Documentación Modelo de Clasificación de Spam con LSTM

1. Introducción

La detección de spam en mensajes de texto es un desafío importante para servicios de comunicación y redes sociales, que buscan reducir el impacto de contenido no deseado en sus plataformas. Este proyecto se centra en construir un modelo de clasificación binaria para identificar mensajes como "spam" o "ham" (no spam) usando redes neuronales LSTM (Long Short-Term Memory), diseñadas para capturar dependencias a largo plazo en secuencias de datos como el lenguaje natural.

Propósito del Proyecto

El objetivo es construir un modelo eficaz que clasifique mensajes de texto con alta precisión y pocos errores de falsos positivos y negativos. Además, se implementa una API REST para hacer que el modelo sea accesible en tiempo real en entornos de producción.

2. Descripción del Dataset

El modelo se entrena utilizando un dataset de mensajes de texto en inglés, etiquetados manualmente en dos categorías:

- **Ham**: Mensajes normales sin contenido de spam.
- **Spam**: Mensajes no deseados, generalmente con intenciones promocionales o fraudulentas.

Estructura del Dataset

El dataset contiene:

- **Columnas**: `Category` (etiqueta: ham o spam) y `Message` (texto del mensaje).
- **Distribución de Clases**: Contiene aproximadamente 5,572 ejemplos, donde alrededor del 86.6% son mensajes "ham" y el 13.4% son mensajes "spam", lo que genera un conjunto de datos desequilibrado.

Dado este desbalance, los modelos pueden aprender a clasificar incorrectamente si no se abordan estrategias para balancear las clases, ya que un modelo podría aprender a predecir siempre "ham" y lograr una alta precisión aparente sin capturar realmente los patrones que definen el spam.

3. Preprocesamiento de Datos

Para utilizar textos en modelos de aprendizaje automático, es necesario transformar el texto en una representación numérica. En este proyecto, se implementan varias técnicas de preprocesamiento para preparar los datos de entrada.

3.1. Limpieza del Texto

La limpieza del texto consiste en normalizar el contenido y eliminar caracteres no relevantes:

- **Eliminación de caracteres especiales y números**: Se eliminan caracteres no alfabéticos, manteniendo solo letras. Esto reduce el ruido y simplifica el texto.
- **Conversión a minúsculas**: Convierte todo el texto a minúsculas para unificar las palabras y evitar duplicaciones por diferencia de mayúsculas/minúsculas.

3.2. Tokenización y Stemming

Para reducir las palabras a una forma raíz común:

- **Tokenización**: Divide cada mensaje en palabras individuales.
- **Stemming**: Aplica el algoritmo de stemming de Porter, que reduce las palabras a su raíz (por ejemplo, "running" a "run"). Esto mejora la coherencia en el análisis, ya que palabras similares se tratan de manera uniforme.

3.3. Eliminación de Stopwords

Las stopwords son palabras comunes en el idioma que no aportan significado específico, como "is", "the" y "and". Eliminarlas ayuda a enfocar el modelo en palabras clave más significativas para la clasificación de spam.

3.4. Codificación One-Hot y Padding

Se utiliza una codificación de cada palabra a un índice único en un vocabulario limitado:

- **Codificación One-Hot**: Convierte las palabras tokenizadas en índices numéricos con una codificación "one-hot". El tamaño del vocabulario se establece en 10,000, lo que significa que cualquier palabra nueva se asignará a un índice dentro de este rango.
- **Padding**: Cada mensaje se rellena o recorta hasta una longitud fija de 200 palabras. Esto permite que todas las secuencias tengan el mismo tamaño, facilitando el procesamiento en la red LSTM, que espera entradas de longitud uniforme.

4. Arquitectura del Modelo

La arquitectura del modelo está diseñada específicamente para capturar las características secuenciales de los datos de texto.

4.1. Estructura del Modelo

El modelo consiste en una red neuronal secuencial con tres capas principales:

1. **Capa de Embedding**:
    - **Función**: Convierte los índices de palabras en vectores densos de una dimensión específica.
    - **Configuración**: Se establece un tamaño de salida de 100 dimensiones, con un tamaño de entrada de 10,000 (vocabulario) y una longitud de secuencia de 200 palabras.
    - **Beneficio**: Permite que el modelo aprenda representaciones de palabras y relaciones entre ellas en un espacio de alta dimensionalidad.
2. **Capa LSTM**:
    - **Función**: Procesa las secuencias de palabras para capturar dependencias a largo plazo, cruciales en el lenguaje.
    - **Configuración**: Se establece con 128 unidades. La LSTM es adecuada para NLP ya que puede retener información pasada y modelar contexto.
    - **Beneficio**: Ayuda a que el modelo comprenda cómo ciertas combinaciones de palabras y patrones contextuales indican spam.
3. **Capa de Salida (Densa)**:
    - **Función**: Produce una salida binaria (spam o ham) basada en la representación generada por la LSTM.
    - **Configuración**: Contiene una única neurona con activación sigmoide. Dado que el problema es de clasificación binaria, la salida sigmoide genera una probabilidad de que el mensaje sea spam.
    - **Umbral de Clasificación**: En producción, el umbral de clasificación se ajusta a 0.02, lo que significa que cualquier predicción superior a este valor se clasifica como spam. Este umbral puede ajustarse para optimizar el rendimiento y reducir errores de falsos positivos y negativos.

4.2. Compilación del Modelo

El modelo se compila utilizando:

- **Optimizador**: Adam, que es un optimizador eficiente que ajusta el aprendizaje automáticamente.
- **Función de Pérdida**: `binary_crossentropy`, adecuada para problemas de clasificación binaria.
- **Métricas**: Se utiliza `accuracy` para evaluar la precisión durante el entrenamiento y la validación.

4.3. Entrenamiento y Validación del Modelo

Para entrenar y validar el modelo:

- **Split de datos**: Los datos se dividen en conjuntos de entrenamiento, validación y prueba. El conjunto de prueba se separa antes de entrenar, mientras que el conjunto de entrenamiento se subdivide en entrenamiento y validación.
- **Técnica de Submuestreo**: Dado el desbalance de clases, se submuestrean los ejemplos de la clase mayoritaria (ham) para equilibrar las clases.
- **Épocas**: Se entrena durante 10 épocas para optimizar el modelo sin sobreajustar.

4.4. Evaluación del Modelo

La evaluación final se realiza en el conjunto de pruebas, utilizando:

- **Exactitud (Accuracy)**: Indica la proporción de predicciones correctas.
- **Matriz de Confusión**: Permite ver el número de verdaderos positivos (TP), falsos positivos (FP), verdaderos negativos (TN) y falsos negativos (FN).

Se obtiene una buena generalización si la cantidad de FP y FN es baja en relación con TP y TN, lo cual es esencial en un sistema de detección de spam para minimizar alertas incorrectas.

5. Guardado del Modelo

El modelo entrenado se guarda en un archivo `.keras` utilizando la función `model.save()`. Este archivo contiene la estructura, los pesos y el estado de entrenamiento del modelo, permitiendo su reutilización en producción sin necesidad de reentrenarlo.

6. Implementación del Servicio Web

El modelo entrenado se implementa en un servicio web usando Flask, facilitando la accesibilidad del modelo en un entorno de producción. A continuación se describe el servicio.

6.1. Estructura del Servicio Web

El servicio web tiene dos rutas principales:

- **Ruta de Predicción (`/predict`)**: Recibe una solicitud POST con el mensaje y devuelve la clasificación como "spam" o "no spam".
- **Ruta de Bienvenida (`/`)**: Responde con un mensaje de bienvenida para confirmar que el servicio está en funcionamiento.

6.2. Flujo de Predicción

El flujo de predicción consta de los siguientes pasos:

1. **Carga del Modelo**: Al iniciarse el servicio, el modelo `.keras` se carga en memoria. Si ocurre algún error, se registra en los logs y se evita que el servicio acepte solicitudes de predicción.
2. **Recepción de Datos**: La ruta `/predict` acepta solicitudes POST con un mensaje JSON que contiene el campo `text`.
3. **Traducción del Texto**: Si el mensaje está en otro idioma, `GoogleTranslator` lo traduce al inglés para asegurar coherencia con los datos de entrenamiento.
4. **Preprocesamiento del Texto**: Se realiza el mismo preprocesamiento que se utilizó en el entrenamiento del modelo (limpieza, tokenización, stemming, codificación one-hot y padding).
5. **Predicción**: El modelo predice la probabilidad de que el mensaje sea spam. Cualquier valor mayor que el umbral (0.02) se clasifica como spam; de lo contrario, se clasifica como no spam.
6. **Respuesta JSON**: El servicio devuelve el resultado en formato JSON, indicando si el mensaje es "spam" o "not spam".

6.3. Manejo de Errores

El servicio incluye mecanismos de manejo de errores para diversas situaciones:

- **Error al Cargar el Modelo**: Si el modelo no se carga correctamente, el servicio rechaza solicitudes de predicción y devuelve un error 500.
- **Mensaje Vacío**: Si el texto proporcionado está vacío, se devuelve un error 400 con un mensaje adecuado.
- **Errores en la Traducción**: Si ocurre un error durante la traducción, el servicio devuelve un error 500 y se registra el error en el log.

6.4. Configuración del Log

Se utiliza `logging` para registrar eventos importantes, como la carga del modelo, los errores de traducción y los resultados de predicción, lo cual es útil para monitorear el funcionamiento del servicio en producción.

7. Conclusión

Este proyecto implementa un sistema robusto para la detección de mensajes de spam utilizando redes neuronales LSTM. Gracias a la arquitectura de LSTM y un preprocesamiento exhaustivo, el modelo es capaz de identificar patrones de spam con alta precisión. Además, el despliegue como servicio web permite que el sistema sea fácilmente accesible en aplicaciones de producción, ofreciendo una solución escalable y extensible para la detección de spam.