



Titre

- *Nom du projet :* **GOMYCARE**
- *Sous-titre :* ‘ Medical diagnostic chatbot ’
- Projet de formation en Data-science (bootcamp).
- Par. Youcef BENGRID.

Problématique

Ce projet consiste à créer un assistant médical capable d'identifier la maladie la plus probable. à partir des symptômes renseignés par l'utilisateur. Il s'appuie sur un jeu de données structuré contenant des maladies, leurs symptômes associés. L'objectif est de développer un modèle prédictif simple, comme un arbre de décision, et de l'intégrer à une interface utilisateur simple, sous forme de Chatbot.

Présentation

Clin d'œil à **GOMYCODE** l'école formatrice de ce bootcamp en data-science, conduit sous la responsabilité pédagogique de la formatrice **Ikram AISSIOU**,

GOMYCARE est le chatbot médical intelligent qui permet d'obtenir rapidement le diagnostic probable d'une maladie à partir des symptômes renseignés. Et pourquoi pas proposer également les premières précautions générales à adopter (Compétence à développée).

Ces atouts :

- Interface simple : sélection des symptômes d'une liste déroulante.
- Diagnostic rapide grâce à un modèle d'IA entraîné.
- Accessible à tous, avec langage médical simple.
- Ne remplace pas un médecin, mais oriente efficacement vers une consultation.

Comment fonctionne **GOMYCARE** ?

- L'utilisateur sélectionne des symptômes d'une liste déroulante et clique sur “Diagnostiquer”
- Le modèle prédit la maladie probable et l'affiche.

Technologies utilisées

Pour le développement, nous avons utilisé les technologies comme

• Python	• ML/A.D	• Streamlit	• ydata_profiling	• joblib	• scikit-learn	• CSS

Développement

J'ai développé un modèle de Machine Learning de classification, basé sur arbre de décision comme algorithme, capable de prédire une maladie à partir d'une liste de symptômes. L'objectif final est d'obtenir un bon compromis entre performance sur les données d'entraînement et généralisation sur les données test.

Par la suite, je l'ai intégré dans un Chatbot médical intelligent, conçu pour aider les utilisateurs à identifier de manière préliminaire des maladies courantes, à partir des symptômes renseignés. Le modèle a été entraîné sur un jeu de données de 4 920 enregistrements, téléchargé depuis Kaggle.

1. Sélection Préparation et prétraitement des données

Avant toute chose j'ai :

- Importer les premières bibliothèques utiles :
 - *pandas, numpy* pour la manipulation des données.
 - *matplotlib.pyplot, seaborn* pour les visualisations .
 - *ydata_profiling* pour une analyse automatique du dataset.
 - *scikit-learn* pour l'encodage, la séparation des données et les modèles ML.
 - *jobjlib* pour sauvegarder le modèle et les métadonnées.
- Chargement le jeu à partir de son url =
<https://raw.githubusercontent.com/itachi9604/healthcare-chatbot/refs/heads/master/Data/dataset.csv>
- Faire les premières vérification `head(5)` et `info`.
- Renommé les colonnes pour mieux structurer les données. : Une colonne "disease" suivie de plusieurs colonnes "symptom_1", "symptom_2"...."symptom_17".
 - 1.Créer une liste vide pour les nouveaux noms de colonnes '`new_column`' .
 - 2.Ajouter "disease" pour la première colonne
 - 3.Ajouter "symptom_1", "symptom_2", etc., pour le reste
 - 4.Appliquer la nouvelle liste de noms au df.
 - 5.verification avec `head(5)`
- Générer un rapport profiling "Symptom Profiling Report".
- Remplacement de toutes les valeurs manquantes (NaN) par une chaîne vide.

2. Transformation des données

- J'ai converti les symptômes multiples par ligne en un format one-hot :
 - Liste unique de tous les symptômes
 - Création d'un DataFrame avec colonnes = tous les symptômes
 - Pour chaque ligne, on mets 1 si le symptôme est présent

3. Machine Learning

Après séparation X (symptômes) et y (maladie) et l'encodage encodage des maladies en utilisant *LabelEncoder*, j'ai partagé mes données (*Split*) en 70% train et 30% test.

Un premier entraînement de l'arbre de décision a été réalisé sans limitation de profondeur, avec une génération d'un rapport de classification.

Le modèle d'arbre de décision entraîné a été évalué sur les données de test et il a affiché une performance exceptionnelle :

– Précision (precision) : 1.00
– Rappel (recall) : 1.00
– Score F1 : 1.00
– Profondeur de l'arbre : 59
– Nombre de feuilles : 72

L'exactitude (*Accuracy*) 100 % sur l'ensemble du jeu de test (1476 exemples) m'a fais conclure que mon model soulève un problème de *Overfitting* où le modèle mémorisait les exemples sans généraliser.

Pour remédier a cet surapprentissage j'ai commencer a faire un élagage (pruning) de mon arbre pour éviter qu'elle ne devienne trop complexe. Différentes profondeurs maximales ont été testées, de 5 à 50. J'ai observé une **amélioration progressive** de la précision sur les données de test à mesure que la profondeur augmentait.

Depth	Accuracy
5	0.15
10	0.29
15	0.41
20	0.52
25	0.66
30	0.77
35	0.90
40	0.97
45	0.97
50	0.98

Quand mon model a une faible profondeur (5 à 15), le modèle sous-apprenait (*underfitting*), avec des précisions inférieures à 50 %. Mais à partir d'une profondeur de 30 à 50, la précision s'est stabilisée autour de 98 % sur les données de test, avec une précision équivalente sur les données d'entraînement, ce qui montre une bonne généralisation.

Pour conclure et optimiser les meilleurs hyperparamètres, j'ai cherché la meilleure combinaison en utilisant *GridSearchCV*. Ce qui ma permis de tester plusieurs combinaisons de paramètres via une validation croisée (cross-validation),

<i>max_depth</i>	Profondeur maximale de l'arbre.	[30, 35, 40, 45, 50]
<i>min_samples_split</i>	Nombre minimal d'échantillons pour diviser un nœud.	2
<i>min_samples_leaf</i>	Nombre minimal d'échantillons dans une feuille.	1
<i>CV</i>	Combinaisons testées avec validation croisée.	5

La meilleure combinaison trouvée était une profondeur maximale de **50**, avec un score moyen de validation croisée de **98,7 %**. Ce résultat a confirmé les tests manuels réalisés précédemment.

En conclusion

- Le modèle final montre un bon équilibre entre complexité et performance.
- L'arbre de décision est lisible et permet d'expliquer les prédictions.
- Les performances sur les données de test montrent que le modèle généralise bien.
- Le surapprentissage initial a été maîtrisé grâce à la réduction de la profondeur et à l'optimisation des hyperparamètres.

4. Sauvegarde des fichiers pour mon chatbot

J'ai sauvegardé :

- Le modèle entraîné : 'model.pkl'
- L'encodeur de labels : 'label_encoder.pkl'
- Les colonnes des symptômes : "symptom_columns.pkl"

5. Constitution du Chatbot

- Interface construite avec Streamlit.
- Un arrangement de l'interface avec CSS.

6. Perspectives de développement

- Conseils médicaux ou liens vers sources fiables
- Dialogue conversationnel (avec LLM)
- Meilleure gestion multilingue
- Apprentissage sur de vrais cas médicaux



Avertissement : le model est entrainé sur un jeu téléchargé pour la formation, et ne remplace pas un diagnostic médical réel.

Remerciements à toute la direction de l'école **Gomycode - Algeria** qui offre un cadre de formation accueillants, agréable, et convivial.

Et un remerciement particulier à **Ikram AISSIOU** formatrice de ce bootcamp data-science, une enseignante compétente dynamique et dévouée.