Université Badji Mokhtar - Annaba

Faculté des Sciences

Département d'Informatique

Travaux Pratiques : Data Mining

Sujet : Fusion et Extraction des N-Grams d'un fichier FASTA

Réalisé par : Wissem Mebarek

Djoudi islem

Encadré par : M. Boulemden

Année universitaire : 2025 / 2026

1. Introduction

Dans le domaine de la bio-informatique, l'analyse de séquences d'ADN permet de mieux comprendre la structure et le fonctionnement du matériel génétique. Ces séquences sont généralement représentées dans des fichiers au format `FASTA`, un format textuel standardisé qui contient des identifiants et des séquences d'acides nucléiques.

Le présent travail pratique (TP) s'inscrit dans le module de **Data Mining**, et vise à manipuler ce type de données à l'aide du langage Python. Plus précisément, il s'agit de :

— Fusionner plusieurs fichiers FASTA pour créer une base de séquences unique.

— Extraire et analyser des motifs de séquences appelés **n-grams**, permettant de détecter des répétitions ou similarités entre séquences.

Ce TP illustre ainsi les étapes de prétraitement, de manipulation et d'analyse de données textuelles appliquées au domaine biologique.

2. Objectifs du TP

Les objectifs du TP sont les suivants :

— **Manipulation des fichiers FASTA** : comprendre leur structure et les lire efficacement avec Python.

— **Fusion des données biologiques** : regrouper plusieurs fichiers pour créer un seul corpus cohérent.

— **Extraction d'informations** : utiliser les n-grams pour détecter des motifs de taille variable (1, 2, 3, ...).

— **Visualisation et interprétation** : présenter les résultats de manière claire et exploitable.

Ces étapes constituent une première approche du processus de *data preprocessing* en Data Mining.

3. Programme `fusion.py`

Ce premier programme est dédié à la fusion des fichiers FASTA. L'objectif est de combiner plusieurs jeux de séquences en un seul fichier `merged_file.fasta`, afin de simplifier les étapes d'analyse ultérieures.

3.1. Fonction `parse_fasta(filepath)`

Cette fonction lit un fichier FASTA ligne par ligne. Lorsqu'une ligne commence par le symbole “>”, elle contient des informations d'en-tête (identifiant, nom de l'organisme,

etc.). Les lignes suivantes contiennent les séquences nucléotidiques. Chaque séquence est enregistrée dans un dictionnaire contenant quatre clés :

— `entry` : identifiant de la séquence,

— `entry_name` : nom d'entrée,

— `organism` : organisme d'origine,

— `sequence` : séquence d'ADN complète.

Cette fonction retourne une liste de dictionnaires simulant un tableau de données.

```python
def parse_fasta(filepath):
    """Lit un fichier FASTA et retourne une liste de dictionnaires (simule un DataFrame)."""
    data = []
    entry = entry_name = organism = ""
    sequence = ""

    with open(filepath, "r") as f:
        for line in f:
            line = line.strip()
            if line.startswith(">"):
                # Sauvegarde de la séquence précédente
                if entry:
                    data.append({
                        "entry": entry,
                        "entry_name": entry_name,
                        "organism": organism,
                        "sequence": sequence
                    })
                    sequence = ""

                # Extraction de l'identifiant
                parts = line.split()
                id_parts = parts[0].split("|")

                entry = id_parts[1] if len(id_parts) > 1 else ""
                entry_name = id_parts[2] if len(id_parts) > 2 else ""

                organism = ""
                for part in parts:
                    if part.startswith("OS="):
                        organism = part[3:]
                        break
            else:
                sequence += line

    # Ajout de la dernière séquence
    if entry:
        data.append({
            "entry": entry,
            "entry_name": entry_name,
            "organism": organism,
            "sequence": sequence
        })

    return data
```

FIGURE 1 – Lecture et parsing des séquences FASTA

3.2. Fonction `merge_fasta_files(input_folder)`

Cette fonction a pour but de fusionner plusieurs fichiers FASTA situés dans un même dossier. Elle parcourt chaque fichier du répertoire, lit son contenu à l'aide de la fonction précédente et ajoute les séquences dans une seule liste globale. Cela permet de regrouper toutes les séquences en un seul ensemble prêt à être sauvegardé.

```python
def merge_fasta_files(file_list):
    """Fusionne plusieurs fichiers FASTA en un seul DataFrame (liste de dictionnaires)."""
    merged_data = []
    for file in file_list:
        print(f"Lecture de {file} ...")
        merged_data.extend(parse_fasta(file))  # ajoute les lignes du fichier
    return merged_data
```

FIGURE 2 – Fusion des fichiers FASTA en une seule structure

3.3. Fonction `save_as_fasta(data, output_file)`

Une fois la fusion terminée, cette fonction sauvegarde toutes les séquences dans un nouveau fichier FASTA. Chaque entrée est formatée selon la norme FASTA :

```
>IDENTIFIANT NOM_ENTRÉE OS=ORGANISME
SEQUENCE
```

Cette étape finalise la première partie du TP, en créant le fichier `merged_file.fasta` utilisé pour l'extraction des n-grams.

```python
def save_as_fasta(df, output_path):
    """Sauvegarde la liste de dictionnaires dans un fichier FASTA."""
    with open(output_path, "w") as f:
        for row in df:
            entry = row.get("entry", "")
            entry_name = row.get("entry_name", "")
            organism = row.get("organism", "")
            sequence = row.get("sequence", "")

            # Écriture du header FASTA
            header = f">sp|{entry}|{entry_name} OS={organism}\n"
            f.write(header)

            # Sépare la séquence en lignes de 60 caractères (comme dans les fichiers FASTA standards)
            for i in range(0, len(sequence), 60):
                f.write(sequence[i:i+60] + "\n")



# Liste de tes 5 fichiers FASTA
files = [
    "bovine.fasta",
    "human.fasta",
    "mouse.fasta",
    "rat.fasta",
    "zibrafish.fasta"
]

# Fusion des fichiers
df_merged = merge_fasta_files(files)


# Nombre total d'entrées
print(f"\nNombre total de séquences fusionnées : {len(df_merged)}")
output_path = "merged_files.fasta"
save_as_fasta(df_merged, output_path)

print(f"\n✅ Fichier FASTA fusionné sauvegardé avec succès dans : {output_path}")
```

FIGURE 3 – Enregistrement du fichier FASTA fusionné

—

4. Programme `ngram.py`

Ce second programme se concentre sur l'analyse du fichier fusionné. L'objectif est d'extraire des motifs (**n-grams**) à partir des séquences ADN et d'observer leur fréquence d'apparition. Un **n-gramme** est une sous-chaîne de longueur n :

— 1-gram = A, T, G, C
— 2-gram = AT, CG, TG, etc.
— 3-gram = ATT, TGA, GCC, etc.

4.1. Fonction `extract_ngrams(df, n)`

Cette fonction parcourt chaque séquence du fichier fusionné. Pour une taille n donnée, elle génère tous les motifs consécutifs de cette longueur et calcule leur fréquence. Le résultat est un dictionnaire associant chaque motif à son nombre d'occurrences pour chaque séquence.

```python
def extract_ngrams(df, n=1):
    """Extrait les n-grammes de taille n pour chaque séquence."""
    results = []

    for row in df:
        seq = row["sequence"]
        entry_id = row["entry"]

        # Dictionnaire des fréquences
        grams_counts = {}

        for i in range(len(seq) - n + 1):
            gram = seq[i:i+n]
            grams_counts[gram] = grams_counts.get(gram, 0) + 1

        # Associe l'ID et les fréquences
        features = {"ID": entry_id}
        features.update(grams_counts)
        results.append(features)

    return results
```

FIGURE 4 – Code de la fonction `extract_ngrams()`

Cette méthode est essentielle car elle permet de transformer une séquence textuelle en un ensemble de caractéristiques numériques, étape cruciale dans toute démarche de Data Mining.

4.2. Fonction `show_results(features, n)`

Cette fonction s'occupe d'afficher les résultats des n-grams extraits. Elle affiche notamment :

— le nombre total de séquences traitées ;

— pour chaque séquence, la liste des n-grams détectés et leur fréquence.

```python
def show_results(features, n):
    """Affiche les 10 premières séquences avec leurs n-grammes."""
    if not features:
        print(f"\nAucun {n}-gramme extrait.")
        return

    print(f"\n=== Extraction des {n}-grammes ===")
    print(f"Nombre total de séquences traitées : {len(features)}\n")

    for i, f in enumerate(features[:10]):  # 10 premières séquences
        print(f"--- Séquence {i+1} ---")
        print("ID :", f["ID"])
        grams_sorted = sorted((k, v) for k, v in f.items() if k != "ID")
        for gram, count in grams_sorted:
            print(f"  {gram}: {count}", end="  ")
        print("\n" + "-" * 40)
```

FIGURE 5 – Code de la fonction `show_results()`

4.3. Résultats obtenus

Les résultats ci-dessous illustrent l'exécution du programme pour les valeurs de $n = 1$, $n = 2$ et $n = 3$. Ces trois cas permettent de comparer la granularité des motifs extraits : plus n est grand, plus les motifs sont précis, mais aussi plus rares.

```
=== Extraction des 1-grammes ===
Nombre total de séquences traitées : 3678

--- Séquence 1 ---
ID : A0A3Q1LMK4
  A: 45   C: 13   D: 28   E: 36   F: 16   G: 41   H: 15   I: 12   K: 23   L: 62   M: 10   N: 17   P: 60   Q: 26   R: 35   S: 61   T: 34   V: 28   W: 8   Y: 5
----------------------------------------
--- Séquence 2 ---
ID : A0A3Q1M2Z5
  A: 24   C: 18   D: 21   E: 28   F: 16   G: 14   H: 9   I: 23   K: 30   L: 43   M: 7   N: 17   P: 28   Q: 16   R: 29   S: 45   T: 23   V: 25   W: 5   Y: 21
----------------------------------------
--- Séquence 3 ---
ID : A0A3Q1MFH0
  A: 84   C: 11   D: 22   E: 27   F: 22   G: 51   H: 17   I: 14   K: 14   L: 41   M: 5   N: 15   P: 55   Q: 22   R: 72   S: 37   T: 20   V: 39   W: 8   Y: 12
----------------------------------------
--- Séquence 4 ---
ID : A0A3Q1MNA1
  A: 27   C: 7   D: 7   E: 6   F: 11   G: 22   H: 9   I: 8   K: 13   L: 37   M: 2   N: 15   P: 31   Q: 13   R: 18   S: 32   T: 12   V: 18   W: 4   Y: 8
----------------------------------------
--- Séquence 5 ---
ID : A0A3Q1N449
  A: 29   C: 11   D: 23   E: 31   F: 13   G: 30   H: 19   I: 16   K: 20   L: 60   M: 10   N: 16   P: 33   Q: 18   R: 39   S: 46   T: 32   V: 25   W: 12   Y: 16
----------------------------------------
--- Séquence 6 ---
ID : A0A452DHW6
  A: 33   C: 10   D: 24   E: 24   F: 11   G: 29   H: 3   I: 19   K: 18   L: 40   M: 9   N: 7   P: 29   Q: 20   R: 31   S: 29   T: 15   V: 20   W: 5   Y: 8
----------------------------------------
```

FIGURE 6 – Résultats des séquences pour les 1-grams

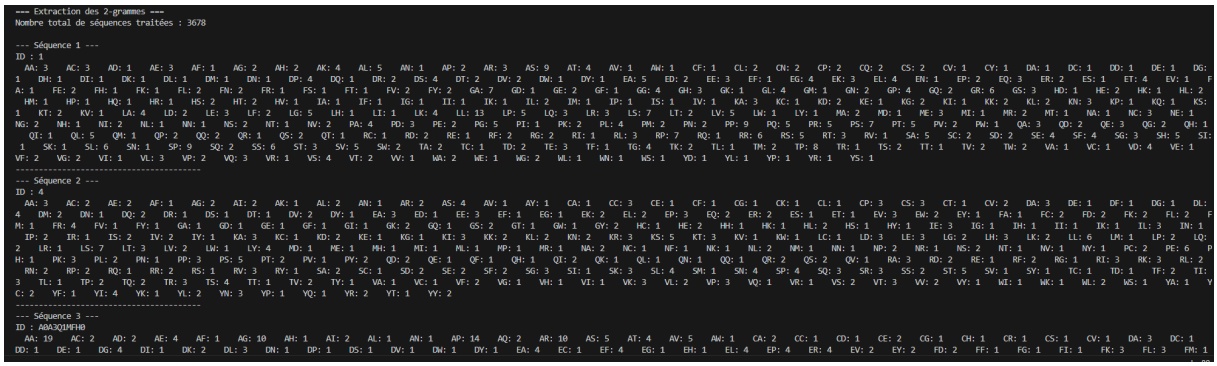

FIGURE 7 – Résultats des séquences pour les 2-grams

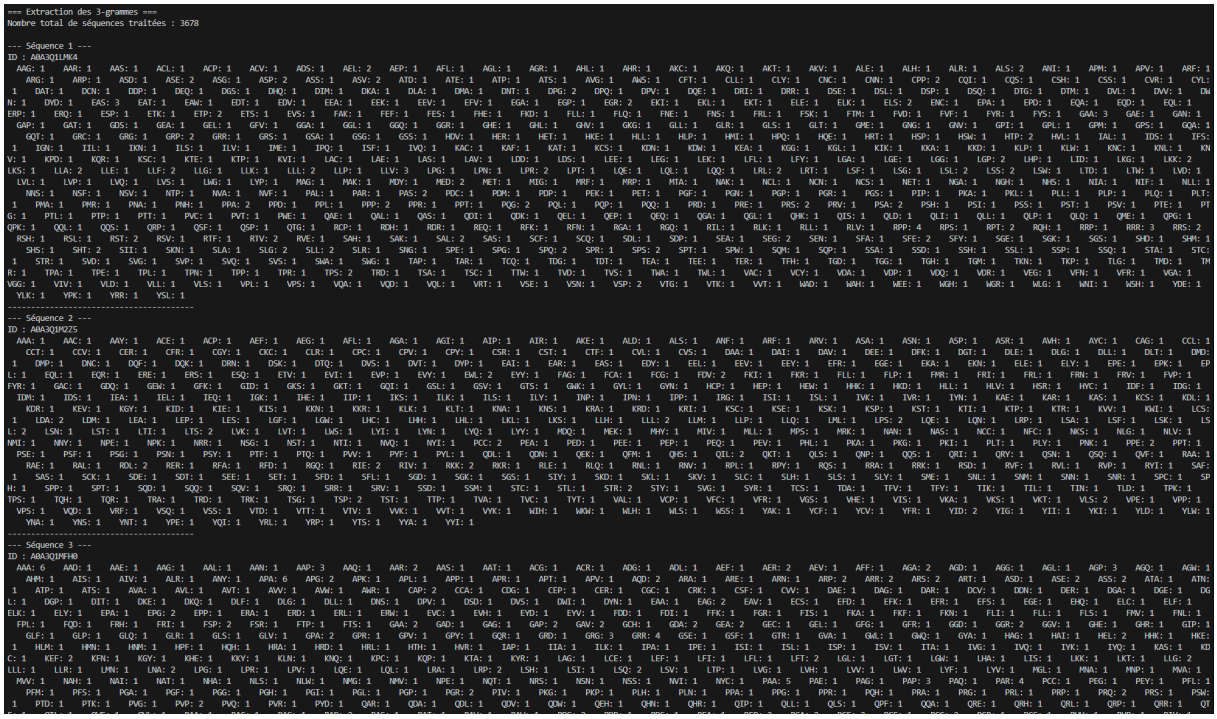

FIGURE 8 – Résultats des séquences pour les 3-grams

—

5. Conclusion

À travers ce TP, nous avons mis en œuvre deux aspects essentiels du Data Mining appliqué à la bio-informatique :

1. le **prétraitement des données biologiques**, via la lecture, la fusion et la normalisation des séquences FASTA ;
2. l'**analyse de motifs** à travers l'extraction des n-grams pour identifier les combinaisons les plus fréquentes.

Ce travail met en évidence l'importance de l'automatisation et du traitement textuel dans l'analyse de données biologiques. Les concepts de n-grams sont également utilisés dans d'autres domaines, tels que le traitement automatique du langage naturel (NLP), la détection de similarité ou encore la classification de séquences.