ChatBot con IBM Watson y Neo4j

Nicolas Añazco, Jordan Murillo
Sistemas, Universidad Politécnica Salesiana, UPS
Cuenca, Ecuador
jmurillov1@est.ups.edu.ec
nanazco@est.ups.edu.ec

In this work we will implement one of the artificial intelligence technologies, in this case it will be chatbots and personal assistants for recommending products or services for users.

***Index Terms*—Neo4j, databases, algorithms, chatbots, IBM Watson, servicios web.**

I. Introducción

ENtre las multiples funcionalidades de la inteligencia artificial se encuentran los famosos chatbots, que son bots especializados y fueron creados para mantener una conversación y ofrecer múltiples respuestas preconcebidas. Parece meridianamente claro que el nivel de servicio y la calidad de este pueden ser grandes beneficios intangibles, pero detrás de todo esto esta simple reflexión pueden estar ocultos otros como la capacidad de obtener bases de datos cualificadas y enriquecidas a través de análisis de la psicográfica de sus usuarios que de paso nos ofrece la posibilidad de abrir la puerta del big data como herramienta.

II. Desarrollo

A. Metodología

Para el desarrollo de la investigación se hizo uso de IBM Watson Assistant (Dialogo de Chat) y la base de datos Neo4j conjuntamente con la documentación que esta provee, en relación a los aspectos principales para almacenamiento de datos y uso de los algoritmos que esta posee. Ademas de contar con acceso a una maquina con Neo4j instalado y configurado. Para hacer uso de esta herramienta eligimos previamente dos algoritmos, Para la informacion a recomendar y surgerir seleccionamos comida rapida y como restaurant es McDonals donde ofrece diversos menus para sus clientes. Luego de obtener datos como son las comidas que ofrece este restaurant, los colocamos en un archivo de excel para su posterior guardado en la base de datos. Para la toma de datos se hara uso de Google Maps, donde se selecciona la ubicacion de los respectivos restaurantes en 4 ciudades del Ecuador.

Fig. 1. Tomado de Datos con Google Maps

1) Descripción detallada del problema.

El problema que vamos a resolver es la poca disponibilidad de tiempo, entonces lo que haremos es con facilidad de que una tienda pueda ofrecer sus servicios o bienas con eso se va a presentar nuestro chatbot para recomendar o surgerir los menus de comidas rapidas que se ofrece y asi mismo el cliente puede hacer cualquier consulta con respecto al restaurant.

2) Propuesta de Solución

Para el desarrollo de la propuesta de solución lo que hicimos es usar Telegram con Watson Assistant con Python haciendo de intermediario para hacer uso de otras API's de Watson sin ningun problema, guardando como respuestas posibles los menus de las comidas que se ofrece y asi mismo cada una de las ubicaciones de los locales en las respectivas ciudades.

El esquema de la aplicación es algo sencillo, ya que la interfaz esta dada por las aplicaciones de mensajería instantanea: Telegram y WhatsApp.

A continuación se muestra un prototipo de la parte gráfica de el sistema.

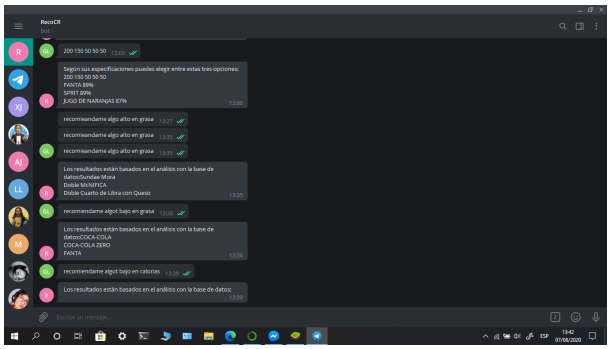

Fig. 2. Solución - Telegram

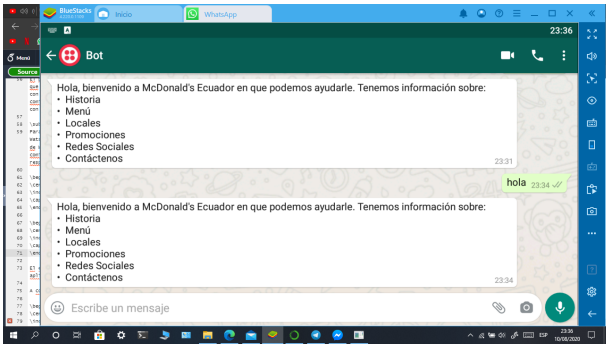

Fig. 3. Solución - WhatsApp

Ahora se presentará el diagrama esquemático de la solución.

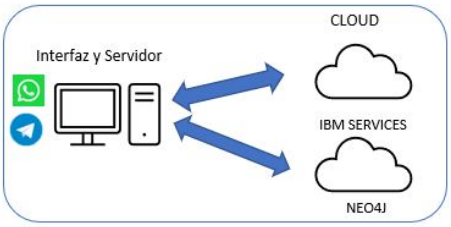

Fig. 4. Diagrama Esquemático

Ahora vamos a presentar la arquitectura del programa desarrollado, para ello podemos fijarnos en la siguiente imagen.

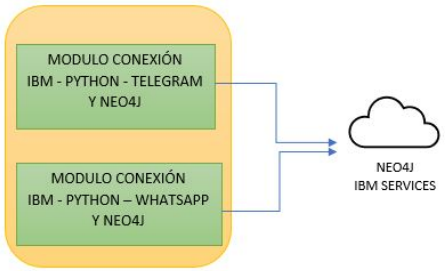

Fig. 5. Arquitectura

3) Descripción de la Solución

Para diseñar la solución, primero estudiamos y entendimos los algoritmos de Neo4j a ser usados así como los servicios de IBM usados, los cuáles explicaremos a continuación. Necesitamos algunas librerías para que esto funcione que son: neo4j, json, ibm-watson, ibm-cloud-sdk-core, io.

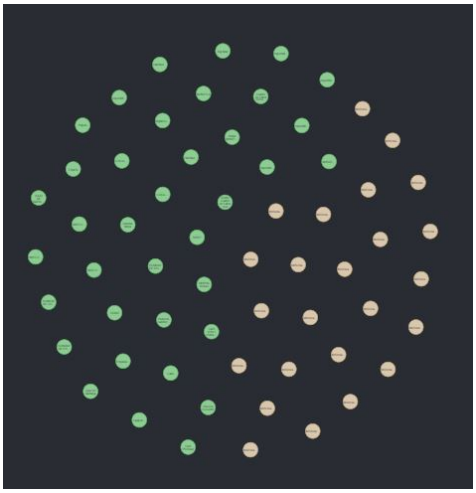

Fig. 6. Datos en Neo4j

A. Algoritmos de Similitud

Para es tipo de algoritmos elegimos al de la Similitud de Coseno, este es el coseno del ángulo entre dos vectores de n dimensiones en un espacio de n dimensiones. Es el producto escalar de los dos vectores dividido por el producto de las longitudes (o magnitudes) de los dos vectores.

```
def get_ed2(tx,vector):
    comunidad = []
    result = tx.run("MATCH (n:comida) "
                    "RETURN n.nombre as Comida, "
                    "gds.alpha.similarity.cosine(n.indoNutricional, $vector) "
                    "AS similarity order by similarity desc limit 3",vector=vector)
    for record in result:
        r1=Resultado(record["Comida"],record["similarity"])
        comunidad.append(r1)
    return comunidad
def getComidas(vector):
    text =""
    uri = "bolt://35.239.4.175:7687"
    driver = GraphDatabase.driver(uri, auth=("neo4j", "neo4j"),encrypted=False)
    with driver.session() as session:
        comunidad= session.read_transaction(get_ed2,vector)
        for i in range(len(comunidad)):
            text += str(comunidad[i].nombre)+" "+str(int(round(comunidad[i].similitud,2)*100))+"% \n"
            #print(comunidad[i].labels)
            #text+=comunidad[i]["nombre"]+" "+comunidad[i]["correo"]+"\n"
            #print(comunidad[i]["name"],"  -> ",comunidad[i]["apellido"],"  -> ", comunidad[i]["edad"])
    driver.close()
    return text
```

Fig. 7. Código Similitud de Coseno en Python

B. Algoritmos de Predicción de Enlaces

Para es tipo de algoritmos elegimos al de Misma Comunidad este compara a través de un característica si dos nodos en este caso pertenecen a la misma comunidad, es decir, la misma comunidad es una forma de determinar si dos nodos tienen una misma característica.

```
def get_ed(tx,ciudad):
    comunidad = []
    result = tx.run("MATCH (l1:Local {lugar:$ciudad}) "
                    "MATCH (l2:Local) WHERE l1<>l2 "
                    "RETURN l2.nombre as Nombre, l2.ubicacion as Ubicacion, gds.alpha.linkprediction.sameCommu
                    "AS Score ORDER BY Score DESC",ciudad=ciudad)
    for record in result:
        p1=Local(record["Nombre"],record["Ubicacion"])
        comunidad.append(p1)
    return comunidad

def getLocales(ciudad):
    text =""
    uri = "bolt://35.239.4.175:7687"
    driver = GraphDatabase.driver(uri, auth=("neo4j", "neo4j"),encrypted=False)
    with driver.session() as session:
        comunidad= session.read_transaction(get_ed,ciudad)
        for i in range(len(comunidad)):
            text += comunidad[i].nombre+" "+comunidad[i].ubicacion+" \n"
            #print(comunidad[i].labels)
            #text+=comunidad[i]["nombre"]+" "+comunidad[i]["correo"]+"\n"
            #print(comunidad[i]["name"],"  -> ",comunidad[i]["apellido"],"  -> ", comunidad[i]["edad"])
    driver.close()
    return text
```

Fig. 8. Código de Misma Comunidad en Python

Bueno esas son las descripciones de los algoritmos que elegimos, ahora presentaremos los servicios de IBM Watson utilizados.

C. WATSON ASSISTANT

Este es el servicio de IBM Watson que se encarga de ayudarte a generar un chatbot adecuadamente de forma muy

sencilla. El servicio combina aprendizaje automático, comprensión del lenguaje natural y un editor de diálogos integrado para crear flujos de conversación entre las aplicaciones y los usuarios.

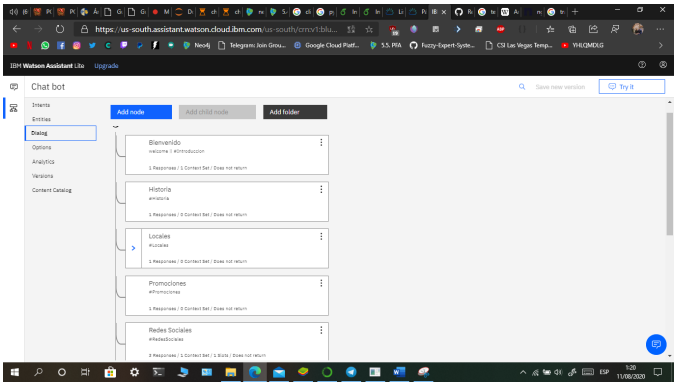

Fig. 9. Diálogo en Watson Assistant

D. LANGUAGE TRANSLATOR

Este servicio lo implementamos para hacer detecciones y traducciones de idioma de una forma fácil y rápida. El servicio traduce texto de un idioma a otro. El servicio ofrece varios modelos de traducción proporcionados por IBM que puede personalizar en función de su terminología y lenguaje únicos. Utilice Language Translator para tomar noticias de todo el mundo y presentarla en su idioma, comunicarse con sus clientes en su propio idioma y mucho más.

```
authenticator = IAMAuthenticator('vh0j48-6QDFVQy6-qehepHiOX_8Dc8_3anwEFFJ-7CRr')
language_translator = LanguageTranslatorV3(
    version='2018-05-01',
    authenticator=authenticator
)

language_translator.set_service_url('https://api.us-south.language-translator.watson.cloud.ib
```

Fig. 10. Código en Python

E. TEXT TO SPEECH

Este servicio lo implementamos para crear archivos de audio con el texto que le enviaremos. El servicio proporciona APIs que utilizan las capacidades de síntesis de voz de IBM para sintetizar texto en voz de sonido natural en una variedad de lenguajes, dialectos y voces. El servicio admite al menos una voz masculina o femenina, a veces ambas, para cada idioma. El audio se transmite de nuevo al cliente con un retardo mínimo.

```
authenticator = IAMAuthenticator('j0kSWqlXptEXfuGqkuvZFLSxmMvfV7aHMYiAf7uze0x4')
text_to_speech = TextToSpeechV1(
    authenticator=authenticator
)

text_to_speech.set_service_url('https://api.us-south.text-to-speech.watson.cloud.ibm.com/instances/7c2c7d18-8cc8-407f-b871-
```

Fig. 11. Código en Python

Los pasos para implementar estos servicios son muy sencillos, y se aprenden fácilmente, ya que Python es muy sencillo de trabajar.

Para las conexion con Telegram hicimos con el creador de bots para Telegram que es BotFather, y nos devuelve lo siguiente.

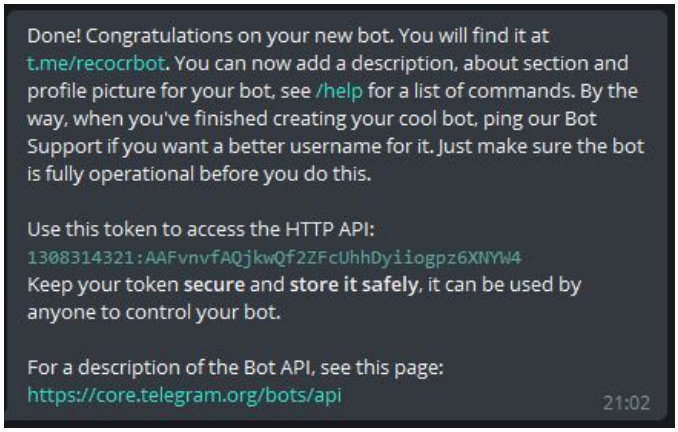

Fig. 12. Mensaje de creción de nuevo bot con BotFather

Para Python se usa las librerías json y requests que ayudan a comunicar con Telegram. A continuación presento como queda la conexión:

```
#Importar librerias
import json
import requests

#Variables para el Token y la URL del chatbot
TOKEN = "1308314321:AAFvnvfAQjkwQf2ZFcUhhDyiiogpz6XNYW4" #Cambialo por tu token
URL = "https://api.telegram.org/bot" + TOKEN + "/"
```

```
def update(offset):
    #Llamar al metodo getUpdates del bot, utilizando un offset
    respuesta = requests.get(URL + "getUpdates" + "?offset=" + str(offset))
    #Telegram devolvera todos los mensajes con id IGUAL o SUPERIOR al offset

    #Decodificar la respuesta recibida a formato UTF8
    mensajes_js = respuesta.content.decode("utf8")

    #Convertir el string de JSON a un diccionario de Python
    mensajes_diccionario = json.loads(mensajes_js)

    #Devolver este diccionario
    return mensajes_diccionario
```

Fig. 13. Conexión de Python con Telegram

Recordar que para esto se necesita un token de seguridad que permite la conexión, el cuál es enviado por el BotFather de Telegram. Por consiguiente presento el método de envio de mensajes:

```
import io
def enviar_mensaje(idchat, texto):
    #Llamar el metodo sendMessage del bot, passando el texto y la id del chat
    requests.get(URL + "sendMessage?text=" + texto + "&chat_id=" + str(idchat))

#Variable para almacenar la ID del ultimo mensaje procesado
ultima_id = 0

def enviar_foto(idchat,img_url):
    url = URL + "sendPhoto"
    remote_image = requests.get(img_url)
    photo = io.BytesIO(remote_image.content)
    photo.name = 'img.png'
    files = {'photo': photo}
    data = {'chat_id' : idchat}
    r= requests.post(url, files=files, data=data)

def enviar_audio(idchat,texto):
    voz(texto)
    url = URL + "sendAudio"
    files={'audio': ("audio.ogg", open('audio.ogg', 'rb'))}
    data = {'chat_id' : idchat, 'caption': "Audio"}
    requests.post(url, files=files, data=data)
```

Fig. 14. Envío de Mensajes, Audio e Imágenes

Aquí el método de selección de mensaje a enviar:

```
while(True):
    mensajes_diccionario = update(ultima_id)
    for i in mensajes_diccionario["result"]:
        #Llamar a la funcion "Leer_mensaje()"
        idchat, nombre, texto, id_update = leer_mensaje(i)
        #Si la ID del mensaje es mayor que el ultimo, se guarda la ID + 1
        if id_update > (ultima_id-1):
            ultima_id = id_update + 1
        #Generar una respuesta a partir de la informacion del mensaje
        print(texto)
        try:
            texto_respuesta=sendMessageIBM(texto)
        except KeyError:
            texto_respuesta="No le entendido"
        if(texto_respuesta == "Estos son los locales para"):
            texto_respuesta+= " " +texto+"\n"
            texto_respuesta+= getLocales(texto)
        if(texto_respuesta == "Datos de"):
            texto_respuesta+= " " +texto+"\n"
            texto_respuesta+= getComis(texto)
        if(texto_respuesta== "Según sus especificaciones puedes elegir entre estas tres opciones:"):
            texto_respuesta+= " " +texto+"\n"
            ar=texto.split(" ")
            ar=[float(i) for i in ar]
            texto_respuesta+= getComidas(ar)
        if(texto_respuesta=="Los resultados están basados en el análisis con la base de datos:"):
            print(texto)
            if("grasa" in texto and "bajo" in texto):
                texto_respuesta+= "\n"+getRecomendacion(4,"ASC")
            if("grasa" in texto and "alto" in texto):
                texto_respuesta+= "\n"+getRecomendacion(4,"DESC")
            if("calorias" in texto and "bajo" in texto):
                texto_respuesta+= "\n"+getRecomendacion(0,"ASC")
            if("calorias" in texto and "alto" in texto):
                texto_respuesta+= "\n"+getRecomendacion(0,"DESC")
            if("carbohidratos" in texto and "bajo" in texto):
```

Fig. 15. Método de selección de mensajes de respuesta

Para las conexión con WhatsApp hicimos con un servicio para trabajar con WhatsApp que se llama Twilio.

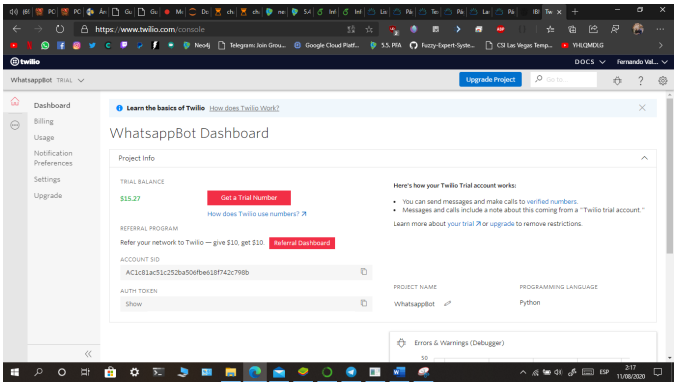

Fig. 16. Consola de Twilio

Así mismo que BotFather este servicio nos da un token que debemos colocar en Python para conectarnos entre los sistemas.

Con esto luego haremos uso de Flask para usar servicios web y de Ngrok que nos permite publicar estos servicios en la nube y de esta forma ser accedidos para obtener la comunicación deseada.

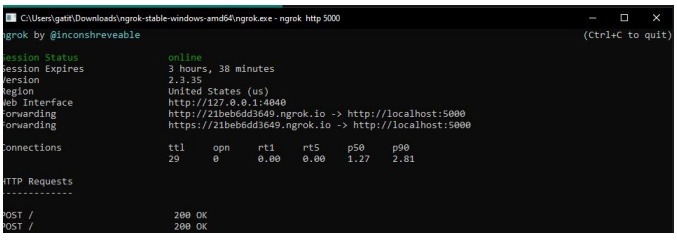

Fig. 17. Ngrok en puerto http 5000 de Flask

Ahora el código en python consiste en una clase que habilite un servicio web para POST y GET que permiten que se envie y se reciban los mensajes de WhatsApp. Lo presentamos a continuación:

```
#!pip install twilio
from twilio.rest import Client
from flask import Flask, request
import json

app = Flask(__name__)
account_sid = 'AC1c81ac51c252ba506fbe618f742c798b'
auth_token = 'd4a3a3f6aa632452db92aa0faefcb3c4'
client = Client(account_sid, auth_token)

def enviar(mensaje):
    message = client.messages.create(
                                body=mensaje,
                                from_='whatsapp:+14155238886',
                                to='whatsapp:+593988815578'
                            )
    return message


def enviar_foto(url):
    message = client.messages.create(
                                body="Foto",
                                media_url=url,
                                from_='whatsapp:+14155238886',
                                to='whatsapp:+593988815578'
                            )
    return message


@app.route("/", methods=['GET', 'POST'])
def receive_message():
    tt=''
    if request.method == 'GET':
        print('Inicio')
        mensaje = 'Mensaje Inicio APP'
        message = enviar(mensaje)
        tt = message.sid
    else:
        num = request.form['From']
```

Fig. 18. Servicio Web para POST y GET

Luego de esto pasaremos al apartado de resultados obtenidos.

4) Resultado de análisis obtenidos

En base a lo obtenido podemos determinar que el uso de Neo4j simplifica bastante los procesos de desarrollo de algoritmos desde cero y conjuntamente con los servicios de IBM Watson podemos crear un chatbot de forma mas sencilla y con funcionalidades optimas.

Ahora veremos al chatbot en ejecución:

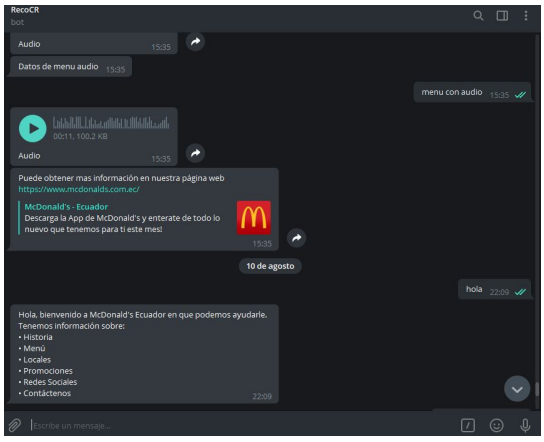

Fig. 19. Ejemplo del CHATBOT Telegram

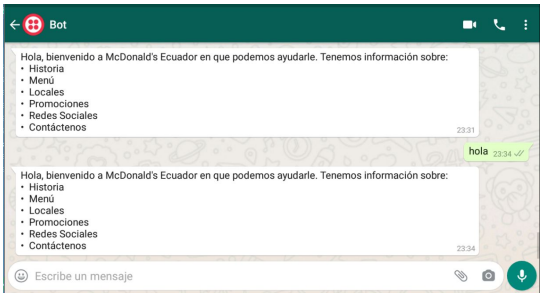

Fig. 20. Ejemplo del CHATBOT WhatsApp

III. Opinión y Recomendaciones

Es recomendable hacerle un buen entrenamiento al chatbot para que asi aprenda y pueda ser un buen asistente y pueda ayudar a los clientes a completar tareas y obtener informacion que buscan.

El chatbot debe devolver informacion entendible para el cliente que lo esta utilizando de acuerdo a los mensajes que escriba en el mismo.

El uso de chatbots ayuda mucho a las empresas y mejora la forma de atender a los clientes y esta disponible 24/7.

References

[1] BD Neoj4. *The Neo4j Graph Data Science Library Manual v1.2-preview* 2020. Disponible en: https://neo4j.com/docs/graph-data-science/1.2-preview/

[2] BD Neo4j *Using Neo4j from Java* 2020. Disponible en: https://neo4j.com/developer/java/

[3] IBM Watson *Text to Speech - IBM Cloud API Docs* 2020. Disponible en: https://cloud.ibm.com/apidocs/text-to-speech?code=python

[4] IBM Watson *Watson Assitant V2 - IBM Cloud API Docs* 2020. Disponible en: https://cloud.ibm.com/apidocs/assistant/assistant-v2?code=python

[5] IBM Watson *Language Translator - IBM Cloud API Docs* 2020. Disponible en: https://cloud.ibm.com/apidocs/language-translator?code=python