Universidad de los Andes
Facultad de Ingeniería
Programa de sistemas y computación

Proyecto 1 – Etapa 2

Clasificación de noticias falsas con modelos de machine learning

Integrantes:
Julián Rolón – 202215839
Javier Barrera – 202214779
Camilo Rey – 202116752

Asignatura: Inteligencia de negocios
Fecha de entrega: 29 de marzo de 2025
Ciudad: Bogotá, Colombia

Tabla de contenido

1. **Desarrollo de los pipelines**

   Este proyecto lleva a cabo la implementación de un pipeline de aprendizaje automático orientado a ayudarnos a clasificar noticias falsas en español. Para ello, se utiliza un conjunto de datos proveniente del archivo fake_news_spanish. csv, donde se combinan las columnas "Descripción" y "Título" para crear una única entrada de texto que servirá como input para los modelos de clasificación. La variable objetivo, denominada "Label", indica si una noticia es falsa o no. Para evitar inconvenientes relacionados con valores nulos, se utiliza la función. fillna(""), asegurando así que el texto esté completo antes de su procesamiento. Esto da como resultado la creación de una nueva columna llamada "Texto", que contendrá los datos textuales combinados, y la variable "y" almacenando las etiquetas asociadas con el tipo.

```
# Cargar datos
df = pd.read_csv("fake_news_spanish.csv", sep=";")
df['texto'] = df['Descripcion'].fillna("") + " " + df['Titulo'].fillna("")
y = df['Label']
```

   En cuanto a los modelos de clasificación, el código define un diccionario llamado MODELS, que incorpora cuatro enfoques diferentes, cada uno con parámetros específicos. La regresión logística se ajusta aplicando un balance de clases mediante el parámetro class_weight, además de implementar regularización L2 y utilizar el solver 'sag'. Por su parte, XGBoost se configura con 70 árboles de decisión y una profundidad máxima de 7. El modelo Random Forest también utiliza 70 árboles, pero sin un límite de profundidad. Finalmente, el modelo Naïve Bayes se basa en la probabilidad y emplea un parámetro de suavizado de alpha=0.5, que regula la influencia de los valores de baja frecuencia en el proceso de aprendizaje.

```
MODELS = {
    "logistic_regression": LogisticRegression(class_weight={0: 1.2, 1: 0.8}, penalty='l2', solver='sag'),
    "xgboost": XGBClassifier(n_estimators=70, max_depth=7),
    "random_forest": RandomForestClassifier(n_estimators=70, max_depth=None),
    "naive_bayes": MultinomialNB(alpha=0.5)
}
```

   Para asegurar un flujo de procesamiento eficiente, el código incorpora un pipeline que automatiza el tratamiento de los datos antes de su clasificación. Este pipeline se compone de tres pasos cruciales. En primer lugar, se lleva a cabo un preprocesamiento mediante la clase Preprocessor(), la cual se encarga de limpiar y normalizar el texto. Este proceso puede incluir la eliminación de signos de puntuación, la conversión a minúsculas y la eliminación de palabras vacías o irrelevantes. A continuación, se realiza la vectorización con TfidfVectorizer(), una técnica que transforma el texto en una matriz numérica, asignando pesos a las palabras según su relevancia en el

conjunto de datos. Finalmente, se añade el clasificador correspondiente, el cual puede ser cualquiera de los cuatro modelos previamente definidos en MODELS.

```
pipeline = Pipeline([
    ('preprocessing', Preprocessor()),
    ('tfidf', TfidfVectorizer()),
    ('classifier', model)
])
```

El entrenamiento del pipeline se ejecuta utilizando los datos textuales y sus respectivas etiquetas (y). Una vez finalizado este proceso, los modelos entrenados se almacenan en la carpeta models/, utilizando el formato Joblib. Este enfoque permite la reutilización de los modelos sin necesidad de volver a entrenarlos, optimizando así el tiempo de procesamiento para futuras predicciones. Para valorar la eficacia del proceso, se utiliza la función time. time(), que permite calcular el tiempo total de entrenamiento de cada modelo y del pipeline en su totalidad. Esta métrica es muy importante para evaluar el rendimiento de los algoritmos y para identificar cuál de los modelos proporciona la mejor combinación de precisión y velocidad de ejecución. Así, la implementación de este pipeline simplifica la automatización en el análisis de noticias falsas, ofreciendo un método estructurado y eficiente para la detección de información engañosa.

2. Desarrollo de la API

Para el desarrollo de esta aplicación nos apoyamos en el uso de **FastAP**. Esta herramienta permitió brindar funcionalidades como predicción y reentrenamiento del modelo desde 0 en una interfaz sencilla que se comunica con el servidor.

Todo el flujo de funcionamiento se organiza a través de distintos archivos, cada uno con una responsabilidad específica, que explicamos a continuación:

2.1 Lógica de la API en el archivo main.py

Este archivo contiene la lógica principal de la API y el servidor. Aquí se definen los endpoints o rutas disponibles para los usuarios:

GET /: Verifica que la API esté disponible.

POST /predict: Recibe un texto (descripción + título) y devuelve la predicción del modelo.

POST /retrain: Recibe un conjunto de datos nuevos para reentrenar el modelo.

Además, en este archivo se activa CORS para permitir que aplicaciones frontend accedan a la API. Se mantiene un diccionario de modelos disponibles (MODELOS) donde cada clave es un nombre de modelo y el valor es su ruta en el sistema de archivos.

structure.py – Validación de datos

Este archivo contiene la definición de los modelos de entrada para los endpoints. Se utiliza la librería pydantic para validar que la estructura de los datos enviados por el cliente sea correcta.

Contiene dos clases principales:

- DataModel: Se usa para la predicción. Requiere el nombre del modelo a usar, la descripción de la noticia y el título.
- ReentrenamientoModel: Se usa para enviar nuevos datos que servirán para reentrenar el modelo. Además de los textos, incluye la etiqueta (0 = falsa, 1 = verdadera).

preprocesamiento.py – Limpieza y normalización de texto

Aquí se encuentra el preprocesador personalizado que se incorpora al pipeline del modelo. Esta clase Preprocessor extiende los componentes de sklearn para poder ser integrada directamente.

Este componente realiza: tokenización del texto, conversión de números a palabras, conversión a minúsculas, eliminación de signos de puntuación, normalización de caracteres especiales, eliminación de stopwords en español, aplicación de stemming y lematización. Esto con el fin de poder brindar a los modelos más uniformidad y limpieza en los datos que van a manejar.

3. **Resultados**

Metricas de los modelos:

3.1 Modelo XGBoost
- Precisión: 0.97 (0), 0.90 (1)
- Recall: 0.85 (0), 0.98 (1)
- F1-Score: 0.91 (0), 0.94 (1)

Exactitud general: 0.93

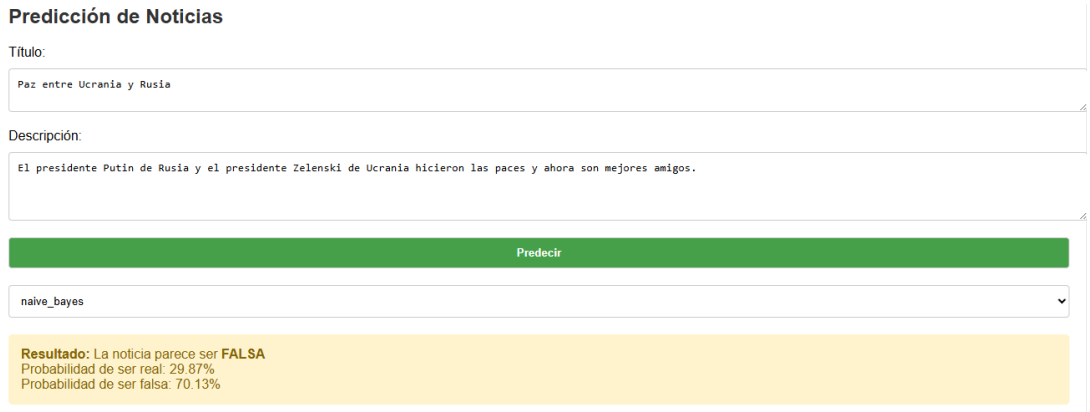
3.2 Modelo Random Forest
- Precisión: 0.95 (0), 0.90 (1)
- Recall: 0.85 (clase 0), 0.97 (1)
- F1-Score: 0.90 (0), 0.94 (1)

Exactitud general: 0.92

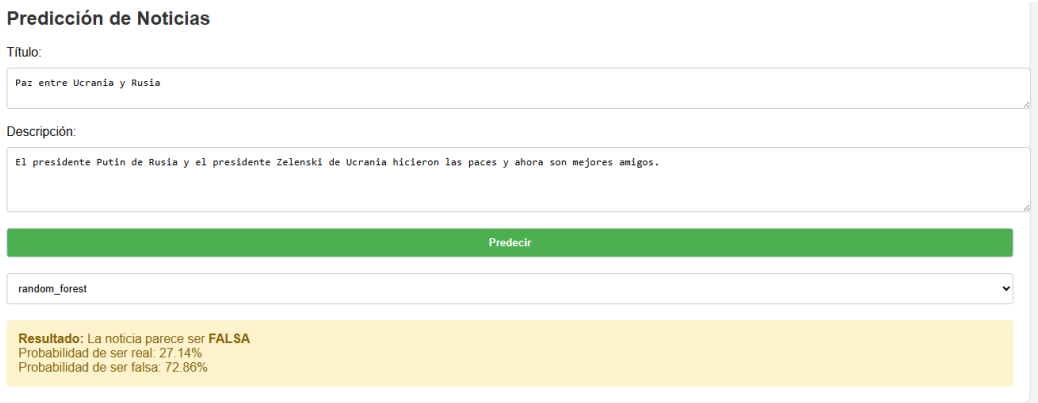
3.3 Modelo Naive Bayes:
- Precisión: 0.93 (0), 0.80 (1)
- Recall: 0.65 (0), 0.96 (1)
- F1-Score: 0.77 (0), 0.87 (1)
Exactitud general: 0.84

Análisis de resultados de los modelos ejecutados
A continuación se les pasara una misma noticia a los diferentes modelos, para ver cómo se desempeñan:

Empezando con naive Bayes. Podemos observar como este modelo clasifica de forma correcta la noticia, dando como resultado una noticia falsa, donde se cree que el 70% de la misma contiene caracteres que la hace ser clasificada de esta manera.

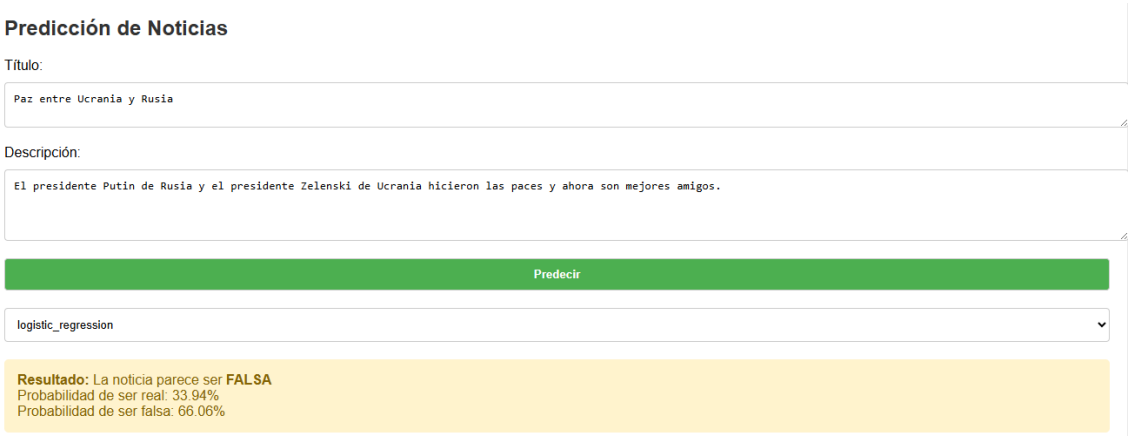


El siguiente modelo que revisamos fue Random forest, donde esté también clasifico de manera correcta la noticia como falsa. Pero teniendo una probabilidad del 72% mayor a la que nos dio el modelo de Naive Bayes.



Continuando con XGboost, es el modelo que más porcentaje de noticia falsa nos dios, siendo este de un 82% siendo el modelo que mejor se desempeña clasificando esta noticia.

Por último, el modelo de logistic_regression tambien clasifico de manera correcta la noticia, pero este modelo nos dio la probabilidad más baja al clasificarlo, dándonos a entender que en algún momento puede presentar fallas clasificando algunas noticias.



Para terminar naive bayes y Xgboost al reentrenarse adicionan esos datos a su mismos para tener más capacidad a la hora de clasificar, mientras que random forest y regresion logistica, cada que se reentrenan sus datos anteriores se eliminan, impidiendo que sea perdurable en el tiempo, pues se le tendrían que adicionar unos datos bastante grandes.

4. **Conclusiones**

   El desarrollo de este proyecto ha permitido la implementación de un pipeline automatizado y una API funcional para la detección de noticias falsas en español. La combinación de técnicas de preprocesamiento, vectorización y modelos de clasificación ha demostrado ser efectiva en la identificación de información engañosa, con métricas de rendimiento que destacan la capacidad de los algoritmos para diferenciar entre noticias falsas y verdaderas.

   En términos de desempeño, XGBoost se posicionó como el modelo más preciso, con una exactitud general del 93% y el mayor nivel de confianza en sus predicciones. Random Forest presentó resultados similares,

mientras que Naïve Bayes, aunque menos preciso, sigue siendo una opción viable por su rapidez y simplicidad. La regresión logística, aunque funcional, mostró la menor probabilidad de clasificación correcta en ciertos casos.

El desarrollo de la API con FastAPI ha facilitado la accesibilidad del modelo, permitiendo la predicción y el reentrenamiento dinámico mediante solicitudes HTTP. La estructuración modular del código, junto con el uso de validación de datos y preprocesamiento eficiente, garantiza un sistema escalable y reutilizable.

5. Trabajo en equipo

Camilo Rey (Líder - Ingeniero de Datos)

Fue el encargado de diseñar la arquitectura general del sistema, incluyendo la estructura de carpetas, la integración entre el backend y el frontend, y la definición de los endpoints de la API REST utilizando FastAPI. Además, desarrolló por completo el frontend interactivo en HTML que permite al usuario realizar predicciones de noticias falsas y subir archivos .csv para reentrenar los modelos.

Camilo lideró la división de tareas entre los integrantes del equipo y supervisó el correcto funcionamiento del sistema completo.

Javier Barrera (Ingeniero de Datos)

Fue el responsable de unificar todos los pipelines de modelos (Naive Bayes, Random Forest, Logistic Regression y XGBoost) bajo una misma lógica de entrenamiento, estructurando y guardando cada uno en archivos independientes con joblib.

Además, desarrolló el procesamiento para permitir la carga de múltiples instancias de noticias para la predicción, lo cual fue clave para permitir el análisis en lote desde la interfaz web.

También diseñó y ajustó la lógica del reentrenamiento incremental para el modelo de Naive Bayes, y colaboró activamente en habilitar este mismo mecanismo para XGBoost, mejorando así el entrenamiento sin perder los datos aprendidos anteriormente.

Julián Rolón (Ingeniero de Software)

Se encargó de realizar el video de presentación de la segunda etapa del proyecto, donde se explica visualmente el funcionamiento de la aplicación.

Colaboró en la documentación técnica, explicando el flujo de datos entre los componentes y la lógica de predicción y reentrenamiento.

También apoyó en el análisis de resultados, asegurando que los modelos alcanzaran métricas aceptables y comprendiendo los posibles errores o desviaciones en el proceso de clasificación.

Bajo el criterio del líder del equipo, se considera que la distribución del trabajo fue equitativa:

Camilo Rey: 33.33%

Javier Barrera: 33.33%

Julián Rolón: 33.33%

Se adjunta el enlace del video:

https://www.youtube.com/watch?v=nwcijzbSVi4&feature=youtu.be&themeRefresh=1