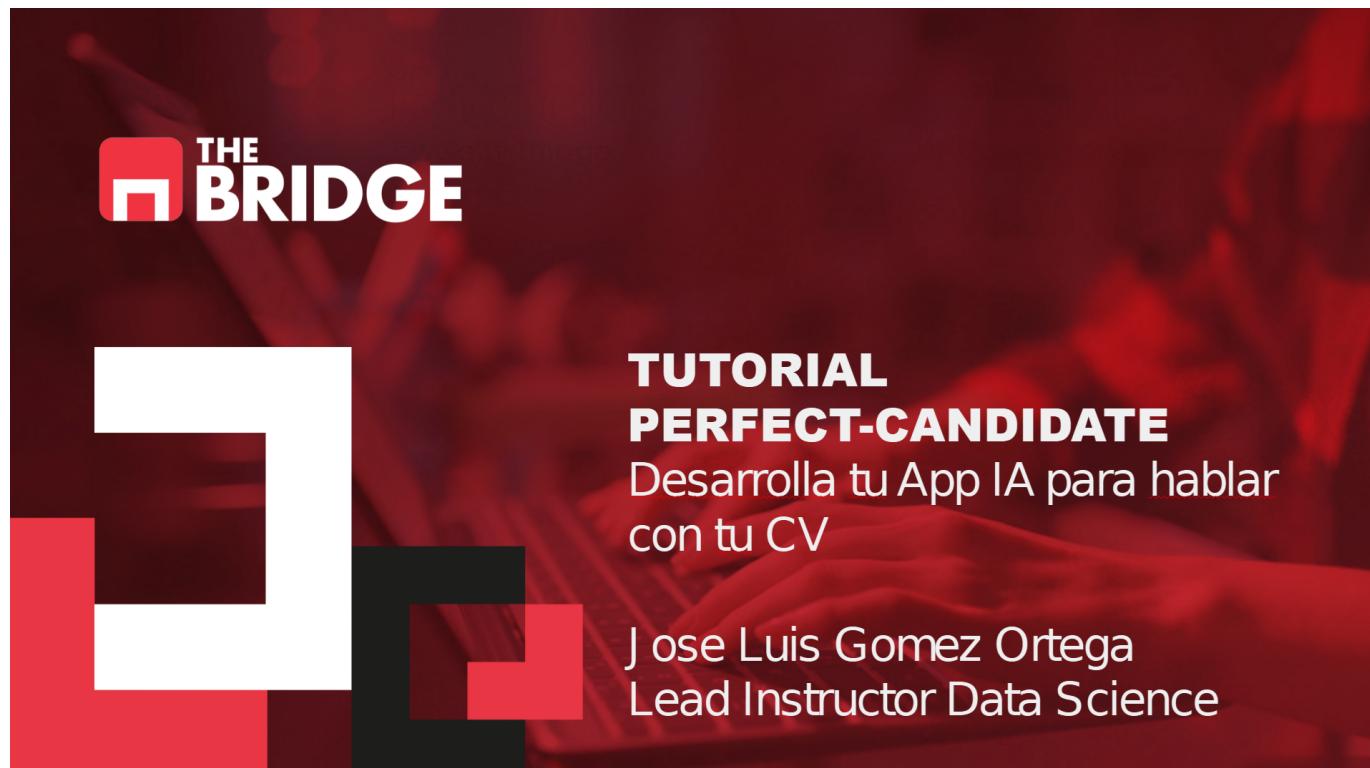


THE BRIDGE

Creando una Aplicación de Asistente de Reclutamiento con Streamlit Perfect-Candidate



Introducción

¡Hola! Os propongo la creación de un asistente de proceso de selección virtual AI powered utilizando **Streamlit**. Esta aplicación está diseñada para ayudar a los reclutadores a evaluar candidatos de manera más eficiente, permitiendo que interactúen directamente con la información del CV y otros documentos de los candidatos a través de un chatbot. Pero, en realidad, esta aplicación está pensada para, no solo enseñar tu CV sino mostrar tus habilidades de programación lo que te puede hacer destacar entre otros candidatos al mostrar tu capacidad para crear herramientas innovadoras. Este tutorial está diseñado con un perfil de Data Science en mente. Pero no se explora ningún concepto especialmente complejo en profundidad, por lo que otros perfiles pueden encontrarlo interesante.

Motivación

Imagina que acabas de terminar tu bootcamp y estás en búsqueda de tu primer trabajo como científico de datos. Has enviado tu CV a varios reclutadores, pero ¿qué más puedes hacer para destacar? Aquí es donde entra nuestra aplicación. Al crear esta herramienta, no solo proporcionas tu CV, sino que también demuestras tu capacidad para trabajar con tecnologías avanzadas como LLMs y en concreto usando la API de **Groq**. Esto puede llamar la atención de los reclutadores/entrevistadores y aumentar tus posibilidades de ser contratado.

¿Qué es Streamlit?

Streamlit es una herramienta con mucho potencial (que quizás ya conozcas!) que permite a los científicos de datos y otros desarrolladores crear aplicaciones web de manera rápida y sencilla utilizando solo Python. No necesitas conocimientos avanzados de desarrollo web; con Streamlit, puedes crear interfaces de usuario interactivas en cuestión de minutos.

¿Prompting Engineering y los Modelos de Groq?

- **Prompting:** Los LLMs como ChatGPT o Gemini sufren cuando no disponen la información que se les pide en su conjunto de entrenamiento inicial. Esto hace que en el mejor de los casos, el modelo responda que no lo sabe o, peor aún, se inventa la respuesta. Hay muchas formas de evitar esto. Nosotros aquí proponemos la más sencilla que es especificarle un rol al modelo de lo que tiene que hacer y en qué caso de uso (AI assistant para un proceso de selección) e incluir en el prompt todos los documentos aportados por el usuario como contexto.
- **Modelos de Groq:** Para demostrar que hay vida más allá de OpenAI y la curiosidad inherente de un ADN bootcamp, probamos con otro proveedor de LLMs, en este caso Groq. Groq proporciona modelos avanzados de procesamiento de lenguaje natural (NLP) que podemos usar para generar respuestas inteligentes y coherentes a las preguntas que los reclutadores puedan tener sobre los candidatos. Para este ejemplo usaremos el nuevo, flamante (en Julio 2024, que esto en semanas se queda obsoleto!) y open source modelo de meta **Llama 3.1 70B** (larga vida a los modelos abiertos 🎉🎉🎉)

Encontrarás todo el código para la aplicación en este repositorio: <https://github.com/MA-Barracas/perfect-candidate>

Paso a Paso de la Creación de la Aplicación

Paso 1: Preparar el Entorno

Primero, necesitamos asegurarnos de tener todas las herramientas necesarias instaladas. Vamos a usar Python, así que asegúrate de tenerlo instalado. Luego, instala las siguientes librerías:

```
pip install streamlit pdfplumber docx2txt openai numpy groq dotenv
```

Paso 2: Configurar las Claves API

Vamos a utilizar los servicios de Groq, así que necesitas obtener tus claves API y configurarlas en un archivo **.env**. Este archivo debe ubicarse en la raíz de tu proyecto y debe contener lo siguiente:

```
GROQ_API_KEY=tu_clave_groq
```

SUPER HIPER MEGA IMPORTANTE: Nunca subas al repo el archivo .env o cualquier parte de código que contenga api keys, contraseñas etc... Si despliegas esta aplicación, bastará que en el propio servicio de despliegue (Streamlit, GCP, AWS...) especifiques la API KEY como variable de entorno.

Paso 3: Estructura del Código

Vamos a desglosar el código línea por línea para entender cómo funciona.

```
import streamlit as st
import pdfplumber
import docx2txt
import openai
import numpy as np
from groq import Groq
import os
from dotenv import load_dotenv
```

Aquí, estamos importando todas las librerías necesarias. **Streamlit** para la interfaz web, **pdfplumber** y **docx2txt** para procesar los documentos, **groq** para los modelos de NLP, y **dotenv** para manejar nuestras variables de entorno.

Paso 4: Cargar las Variables de Entorno

```
load_dotenv()
groq_client = Groq(api_key=os.environ.get("GROQ_API_KEY"))
```

Cargamos las claves API desde nuestro archivo **.env** y las usamos para inicializar el cliente de Groq.

Paso 5: Funciones de Utilidad

Parsear Archivos PDF

```
def parse_pdf(file):
    text = ""
    with pdfplumber.open(file) as pdf:
        for page in pdf.pages:
            text += page.extract_text()
    return text
```

Aquí, usamos **pdfplumber** para extraer texto de un archivo PDF.

Parsear Archivos DOCX

```
def parse_docx(file):
    text = docx2txt.process(file)
    return text
```

Similar a la función anterior, pero para archivos DOCX utilizando **docx2txt**.

Manejar Subida de Archivos

```
def handle_file_upload(file, file_type):
    if file_type == 'pdf':
        return parse_pdf(file)
    elif file_type == 'docx':
        return parse_docx(file)
    elif file_type == 'txt':
        return file.read().decode('utf-8')
    else:
        return ""
```

Esta función decide qué función de parsing usar basada en el tipo de archivo subido.

Paso 6: Obtener Respuestas del Modelo Groq

```
def get_groq_response(messages):
    completion = groq_client.chat.completions.create(
        model="llama-3.1-70b-versatile",
        messages=messages,
        temperature=0.5,
        max_tokens=400,
        top_p=1,
        stream=True,
        stop=None,
    )
    response = ""
    for chunk in completion:
        response += chunk.choices[0].delta.content or ""
    return response
```

Esta función envía un conjunto de mensajes al modelo de Groq y recibe una respuesta. El parámetro **temperature** controla la creatividad de las respuestas, y **max_tokens** define la longitud máxima de la respuesta.

Paso 7: Crear la Aplicación con Streamlit

Configurar la Página

```
def main():
    st.set_page_config(page_title="PerfectCandidate", page_icon=":rocket:",
    layout="wide")
```

Aquí configuramos el título, el icono y el layout de nuestra aplicación.

Título y Columna de Imagen

```
col1, col2 = st.columns((0.7,0.3))

with col1:
    st.title("🚀 Asistente PerfectCandidate! 🚀")
with col2:
    st.image("./img/bot.png")
```

Creamos dos columnas, una para el título y otra para una imagen decorativa.

Subida de Archivos

```
cv_file = st.sidebar.file_uploader("Upload your CV (mandatory)", type=['pdf',
'docx', 'txt'])
rec_letter_file = st.sidebar.file_uploader("Upload your Recommendation Letter
(optional)", type=['pdf', 'docx', 'txt'])
interests_file = st.sidebar.file_uploader("Upload your Personal Interests
Document (optional)", type=['pdf', 'docx', 'txt'])
job_desc_file = st.sidebar.file_uploader("Upload the Job Description
(optional)", type=['pdf', 'docx', 'txt'])
```

Usamos el sidebar de Streamlit para permitir la subida de varios tipos de archivos.

Procesar los Archivos Subidos

```
if cv_file:
    cv_text = handle_file_upload(cv_file, cv_file.name.split('.')[ -1])

else:
    st.error("***Se debe introducir al menos el CV del candidato***")
    return

rec_letter_text = handle_file_upload(rec_letter_file,
rec_letter_file.name.split('.')[ -1]) if rec_letter_file else "No info disponible"
interests_text = handle_file_upload(interests_file,
interests_file.name.split('.')[ -1]) if interests_file else "No info disponible"
job_desc_text = handle_file_upload(job_desc_file,
job_desc_file.name.split('.')[ -1]) if job_desc_file else "No info disponible"
```

Procesamos los archivos subidos y obtenemos los textos y embeddings necesarios.

Estado de la Sesión

```
if 'messages' not in st.session_state:  
    st.session_state['messages'] = []
```

Mantenemos el historial de conversación utilizando el estado de la sesión de Streamlit. Esto no permite ir almacenando toda la conversación paso a paso. Mediante esto, podemos tener una charla en la que nuestro bot "recuerda" todo lo que hemos dicho con anterioridad. Nota: No te preocupes que por mucho que se alargue la conversación, Llama 3.1 70B tiene una ventana de contexto de 128k.

Entrada de Chat y Mensajes

```
input_text = st.chat_input("Pregunta lo que quieras respecto al candidato:")  
if input_text:  
    user_message = {"role": "user", "content": input_text}  
    st.session_state.messages.append(user_message)  
  
    context = """Eres un experto asistente que ayudas a los reclutadores a ver  
el potencial  
        en el candidato del CV adjunto. Con la información aportada,  
        ayudarás a los reclutadores a entender las cualidades de los  
candidatos  
        y su potencial para el puesto en caso de conocer este. En caso  
de no conocer el rol  
        al que postula, simplemente se valorarán sus competencias para  
cualquier puesto en general.  
        Sé conciso, riguroso, equitativo y constructivo en tu  
feedback.  
        """  
  
    if cv_text:  
        context += "\n\nCV:\n" + cv_text  
    if rec_letter_text:  
        context += "\n\nLetra de recomendación:\n" + rec_letter_text  
    if interests_text:  
        context += "\n\nIntereses personales:\n" + interests_text  
    if job_desc_text:  
        context += "\n\nDescripción del puesto al que postula:\n" +  
job_desc_text  
  
messages_with_context = st.session_state.messages.copy()  
messages_with_context.insert(0, {"role": "system", "content": context})  
  
response = get_groq_response(messages_with_context)
```

```
    st.session_state.messages.append({"role": "assistant", "content":  
        response})
```

Aquí gestionamos la entrada del chat, construimos el contexto necesario y obtenemos la respuesta del modelo Groq. Además, incluimos la llamada y la respuesta a la memoria de la sesión para seguir "recordando".

Mostrar Conversación

```
for msg in st.session_state.messages:  
    if msg['role'] == 'user':  
        st.chat_message("user").markdown(msg['content'])  
    else:  
        st.chat_message("assistant").markdown(msg['content'])
```

Mostramos la conversación entre el usuario y el asistente.

Paso 8: Ejecutar la Aplicación

Finalmente, ejecutamos nuestra aplicación:

```
if __name__ == "__main__":  
    main()
```

¿Qué Esperamos que Suceda?

- **Subida de Archivos:** Los usuarios pueden subir sus CVs y otros documentos relevantes.
- **Procesamiento del Texto:** El texto de estos documentos es procesado y convertido en embeddings.
- **Interacción de Chat:** Los usuarios pueden hacer preguntas y recibir respuestas detalladas sobre los candidatos.
- **Evaluación del Candidato:** Los reclutadores pueden obtener una evaluación detallada del candidato basada en el CV y otros documentos.

Finalidad y Casos de Uso

Con la excusa de que el reclutador use la app para "hablar" con nuestro CV, tenemos la oportunidad de repasar nuestra info pero de una forma diferente, amena y que dice mucho de nuestra capacidad de generar aplicaciones basadas en IA para casos prácticos.

The screenshot shows a Streamlit application interface. On the left, there are four input fields for file uploads:

- Upload your CV (mandatory):** A placeholder says "Drag and drop file here Limit 200MB per file • PDF, DOCX, TXT". Below it is a "Browse files" button.
- Upload your Recommendation Letter (optional):** A placeholder says "Drag and drop file here Limit 200MB per file • PDF, DOCX, TXT". Below it is a "Browse files" button.
- Upload your Personal Interests Document (optional):** A placeholder says "Drag and drop file here Limit 200MB per file • PDF, DOCX, TXT". Below it is a "Browse files" button.
- Upload the Job Description (optional):** A placeholder says "Drag and drop file here Limit 200MB per file • PDF, DOCX, TXT". Below it is a "Browse files" button.

On the right, there is a central area with the title **Asistente PerfectCandidate!** accompanied by two small rocket icons. Below the title is a cartoon illustration of a white robot head wearing headphones and holding a blue speech bubble that says "HI!".

Underneath the title, there are several AI-generated responses in a list format:

- Icon: 🤖 como se llama el candidato
- Icon: 🤖 El candidato se llama Jesús Campos Ferrer.
- Icon: 🤖 es interesante este puesto para Jesús?
- Icon: 🤖 No parece que el puesto de Administrador sea muy interesante para Jesús Campos Ferrer, considerando su perfil profesional y experiencia. Jesús es un científico de datos con más de 3 años de experiencia en el diseño de soluciones estratégicas basadas en datos, y ha trabajado como Científico de Datos Sénior en Wavewood Corporación. Su experiencia y habilidades están enfocadas en el análisis de datos, la visualización de datos y el desarrollo de modelos de aprendizaje automático.

At the bottom, there is a text input field labeled "Pregunta lo que quieras respecto al candidato:" followed by a right-pointing arrow button.

Aquí ↑↑ puedes ver un ejemplo con unos archivos fake de prueba (están en la carpeta ejemplos del proyecto) donde se ve como el asistente reconoce al candidato e incluso desaconseja el puesto al que postula por no ser relevante para su perfil.

Retos y Líneas Futuras

- **Integración con Bases de Datos:** Integrar la aplicación con bases de datos de candidatos para una evaluación más completa. Podríamos ir almacenando todas las conversaciones y así ir generando un historial de todas nuestras entrevistas. Con suerte, pocas 😊
- **Optimización de la Interfaz:** Mejorar la interfaz de usuario para hacerla más intuitiva y amigable. Hacer que los mensajes se muestren en streaming, o que se vea de forma más visual qué documentos se han aportado en todo momento.
- **Generar un pequeño resumen:** resumir el CV del candidato automáticamente muy brevemente al subir el CV y mostrarlo.

Despliegue

Puedes desplegar esta aplicación directamente en los servidores de Streamlit. Solo necesitas crear una cuenta en [Streamlit Sharing](#) y seguir las instrucciones para desplegar tu aplicación. Básicamente es subir la aplicación a un repo, loguearte en streamlit con tu cuenta de github y desplegar. **No olvides añadir la API KEY de OpenAi y la de Groq a las variables de entorno!!!** He subido una copia de la app por si alguien quiere trastear (<https://ma-barracas-perfect-candidate-app-ij0pkh.streamlit.app/>) mientras la tengo un tiempo desplegada.

Conclusiones

Crear una aplicación como PerfectCandidate no solo te ayuda a destacar entre otros candidatos, sino que también te brinda la oportunidad de demostrar tus habilidades en el uso de tecnologías avanzadas. Esta herramienta puede ser útil para destacar en tu proceso de reclutamiento demostrando que vas más allá en el uso de nuevas tecnologías. Y si no tienes claro si usarla o no en un contexto real, en el peor de los casos siempre es interesante ver aplicaciones con potencial de uso real aunque sea para practicar y reforzar tus conocimientos! 😊

Hay muchas maneras de mejorar la app. Si se te ocurre alguna no dudes en contactar conmigo 

