
25            gauche, d = T.pop()
26            if correspondance[char] == gauche:
27                L.append(d)
28            else:
29                return ""
30
31        if not T:
32            return L
33
34    return ""
```

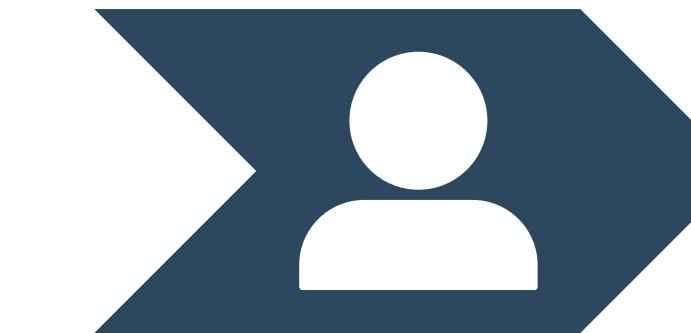
FONCTIONNEMENT DE KNN



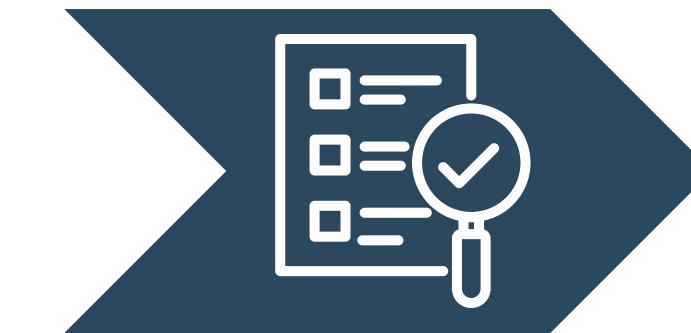
Chargement et conversion des données



Choix et calcul des distances



Sélection des voisins et classification



Affichage des résultats

```
10  def load_data_set(filename):
11      try:
12          with open(filename, newline='') as iris:
13              return list(reader(iris, delimiter=','))
14      except FileNotFoundError as e:
15          raise e
16
17
18  def convert_to_float(data_set, mode):
19      new_set = []
20      try:
21          if mode == 'training':
22              for data in data_set:
23                  new_set.append([float(x) for x in data[:len(data)-1]] + [data[len(data)-1]])
24
25          elif mode == 'test':
26              for data in data_set:
27                  new_set.append([float(x) for x in data])
28
29      else:
30          print('Invalid mode, program will exit.')
31          exit()
32
33      return new_set
34
35  except ValueError as v:
36      print(v)
37      print('Invalid data set format, program will exit.')
38      exit()
```

```
60  def calculer_distance(point1, point2, type_distance):  
61      """  
62          Calcule la distance entre deux points en utilisant la métrique spécifiée  
63      """  
64      # Convertir les listes en tuples pour compatibilité avec les fonctions de distance  
65      point1_tuple = tuple(point1)  
66      point2_tuple = tuple(point2)  
67  
68      if type_distance == 'euclidienne':  
69          return calculEuclidianDistance(point1_tuple, point2_tuple)  
70  
71      elif type_distance == 'manhattan':  
72          return calculManhattanDistance(point1_tuple, point2_tuple)  
73  
74      elif type_distance == 'chebyshev':  
75          return calculChebyshevDistance(point1_tuple, point2_tuple)  
76  
77      else:  
78          print("Type de distance invalide, utilisation par défaut de la distance Euclidienne")  
79          return calculEuclidianDistance(point1, point2)  
80
```

```
40
41 def obtener_classes(ensemble_entrainement):
42     return list(set([c[-1] for c in ensemble_entrainement]))
43
44
45 def trouver_voisins(distances, k):
46     return distances[0:k]
47
48
49 def determiner_reponse(voisins, classes):
50     votes = [0] * len(classes)
51
52     for instance in voisins:
53         for ctr, c in enumerate(classes):
54             if instance[-2] == c:
55                 votes[ctr] += 1
56
57     return max(enumerate(votes), key=itemgetter(1))
58
```

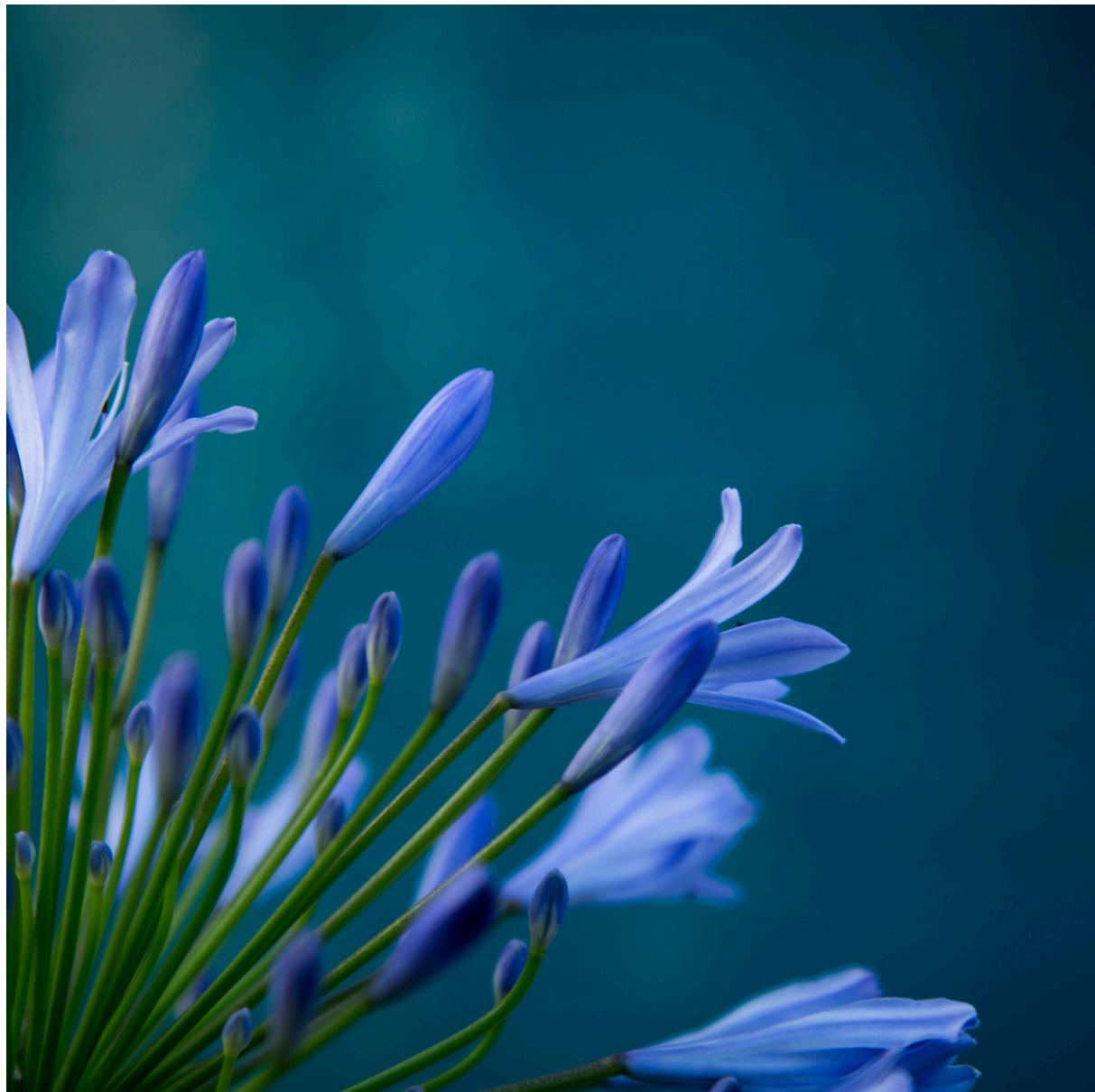
```
82     def knn(ensemble_entrainement, ensemble_test, k, type_distance):
83         distances = []
84         limite = len(ensemble_entrainement[0]) - 1
85
86         # Générer les classes à partir des données d'entraînement
87         classes = obtenir_classes(ensemble_entrainement)
88
89         try:
90             for instance_test in ensemble_test:
91                 for ligne in ensemble_entrainement:
92                     # Calculer la distance en utilisant la métrique sélectionnée
93                     dist = calculer_distance(ligne[:limite], instance_test, type_distance)
94                     distances.append(ligne + [dist])
95
96             distances.sort(key=itemgetter(len(distances[0])-1))
97
98             # Trouver les k plus proches voisins
99             voisins = trouver_voisins(distances, k)
100
101            # Déterminer la classe avec le maximum de votes
102            index, valeur = determiner_reponse(voisins, classes)
103
104            # Afficher la prédiction
105            print("La classe prédite pour l'échantillon " + str(instance_test) + " est : " + classe
106            print("Nombre de votes : " + str(valeur) + " sur " + str(k))
107            print("Métrique de distance utilisée : " + type_distance.capitalize())
108            print('-' * 50)
109
110            # Vider la liste des distances
111            distances.clear()
112
113        except Exception as e:
114            print(e)
```

```
82     def knn(ensemble_entrainement, ensemble_test, k, type_distance):
83         distances = []
84         limite = len(ensemble_entrainement[0]) - 1
85
86         # Générer les classes à partir des données d'entraînement
87         classes = obtenir_classes(ensemble_entrainement)
88
89         try:
90             for instance_test in ensemble_test:
91                 for ligne in ensemble_entrainement:
92                     # Calculer la distance en utilisant la métrique sélectionnée
93                     dist = calculer_distance(ligne[:limite], instance_test, type_distance)
94                     distances.append(ligne + [dist])
95
96             distances.sort(key=itemgetter(len(distances[0])-1))
97
98             # Trouver les k plus proches voisins
99             voisins = trouver_voisins(distances, k)
100
101            # Déterminer la classe avec le maximum de votes
102            index, valeur = determiner_reponse(voisins, classes)
103
104            # Afficher la prédiction
105            print("La classe prédite pour l'échantillon " + str(instance_test) + " est : " + classe
106            print("Nombre de votes : " + str(valeur) + " sur " + str(k))
107            print("Métrique de distance utilisée : " + type_distance.capitalize())
108            print('-' * 50)
109
110            # Vider la liste des distances
111            distances.clear()
112
113        except Exception as e:
114            print(e)
```

```
117 def principal():
118     try:
119         # Obtenir la valeur de k
120         k = int(input("Entrez la valeur de k : "))
121
122         # Demander à l'utilisateur de choisir une métrique de distance
123         print("\nMétriques de distance disponibles:")
124         print("1. Distance Euclidienne ( cercle )")
125         print("2. Distance Manhattan ( losange )")
126         print("3. Distance Chebyshev ( carré )")
127
128         choix = int(input("\nChoisissez une métrique de distance (1-3) : "))
129
130         if choix == 1:
131             type_distance = 'euclidienne'
132         elif choix == 2:
133             type_distance = 'manhattan'
134         elif choix == 3:
135             type_distance = 'chebyshev'
136         else:
137             print("Choix invalide, utilisation par défaut de la distance Euclidienne")
138             type_distance = 'euclidienne'
139
140         # Charger les ensembles de données d'entraînement et de test
141         fichier_entrainement = input("\nEnterz le nom du fichier de données d'entraînement : ")
142         fichier_test = input("\nEnterz le nom du fichier de données de test : ")
143         ensemble_entrainement = convertir_en_flottant(charger_ensemble_donnees(fichier_entrainement))
144         ensemble_test = convertir_en_flottant(charger_ensemble_donnees(fichier_test), 'test')
145
146         if not ensemble_entrainement:
147             print("Ensemble d'entraînement vide")
148
```

```
149 elif not ensemble_test:  
150     print("Ensemble de test vide")  
151  
152 elif k > len(ensemble_entrainement):  
153     print("Le nombre attendu de voisins est supérieur au nombre d'échantillons d'entraînement")  
154  
155 else:  
156     print(f"\nExécution de KNN avec la métrique {type_distance.capitalize()}...\n")  
157     knn(ensemble_entrainement, ensemble_test, k, type_distance)  
158
```

STRUCTURE DES DONNÉES POUR KNN



Type de Données :

Données numériques continues ou catégorielles (ex. mesures de fleurs, images, etc.).

Organisation :

Tableau où chaque ligne est un échantillon et chaque colonne une caractéristique.

Utilisation :

Recherche des K voisins les plus proches pour classifier ou prédire une valeur.

FONCTIONNEMENT DE KMEANS

Chargement des données :

Lecture du fichier CSV et extraction des colonnes utiles (revenu annuel et score de dépense des clients).



Initialisation des centroïdes :

Choix intelligent de k points initiaux avec la méthode K-Means++.

Assignment des clusters

Affectation de chaque point au cluster dont le centroïde est le plus proche (distance euclidienne).

```
83
84 def charger_donnees(chemin_fichier: str) -> np.ndarray:
85     """Charger le jeu de données à partir d'un fichier CSV."""
86     try:
87         dataset = pd.read_csv(chemin_fichier)
88         return dataset.iloc[:, [3, 4]].values
89     except FileNotFoundError:
90         print(f"Erreur : Le fichier {chemin_fichier} n'a pas été trouvé.")
91         raise
92     except Exception as e:
93         print(f"Une erreur s'est produite : {e}")
94         raise
95
```

```
7 class KMeansClustering:  
8  
9  
10    def __init__(self, n_clusters: int = 5, max_iter: int = 300, random_state: int = 0):  
11        self.n_clusters = n_clusters  
12        self.max_iter = max_iter  
13        self.random_state = random_state  
14        self.cluster_centers_ = None  
15        self.inertia_ = 0  
16  
17  
18    def initialiser_centroides(self, X: np.ndarray) -> np.ndarray:  
19        """Initialiser les centroïdes en utilisant la méthode k-means++."""  
20        np.random.seed(self.random_state)  
21  
22        centroides = [X[np.random.randint(0, X.shape[0])]]  
23  
24        for _ in range(1, self.n_clusters):  
25            distances = np.array([min([np.linalg.norm(x - c) for c in centroides]) for x in X])  
26            probs = distances**2 / np.sum(distances**2)  
27            cumulative_probs = np.cumsum(probs)  
28            r = random.random()  
29            idx = np.where(cumulative_probs >= r)[0][0]  
30            centroides.append(X[idx])  
31  
32        return np.array(centroides)  
33
```

```
34
35     def affecter_clusters(self, X: np.ndarray, centroides: np.ndarray) -> np.ndarray:
36         """Affecter les clusters en fonction du centreide le plus proche."""
37         distances = np.zeros((X.shape[0], self.n_clusters))
38         for k in range(self.n_clusters):
39             distances[:, k] = np.linalg.norm(X - centroides[k], axis=1)
40         return np.argmin(distances, axis=1)
41
42
43     def mettre_a_jour_centroides(self, X: np.ndarray, labels: np.ndarray) -> np.ndarray:
44         """Mettre à jour les centroïdes en fonction des clusters assignés."""
45         centroides = np.zeros((self.n_clusters, X.shape[1]))
46         for k in range(self.n_clusters):
47             if np.sum(labels == k) > 0:
48                 centroides[k] = np.mean(X[labels == k], axis=0)
49         return centroides
50
51
52     def calculer_inertie(self, X: np.ndarray, labels: np.ndarray, centroides: np.ndarray) -> float:
53         """Calculer l'inertie (somme des distances au carré du centreide le plus proche)."""
54         inertie = 0
55         for k in range(self.n_clusters):
56             if np.sum(labels == k) > 0:
57                 inertie += np.sum(np.linalg.norm(X[labels == k] - centroides[k], axis=1)**2)
58         return inertie
```

```
61 def ajuster(self, X: np.ndarray) -> np.ndarray:
62     """Ajuster le modèle aux données."""
63     self.cluster_centers_ = self.initialiser_centroides(X)
64
65     for _ in range(self.max_iter):
66         labels = self.affecter_clusters(X, self.cluster_centers_)
67         nouveaux_centroides = self.mettre_a_jour_centroides(X, labels)
68
69         if np.all(self.cluster_centers_ == nouveaux_centroides):
70             break
71
72         self.cluster_centers_ = nouveaux_centroides
73
74     labels = self.affecter_clusters(X, self.cluster_centers_)
75     self.inertia_ = self.calculer_inertie(X, labels, self.cluster_centers_)
76
77     return labels
78
79 def ajuster_predire(self, X: np.ndarray) -> np.ndarray:
80     """Ajuster le modèle et prédire les labels des clusters."""
81     return self.ajuster(X)
82
```

STRUCTURE DES DONNÉES POUR KMEANS



Format des données :

Données numériques continues (salaire, score de dépense).



Organisation :

Une valeur numérique continue représentant le salaire du client, utilisée comme première dimension pour le regroupement des comportements d'achat.

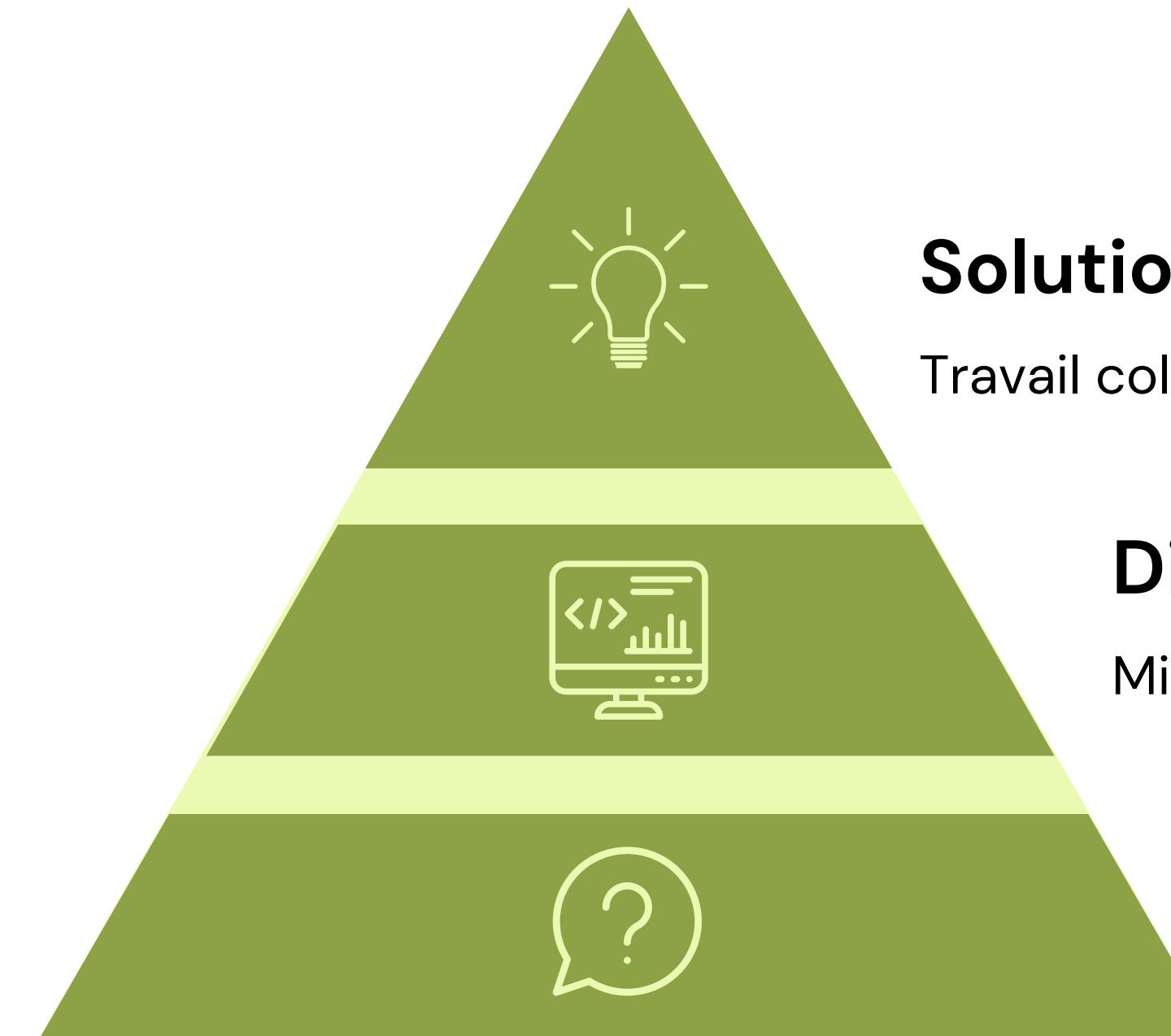


Utilisation :

Regroupement des clients selon leur salaire et leurs habitudes de dépense.



LIMITATIONS/PROBLÈMES RENCONTRÉS ET **SOLUTIONS APPORTÉES**



Solutions trouvées:

Travail collaboratif et partage des connaissances

Difficultés de programmation

Mise en application des concepts théoriques

Compréhension des consignes

Instructions parfois vagues et manque de clarté



Lannion

MÉTHODOLOGIE DE TEST ET VALIDATION

**Recherche de
bases de tests**

**Création de
scénarios de test**

**Mise en place d'un
environnement de test**

- Tests sur des jeux de données de référence pour des conditions standardisées.
- Crédit de cas précis avec résultats attendus pour valider chaque algorithme.
- Mise en place d'un cadre fiable pour des vérifications systématiques.

CONCLUSION

- Objectif : Comprendre le fonctionnement, les applications et les limites des algorithmes.
- Difficultés : Consignes floues, implémentation en Python, validation des résultats.
- Solutions : Recherches, ressources en ligne, travail de groupe.
- Apprentissages : Meilleure maîtrise des algorithmes, renforcement des compétences en programmation et analyse de données.