PROYECTO 1 ETAPA 2

Juan David Valencia - 202121467 (Líder de datos)

Leonardo Rueda - 202123599 (Líder de proyecto)

Juan Camilo Ibáñez - 201924835 (Líder de analítica)

Pipeline

Preprocesamiento de texto

La clase TextPreprocessor se encarga de limpiar los textos antes de ser procesados por el modelo. Incluye pasos como la expansión de contracciones, tokenización, conversión a minúsculas, eliminación de puntuación y caracteres irrelevantes, eliminación de stopwords y stemming y lematización.

Vectorización y clasificación

Se usa TfidfVectorizer para convertir el texto en valores numéricos, limitando a 1000 palabras clave. Posteriormente, se entrena un RandomForestClassifier con 50 árboles.

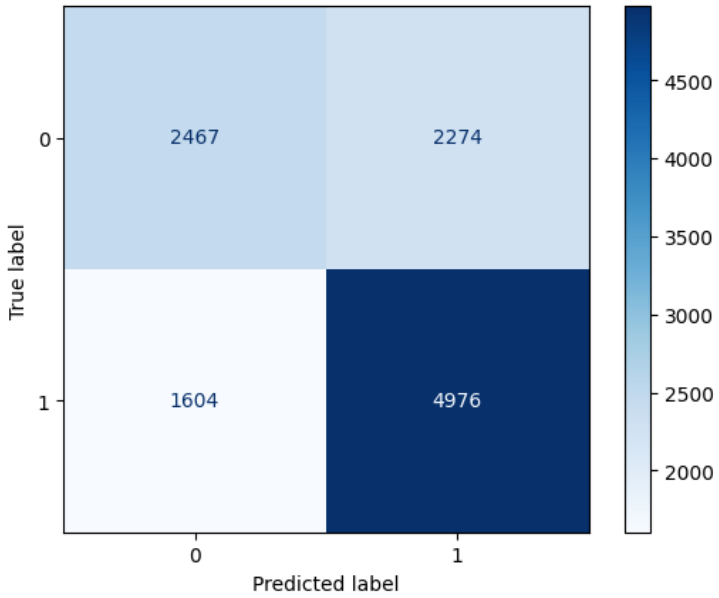
Entrenamiento del modelo

El modelo se entrena con un conjunto de noticias etiquetadas mediante la instrucción:

```
pipeline.fit(X_train, y_train)
```

Evaluación del modelo

Matriz de confusión tras evaluar en datos de prueba:



Métricas del modelo

Métrica	*Entrenamiento*	*Prueba*
Exactitud	89%	66%
Recall	93%	75.6%
Precisión	88%	68.6%
F1-score	90.7%	71.9%

Guardado y carga del modelo

El modelo se guarda y carga con:

```
dump(pipeline, "model.joblib")

pipeline_loaded = load("model.joblib")
```

Definiciones de re-entrenamiento

	Ajuste de parámetros (Fine-Tuning)	*Aprendizaje incremental*	*Re-entrenamiento completo*
Definición	Adaptación de un modelo pre-entrenado con datos específicos del dominio.	Aprendizaje continuo a partir de nuevos datos sin olvidar el conocimiento pasado.	Re-entrenamiento del modelo desde cero utilizando todo el conjunto de datos actualizado.
Ventaja principal	Aprovecha el conocimiento previo, entrenamiento más rápido con datos limitados.	Uso eficiente de recursos, adaptación en tiempo real.	Aborda eficazmente la deriva de datos, potencialmente mayor precisión.
Desventaja principal	Riesgo de sobreajuste y olvido catastrófico.	Olvido catastrófico, complejidad para manejar la deriva de conceptos.	Alto costo computacional y tiempo, potencial interrupción del servicio.

Dado que en este proyecto se utiliza Random Forest, el único enfoque viable es el re-entrenamiento completo. Esto se debe a que Random Forest no admite aprendizaje incremental, por lo que cada vez que se incorporan nuevos datos, es necesario reconstruir el modelo desde cero para garantizar su precisión y estabilidad.

Fuentes utilizadas

- https://medium.com/@sujathamudadla1213/advantages-and-disadvantages-of-fine-tuning-a-model-3c67231bc692
- https://www.hpe.com/us/en/what-is/fine-tuning.html

- https://www.analyticsvidhya.com/blog/2023/08/incremental-learning/
- https://www.analyticsvidhya.com/blog/2024/06/fine-tuning-vs-full-training-vs-training-from-scratch/
- https://www.iguazio.com/glossary/model-retraining/

API

La API permite realizar la detección de noticias falsas en español mediante un modelo de Machine Learning previamente entrenado. Además, ofrece la capacidad de reentrenar el modelo con nuevos datos para mejorar su desempeño.

Usuario y rol en la organización

La aplicación está diseñada para ser utilizada por analistas de datos y verificadores de noticias en medios de comunicación, agencias de verificación de información y plataformas de redes sociales. También puede ser útil para investigadores que estudian la propagación de noticias falsas.

Conexión con el proceso de negocio

En un entorno donde la desinformación es un problema crítico, esta aplicación permite a los analistas identificar automáticamente noticias falsas con base en modelos de aprendizaje automático. Facilita la toma de decisiones rápidas sobre la credibilidad del contenido antes de su difusión.

Importancia de la aplicación

Esta aplicación ofrece varios beneficios clave: reduce el tiempo y esfuerzo humano requerido para analizar grandes volúmenes de información, mejora la precisión utilizando modelos de machine learning entrenados con datos históricos, permite el reentrenamiento continuo para mejorar su precisión con nuevos datos y, a través de la API REST, puede integrarse con otras plataformas y sistemas internos para una verificación automática de noticias en tiempo real.

Endpoints y funcionamiento

La API cuenta con dos endpoints principales:

- POST /predict: Recibe una lista de noticias y devuelve las predicciones junto con la probabilidad de cada clasificación.
- POST /retrain: Recibe un conjunto de datos con etiquetas y reentrena el modelo, devolviendo las nuevas métricas de desempeño.

Endpoint /predict

Método HTTP: POST

Descripción: Recibe una lista de descripciones de noticias y devuelve una clasificación (falsa o verdadera) junto con la probabilidad de la predicción.

Entrada: JSON con una lista de objetos en el siguiente formato:

```
[
  {"Descripcion": "Texto de la noticia a analizar"},
  {"Descripcion": "Otra noticia a analizar"}
]
```

Salida:

```
{
  "predictions": ["Falsa", "Verdadera"],
  "probabilities": [[0.8, 0.2], [0.3, 0.7]]
}
```

Funcionamiento:

1. Recibe los datos y los convierte en un DataFrame.
2. Llama a predict_proba del módulo utils.py, que utiliza el modelo cargado para generar predicciones y probabilidades.
3. Devuelve la clasificación de cada noticia y su probabilidad.

Endpoint /retrain

Método HTTP: POST

Descripción: Recibe datos etiquetados para reentrenar el modelo y mejorar su desempeño.

Entrada: JSON con una lista de noticias y su etiqueta correspondiente:

```
[
  {"Descripcion": "Texto de la noticia", "Label": "Falsa"},
  {"Descripcion": "Otra noticia", "Label": "Verdadera"}
]
```

Salida:

```
{
   "precision": 0.85,
   "recall": 0.82,
   "f1": 0.83
}
```


Funcionamiento:

1. Recibe los datos y los convierte en un DataFrame.
2. Llama a retrain_model del módulo utils.py, que divide los datos en entrenamiento y prueba, procesa los textos, y reentrena el clasificador.
3. Guarda el nuevo modelo en assets/model.joblib.
4. Devuelve las métricas de desempeño (precisión, recall y F1-score).

Dependencias

La API está desarrollada con FastAPI y requiere las siguientes librerías: Fastapi, pandas, sklearn, joblib, uvicorn (para ejecutar la API)