Documentación de la prueba para Coink

Introducción

Coink es una startup del sector Fintech con la misión de promover la inclusión financiera a través de un producto innovador de tecnología colombiana. En esta prueba, se desarrollaron tres tareas principales para evaluar la creatividad, habilidad para aprender información nueva y conocimientos de Machine Learning.

Punto 1: Métrica para evaluar el Desempeño de los Usuarios

Descripción del problema:

Se proporcionó una base de datos de depósitos en Oinks que contiene las cabeceras 'user_id', 'operation_value', 'operation_date', 'maplocation_name' y 'user_createddate'. Se solicitó diseñar una métrica para evaluar qué tan buenos son los usuarios de Coink y calificar a los usuarios contenidos en la muestra.

Solución propuesta:

Para definir una métrica de evaluación, se decidió calcular el 'User Score', una combinación ponderada de varios factores:

1. **Frecuencia de Depósitos**: Número de depósitos realizados por el usuario.
2. **Consistencia en los Depósitos**: Medida de la regularidad de los depósitos.
3. **Monto Total Depositado**: Suma de los valores de los depósitos realizados por el usuario.
4. **Tiempo de Uso**: Tiempo desde que el usuario creó su cuenta hasta la fecha del último depósito.

Cada uno de estos factores se combinó en una puntuación final.

Implementación:

La implementación la podemos visualizar en el archivo dispuesto en la carpeta userScore 'desempeñoUsusarios.py'

Punto 2: Implementación de un Modelo de ML en AWS

Descripción del problema:

Se solicitó construir un modelo de Maching Learnig que sea implementado en AWS y que pueda ser ejecutado a demanda con un API.

Solución Propuesta:

Para implementar un modelo de Maching Learnig en AWS, se utilizan los siguientes servicios:

1. **Amazon SageMaker**: Para entrenar y desplegar el modelo.
2. **AWS Lambda**: Para manejar las solicitudes del API.
3. **Amazon API Gateway**: Para proporcionar una interfaz HTTP para el modelo.

Proceso de Implemantación:

1. Entrenar el Modelo en Amazon SageMaker:
    a. Cargar los datos en un bucket de S3.
    b. Utilizar SageMaker para entrenar el modelo.

    c. Desplegar el modelo como un endpoint de SageMaker.
2. Configurar AWS Lambda:
    a. Crear una función Lambda para invocar el endpoint de SageMaker.
    b. Configurar la función Lambda para recibir solicitudes HTTP.
3. Configurar Amazon API Gateway:
    a. Crear una nueva API.
    b. Configurar un endpoint que invoque la función Lambda.

Explicación:

Para poder implementar un modelo de ML en AWS, primero debemos entrenar el modelo utilizando Amazon SageMaker. Los datos de entrenamiento se almacenan en un Bucket de S3 y se utilizan para entrenar el modelo en SageMaker. Ya entrenado el modelo, este se despliega como un endpoint de SageMaker, lo cual, me permite realizar predicciones a través de solicitudes HTTP.

Ahora, creamos una función AWS Lambda para poder manejar las solicitudes entrantes y llamar al endpoint de SageMaker. La función Lambda recibe los datos de entrada, realiza una solicitud de predicción al endpoint de SageMaker y devuelve los resultados.

Finalmente, utilizamos Amazon API Gateway para proporcionar una interfaz HTTP para el modelo. Se configura esta con un endpoint en API Gateway que invoca la función Lambda, permitiendo a los usuarios enviar solicitudes HTTP para obtener predicciones del modelo. Con este enfoque nos aseguramos que el modelo puede ser ejecutado bajo demanda con una API escalable y eficiente.

Punto 3: Modelos de Aprendizaje Supervisado para Predecir el JobSatisfaction

Descripción del Problema:

Se proporcionó una base de datos con múltiples características y se solicitó dos modelos de aprendizaje supervisado para predecir la variable ‘JobSatisfaction’

Solución Propuesta:

Se seleccionaron dos modelos para esta tarea: Ramdon Forest y XGBoost. Los modelos fueron entrenados y evaluados utilizando métricas de error cuadrático medio (MSE).

Por qué se eligieron Ramdon Forest y XGBoost

Ramdon Forest:

- **Robustez y precisión**: Randon Forest es un modelo de ensamblado basado en la construcción de múltiples árboles de decisión. Es conocido por su alta precisión y capacidad para manejar grandes cantidades de datos con alta dimensionalidad sin sobre ajustar.
- **Manejo de Datos Missing y Variables Categóricas**: Es muy efectivo para manejar datos faltantes y puede procesar variables categóricas sin necesidad de una codificación extensiva.
- **Reducción de la Varianza**: Al promediar los resultados de múltiples árboles, Ramdon Forest reduce la varianza del modelo, lo que mejora su generalización a datos no vistos.

XGBoost:

- **Alta Performance y Eficiencia**: XGBoost es una implementación optimizada de gradient boosting que se destaca por su alta performance en competencias de Machine Learnig. Es extremadamente eficiente en términos de tiempo de entrenamiento y rendimiento de predicción.

- **Control de Overfiting**: Proporciona mecanismos avanzados para controlar el overfiting a través de parámetros como ‘max_depth’, ‘learnig_rate’ y n_estimators’.
- **Flexibilidad**: Es altamente flexible y puede ser ajustado con una variedad de parámetros para mejorar la precisión y adaptarse a diferentes tipos de datos y problemas.

Implementación:

Esta la podemos consultar en el archivo dispuesto en la carpeta de modelos llamado ‘models.py’.

Monitoreo de Modelos

Para monitorear los modelos desplegados se pueden implementar las siguientes estrategias:

1. **Seguimiento de Precisión y Error del Modelo**: Evaluar continuamente el rendimiento del modelo en datos nuevos y configurar alertas para detectar incrementos significativos en el error.
2. **Detección de Cambios en la Distribución de los Datos**: Monitorear las distribuciones de las características de entrada y detectar cambios significativos que puedan indicar la necesidad de reentrenar el modelo.
3. **Reentrenamiento Automático**: Programar reentrenamientos periódicos del modelo utilizando datos nuevos. Automatizar el reentrenamiento y el despliegue del modelo cuando se detecten disminuciones en la precisión (en el archivo dispuesto en la carpeta modelos llamado ‘monitoreo.py’ podemos tener una fase inicial en el monitoreo automático que se puede configurar en una AWS Lambda para conocer periódicamente la precisión del modelo).

Implementar un sistema de logging para registrar todas las predicciones y los errores del modelo, y utilizar herramientas de monitoreo como Amazon CloudWatch para configurar alertas y mantener un registro continuo del rendimiento del modelo.