Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes - RABAT

Rapport de PFA : Healthcare Chatbot design

Réalisé par :

El Mehdi Oudaoud
Fatima-Ezzahra Lahnine

Encadré par :

Pr. Jalil el hassouni

Année Scolaire 2020/2021

Résumé

Ce Projet propose un design de chatbot pour soins de santé qui répond aux questions les plus fréquemment posées et fait une vérification des symptômes entrées par les clients.

Ce bots permette les utilisateurs de décrire leurs symptômes et leur état de santé en posant une suite de questions. Le bot ensuit comprend les inputs de l'utilisateur en utilisant le traitement du langage naturel (NLP), quelle que soit la variation de l'entrée. Cela est essentiel pour obtenir des réponses précises.

Le bot est intégré dans une application web qui permet l'utilisateur de préciser une partie du corps concernant sa demande, et liste les différents symptômes liés à cette partie. Il peut choisir une des réponses listées qui va être transmise au bot, puis ce dernier répond le client par la description du symptôme choisi et lui donne quelque précautions et conseils à prendre.

Abstract

This project proposes a healthcare chatbot design that answers frequently asked questions and verifies the symptoms entered by customers.

The bot allows users to describe their symptoms and conditions by asking a series of questions. The bot then understands the user's input using natural language processing (NLP), regardless of the variation in the input. This is essential to get accurate answers.

The bot is embedded in a web application that allows the user to specify a body part regarding his query, and lists the different symptoms related to that part. He can choose one of the listed answers which will be transmitted to the bot, then the latter responds to the customer with a description of the chosen symptom and gives him some precautions and advice to take.

Table des matières

Introduction

Les chatbots sont considérés comme une alternative plus fiable et plus précise aux recherches en ligne que les patients effectuent lorsqu'ils essaient de comprendre la cause de leurs symptômes. Les prestataires de soins de santé pensent que les chatbots pourraient aider les patients qui ne sont pas sûrs de savoir où ils doivent aller pour recevoir des soins.

Les prestataires de soins de santé mettent actuellement en œuvre des bots qui permettent aux utilisateurs de vérifier leurs symptômes et de comprendre leur état de santé depuis le confort de leur domicile. Les chatbots qui utilisent le traitement du langage naturel (NLP) peuvent comprendre les demandes des patients, quelle que soit la variation de l'entrée. Cela est essentiel pour atteindre une grande précision dans les réponses, ce qui est indispensable pour les vérificateurs de symptômes.

Grâce à la connaissance de l'entrée, le bot peut évaluer les informations et aider les utilisateurs à déterminer la cause de leurs symptômes. Avec toutes les données fournies par le robot, les utilisateurs peuvent déterminer si un traitement professionnel est nécessaire ou si des médicaments en vente libre sont suffisants.

Chapitre 1

Concepts et implémentation du NLP

Un framework de chatbot a besoin d'une structure dans laquelle les intentions de conversation sont définies. Un moyen propre de le faire est d'utiliser un fichier JSON.

Nous allons créer un cadre de chatbot et construire un modèle conversationnel pour un domaine de soins de santé. Le chatbot doit répondre à des questions simples sur les symptômes, les maladies, etc.

Nous allons travailler en 3 étapes :

Nous allons transformer les définitions de l'intention conversationnelle en un modèle Pytorch.

Ensuite, nous allons construire un cadre de chatbot pour traiter les réponses.

Enfin, nous montrerons comment le contexte de base peut être incorporé dans notre processeur de réponses.

Nous utiliserons tflearn, une couche au-dessus de Pytorch, et bien sûr Python.

1.1 Créer des données d'entraînement

Nous avons créé des données d'entraînement dans un fichier Json (intents.json). Il a la structure suivante :

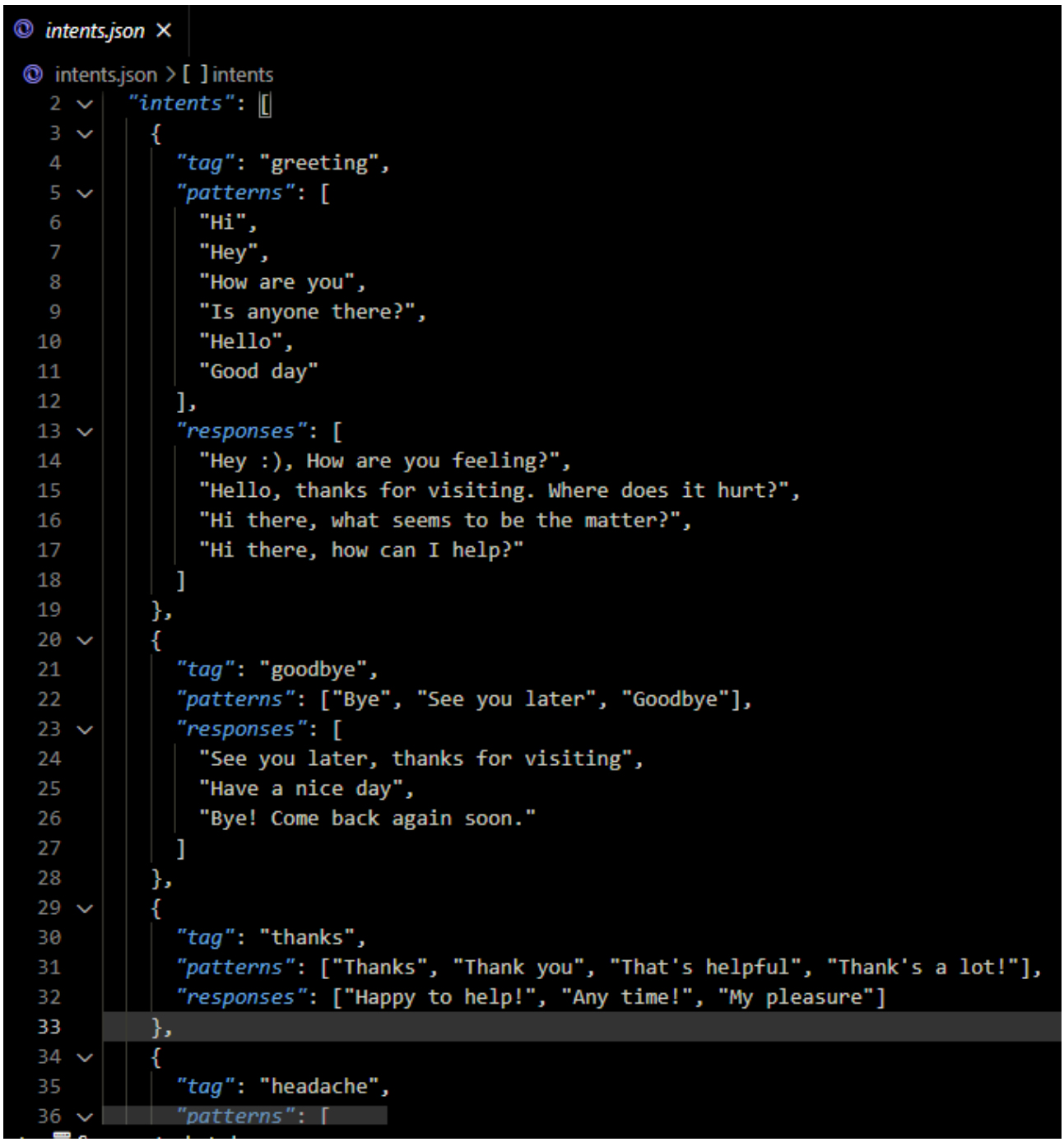

FIGURE 1.1 – Chatbot intents

1.2 Principes de base du NLP

Nous ne pouvons pas simplement transmettre la phrase d'entrée telle quelle à notre réseau neuronal. Nous devons d'une manière ou d'une autre convertir les chaînes de motifs en nombres que le réseau peut comprendre. Pour cela, nous convertissons chaque phrase en un bag of words (bow). Pour ce faire, nous devons collecter des mots d'entraînement, c'est-à-dire tous les mots que notre bot peut examiner dans les données d'entraînement. Sur la base de tous ces mots, nous pouvons ensuite calculer le sac de mots pour chaque nouvelle phrase. Un sac de mots a la même taille que le tableau de tous les mots, et chaque position contient un 1 si le mot est disponible dans la phrase entrante, ou 0 sinon. Voici un exemple visuel :

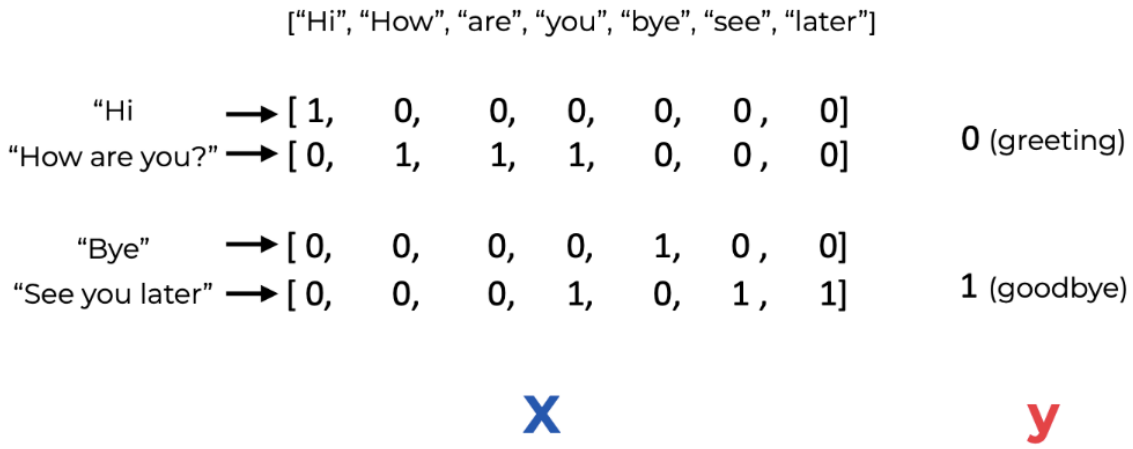

FIGURE 1.2 – Training data bag of words

Avant de pouvoir calculer l'arc, nous appliquons deux autres techniques NLP : la tokenisation et la séparation des mots.

— **Tokenisation :** Découpage d'une chaîne de caractères en unités significatives (par exemple, des mots, des caractères de ponctuation, des chiffres).

— **Stemming :** Un processus de normalisation linguistique, qui réduit les mots à leur racine ou supprime les affixes de dérivation. Par exemple, connexion, connecté, mot de connexion se réduisent à un mot commun "connecter".

Pour nos étiquettes, nous les trions par ordre alphabétique, puis nous utilisons l'index comme label de classe. L'ensemble de notre pipeline de traitement initial ressemble à ceci :

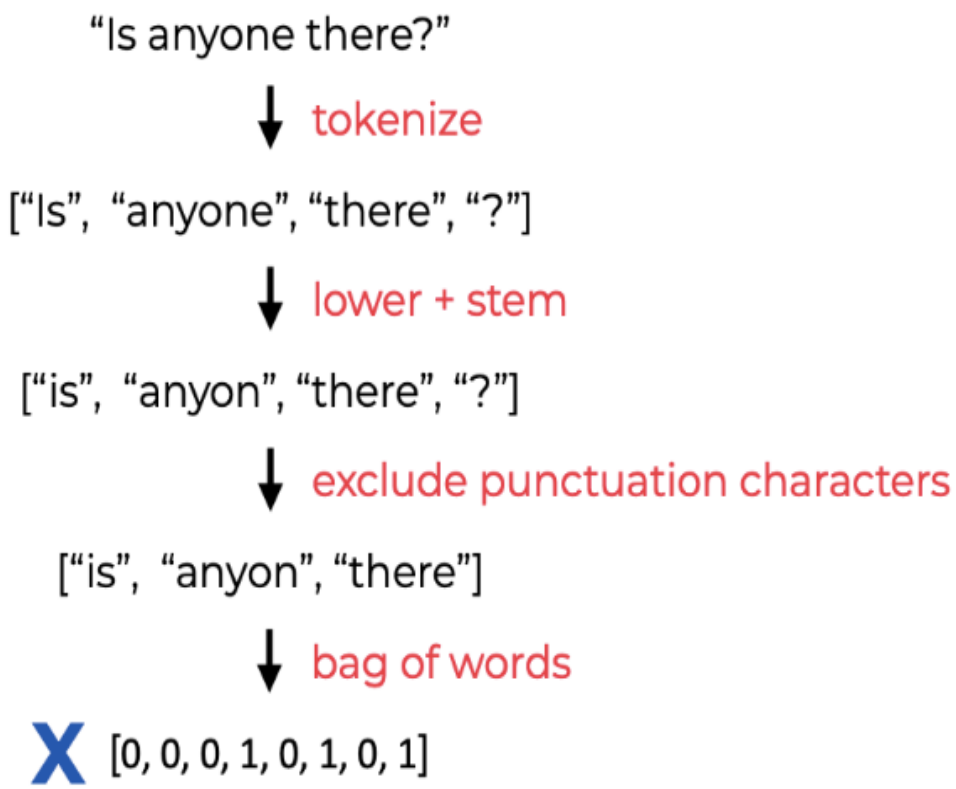

FIGURE 1.3 – NLP preprocessing pipeline

1.3 Implémenter les outils du NLP

Pour cela, nous utilisons le module nltk. NLTK (Natural Language Toolkit) est une plateforme de pointe pour la création de programmes Python destinés à travailler avec des données sur le langage humain.

C'est un package Python puissant qui fournit un ensemble d'algorithmes divers pour les langues naturelles. Il est gratuit, opensource, facile à utiliser, avec une grande communauté et bien documenté. NLTK comprend les algorithmes les plus courants tels que la tokenisation, l'étiquetage des parties du langage, la formation de la racine, l'analyse des sentiments, la segmentation des sujets et la reconnaissance des entités nommées. NLTK aide l'ordinateur à analyser, prétraiter et comprendre le texte écrit.

nltk_utils.py > ...

```python
import numpy as np
import nltk
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def tokenize(sentence):

    return nltk.word_tokenize(sentence)

def stem(word):

    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, words):

    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag

```

FIGURE 1.4 – Implémentation des outils NLP

1.4 Implémentation du Neural Network

L'implémentation est simple avec un réseau neuronal Feed Forward à deux couches cachées :

model.py > NeuralNet > forward

```python
import torch
import torch.nn as nn


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

FIGURE 1.5 – Implémentation du Neural Network

1.5 Implémentation du Training Pipeline

Dans cette partie, on va expliquer le chargement du modèle formé et les prédictions pour les nouvelles phrases.

On commence par charger notre training data qui existe dans le fichier **intents.json**, puis on passe au traitement de chaque phrase du fichier, en appliquant les méthodes du fichier **nltk_utils.py** : Tokenizing et Stemming :

train.py > ...

```python

with open('intents.json', 'r') as f:
    intents = json.load(f)

all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))

# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)
```

FIGURE 1.6 – Chargement et traitement du training data

Ensuit, on crée notre training data d'une manière à avoir les deux vecteurs X et Y, pour avoir à la fin notre Bag of Words list :

```python
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)
```

FIGURE 1.7 – Création du bag of words

Enfin, on applique le training model au dataset :

```python
dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = NeuralNet(input_size, hidden_size, output_size).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

FIGURE 1.8 – Application du training model

Vers la fin du processus du trainig, on sauvegarde notre data résultante dans un fichier **data.pth**.

1.6 Implémentation du Chat

Dans cette partie, on va charger le modèle entraîné et faites des prédictions pour les nouvelles phrases.
On commence d'abord par récupérer les données du fichier **data.pth**.

La fonction chatbot_response prend comme paramètre la phrase entrée par l'utilisateur du chatbot et applique les outils du nltk, Tokenizing et Stemming. Elle calcule ensuite la probabilité d'avoir un élément existant parmi les tags de notre training data, et fait un test pour savoir si le tag du pattern entré par l'utilisateur correspond à un tag dans data.pth. Enfin elle renvois une réponse selon le résultat du test.

```python
def chatbot_response(sentence):

    #if sentence == "results":
        #res = listToString(what_are_your_symptoms(filter_disease(intent_tags_list)))

    sentence = tokenize(sentence)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    print("\n *************")
    print(sentence)
    print("\n *************")

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]
    print(tag)

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                res = random.choice(intent['responses'])
                intent_tags_list.append(tag)
    elif "results" in sentence:
        dic= test.what_are_your_symptoms(filter_disease(intent_tags_list))
        res=dic
        print("\n ****/////***")
        print(dic)
        print("\n ***/////****")
    else:
```

FIGURE 1.9 – Génération des réponses du chatbot

Lorsque l'utilisateur envoie "results" comme phrase, il reçoit comme réponse le symptôme le plus proche aux description donnés. La fonction what_are_your_symptoms prend en paramètre la liste de symptômes entrés par l'utilisateur, fait un calcule de probabilité d'avoir une des maladies citées dans le fichier **disease.json** en ce basant sur les symptômes de chaque maladies, et retourne à la fin un dictionnaire qui contient une clé indiquant la maladie et sa valeur, et d'une autre clé qui indique la probabilité et sa valeur.

test.py > what_are_your_symptoms

```python
def what_are_your_symptoms(patient_symptoms):
    print(patient_symptoms)
    """patient_symptoms=[
      "pain chest",
      "shortness of breath",
      "dizziness",
    ]"""

    potential_diagnosis = []#[[desease,percent]]
    proba = []#these two are parrallel lists
    with open('disease.json','r') as f:
        diseases = json.load(f)
        for patient_symptom in patient_symptoms:
            for disease in diseases:
                if patient_symptom in diseases[disease]['symptoms']:
                    total_symp = len(diseases[disease]['symptoms'])
                    print("symptoms",diseases[disease]['symptoms'])
                    if diseases[disease]['disease'] in potential_diagnosis:
                        i = potential_diagnosis.index(diseases[disease]['disease'])
                        proba[i] += 1/total_symp
                    else:
                        potential_diagnosis.append(diseases[disease]['disease'])
                        proba.append(1/total_symp)

        if proba == []:
            j = 0
            potential_diagnosis.append(["you are maybe exausted",])
            mx = "?"
        else:
            mx = max(proba)
            mx = "{:.4f}".format(mx)
            j = proba.index(mx)
        #print(potential_diagnosis)
    return {"illness":potential_diagnosis[j][0],"probability":mx}
```

FIGURE 1.10 – Le résultat final du conseil

Bibliographie

[1] Pytorch documentation. <https://pytorch.org/docs/stable/index.html>.

[2] NLTK 3.6.2 documentation. <https://www.nltk.org/>.

[3] gk. Contextual chatbots with tensorflow. <https://chatbotsmagazine.com/contextual-chat-bots-with-tensorflow-4391749d0077>.