Prueba teórica - Gestor SR con énfasis en IA.

Juan Felipe Quinto Rios

Problema 1

En la empresa GA, en el área de compras necesitan CLASIFICAR y organizar los correos que llegan a la bandeja de entrada entre 4 tipos de correos (Compras cementos, Compras energía, Compras concretos y correos generales o de otra índole). Esta tarea se le encomienda a usted, como es el Gestor SR en temas de analítica e IA puede solicitar al área interesada los recursos humanos que necesite para llevar a cabo este proyecto, también puede solicitar en tecnología todo lo que necesite, además tiene las bandejas de entrada de correos históricos de los analistas que reciben estas solicitudes con aproximadamente: 5500 correos de compras cementos, 2700 correos de compras de energía, 1100 correos de compras concretos y 12876 correos generales o de otra índole.

- Explique como resolvería este problema, metodología, algoritmos, modelos, arquitectura del proyecto etc.

Respuesta:

Se podria inferir a partir del enunciado que el área de compras cuenta con más de veintidós mil correos históricos clasificados por uso de los anistas de negocio (5500 de cementos, 2700 de energía, 1100 de concretos y cerca de 13000 generales). Lo primero será diseñar el flujo ideal: cada correo ingresa a la bandeja, pasa por un proceso de limpieza, transformación y clasificación, y finalmente llega al analista adecuado. Se aprovechará el ecosistema basado en nube para bridar solución mediante al orquestacion de cada paso sin depender de scripts dispersos.

En la fase de exploracion con datos reales, se revisará cómo están almacenados esos correos históricos: si están en un buzón, podria usar la Graph API de Azure para extraer el asunto, el cuerpo y los metadatos de cada mensaje, y luego trasladarlos a un contenedor en Azure Blob Storage. De esa forma, se asegura que todos los correos queden en formato uniforme (texto plano) y se mantiene un repositorio centralizado para el trabajo colaborativo con el equipo de datos. Paralelamente, se establecerá en Azure Data Factory un pipeline que automatice la ingestión de nuevos correos en tiempo real: cada vez que llegue un mensaje a la bandeja de compras, el pipeline lo ingestará en una carpeta raw-emails/incoming/ en Blob Storage.

Con los correos ya en la nube, se continuna con etapa de limpieza y el preprocesamiento en un notebook. En Azure Databricks, usaré Spark para preparar los datos: eliminar las

etiquetas HTML, firmas y texto irrelevante con expresiones regulares, normalizando todo a minúsculas. A medida que se identifiquen patrones, crearé un script en PySpark que detecte y elimine datos sensibles (como direcciones o números de cuenta que no son necesarios,). También incluiré en Databricks una rutina de lematización (proceso que busca reducir una palabra en su forma base o lema, que es la forma que se encuentra en un diccionario) usando spaCy en español (o alguna libreria multilenguaje si es el caso), para asegurar que palabras como "cotización" y "cotizar" se traten como la misma raíz. Este procesamiento lo orquestaré con un job en Azure Databricks que se ejecute automáticamente al detectar nuevos blobs en la carpeta de entrada.

Una vez que el texto esté limpio y normalizado, se transformarán los correos en vectores. Para la primera iteración, se implementará un pipeline ligero con TF-IDF (Term Frequency-Inverse Document Frequency) en scikit-learn, que es una técnica que se utiliza para medir la importancia de una palabra en un documento en relación a una colección de documentos, esta será mi baseline. Entrenaré un modelo de Logistic Regression con regularización L2 para establecer un punto de referencia, y registraré los artefactos (vectores y modelo) en MLflow para poder comparar en el futuro. Así, obtendré una métrica de accuracy global y la matriz de confusión, manteniendo control sobre si la clase "Concretos" se queda rezagada en comparación con las demás.

Con el baseline validado, el siguiente paso: probar un modelo más sofisticado. En Azure ML, se puede aprovecahar los GPU para afinar un modelo de BERT en español (por ejemplo, el checkpoint microsoft/Biotex-MedTR). Este modelo trae integrados mecanismos de comprensión de contexto y, al entrenarlo con nuestro histórico (incluyendo balanceo de clases para aumentar ejemplos de "Concretos", clase minoritaria), espero incrementar varios puntos el F1 de esa clase minoritaria. Se configura un Compute Cluster en AML y realiza un versionado cada experimento en MLflow—guardando pesos, hiperparámetros y métricas. De esa forma, posteriormente se compara en Azure ML Studio cuánta mejora proporciona la capacidad contextual de BERT sobre el enfoque TF-IDF puro.

Una vez obtenga el modelo optimizado, continuamos con al el despliegue como un servicio. parae ello se crea un Dockerfile basado en Python 3.9, instalando llibrerias necesarias como transformers, torch, scikit-learn y el SDK de Azure ML. Se empaqueta el endpoint de inferencia como contenedor y se registra en el Azure Container Registry (ACR). En Azure ML, se genera un entorno de inferencia y se configura un AKS (Azure Kubernetes Service) como destino del despliegue. Con esto, se dispone de un endpoint HTTPS que recibe el texto de un correo y devuelva la etiqueta (Compras Cementos, Compras Energía, Compras Concretos o General), junto con las probabilidades correspondientes. Para conectar con la bandeja de correo, se desarrolla una función en Azure Functions que, cada vez que llegue un correo a la bandeja de compras, envíe un POST al endpoint en AKS y aplique la etiqueta en Exchange automáticamente.

Por ultimo, se configura un tablero de monitoreo en Azure Monitor y Log Analytics. Se registrarn las métricas clave: la tasa de aciertos por clase, la latencia de respuesta del modelo y el número de peticiones diarias. adicional, se podria crear un script en Python que periódicamente extraiga las predicciones y sus probabilidades, y las envíe a MLflow como nuevos "runs" de validación. Así, si algún día el accuracy de la clase "Energía" baja más de un 10 %, Azure Monitor disparará una alerta y se notificará al equipo de Data Science.

En cuanto a la metodología, lo esencial será mantener un pipeline modular y reproducible:

- Centralizar los correos en Azure Blob Storage y unificar su formato.
- Automatizar el preprocesamiento en Databricks y Data Factory.
- Versionar experimentos y modelos en MLflow dentro de Azure ML.
- Desplegar el servicio contenedorizado en AKS, integrando con Azure Functions y Exchange.
- Monitorear performance y posible drift a futuro con Azure Monitor y MLflow

Problema 2

Seis meses después de haber desplegado un modelo de regresión en producción, los usuarios se dan cuenta que las predicciones que este está dando no son tan acertadas, se le encarga a usted como Gestor SR en temas de IA que revise que puede estar sucediendo.

- ¿Cree que el modelo esté sufriendo Drift?
- ¿Cómo puede validarlo?
- ¿De ser así, que haría usted para corregir esto?
- Explique sus respuestas.

Respuesta:

Si los usuarios comienzan a notar que las predicciones ya no son tan acertadas, lo primero que debe hacer es observar con detalle el entorno: Indigar cual es fue el RMSE que se obtuvo en test duranae la fase de entrenamiento, mirar que cambios han ocurrido un cambio en las condiciones del negocio o en los datos.

Para verificarlo, se recopialan datos recientes: solicitaré al equipo de Data un histórico de predicciones vs. valores reales de los últimos tres meses. Luego, en un notebook de Azure ML, ejecutaré scripts que calculen el RMSE mensual, el MAE y el R^2 para cada periodo. Si detecto que el MAE subió un 20 % en los últimos dos meses, eso será el indicio de que algo cambió.

Simultáneamente, iniciaré un experimento en MLflow para comparar las distribuciones de las variables de entrada: extraeré los valores de cada feature y los compararé con las muestras originales de entrenamiento usando la prueba Kolmogorov–Smirnov de scipy.stats. Si hallo, por ejemplo, que el rango de prediccion se desplazó significativamente ($p<0.01$), estaré ante un data drift. Si, además, la correlación entre esa variable y la predicha ya no coincide con la relación original, confirmaré que existe también un concepto drift.

A partir de esa evidencia, se pude trabajar en diseñar la ruta de corrección optima. Lo primero será documentar los hallazgos en MLflow, donde ya estarán versionados los datos de entrenamiento, los experimentos previos y las métricas históricas. MLflow me permitirá comparar gráficas de rendimiento y ver cómo evolucionaron las distribuciones de cada feature. Posteriormente la idea seria configurarar un pipeline de retraining en Azure ML Pipelines: usaré los últimos seis meses de datos junto con el histórico original, prepararé las nuevas observaciones y volveré a entrenar el modelo, bien sea reentrenando desde cero o usando un enfoque incremental (warm-start) en scikit-learn, para ajustar hiperparámetros con un pequeño GridSearch. Una vez obtenga el nuevo modelo, lo registraré en MLflow.

Este pipeline se programará para ejecutarse automáticamente cuando Azure Monitor detecte drift. Por ejemplo, si la métrica model_error_monthly supera un umbral predefinido, Azure Monitor enviará una señal que desencadene el reentrenamiento. El proceso culminará con un despliegue tipo en AKS: la versión antigua seguirá activa mientras validamos la nueva versión con un pequeño porcentaje del tráfico real. Si las métricas en staging confirman que el nuevo modelo mejora el RMSE en al menos un 5 %, entonces cambiaré el tráfico al 100 % para la versión actualizada. Mientras tanto, en el rol de datos se buscará explorar la incorporación de nuevas variables al modelo original. Para ello, los notebooks colaborativos de Azure ML y los recursos de cómputo bajo demanda para prototipar nuevas features, hacer validación cruzada y decidir si agregarlas a la siguiente iteración.

Problema 3

Su equipo de trabajo está trabajando en un chatbot con generación de texto utilizando el modelo GPT-3.5, según cómo funciona este modelo, ¿cómo haría usted para hacer que las respuestas del chatbot estén siempre relacionadas a conseguir cierta información particular del usuario y no empiece a generar texto aleatorio sobre cualquier tema? Explique su respuesta.

Respuesta:

Partiré de la premisa de que el modelo genera texto muy natural, pero corre el riesgo de divagar si no lo configuro correctamente. Mi objetivo será que las respuestas siempre vayan dirigidas a información específicada por el usuario, sin añadidos ni distracciones. Para ello

se trabaja en la arquitectura de backend del modelo, mediante un prompt que permita ubicar al Chatbot en su foco de funcionamiento, de esta manera tener un mayor conexto de su tarea y de lo que se le suiere hacer. Para ello se puede crear un prompt inicial en forma de *System massage* predefinido que acompañe al mensaje del usuario, brindando la informacion necesaria al LLM sobre la tarea en especifico.

Tambien se pueden hacer ajustes o fine tuning de los hiperparmetros del modelo, en este caso al hacer la llamada a la API de OpenAI con este mensaje de sistema y parámetros como temperature (disminuyendo su creatividad y siendo mas objetivo a la hora de generar texto) y max_tokens acotado, de esta manera me aseguraré de que el modelo se enfoque en la consulta y no genere párrafos extensos e innecesarios, entiendo obviamente la dinamica de uso dentro de negocio.

Se sabe que este tipo de servicios pueden establecer funciones agenticas, en donde se controle su uso especifico para un caso dentro del negocio determinado, de esta forma se podrian brindar funciones que el chatbot pueda usar cuando la solicitud del usuario se sale del alcance de uso predefinido. Esto se puede trabajar mediante funciones que el agente utillice una vez vea que el sentido de la solicitud del usuario no va por el camino acorde definido para su uso.

Como alternativa ventajosa, puedo incorporar un enfoque de RAG (Retrieval-Augmented Generation): en lugar de dejar que GPT-3.5 genere cualquier pregunta, montaré un vector store con plantillas validadas para cada dato. Tambien se puede aprovechar para brindar contexto tecnico o lenguaje especifico dentro del negocio para la funcion determinada del chatbot, como lo son el caso de las siglas especificas de negocio.

En definitiva, mi estrategia será: definir un prompt de sistema muy directo, validar cada respuesta en el backend con reglas de negocio, y, opcionalmente, usar RAG para asegurar uniformidad en las preguntas. De este modo, el chatbot nunca se desviará y se limitará a generar exactamente las preguntas necesarias para recopilar los datos objetivo.