Pruebas de Software Servicio Predicción Modelo de Machine Learning

1. Introducción

Propósito:
Este documento tiene como objetivo describir y documentar las pruebas realizadas sobre el servicio de detección de spam basado en un modelo de aprendizaje profundo alojado en Flask. El propósito de estas pruebas es asegurar que el servicio funcione correctamente en condiciones normales y bajo cargas variadas, que cumpla con las expectativas de precisión y que pueda integrarse de manera efectiva con otros sistemas que hagan uso de sus predicciones.

Alcance de las pruebas:
Se documentarán los siguientes tipos de pruebas:

- **Pruebas Unitarias:** Validación de funcionalidades individuales del servicio, como las rutas y el procesamiento de texto.
- **Pruebas de Precisión:** Evaluación de la precisión del modelo de detección de spam en la clasificación de mensajes.
- **Pruebas de Carga:** Verificación del rendimiento del servicio bajo diferentes niveles de carga.
- **Pruebas de API:** Validación del funcionamiento correcto de los endpoints de la API, incluyendo respuestas adecuadas a entradas válidas e inválidas.

Objetivos:

1. Asegurar que las funcionalidades individuales del servicio (por ejemplo, procesamiento de texto, predicciones y manejo de errores) funcionan correctamente.
2. Verificar que el modelo clasifica con precisión los mensajes como "spam" o "not spam".
3. Evaluar el rendimiento del servicio bajo diferentes niveles de carga.
4. Asegurar que la API responde de manera consistente y adecuada a diferentes tipos de entradas.

2. Metodología de Pruebas

Tipos de pruebas:

- **Pruebas Unitarias:** Se realizan para validar la funcionalidad de los endpoints y el manejo de errores en el servicio. Cada prueba unitaria evalúa una parte específica del código, incluyendo casos en los que el modelo no se cargue, problemas de traducción, entradas vacías o mal formadas, y verificaciones de las respuestas para textos de prueba.

- **Pruebas de Precisión:** Se realizan con un conjunto de datos conocido para evaluar la capacidad del modelo de clasificar los mensajes correctamente. Estas pruebas verifican el porcentaje de precisión del modelo en un subconjunto de datos de mensajes "spam" y "ham" (no spam).
- **Pruebas de Carga:** Utilizando la herramienta **Locust**, se realizan pruebas para observar el comportamiento del servicio bajo distintas cargas de usuarios simulados, midiendo tiempos de respuesta y estabilidad bajo cargas ligeras y moderadas.
- **Pruebas de API:** Verifican la funcionalidad de la API del servicio en cuanto a los resultados esperados y el manejo de entradas válidas e inválidas. Estas pruebas se centran en la ruta principal y en la ruta `/predict`, asegurando que la API maneje adecuadamente diferentes tipos de solicitudes.

Herramientas de prueba:

- **Unittest:** Utilizado para las pruebas unitarias, permitiendo evaluar las respuestas y el manejo de errores en el servicio.
- **Pandas y Requests:** Herramientas empleadas para cargar un conjunto de datos de prueba y enviar solicitudes al servicio en las pruebas de precisión.
- **Locust:** Herramienta de pruebas de carga que permite simular múltiples usuarios y medir el rendimiento del servicio bajo condiciones de carga.
- **Pytest y Requests:** Usadas en las pruebas de API para validar el funcionamiento de los endpoints.

Criterios de aceptación:

- **Pruebas Unitarias:** Se consideran exitosas si cada prueba individual retorna los valores esperados, incluyendo los mensajes de error y respuestas del endpoint `/predict`.
- **Pruebas de Precisión:** Se establece un umbral de precisión mínimo del 80% para considerar que el modelo cumple con los estándares deseados.
- **Pruebas de Carga:** Los tiempos de respuesta promedio no deben superar los 500 ms bajo cargas moderadas. El servicio debe mantener una tasa de éxito del 95% o superior.
- **Pruebas de API:** Cada endpoint debe responder correctamente a entradas válidas e inválidas. La respuesta del endpoint `/predict` debe incluir una predicción adecuada en función del texto de entrada y manejar adecuadamente errores comunes (texto vacío, errores de traducción, modelo no cargado).

3. Pruebas Unitarias

Propósito:
Validar la funcionalidad de los componentes individuales del servicio de detección de spam, especialmente el endpoint de predicción (`/predict`) y la ruta principal (`/`). Las pruebas unitarias ayudan a asegurar que el servicio responde correctamente a diversas entradas, maneja errores de forma adecuada y que las funcionalidades clave (como la carga del modelo y el manejo de traducción) funcionan como se espera.

Herramientas:

- **unittest**: Framework de pruebas utilizado para realizar pruebas unitarias de las rutas y funcionalidades del servicio.
- **unittest.mock (patch)**: Utilizado para simular situaciones de error (como el modelo no cargado o un fallo en la traducción) y verificar que el sistema maneja estas excepciones adecuadamente.

Estrategia de Pruebas:

1. **Cobertura de Pruebas:**
   Se espera una cobertura de código que valide los casos principales del endpoint `/predict`, la ruta principal `/`, y el manejo de posibles errores como:
   - Texto vacío o faltante.
   - Formato JSON mal formado.
   - Modelo no cargado.
   - Fallo en la traducción.
2. **Criterios de Ejecución:**
   Cada caso de prueba se ejecuta enviando solicitudes a la API utilizando el cliente de prueba de Flask, con verificaciones sobre el código de respuesta HTTP y el contenido de la respuesta JSON.

Casos de Prueba:

1. **Test: test_home_route**
   - **Descripción:** Verifica que la ruta principal (`/`) responde correctamente.
   - **Entradas:** Solicitud GET a `/`.
   - **Acciones:** Realizar una solicitud GET.
   - **Resultados Esperados:** Código de estado 200 y mensaje `"Welcome to the Spam Detection Service!"`.
   - **Resultados de Ejecución:** PASSED.
2. **Test: test_predict_spam_message**
   - **Descripción:** Verifica la predicción de un mensaje con características de spam.
   - **Entradas:** POST con el texto `"Congratulations! You won a free prize. Call now!"`.
   - **Acciones:** Enviar el mensaje al endpoint `/predict`.
   - **Resultados Esperados:** Código de estado 200 y predicción en `["spam", "not spam"]`.
   - **Resultados de Ejecución:** PASSED.
3. **Test: test_predict_ham_message**
   - **Descripción:** Verifica la predicción de un mensaje que no es spam.
   - **Entradas:** POST con el texto `"Hey, just wanted to say hello!"`.
   - **Acciones:** Enviar el mensaje al endpoint `/predict`.
   - **Resultados Esperados:** Código de estado 200 y predicción `"not spam"`.
   - **Resultados de Ejecución:** PASSED.

4. **Test: test_empty_text_error**
    - **Descripción:** Verifica la respuesta del servicio cuando el texto es vacío.
    - **Entradas:** POST con un texto vacío (`"text": ""`).
    - **Acciones:** Enviar el mensaje vacío al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 400 y mensaje de error `"El texto proporcionado está vacío"`.
    - **Resultados de Ejecución:** PASSED.
5. **Test: test_missing_text_key_error**
    - **Descripción:** Verifica la respuesta del servicio cuando falta la clave `"text"` en el JSON.
    - **Entradas:** POST sin la clave `"text"`.
    - **Acciones:** Enviar el JSON vacío al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 400 y mensaje de error `"El texto proporcionado está vacío"`.
    - **Resultados de Ejecución:** PASSED.
6. **Test: test_model_not_loaded_error**
    - **Descripción:** Simula la situación en la que el modelo no está cargado y verifica el manejo del error.
    - **Entradas:** POST con un mensaje de texto cualquiera y `model = None`.
    - **Acciones:** Enviar el mensaje al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 500 y mensaje de error `"El modelo no está disponible"`.
    - **Resultados de Ejecución:** PASSED.
7. **Test: test_translation_error_handling**
    - **Descripción:** Simula un fallo en la traducción y verifica el manejo del error.
    - **Entradas:** POST con el texto `"Mensaje de prueba"`.
    - **Acciones:** Simular una excepción en el traductor y enviar el mensaje al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 500 y mensaje de error `"Error al traducir el texto"`.
    - **Resultados de Ejecución:** PASSED.
8. **Test: test_invalid_json_format**
    - **Descripción:** Verifica la respuesta del servicio cuando el JSON tiene un formato incorrecto.
    - **Entradas:** POST con formato JSON inválido (`data="Invalid JSON format"`).
    - **Acciones:** Enviar el mensaje con formato JSON inválido.
    - **Resultados Esperados:** Código de estado 500 y mensaje de error `"Error en el servidor"`.
    - **Resultados de Ejecución:** PASSED.

Resumen de Ejecución de Pruebas Unitarias:

- **Pruebas Totales:** 8.
- **Pruebas Exitosas:** 8.
- **Pruebas Fallidas:** 0.

Observaciones:
Todas las pruebas unitarias pasaron correctamente, lo que indica que el servicio responde adecuadamente a entradas válidas e inválidas, maneja bien los errores simulados y sigue los criterios de aceptación definidos.

4. Pruebas de Carga

Propósito:
Evaluar el rendimiento del servicio de detección de spam bajo diferentes cargas de usuario concurrentes. Estas pruebas permiten determinar la capacidad del sistema para manejar múltiples solicitudes y analizar su comportamiento en cuanto a tiempos de respuesta y estabilidad bajo condiciones de alta demanda.

Herramientas:

- **Locust:** Herramienta de pruebas de carga utilizada para simular múltiples usuarios enviando solicitudes al servicio. Locust permite generar una carga controlada y medir el rendimiento del sistema en términos de solicitudes por segundo y tiempos de respuesta.

Escenarios de Prueba:

1. **Escenario de Baja Carga:**
   Simular un bajo número de usuarios (10-20) interactuando con el sistema. Este escenario verifica el comportamiento del servicio en condiciones mínimas de uso.
2. **Escenario de Carga Promedio:**
   Simular una carga promedio de usuarios (50-80) para observar cómo el sistema maneja un uso típico y evaluar su capacidad de respuesta en condiciones moderadas.
3. **Escenario de Carga Máxima:**
   Simular una carga elevada con hasta 100 usuarios concurrentes para evaluar el límite de rendimiento del servicio y observar posibles degradaciones en el tiempo de respuesta o tasas de error bajo alta demanda.

Métricas:

- **Total de Solicitudes por Segundo (RPS):**
  Número de solicitudes que el sistema puede manejar por segundo. Se evaluó el sistema para alcanzar hasta 18 solicitudes por segundo bajo carga máxima.
- **Tiempos de Respuesta (ms):**
  Medición del tiempo de respuesta en los percentiles 50 y 95. Los resultados muestran que el tiempo de respuesta promedio aumenta con la carga de usuarios:
    - **Percentil 50:** Tiempo medio que la mitad de las solicitudes cumple; alcanzó valores de hasta 5,000 ms en este escenario.
    - **Percentil 95:** Tiempo que el 95% de las solicitudes cumplen, mostrando un tiempo de respuesta de hasta 7,000 ms en condiciones de alta carga.

- **Consumo de Recursos:**
  Aunque no se incluyeron datos específicos de CPU y memoria en los gráficos proporcionados, estos datos son recomendables para monitorear en futuras pruebas.
- **Tasa de Éxito:**
  No se registraron fallos de solicitud en los resultados obtenidos, indicando una alta tasa de éxito y estabilidad en la respuesta del servicio, incluso bajo carga máxima.

Criterios de Aceptación:

- **Rendimiento bajo Carga Promedio:**
  El tiempo de respuesta promedio debería ser inferior a 500 ms en condiciones de carga promedio (50-80 usuarios).
- **Rendimiento bajo Carga Máxima:**
  El tiempo de respuesta debería mantenerse por debajo de 1,000 ms en el percentil 95 bajo condiciones de carga máxima (hasta 100 usuarios). La tasa de éxito debe ser de al menos 95%.

Resultados de Ejecución:

De acuerdo con los gráficos y la página de reporte de Locust:

- **Solicitudes por Segundo (RPS):** Se alcanzaron picos de hasta 18 RPS, con una tendencia estable y sin caídas significativas, lo cual sugiere que el servicio soporta esta tasa de solicitudes bajo carga intensa.
- **Tiempo de Respuesta:** Los tiempos de respuesta aumentaron progresivamente con la carga, alcanzando un promedio en el percentil 95 de hasta 7,000 ms, lo cual supera el umbral deseado para el rendimiento bajo carga máxima. Esto indica que el servicio puede experimentar una disminución en la velocidad de respuesta cuando se le exige al límite.
- **Tasa de Éxito:** No se registraron errores en las solicitudes, lo cual indica un desempeño estable en cuanto a la tasa de éxito.

Observaciones:

- **Optimización de Respuesta:** Los tiempos de respuesta observados en los percentiles más altos bajo carga máxima sugieren que el servicio podría beneficiarse de optimizaciones adicionales para mejorar la eficiencia y reducir la latencia.
- **Escalabilidad:** El servicio mantuvo estabilidad en la tasa de solicitudes sin fallos, indicando que es capaz de manejar cargas elevadas, aunque con una latencia incrementada.

5. Pruebas de API

Propósito:
Validar el correcto funcionamiento de los endpoints del servicio de detección de spam, asegurando que la API responde adecuadamente a diferentes tipos de solicitudes, tanto válidas como inválidas, y que maneja los errores de forma apropiada.

Herramientas:

- **pytest:** Framework de pruebas utilizado para ejecutar las pruebas de la API y verificar el comportamiento esperado de cada endpoint.
- **requests:** Biblioteca de Python para realizar solicitudes HTTP a la API y simular interacciones con el servicio.

Especificación de las Pruebas de API:

- **Endpoint:** `/` (ruta principal)
  - **Método:** GET
  - **Propósito:** Verificar que el servicio está activo y responde con un mensaje de bienvenida.
  - **Resultados Esperados:** Código de estado 200 y mensaje `"Welcome to the Spam Detection Service!"`.
- **Endpoint:** `/predict`
  - **Método:** POST
  - **Propósito:** Validar la predicción de mensajes de texto para identificar si son spam o no.
  - **Parámetros:** JSON con el campo `"text"`.
  - **Casos de prueba:**
    1. **Texto de spam:** Verificar la predicción de un mensaje con características de spam.
    2. **Texto no spam:** Verificar la predicción de un mensaje sin características de spam.
    3. **Texto vacío:** Verificar la respuesta cuando el texto es vacío.
    4. **Error de formato de JSON:** Evaluar el manejo de un formato JSON inválido (no se incluye en la captura pero es un caso importante).

Casos de Prueba:

1. **Test: test_home**
   - **Descripción:** Verifica que la ruta principal (`/`) responde correctamente.
   - **Entradas:** Solicitud GET a `/`.
   - **Acciones:** Realizar una solicitud GET.
   - **Resultados Esperados:** Código de estado 200 y mensaje `"Welcome to the Spam Detection Service!"`.
   - **Resultados de Ejecución:** PASSED.

2. **Test: test_predict_spam**
    - **Descripción:** Verifica la predicción de un mensaje con características de spam.
    - **Entradas:** POST con el texto `"Free money now!!!"`.
    - **Acciones:** Enviar el mensaje al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 200 y predicción en `["spam", "not spam"]`.
    - **Resultados de Ejecución:** PASSED.
3. **Test: test_predict_not_spam**
    - **Descripción:** Verifica la predicción de un mensaje que no es spam.
    - **Entradas:** POST con el texto `"Hello, I just wanted to check in on the project status."`.
    - **Acciones:** Enviar el mensaje al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 200 y predicción en `["spam", "not spam"]`.
    - **Resultados de Ejecución:** PASSED.
4. **Test: test_predict_empty_text**
    - **Descripción:** Verifica la respuesta del servicio cuando el texto es vacío.
    - **Entradas:** POST con un texto vacío (`"text": ""`).
    - **Acciones:** Enviar el mensaje vacío al endpoint `/predict`.
    - **Resultados Esperados:** Código de estado 400 y mensaje de error `"El texto proporcionado está vacío"`.
    - **Resultados de Ejecución:** PASSED.

Resumen de Ejecución de Pruebas de API:

- **Pruebas Totales:** 4.
- **Pruebas Exitosas:** 4.
- **Pruebas Fallidas:** 0.

Observaciones:

Todas las pruebas de API pasaron correctamente, lo que indica que el servicio responde adecuadamente tanto a solicitudes válidas como a casos de error, y sigue los criterios de aceptación definidos para el endpoint `/predict` y la ruta principal `/`.

6. Pruebas de Precisión

Propósito:
Evaluar la precisión del modelo de detección de spam implementado en el servicio, utilizando un conjunto de datos etiquetado. Estas pruebas permiten verificar que el modelo clasifique correctamente los mensajes de texto como "spam" o "not spam", proporcionando una métrica cuantitativa sobre su efectividad en la tarea de clasificación.

Herramientas:

- **Pandas:** Utilizado para cargar y procesar el conjunto de datos de mensajes de prueba.
- **Requests:** Biblioteca para enviar solicitudes HTTP al servicio de detección de spam, simulando mensajes que el modelo debe clasificar.
- **tqdm:** Empleado para mostrar el progreso de procesamiento de los mensajes, especialmente útil para conjuntos de datos grandes.

Especificación de las Pruebas de Precisión:

- **Conjunto de Datos:**
  Se utilizó un conjunto de datos de mensajes etiquetados, con las categorías "ham", "spam" y "smishing". Los mensajes etiquetados como "ham" se consideran "not spam", mientras que los mensajes etiquetados como "spam" o "smishing" se consideran "spam".
- **Parámetros de Prueba:**
  - Número de mensajes: 100 muestras tomadas del archivo proporcionado para evaluar la precisión del modelo.
  - Etiquetas: `ham`, `spam`, `smishing` (convertidas a "not spam" o "spam" para coincidir con la respuesta esperada del servicio).

Criterios de Aceptación:
La prueba de precisión se considera aceptable si el modelo logra una precisión del 80% o superior en la clasificación de los mensajes de prueba. Esta precisión se calcula como el porcentaje de mensajes correctamente clasificados por el modelo en comparación con las etiquetas esperadas.

Resultados de Ejecución:

De acuerdo con los resultados obtenidos:

- **Precisión del Modelo:** 80.0%.
  - El modelo alcanzó una precisión del 80%, cumpliendo con el umbral mínimo esperado para ser considerado efectivo en la clasificación de mensajes.

Observaciones:

- **Desempeño del Modelo:** La precisión del 80% indica que el modelo tiene una efectividad aceptable en la clasificación de mensajes, aunque existe un margen de mejora, especialmente en la identificación de mensajes que pueden no ser evidentes como "spam" o "not spam".
- **Recomendación:** Para mejorar la precisión, se podría considerar una revisión de los datos de entrenamiento y la optimización del modelo, especialmente en la clasificación de mensajes ambiguos o aquellos que puedan tener características tanto de spam como de mensajes normales.

7. Reporte de Resultados de Pruebas

Resumen de Ejecución de Pruebas:

- **Pruebas Unitarias:**
  - **Pruebas Totales:** 8
  - **Pruebas Exitosas:** 8
  - **Pruebas Fallidas:** 0
  - **Observaciones:** Todas las pruebas unitarias pasaron, lo que demuestra que el servicio responde adecuadamente a entradas válidas e inválidas y maneja los errores de manera adecuada.
- **Pruebas de Carga:**
  - **Máximo de Solicitudes por Segundo (RPS):** 18
  - **Tiempo de Respuesta Promedio (Percentil 50):** 5,000 ms
  - **Tiempo de Respuesta en el Percentil 95:** 7,000 ms
  - **Tasa de Éxito:** 100%
  - **Observaciones:** Aunque el servicio mantuvo una tasa de éxito del 100%, el tiempo de respuesta promedio bajo carga alta superó los límites deseados. Esto sugiere la necesidad de optimizaciones adicionales en el procesamiento del servicio para mejorar el rendimiento bajo condiciones de alta demanda.
- **Pruebas de API:**
  - **Pruebas Totales:** 4
  - **Pruebas Exitosas:** 4
  - **Pruebas Fallidas:** 0
  - **Observaciones:** La API funcionó correctamente para todas las entradas válidas y manejó los errores de manera adecuada. Esto confirma que los endpoints son robustos y responden consistentemente según los criterios establecidos.
- **Pruebas de Precisión:**
  - **Precisión del Modelo:** 80.0%
  - **Observaciones:** El modelo alcanzó el umbral mínimo de precisión esperado. Sin embargo, un análisis más profundo y optimización en la clasificación de mensajes ambiguos podría ayudar a mejorar esta métrica.

Análisis de Resultados: Las pruebas indican que el servicio es funcional y estable, con un rendimiento adecuado bajo carga moderada y una precisión aceptable en la detección de spam. No obstante, bajo carga máxima, el tiempo de respuesta incrementado podría impactar la experiencia del usuario. Además, mejorar la precisión del modelo incrementaría su efectividad en aplicaciones reales.

Observaciones y Recomendaciones:

- **Puntos de Mejora:**
  - **Optimización de Rendimiento:** El incremento en el tiempo de respuesta bajo carga máxima sugiere que el servicio podría beneficiarse de optimizaciones en su arquitectura o en el procesamiento del modelo.

    - **Aumento de la Precisión del Modelo:** Se recomienda optimizar el modelo de detección de spam para mejorar la precisión, especialmente en la clasificación de mensajes ambiguos o de difícil interpretación.
- **Acciones Correctivas:**
    - **Revisión y Optimización del Código:** Evaluar y optimizar las partes del código donde el tiempo de respuesta aumenta bajo carga alta.
    - **Ajustes en el Modelo:** Revisar el conjunto de datos de entrenamiento y ajustar los hiperparámetros del modelo para mejorar su capacidad de generalización y precisión.

8. Anexos

Archivos de Configuración de Pruebas: Incluye los scripts y configuraciones utilizados para ejecutar las pruebas unitarias, de carga, de precisión y de la API.

Logs de Ejecución: Los resultados de ejecución de las pruebas de carga, unidad, precisión y API se encuentran documentados en los archivos de salida respectivos. Los archivos de log y capturas de pantalla de los resultados específicos están disponibles como evidencia de los resultados.

Documentación de Herramientas: Para la ejecución de las pruebas, se utilizaron:

- **Unittest:** Herramienta de pruebas en Python utilizada para pruebas unitarias.
- **Pytest:** Usada en pruebas de API, compatible con Unittest.
- **Locust:** Herramienta para pruebas de carga, configurada para simular múltiples usuarios enviando solicitudes al servicio.
- **Pandas y Requests:** Utilizados en las pruebas de precisión para cargar y enviar datos al servicio.