



## **Projet Natural Language Processing**

### **Medibot : Chatbot Médical Intelligent basé sur le NLP et le Machine Learning**

#### **Mastère Data Engineering & IA**

Réalisé par :

- Darryll Genève Junior JOSEPH
- Mariem BEN SALAH

Année scolaire : 2024 – 2025

## **1. Introduction :**

Au fur et à mesure que les volumes de données textuelles se multiplient chaque jour, qu'il s'agisse des réseaux sociaux, des sites d'évaluations ou des systèmes de service clientèle, les entreprises se trouvent progressivement confrontées à la nécessité de décoder, examiner et exploiter ces textes de manière automatique. Par conséquent, l'analyse automatique du langage naturel (NLP – Natural Language Processing) apparaît comme une option indispensable pour tirer de la valeur des textes non structurés, affiner le processus décisionnel, identifier les tendances naissantes et perfectionner les échanges avec les utilisateurs.

Dans ce cadre, notre initiative était de développer un chatbot médical intelligent, apte à saisir les demandes formulées en langage naturel par les utilisateurs, à extraire automatiquement les symptômes mentionnés, et ensuite à anticiper la maladie probable grâce à des méthodes d'apprentissage automatique. Le chatbot devait également être en mesure de formuler une réponse pertinente à partir d'une base de connaissances médicales.

Pour réaliser cet objectif, nous avons élaboré un pipeline d'analyse de textes exhaustif qui comprend l'extraction, le nettoyage et la modélisation automatique grâce à des algorithmes de Machine Learning non supervisé (clustering KMeans). Ensuite, nous exposons le modèle par le biais de trois APIs RESTful distinctes : une API consacrée à la gestion de la base de données médicale, une API pour le système de dialogue (chatbot) et une API dédiée à l'estimation des maladies.

Le projet a été conçu avec une architecture généraliste, modulaire, évolutive et aisément déployable dans des environnements d'application en place, tout en permettant une éventuelle transition vers une infrastructure orientée cloud.

Pour consulter l'implémentation détaillée, le dépôt GitHub du projet est accessible via le lien suivant : <https://github.com/Darryll1/NLP.git>

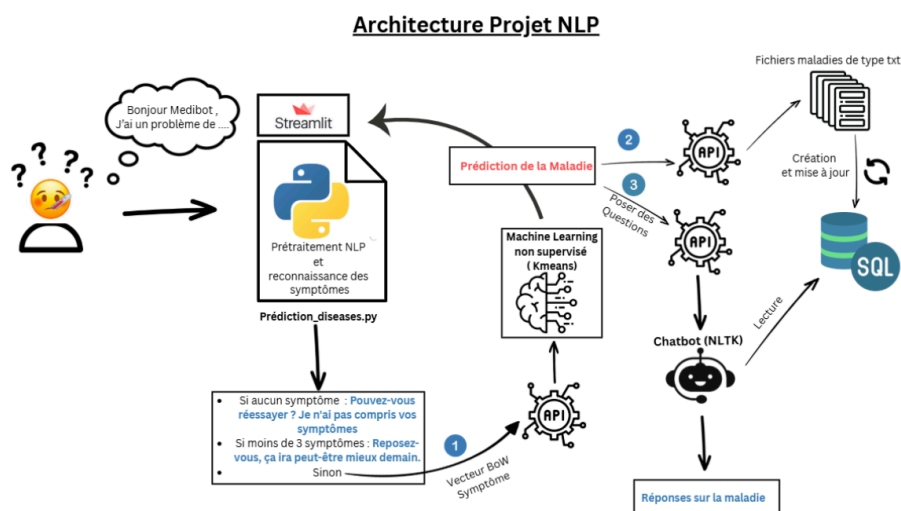
## 2. Présentation du projet et de la problématique métier :

L'élaboration d'un assistant virtuel médical intelligent vise à améliorer la gestion initiale des patients et à faciliter l'accès aux informations médicales. Dans le contexte actuel de surcharge des dispositifs de santé et d'une demande pressante des usagers pour des réponses instantanées à leurs questions de santé, il est primordial d'offrir des outils numériques aptes à délivrer rapidement une orientation préliminaire fiable.

Notre initiative vise à résoudre ce défi en développant un assistant virtuel capable d'interpréter les interrogations médicales formulées en langage naturel, de repérer automatiquement les symptômes cités par l'utilisateur, et de prévoir la pathologie la plus susceptible d'être liée à ces symptômes. Ce chatbot est basé sur un modèle d'apprentissage automatique, alimenté par une base de données médicale structurée, et peut être consulté grâce à une structure d'API modulaire.

L'idée est de fournir un service intelligent qui, non seulement, détecte les indices cliniques mentionnés dans les demandes des utilisateurs, mais qui offre aussi des informations adaptées et basées sur des données scientifiques fiables, tout en garantissant une interaction facile, rapide et intuitive.

Ce projet utilise une approche axée sur les données, associant le prétraitement de texte, le modelage par apprentissage machine, la gestion d'une base de données médicale et la création d'une API RESTful pour garantir l'accessibilité et la scalabilité du service.



L'image présente l'**architecture du projet NLP (traitement du langage naturel)** pour la **prédiction de maladies** à partir de symptômes décrits par un utilisateur.

### 3. Technologies et outils utilisés :

Le développement du chatbot médical intelligent repose sur un ensemble cohérent de bibliothèques et de technologies adaptées aux exigences du traitement du langage naturel, de l'apprentissage automatique, de la conservation des données et du déploiement d'APIs. L'architecture repose sur plusieurs modules autonomes, chacun assigné à une fonction spécifique, ce qui a guidé des choix techniques correspondant à leur rôle respectif.

L'intégralité des APIs du projet est élaborée à l'aide de FastAPI, un framework Python innovant destiné à concevoir des interfaces RESTful de manière rapide. FastAPI est reconnu pour sa rapidité d'exécution, sa capacité native à gérer l'asynchrone et sa documentation automatique des routes via Swagger UI. Comme l'illustre le code ci-dessous provenant du fichier API\_Chatbot.py, la déclaration de l'application FastAPI dans nos scripts est simple et structurée, ce qui facilite l'exposition rapide de services web robustes :

```
from fastapi import FastAPI, Request, Body
app2 = FastAPI()
@app2.get("/")
def read_root():
    return {"message": "API is working"}
```

Pour le stockage et l'organisation des connaissances médicales, le projet se sert d'SQLite, une base de données relationnelle compacte et intégrée ce qui la rend appropriée pour les environnements de développement locaux. Cette solution facilite la consolidation des maladies et de leurs descriptions au sein d'une seule table, tout en offrant une mise en application très simple. Comme démontré dans API\_BD.py, l'établissement de la connexion et la création de la base de données s'effectuent en quelques lignes de code :

```
@app3.post("/creer-bdd")
def creer_base_de_donnees():
    conn = sqlite3.connect("sante.db")
    cursor = conn.cursor()

    # Création de la table si elle n'existe pas
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS maladie (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL UNIQUE,
```

```
informations TEXT NOT NULL
```

```
)  
""")
```

Concernant le traitement de texte, l'intégration de la bibliothèque NLTK (Natural Language Toolkit) a permis d'exécuter les phases de prétraitement : tokenisation, élimination des stopwords et segmentation des phrases. Il est indispensable de procéder à ces transformations pour normaliser les requêtes des utilisateurs. L'emploi d'un RegexpTokenizer facilite en particulier l'extraction des mots porteurs de sens :

```
from nltk import RegexpTokenizer  
def tokenisation(text):  
    tokenizer = RegexpTokenizer(r'\w+')  
    tokens1=[]  
    tokens1+=tokenizer.tokenize(text.lower())  
    return tokens1
```

Après le nettoyage des textes, ceux-ci sont transformés en vecteurs numériques grâce à deux méthodes complémentaires :

- Dans le script Prediction\_diseases.py, le Bag of Words (via CountVectorizer) est utilisé pour former le modèle de regroupement. Dans API\_Chatbot.py,
- Le TF-IDF (via TfidfVectorizer) est mis en œuvre pour évaluer la similarité entre l'interrogation de l'utilisateur et la base de données des connaissances.

Ces techniques proposent une solution équilibrée entre facilité d'utilisation et performance pour des volumes de données moyens. Voici un exemple de vectorisation TF-IDF extrait du code du chatbot :

```
from sklearn.feature_extraction.text import TfidfVectorizer  
TfidfVec = TfidfVectorizer(tokenizer = tokenisation, stop_words=mots) #english  
tfidf = TfidfVec.fit_transform(tokens_phrase)
```

L'algorithme KMeans de scikit-learn est utilisé lors de l'étape de modélisation pour organiser les vecteurs présentant des symptômes similaires en groupes. Cette décision se base sur la facilité de mise en œuvre et l'adéquation à une tâche de classification non supervisée.

Finalement, des bibliothèques additionnelles telles que NumPy ou Pandas viennent compléter l'écosystème du projet pour une gestion efficace des vecteurs, la lecture de fichiers CSV, et la mise en place des ensembles de données.

En résumé, le projet utilise une stack technique uniforme qui combine, FastAPI pour la mise à disposition des APIs, NLTK pour l'analyse de texte, Scikit-learn pour l'élaboration de modèles, TF-IDF / BoW pour la transformation en vecteurs, SQLite pour le stockage local et Pandas/NumPy pour le traitement des données.

L'ensemble est élaboré pour être lisible, modulaire et facilement entretenable, tout en fournissant une base robuste pour une éventuelle montée en charge ou industrialisation.

#### **4. Pipeline de traitement de texte (Cleaning & Preprocessing) :**

Afin de faciliter la compréhension des requêtes formulées en langage naturel par l'utilisateur, un processus de traitement textuel simple mais performant a été instauré pour l'agent conversationnel. Ce pipeline vise à mettre en parallèle la phrase fournie par l'utilisateur avec un corpus de connaissances médicales textuelles, dans le but de choisir automatiquement la réponse la plus appropriée.

L'intégralité de ce processus est spécifiée dans la fonction `repondre(...)`, qui regroupe les étapes ci-après :

Premièrement, en se basant sur le nom de la maladie choisie, le chatbot obtient le texte médical correspondant (généralement un fichier .txt enregistré dans une base SQLite) par le biais d'une requête. Le texte est transformé en minuscules avant d'être segmenté en phrases individuelles à l'aide de `nltk.sent_tokenize`. La question formulée par l'utilisateur est intégrée à cette liste.

Par la suite, nous effectuons un nettoyage lexical en utilisant une fonction de tokenisation sur mesure qui s'appuie sur `RegexpTokenizer`. Cette fonctionnalité récupère les termes importants en éliminant la ponctuation et en transformant tous les caractères en lettres minuscules.

```
from nltk import RegexpTokenizer
def tokenisation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    tokens1=[]
    tokens1+=tokenizer.tokenize(text.lower())
    return tokens1
```

Une fois le corpus préparé, chaque phrase (y compris celle de l'utilisateur) est transformée en vecteurs à l'aide de la méthode TF-IDF via `TfidfVectorizer` du module `scikit-learn`. Cette méthode consiste à convertir chaque phrase en un vecteur de valeur, basé sur la pertinence des mots employés.

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
mots = stopwords.words('french')
tfidf = TfidfVec.fit_transform(tokens_phrase)
```

Une mesure de similarité cosinus est employée pour mettre en parallèle la requête de l'utilisateur avec les autres expressions. Elle permet de déterminer quelle phrase a la plus grande proximité sémantique avec la question formulée. Si une correspondance adéquate n'est pas identifiée, une réponse standard est fournie ; dans le cas contraire, la requête la plus pertinente est extraite et présentée à l'utilisateur.

```
from sklearn.metrics.pairwise import cosine_similarity
vals = cosine_similarity(tfidf[-1],tfidf)
idx = vals.argsort()[0][-2]
```

Par conséquent, sans l'utilisation de modèles sophistiqués ou d'intelligence artificielle générative, le système réussit à saisir l'intention communiquée et à y répondre de façon spécifique, tout en demeurant simple, rapide et contrôlable.

## **5. Modélisation et apprentissage automatique (Machine Learning) :**

Le projet utilise l'algorithme `KMeans` comme méthode de clustering non supervisée pour la modélisation, avec pour objectif de rassembler les patients présentant des symptômes comparables en tenant compte de la proximité de leur profil sous forme vectorielle. Cette méthode a prouvé son efficacité pour repérer des séries de symptômes répétitifs et les associer aux affections les plus susceptibles.

Le fichier `Training.csv`, qui contient les symptômes codés en variables binaires (présent/absent) pour chaque patient et la maladie diagnostiquée, sert de base pour l'entraînement. Après le chargement du fichier, on procède à la suppression de certaines colonnes superflues, on traduit les désignations des symptômes en français

et on remplit les valeurs absentes en utilisant la méthode de moyenne (SimpleImputer).

Comme le démontre l'extrait ci-dessous, ces informations sont par la suite normalisées grâce à StandardScaler, une phase cruciale pour assurer la cohérence de l'échelle des variables avant de former le modèle :

```
# GESTION DES VALEURS MANQUANTES (NaN)
imputer = SimpleImputer(strategy="mean") # Remplace les NaN par la moyenne
X_imputed = imputer.fit_transform(X)

# Normalisation des données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)
```

On utilise alors l'algorithme KMeans, avec un total de 41 clusters prédéterminé, correspondant au nombre distinct de maladies à identifier. Chaque agrégat représentera un ensemble de patients semblables. Le code suivant est un exemple illustratif de cette étape :

```
#Clustering avec k=41
k = 41
modele = KMeans(n_clusters=k, random_state=42, n_init=10)
df["Cluster"] = modele.fit_predict(X_scaled)
```

Après l'acquisition des clusters, une cartographie automatique est produite pour lier chaque cluster à la pathologie la plus courante parmi les observations qui y sont attribuées. Cela facilite la conversion de chaque prédiction de regroupement en prédiction pathologique, comme démontré ci-dessous :

```
# Trouver la maladie la plus fréquente dans chaque cluster
mapping_clusters = df.groupby("Cluster")["Pronostic"].agg(lambda x:
x.value_counts().idxmax())

print(mapping_clusters) # Affiche quel cluster est associé à quelle maladie
#Transformer les clusters en labels de maladies prédits
df["Cluster_Pred_Maladie"] = df["Cluster"].map(mapping_clusters)
```

Ce modèle est ensuite intégré dans une **API FastAPI** via un endpoint /Kmeans, qui prend en entrée un vecteur de symptômes et retourne une maladie prédite.



## 6. Évaluation des performances du modèle

Pour évaluer la fiabilité de notre méthode de modélisation, nous avons effectué une phase d'évaluation en confrontant les résultats du regroupement aux diagnostics authentiques présents dans l'ensemble de données d'origine. Même si KMeans est une technique non supervisée, il est possible d'évaluer ses résultats de manière supervisée en associant chaque groupe à la pathologie la plus courante qu'il renferme.

La validation est effectuée en utilisant le rapport de classification scikit-learn, grâce à la fonction `classification_report`. Il offre la possibilité d'obtenir les indicateurs de performance majeurs suivants :

- **accuracy** (exactitude globale),
- **precision** (justesse des prédictions positives),
- **recall** (capacité à détecter tous les cas d'une maladie),
- **f1-score** (moyenne harmonique précision/rappel).

Cette phase est illustrée par le code suivant, extrait du script principal d'entraînement :

```
from sklearn.metrics import classification_report
#Comparer avec les vraies classes
y_true = df["Pronostic"]
y_pred = df["Cluster_Pred_Maladie"]
#Générer le rapport de classification
print(classification_report(y_true, y_pred, zero_division=1))
```

Les conclusions du rapport ont été présentées dans la console, ce qui a permis de distinguer clairement les maladies bien identifiées de celles mal différenciées, et d'adapter si nécessaire le nombre de regroupements ou les méthodes de prétraitement. Cette technique permet d'utiliser une démarche non supervisée tout en l'évaluant avec des indicateurs habituellement employés dans l'apprentissage supervisé, un aspect crucial dans le cadre d'un projet de recherche ou de prototypage rapide.

## 7. Développement de l'API RESTful :

Le projet tout entier est basé sur la présentation de fonctionnalités par le biais d'APIs REST, élaborées avec FastAPI, un framework apprécié pour sa facilité d'utilisation, sa vitesse et sa documentation interactive intégrée. Chaque élément majeur du système le chatbot, la prédiction, la gestion des données est contenue dans sa propre API.

L'exemple suivant illustre comment déclarer une API avec FastAPI de manière simple et claire :

```
from fastapi import FastAPI, Request, Body
app2 = FastAPI()
```

Chaque API offre une route principale POST pour traiter les requêtes JSON. Par exemple, l'API de prédiction de maladie offre un point d'accès /Kmeans qui accepte un vecteur de symptômes (Bag of Words), traité et évalué par le modèle KMeans, puis renvoie le diagnostic approprié :

```
@app.post("/Kmeans")
async def disease_prediction(request: Request):
```

Cette architecture permet une séparation claire des responsabilités, une maintenabilité facilitée, et une scalabilité future possible vers des services indépendants conteneurisés.

## 8. Base de données et enrichissement des connaissances médicales :

Une base de données **SQLite** a été utilisée pour centraliser les fiches de maladies, stockées initialement sous forme de fichiers .txt. Le script d'initialisation (creer\_bdd) parcourt automatiquement le répertoire Maladies/, lit chaque fichier et alimente une table nommée maladie, tout en évitant les doublons :

```
@app3.post("/creer-bdd")
def creer_base_de_donnees():
    conn = sqlite3.connect("sante.db")
    cursor = conn.cursor()

    # Création de la table si elle n'existe pas
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS maladie (
id INTEGER PRIMARY KEY AUTOINCREMENT,
nom TEXT NOT NULL UNIQUE,
informations TEXT NOT NULL
)
""")
```

Ce mécanisme d'enrichissement automatique permet d'élargir facilement le socle de connaissances en ajoutant de nouveaux fichiers maladies. Ces textes alimentent ensuite le moteur de recherche sémantique du chatbot, qui y puise la réponse la plus pertinente à l'aide de TF-IDF et de la similarité cosinus.

## 9. Interface de l'application streamlit :



**Medibot - Votre chatbot médical**



Bonjour ! Je suis Medibot 🇫🇷, votre assistant médical virtuel. Je suis là pour écouter vos symptômes et vous aider à comprendre ce qui pourrait être la cause de votre mal-être. Dites-moi, qu'est-ce qui vous dérange ?

Veuillez entrer vos symptômes

Valider les symptômes

## 10. Fonctionnement du chatbot :

L'utilisateur saisit une description libre de ses symptômes. Le chatbot traite le texte, extrait les symptômes pertinents, puis agit selon trois cas :

**Cas 1** : Entrée incompréhensible ou sans symptômes reconnus

Saisie de l'utilisateur : "bla bla, il fait beau aujourd'hui !"

Réponse de Medibot : "Pouvez-vous réessayer ? Je n'ai pas compris vos symptômes."

Capture :



**Cas 2** : Moins de 3 symptômes détectés

Saisie de l'utilisateur : j'ai des maux de tête

Réponse de Medibot : "Reposez-vous, ça ira peut-être mieux demain. »

Capture :



### Cas 3 : Trois symptômes ou plus détectés

Le chatbot prédit une maladie en se basant sur le modèle Kmeans.

#### Exemple 1

Saisie de l'utilisateur : Bonjour docteur, je suis venu vous voir parce que je me sens vraiment mal. Depuis quelques jours, j'ai remarqué que mes muscles sont devenus très faibles, je me sens comme si j'étais épuisé tout le temps. J'ai également des irritations dans la gorge qui sont très douloureuses et qui m'empêchent de avaler correctement. Et pour couronner le tout, j'ai une fièvre élevée qui ne semble pas baisser, malgré les médicaments que j'ai pris. Mais ce qui me préoccupe le plus, c'est que j'ai eu un rapport sexuel sans préservatif il y a quelques jours et je crains d'avoir contracté une maladie sexuellement transmissible. J'ai été imprudent et je regrette mon erreur. Je suis très inquiet, docteur, et j'espère que vous pourrez m'aider à comprendre ce qui se passe et à me soigner. Je suis prêt à faire des examens et à suivre un traitement, mais je veux savoir ce qui se passe et comment je peux m'en sortir.

## Réponse de Medibot :

symptomes enregistré : Bonjour docteur, je suis venu vous voir parce que je me sens vraiment mal. Depuis quelques jours, j'ai remarqué que mes muscles sont devenus très faibles, je me sens comme si j'étais épuisé tout le temps. J'ai également des irritations dans la gorge qui sont très douloureuses et qui m'empêchent de avaler correctement. Et pour couronner le tout, j'ai une fièvre élevée qui ne semble pas baisser, malgré les médicaments que j'ai pris. Mais ce qui me préoccupe le plus, c'est que j'ai eu un rapport sexuel sans préservatif il y a quelques jours et je crains d'avoir contracté une maladie sexuellement transmissible. J'ai été imprudent et je regrette mon erreur. Je suis très inquiet, docteur, et j'espère que vous pourrez m'aider à comprendre ce qui se passe et à me soigner. Je suis prêt à faire des examens et à suivre un traitement, mais je veux savoir ce qui se passe et comment je peux m'en sortir.

MediBot 🤖 : Ce qui m'interpelle 😬



MediBot 🤖 : Aïe ! Il y a de fortes chances que vous souffrez de: SIDA

Vous allez être redirigé !

Ensuite, l'utilisateur peut poser des questions complémentaires sur la maladie (ex. : causes, traitements, durée, contagiosité...).

MediBot 🤖 : ? Voulez-vous poser des questions sur SIDA ?

Oui

Non

Si le choix de l'utilisateur est Non :



Si le choix de l'utilisateur est Oui :

MediBot 🤖 : Que souhaitez-vous savoir sur SIDA ? Si vous n'avez pas de question, tapez simplement 'Au revoir'.

Entrez votre question :

Valider la question

Exemple de questions :

MediBot 🤖 : Que souhaitez-vous savoir sur SIDA ? Si vous n'avez pas de question, tapez simplement 'Au revoir'.

Entrez votre question :

Valider la question

MediBot 🤖 :

le sida, ou syndrome d'immunodéficience acquise, est une maladie causée par le virus de l'immunodéficience humaine (vih).

MediBot 🤖 : Que souhaitez-vous savoir sur SIDA ? Si vous n'avez pas de question, tapez simplement 'Au revoir'.

Entrez votre question :

comment le SIDA se transmet-il?

Valider la question

MediBot 🤖 :

Le VIH se transmet principalement par voie sexuelle lors de rapports non protégés, par contact avec du sang contaminé, notamment à travers le partage de seringues infectées ou des transfusions sanguines non sécurisées, et de la mère à l'enfant lors de la grossesse, de l'accouchement ou de l'allaitement.

MediBot 🤖 : Que souhaitez-vous savoir sur SIDA ? Si vous n'avez pas de question, tapez simplement 'Au revoir'.

Entrez votre question :

au revoir

Valider la question

MediBot 🤖 : 🙌 Au revoir et prenez soin de vous !



## **11.Conclusion :**

Ce projet a permis d'appliquer, de manière exhaustive et organisée, une chaîne de traitement qui intègre les divers aspects de l'intelligence artificielle dans le domaine des données médicales. En développant un chatbot capable de comprendre une requête d'utilisateur en langage naturel et d'offrir un diagnostic prédictif basé sur un apprentissage non supervisé, nous avons pris en charge toutes les phases cruciales du cycle de développement d'un projet d'IA.

Depuis l'étape de préparation des données, nous avons appliqué strictement des méthodes de nettoyage, de traduction, d'imputation et de normalisation pour garantir une base de données utilisable. Cette tâche a efficacement nourri un modèle de regroupement (KMeans), apte à rassembler les profils semblables et à déterminer, pour chaque situation, la pathologie la plus probable.

L'option d'utiliser un modèle non supervisé pour la prédiction découle d'une démarche de classification exploratoire, motivée par la structure binaire et partiellement labellisée du jeu de données. On a évalué les performances du modèle en utilisant des métriques standards (précision, rappel, f1-score), obtenues en mettant en parallèle les regroupements prévus et les diagnostics réels.

L'implémentation du modèle à travers une API RESTful, en parallèle d'un moteur de réponse intelligent fondé sur des critères de ressemblance textuelle, offre la possibilité d'une fourniture instantanée du service via une interface web. L'approche par modules adoptée facilite l'extension du système et permet son intégration potentielle dans des applications médicales plus vastes.