Prueba técnica - Data Scientist

**Prueba Técnica para el Puesto de Científico de Datos Junior**

**Instrucciones generales:** Esta prueba técnica está compuesta por cuatro partes. Completa tantas partes como te sea posible. Te recomendamos que leas todas las partes antes de ponerte a solucionar alguna de ellas.

**Parte 1: Clasificación Semántica de Textos**

**Introducción:**

Como Científico de Datos Junior, desempeñarás un papel clave en el apoyo al equipo de Atención al Cliente. Tu tarea para esta esta parte de la prueba técnica será crear un sistema de clasificación de textos que categorice los correos electrónicos de los clientes. Esto ayudará a agilizar el manejo de consultas y preocupaciones de los clientes.

**Objetivo:** Desarrollar un clasificador que divida los correos electrónicos de los clientes en categorías significativas para ayudar al equipo de Atención al Cliente a gestionar y responder de manera más efectiva.

**Pasos:**

* **Definir Categorías:** Define las categorías apropiadas basándote en los datos proporcionados. Asegúrate de que sean equilibradas entre ser demasiado generales y demasiado específicas.
* **Preparación de Datos:** Trabajarás con tres tablas:
* **Clientes:** Contiene información sobre los clientes.
* **Correos Electrónicos:** Compilación de correos electrónicos de clientes.
* **Impagos:** Un subconjunto de clientes que no deseamos analizar debido a su estado de cuenta moroso. Asegúrate de que estos correos sean filtrados en tu análisis.
* **Desarrollar Pipeline:** Libertad para usar la herramienta que se considere oportuna.

Nota: para esta parte, las categorías de ejemplo que se mencionan en la Parte 4 son completamente irrelevantes. No recomendamos basarse en ellas, ya que son ejemplos meramente ilustrativos de la tarea a realizar en la Parte 4.

**Entorno de Desarrollo para la Parte 1:**

Para esta parte, tendrás que trabajar dentro del directorio “parte1”. Dentro te proporcionaremos una carpeta que contiene un archivo **compose.yaml** que se encarga de levantar dos servicios: una base de datos MySQL y una aplicación de Python que implementarás.

**Instrucciones**:

* **Configuración del Entorno**:
* Una vez hayas descargado la carpeta proporcionada, ejecuta el comando **docker compose up** para levantar ambos servicios (la base de datos y la aplicación).
* Eres libre de modificar el Dockerfile del servicio **app** para añadir las herramientas y utilidades que necesites.
* **Desarrollo del Sistema de Clasificación**:

En la subcarpeta **./parte1 /app**, encontrarás un archivo **main.py**. Tu tarea será:

* Crear una API que, al recibir una fila de la tabla **emails** como entrada, devuelva la predicción de la categoría correspondiente. (en el README tienes más detalles)
* Integrar el código adicional y cualquier otro recurso necesario para que la pipeline de clasificación funcione correctamente.

**Entregable para la Parte 1:**

* La carpeta “parte1” proporcionada con las modificaciones realizadas en **main.py** y cualquier otro archivo necesario
* Incluye en la carpeta “parte1” un fichero CSV con dos columnas: ID de correo y categoría asignada por tu clasificador. Cada fila del CSV se corresponderá con uno de los correos que hay en la base de datos.

**Parte 2: Análisis e Información**

**Objetivo:** Analizar los datos categorizados para identificar patrones, obtener información y proponer mejoras para el departamento de Atención al Cliente, utilizando como base los emails proporcionados en la base de datos.

**Pasos:**

* **Exploración de Datos:** Profundiza en los correos categorizados para entender la distribución, como:
* Volumen de correos por categoría.
* Distribución temporal de correos (por ejemplo, ¿hay problemas que surgen en momentos específicos del año/mes/día?).
* **Proporcionar Recomendaciones:** Basado en tu análisis:
* Sugiere posibles estrategias para una mejor asignación de recursos dentro del departamento.
* Identifica posibles áreas de formación o capacitación para el equipo de Atención al Cliente.
* Propón medidas preventivas que podrían reducir el volumen de ciertas categorías de problemas.

**Entorno de Desarrollo para la Parte 2:**

Dentro del directorio “parte2”, debes desarrollar un notebook en el que realices el análisis descrito en la "Parte2" del ejercicio. En este notebook, explora los correos categorizados, identifica patrones y presenta tus recomendaciones.

**Entregable para la Parte 2:**

* Dentro del directorio “parte2”, un notebook con el análisis y conclusiones extraídas de los datos.

Ten en cuenta que, dada la reducida cantidad de datos que te proporcionamos, no esperamos un análisis muy exhaustivo de los datos.

**Parte 3: Generación de Borradores de Respuesta**

**Objetivo:** Desarrollar un sistema que genere borradores de respuesta a correos electrónicos, basándose en ejemplos proporcionados y personalizándolos según el contenido del correo recibido. El sistema deberá clasificar previamente los correos utilizando la API desarrollada en la Parte 1 y, posteriormente, generar una respuesta personalizada utilizando la API de OpenAI GPT.

**Detalles de la Implementación:**

* **Clasificación del Correo:** Utilizar la API creada en la Parte 1 para obtener la categoría del correo.
* **Generación de Respuesta:**
* Basándose en los ejemplos proporcionados, el sistema deberá generar respuestas personalizadas que sigan el estilo y contenido esperado para cada tipo de correo.
* La generación de las respuestas debe ser gestionada de manera automática, basándose en los ejemplos dados y enriquecida con GPT para adaptarse al contexto del correo.
* **Casos No Cubiertos por los Ejemplos:**
* Para correos que no correspondan a los ejemplos dados, el sistema deberá generar una respuesta estándar indicando que un agente revisará la consulta y se pondrá en contacto con el cliente.

**Ejemplos de Correos y Respuestas Esperadas**

**Ejemplo 1:**

* Correo:
* Hola buenas tardes, necesito que me envíen el contrato de electricidad y gas. Nunca me lo han enviado.
* Ejemplo de respuesta esperada:
* Buenas tardes, gracias por su consulta. Puede descargar su contrato de electricidad y gas directamente desde nuestra aplicación móvil o desde el área de clientes en nuestra página web.

**Ejemplo 2**

* Correo:
* Hola, buenos días. Mira que me habéis cobrado las facturas de la luz y el gas, pero no me habéis mandado la factura. ¿Vale? Gracias.
* Ejemplo de respuesta esperada:
* Buenos días, gracias por su consulta. Puede encontrar y descargar copias de todas las facturas asociadas a sus contratos en nuestra aplicación móvil o en el área de clientes en nuestra página web.

**Ejemplo 3**

* Correo:
* Buenas, quisiera conocer las tarifas actuales para nuevos clientes.
* Ejemplo de respuesta esperada:
* Hola, gracias por su interés. Toda la información sobre nuestras tarifas actuales para nuevos clientes está disponible en nuestra página web.

**Ejemplo 4**

* Correo:
* Buenos días, he tenido un problema con el suministro de electricidad. Por favor, ayúdenme.
* Ejemplo de respuesta esperada:
* Hola, gracias por contactarnos. Un agente revisará el problema que indica con el suministro de electricidad tan pronto como sea posible y se pondrá en contacto con usted.

**Entorno de desarrollo:**

* Dentro del directorio “parte3” te proporcionamos un fichero main.py a modo de guía de lo que se desea conseguir.
* Has de implementar dos partes: la llamada a la API implementada en la parte 1 para clasificar el correo, y la generación automática de la respuesta basándote en el cuerpo del correo y la categoría asignada.
* Te proporcionamos una API key de una cuenta de OpenAI que tiene permiso para usar únicamente el modelo ”gpt-4o-mini”. Esta cuenta tiene un límite de uso restringido y debes hacer un uso responsable de la misma, con el único objetivo de completar la prueba técnica. Puedes utilizar este modelo para otras partes de la prueba si lo consideras apropiado.
* Encontrarás la API key en el fichero “parte3/apikey.txt”.

**Entregable para la Parte 3:**

* La carpeta “parte3” proporcionada con las modificaciones realizadas en **main.py** y cualquier otro fichero de código utilizado.

**Parte 4: Sistema de Clasificación Flexible**

**Introducción**

En el ámbito de los agentes de atención al cliente, la gestión eficaz de los correos electrónicos puede marcar la diferencia entre una experiencia excepcional para el cliente y una frustrante. No todos los correos son iguales: algunos son urgentes, otros contienen promociones, y otros pueden implicar quejas o denuncias. Los sistemas de clasificación automatizada ayudan a los agentes a priorizar y abordar las consultas de manera más eficiente.

En las partes anteriores, creaste un sistema de clasificación semántica, analizaste los datos categorizados e implementaste un sistema de respuestas automáticas. Ahora, en la Parte 4, irás un paso más allá al diseñar un sistema de clasificación flexible que pueda adaptarse a criterios personalizados definidos en lenguaje natural. Es decir, tu tarea en esta parte será la de implementar un sistema que, dada una descripción en lenguaje natural del comportamiento esperado de un clasificador, sea capaz de crear un clasificador en Python que implemente dichos criterios.

**Contexto**

Imagina que un supervisor del departamento de Atención al Cliente necesita clasificar los correos según reglas específicas que cambian frecuentemente, como:

* Clasificar los correos marcados como "urgente" en el asunto con prioridad máxima.
* Identificar correos promocionales basados en palabras clave como "oferta" o "promoción".
* Manejar excepciones, como direcciones de correo específicas (por ejemplo, test@gmail.com) que requieren una categoría especial.

El objetivo de esta parte es crear un sistema que permita configurar este tipo de reglas fácilmente utilizando descripciones en lenguaje natural. Por ejemplo, alguien sin conocimientos técnicos debería poder escribir algo como:

"Si el asunto contiene 'urgente', clasifícalo como 'urgente'. Si el cuerpo contiene 'oferta', clasifícalo como 'promo'. En cualquier otro caso, usa el clasificador por defecto."

Tu tarea será crear un sistema capaz de traducir estas descripciones en un sistema de clasificación funcional que evalúe criterios de forma secuencial, aplicando reglas personalizadas y, si ninguna aplica, delegando la decisión a un clasificador por defecto (el sistema creado en la Parte 1).

**Ejemplos**

Algunos ejemplos de lo que se desea, tomando el siguiente email como ejemplo:

* ID de cliente: 123
* Asunto: “Urgente: no ignorar”
* Cuerpo: “Quiero hacerle una oferta.
* De: [ventas@gmail.com](mailto:ventas@gmail.com)

**Nota:** esta tabla se puede interpretar de la siguiente forma: ¿Dado el email de ejemplo, qué categoría le asignaría un clasificador que siguiera el criterio descrito en lenguaje natural?

|  |  |
| --- | --- |
| **Criterio de clasificación (en lenguaje natural)** | **Categoría** |
| Si el asunto contiene la palabra “urgente”, clasificarlo como “urgente”. Si el cuerpo contiene la palabra “oferta”, clasificarlo como “promo”. Si no, usar el clasificador por defecto. | Urgente |
| Si el cuerpo contiene la palabra “denuncia”, clasificarlo como “denuncia”. Si el cuerpo contiene la palabra “oferta”, clasificarlo como “promo”. Usar el clasificador default para cualquier otro caso. | Promo |
| Usar siempre el clasificador por defecto | <insertar aquí la categoría (definida por el candidato) que devuelva por API el clasificador de la parte 1.> |
| If the sender of the email is exactly “test@gmail.com”, classify the email as “test”. Also, classify it as “promo” if the sender address contains the word “ventas”. In all other cases, use the default classifier. Thank you! | Promo |

Se debe implementar un mecanismo de clasificación que permita evaluar uno o más criterios de forma secuencial. Tomando como ejemplo el primer ejemplo, el mecanismo de clasificación resultante debería:

* Evaluar si el asunto contiene la palabra “urgente”.
* Si la contiene, entonces asignar al correo la categoría “urgente” y terminar.
* Si no la contiene pasar al siguiente criterio:
* Evaluar si el cuerpo contiene la palabra “oferta”.
* Si la contiene, entonces asignar al correo la categoría “promo” y terminar.
* Si no la tiene, pasar al siguiente criterio.
* Utilizar el clasificador por defecto (es decir, llamar a la API definida en la parte 1). Devolver la categoría que devuelva la API y terminar.

Se consideran únicamente dos tipos de criterios de clasificación concretos, que pueden ser ejecutados secuencialmente hasta que algún criterio/clasificador devuelva una categoría no nula.

* **Clasificador por defecto**: usar el clasificador creado en la parte 1, llamando a la API. En caso de que la API devuelva error, la categoría será None.
* **Clasificador basado en una condición de tipo equals / contains**: Este clasificador implementa una condición sencilla del tipo: la parte X del correo es igual a / contiene la palabra Y. En caso de que la condición se cumpla, se asigna una categoría determinada.
* Las partes del correo a considerar son: “asunto”, “cuerpo”, “remitente”.
* Las condiciones que se implementan pueden ser o bien de tipo “equals” (y == x) o “contains” (y in x).
* Ejemplos:
* El asunto del correo es “Urgente: no ignorar” (debe coincidir al 100%).
* El cuerpo del correo contiene la palabra “promoción” (basta con que el texto “promoción” esté presente en el cuerpo del correo).
* El remitente del correo es “test@gmail.com”
* Un clasificador basado en palabras clave solo puede contener una única condición. Si la condición se cumple para un correo concreto, se asigna la categoría asociada al clasificador. En caso contrario, no se asigna una categoría.
* **Las categorías de estos clasificadores no tienen por qué pertenecer al conjunto de categorías definidas para la Parte 1**
* El clasificador por defecto siempre asignará categorías definidas en la Parte 1 (por ejemplo, si las categorías definidas en la parte 1 son A, B, C, entonces el clasificador por defecto solo podrá asignar esas tres categorías).
* Los clasificadores basados en condiciones de tipo equal/contains podrán devolver categorías de ese conjunto, u otras adicionales (por ejemplo, A, B, C, D, E).

Para guiar el desarrollo de la Parte 4, se proporciona una serie de clases base de Python sobre las has de trabajar para implementar su mecanismo de clasificación. Estas clases se encuentran en el fichero base.py

El paso de una descripción en lenguaje natural a un clasificador concreto ha de hacerse en dos pasos:

* Pasar del lenguaje natural a un JSON que defina el comportamiento del clasificador.
* Este es precisamente el propósito de la clase base “NaturalLanguageConfigParser”, que acepta una cadena de texto (frase en lenguaje natural) y devuelve un diccionario (análogo al JSON).
* Deserializar el JSON a un objeto de Python que implemente la lógica de clasificación descrita.
* Este es precisamente el propósito de la clase base “ClassifierDeserializer”, que recibe un diccionario y devuelve una instancia de un Classifier.

Tareas:

* Crear implementaciones de la clase base “Classifier” que soporten los diferentes criterios de clasificación especificados en la Parte 4.
* Nótese que, para poder implementar el clasificador por defecto, se tiene que haber completado la Parte 1 (APIClassifier).
* Definir un formato JSON que represente la configuración de un clasificador.
* Crear una implementación de la clase base “ClassifierDeserializer” que permita pasar de una configuración en formato JSON a una instancia del tipo Classifier.
* Crear una implementación de la clase base “NaturalLanguageConfigParser” que permita pasar de un texto en lenguaje natural a un JSON (como diccionario de Python) que respete el formato que has definido.
* Se proporciona un fichero test.py que contiene una serie de prueba unitarias con inputs y outputs esperados. **Tu solución ha de ser lo suficientemente flexible para pasar estos tests.**
* **No se te permite modificar el fichero de tests.** Para conseguir que los tests pasen, tendrás que rellenar las funciones“get\_nl\_config\_parser” y “get\_classifier\_deserializer” del fichero dependencies.py, devolviendo instancias de las clases que implementes.

**Entorno de Desarrollo para la Parte 4:**

Dentro del directorio “parte4”, tienes libertad para crear los ficheros de Python que consideres. Dentro de este directorio, te proporcionaremos un fichero base.py que contendrá las clases base sobre las que tendrás que trabajar para esta parte. El objetivo último es poder rellenar las funciones del fichero dependencies.py y conseguir que todos los tests del fichero test.py pasen (sin modificar el fichero test.py). Para ello, tendrás que crear implementaciones para las diferentes interfaces de base.py, y usarlas en el fichero dependencies.py.

**Entregable para la Parte 4:**

* La carpeta “parte4”, con los ficheros base.py, test.py y dependencies.py (rellenado este último para que los tests pasen), y los ficheros adicionales que utilices para tus implementaciones.