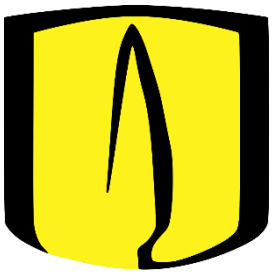
Proyecto 1 – Etapa 2

Herramientas de Analítica de Textos para la Identificación de Desinformación Política



Grupo 9

Carol Sofía Florido Castro - 202111430
Juan Martin Vásquez Cristancho - 202113314
Natalia Villegas Calderón – 202113370

Universidad de Los Andes

Departamento de Ingeniería de Sistemas y Computación

Inteligencia de Negocios - ISIS 3301

Bogotá D.C., Colombia

2025

1. Ingeniería de datos

El objetivo del proyecto es desarrollar un sistema robusto para la clasificación de noticias con el fin de detectar desinformación política. Este informe explica el proceso de ingeniería de datos y las decisiones técnicas adoptadas, desde la preparación de los datos hasta la persistencia del modelo a través de un pipeline integral.

1.1. Adquisición y limpieza de Datos

Para garantizar la calidad de la información, se inició con una estricta fase de limpieza de datos. Se eliminaron registros incompletos en campos críticos —como “Título” y “Descripción”—, ya que estos son esenciales para el análisis textual y la extracción de características relevantes. Además, se combinó el contenido de “Título” y “Descripción” en un único campo denominado “Texto”. Esta decisión se fundamenta en la necesidad de obtener una representación más completa del contenido, permitiendo al modelo capturar tanto el contexto como las sutilezas semánticas de cada noticia.

```python
def dropna_and_combine_text(df):
    df_clean = df.dropna(subset=['Titulo', 'Descripcion']).copy()
    df_clean['Texto'] = df_clean['Titulo'] + ' ' + df_clean['Descripcion']
    return df_clean['Texto']
```

Ilustración 1 Código de adquisición y limpieza de datos

1.2. Procesamiento de texto

El proceso de preprocesamiento del texto comienza con la normalización de las palabras, donde se expanden contracciones para evitar ambigüedades y mejorar la calidad del análisis. Luego, el texto se divide en unidades más pequeñas, generalmente palabras, permitiendo su tratamiento individual. Posteriormente, se aplican técnicas de limpieza, eliminando caracteres innecesarios, signos de puntuación y palabras irrelevantes que no aportan valor al modelo. Finalmente, se lleva a cabo un proceso de reducción léxica para estandarizar las palabras y minimizar la variabilidad, asegurando que términos con un significado similar sean tratados de manera uniforme. Todo este flujo garantiza que la información procesada sea más estructurada y adecuada para su uso en modelos de aprendizaje automático.

```python
def procesar(texto):
    texto = contractions.fix(texto)
    palabras = word_tokenize(texto)
    palabras = preprocesar(palabras)
    palabras = aplicarStemmingYLematizacion(palabras)
    return ' '.join(palabras)
```

Ilustración 2 Código flujo de procesamiento de texto

1.2.1. Expansión de contracciones en el texto

En este paso se realiza la expansión de contracciones en el texto, es decir, se reemplazan palabras abreviadas o contraídas por su forma completa. Esto es especialmente útil en el procesamiento de lenguaje natural, ya que ayuda a reducir la ambigüedad y mejora la coherencia del texto.

```
def procesar(texto):
    texto = contractions.fix(texto)
    palabras = word_tokenize(texto)
    palabras = preprocesar(palabras)
    palabras = aplicarStemmingYLematizacion(palabras)
    return ' '.join(palabras)
```

Ilustración 3 Código expansión de contracciones en el texto

1.2.2. Tokenización

El texto se divide en *tokens* o unidades básicas (usualmente palabras). La tokenización segmenta la cadena de texto en componentes manejables, lo que permite aplicar transformaciones posteriores de manera individualizada. Este proceso es crítico porque define la base sobre la cual se construirá el vocabulario y se realizará la vectorización.

```
def procesar(texto):
    texto = contractions.fix(texto)
    palabras = word_tokenize(texto)
    palabras = preprocesar(palabras)
    palabras = aplicarStemmingYLematizacion(palabras)
    return ' '.join(palabras)
```

Ilustración 4 Código tokenización

1.2.3. Preprocesamiento del Texto

El preprocesamiento del texto es una etapa fundamental en proyectos de procesamiento de lenguaje natural (NLP), ya que impacta directamente en la calidad de las características que el modelo utilizará para aprender. En este proyecto, se aplicaron varias técnicas de preprocesamiento con el objetivo de normalizar, limpiar y transformar los datos textuales, asegurando que el modelo reciba información consistente y relevante. A continuación, se explica cada paso con mayor profundidad:

```
def preprocesar(palabras):
    palabras = aMinusculas(palabras)
    palabras = eliminarNumeros(palabras)
    palabras = eliminarPuntuacion(palabras)
    palabras = removerNoAscii(palabras)
    palabras = eliminarStopwords(palabras)
    return palabras
```

Ilustración 5 Código preprocesamiento de texto

1.2.3.1. Normalización

El primer paso consiste en transformar todo el texto a minúsculas. Esto se realiza para evitar que palabras idénticas en diferentes casos (por ejemplo, “Política” y “política”) sean tratadas como términos distintos, lo que incrementaría la dimensionalidad del espacio de características sin aportar valor adicional. La normalización garantiza que se reduce la variabilidad léxica innecesaria.

```python
def aMinusculas(palabras):
    return [palabra.lower() for palabra in palabras]
```

Ilustración 6 Código normalización

1.2.3.2. Eliminación de Ruido

Eliminación de Números y Puntuación: Los números y signos de puntuación a menudo aportan poco a la semántica en tareas de clasificación de texto, especialmente en contextos donde el significado radica en el contenido textual descriptivo. Eliminar estos elementos ayuda a reducir el ruido, disminuyendo la cantidad de *tokens* irrelevantes que podrían interferir en el proceso de aprendizaje.

```python
def eliminarNumeros(palabras):
    return [re.sub(r'\d+', '', palabra) for palabra in palabras]

def eliminarPuntuacion(palabras):
    return [palabra for palabra in palabras if palabra not in string.punctuation]
```

Ilustración 7 Código eliminación de números y puntuación

Filtrado de Caracteres No ASCII: Se aplicó una verificación para retener solo aquellos caracteres que son ASCII. Esto es particularmente importante en contextos donde se desea trabajar con un lenguaje específico (en este caso, español) y se desea evitar la inclusión de caracteres especiales o símbolos que puedan ser producto de errores de codificación. Esto simplifica la limpieza y asegura un conjunto de datos más homogéneo.

```python
def removerNoAscii(palabras):
    return [palabra for palabra in palabras if palabra.isascii()]
```

Ilustración 8 Código filtrado de caracteres no ASCII

1.2.3.3. Eliminación de *Stopwords*

Las *stopwords* son palabras de alta frecuencia que, en la mayoría de los casos, aportan poco valor semántico para la tarea de clasificación. Ejemplos comunes en español incluyen “el”, “la”, “y”, “de”, entre otros. La eliminación de estas palabras se fundamenta en dos aspectos técnicos: por un lado al eliminar términos de alta frecuencia, pero baja información, se reduce el tamaño del vocabulario y, por ende, la complejidad del espacio vectorial. Por otro lado, con un vocabulario más centrado en palabras significativas, el modelo puede aprender patrones y relaciones que son realmente indicativos de la clase objetivo.

```python
def eliminarStopwords(palabras):
    sw = set(stopwords.words('spanish'))
    return [palabra for palabra in palabras if palabra not in sw]
```

Ilustración 9 Código eliminación de Stopwords

1.2.4. Stemming y Lematización

Stemming: El *stemming* es un proceso que reduce las palabras a su raíz (*stem*). Por ejemplo, palabras como “jugando”, “jugará” y “jugado” se convierten en una forma común que representa el concepto básico “jugar”. Este proceso es particularmente útil para agrupar variantes morfológicas

de una palabra, reduciendo la dispersión en el vocabulario. Sin embargo, el *stemming* puede ser agresivo, eliminando sufijos de manera que la raíz resultante no siempre es una palabra válida.
Lematización: La lematización es una técnica más sofisticada que transforma cada palabra a su forma canónica (lema) utilizando información gramatical y léxica. A diferencia del *stemming*, la lematización toma en cuenta el contexto y la parte de la oración (sustantivo, verbo, adjetivo, etc.), lo que genera resultados más precisos y significativos. En este proyecto se aplicó una combinación de *stemming* y lematización (utilizando herramientas como *SnowballStemmer* y *WordNetLemmatizer*) para beneficiarse de la reducción de variabilidad sin perder la semántica esencial de cada término.

```python
def aplicarStemmingYLematizacion(palabras):

    stemmer = SnowballStemmer('spanish')
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(stemmer.stem(palabra)) for palabra in palabras]
```

Ilustración 10 Código stemming y lematización

Cada una de estas etapas se implementó de forma secuencial y encapsulada en funciones que permiten aplicar el mismo conjunto de transformaciones de manera consistente tanto en los datos de entrenamiento como en los nuevos datos para inferencia. La modularidad en el diseño de estas funciones facilita su integración en un pipeline, asegurando que cada nuevo dato pase por los mismos procesos de normalización, limpieza y transformación. Esto es crítico para mantener la coherencia entre el entrenamiento y la inferencia, evitando discrepancias que puedan degradar el rendimiento del modelo.

1.3. Extracción de Características mediante TF-IDF

Una vez que el texto ha sido preprocesado, se emplea el método TF-IDF (*Term Frequency-Inverse Document Frequency*) para transformar los datos textuales en una representación numérica. Esta técnica pondera la importancia de cada término dentro de un documento y en relación con el corpus completo, ayudando a identificar las palabras que tienen mayor capacidad discriminatoria para la clasificación.

```python
tf_idf_vectorizer = TfidfVectorizer(decode_error='ignore')
X_tf_idf = tf_idf_vectorizer.fit_transform(X_train['Texto'])
y_tf_idf = Y_train['Label']
```

Ilustración 11 Código vectorización

```python
vectorizer = joblib.load("vectorizer.joblib")

def vectorization_function_transform(x):
    return vectorizer.transform(x).toarray()
```

Ilustración 12 Código cargar vectorizador y función vectorizar texto

1.4. Selección y Optimización del Modelo

Se eligió el *Gradient Boosting Classifier* como el modelo de clasificación debido a su capacidad para combinar múltiples modelos débiles en un clasificador altamente robusto y preciso. La optimización del modelo se realizó mediante una búsqueda exhaustiva de hiperparámetros utilizando *GridSearchCV.* Se evaluaron diferentes combinaciones de parámetros —como el criterio de división, el número de estimadores y la profundidad máxima de los árboles— en un proceso de validación cruzada (CV=3). Esta metodología permitió identificar la configuración óptima,

asegurando que el modelo minimice tanto el sesgo como la varianza, y se adapte de manera efectiva a la estructura de los datos preprocesados.

```
param_grid = {'criterion':['friedman_mse'],'n_estimators': [300, 500], 'max_depth': [3, 5]}
gb_clf = GradientBoostingClassifier(random_state=0)
mejor_modelo = GridSearchCV(gb_clf, param_grid, cv=3, n_jobs=-1)
mejor_modelo.fit(X_smote, Y_smote.values.ravel())
mejor_modelo.best_params_
```

Ilustración 13 Código creación y optimización del modelo

1.5. Construcción del pipeline

El pipeline comienza eliminando registros con valores nulos en campos críticos, como “Título” y “Descripción”, y combinando estos en un solo campo denominado “Texto” para asegurar que se capture el contexto completo. Posteriormente, se aplica un preprocesamiento exhaustivo en el que el texto se normaliza a minúsculas, se tokeniza y se limpia eliminando números, signos de puntuación y caracteres no ASCII; además, se filtran las *stopwords* y se utilizan técnicas de *stemming* y lematización para reducir la variabilidad léxica. Una vez transformado, el texto se vectoriza mediante TF-IDF, creando vectores numéricos que ponderan la relevancia de cada término dentro del documento y del corpus total, lo que garantiza un espacio de características consistente. Finalmente, este vector es ingresado en el modelo de clasificación optimizado, en este caso un *Gradient Boosting Classifier*, que genera la predicción final. Integrar todas estas etapas en un pipeline permite procesar de forma coherente y reproducible cualquier nuevo dato, asegurando la consistencia entre el entrenamiento y la inferencia, y facilitando la persistencia y el despliegue del sistema.

```
pipeline = Pipeline([
    ('dropna_and_combine', FunctionTransformer(dropna_and_combine_text)),
    ('text_preprocessing', FunctionTransformer(text_preprocessing_function)),
    ('vectorization', FunctionTransformer(vectorization_function_transform)),
    ('classification', model)
])
```

Ilustración 14 Código pipeline

1.6. Construcción de la API

La API ha sido desarrollada utilizando FastAPI, lo que permite definir *endpoints* RESTful de manera rápida y eficiente, con validación de datos a través de Pydantic. Esta solución se diseñó para cubrir dos necesidades fundamentales: la generación de predicciones a partir de datos textuales y el reentrenamiento del modelo de clasificación de noticias. La modularidad y la consistencia en el procesamiento de datos son elementos clave para asegurar que el sistema opere de forma confiable tanto en producción como en entornos de prueba.

```
pipeline = load("pipeline.joblib")
app = FastAPI()
You, 4 hours ago | 1 author (You)
class NewsItem(BaseModel):
    ID: str
    Titulo: str
    Descripcion: str
    Fecha: str
You, 16 hours ago | 1 author (You)
class RetrainData(BaseModel):
    ID: str
    Titulo: str
    Descripcion: str
    Fecha: str
    Label: int
```

Ilustración 15 Entrada esperada para la API

1.6.1. Endpoint de Predicción Individual (/predict/)

Este *endpoint* recibe un único objeto que contiene los campos “Titulo”, “Descripción”, “ID” y “Fecha”. Los datos se transforman en un *DataFrame* y se procesan mediante un *pipeline* previamente

persistido, el cual integra todas las transformaciones necesarias (limpieza, preprocesamiento, vectorización) y el modelo de clasificación. El resultado devuelto es la etiqueta predicha y las probabilidades asociadas, permitiendo una interpretación directa de la salida del modelo.

```python
@app.post("/predict/")
def predict(news: NewsItem):
    df = pd.DataFrame([news.dict()])
    prediction = pipeline.predict(df)[0]
    probabilidades = pipeline.predict_proba(df)[0]
    return {"prediction": int(prediction), "probabilidades": probabilidades.tolist()}
```

Ilustración 16 EndPoint predicción individual

1.6.2. Endpoint de Predicción en Lote (/predictMany)

Para facilitar el procesamiento masivo, este *endpoint* acepta una lista de objetos con los mismos campos utilizados en la predicción individual. Al igual que en el caso anterior, se transforma la lista en un *DataFrame* y se aplica el pipeline para generar las predicciones. Se retorna un conjunto de etiquetas y las probabilidades correspondientes para cada entrada, lo que simplifica la integración con aplicaciones que requieren análisis de grandes volúmenes de datos.

```python
@app.post("/predictMany")
def predict(news_list: List[NewsItem]):
    df = pd.DataFrame([news.dict() for news in news_list])
    predictions = pipeline.predict(df)
    probabilidades = pipeline.predict_proba(df)
    return {
        "predictions": predictions.tolist(),
        "probabilidades": [list(probs) for probs in probabilidades]
    }
```

Ilustración 17 Código EndPoint predicción en conjunto

1.6.3. Endpoint de Reentrenamiento (/retrain/)

Este *endpoint* se encarga de actualizar el modelo de clasificación. Recibe una lista de datos de reentrenamiento. Los datos se cargan en un *DataFrame* y se separan en características y etiquetas. Posteriormente, se aplica el mismo preprocesamiento que utiliza el *pipeline* (excluyendo la etapa final de clasificación) y se realiza una división entre conjuntos de entrenamiento y validación. El modelo (un *Gradient Boosting Classifier*) se entrena desde cero utilizando esta partición, y se calculan métricas de desempeño (F1, *Recall, Precision y Accuracy*) para ambos conjuntos. Una vez evaluado, el *pipeline* se actualiza reemplazando el clasificador anterior por el nuevo modelo y se persiste tanto el pipeline completo como el modelo reentrenado, asegurando que las futuras predicciones se realicen utilizando la versión más reciente.

```python
@app.post("/retrain/")
def retrain(data: list[RetrainData]):
    global df_historical
    df_historical = pd.DataFrame([item.dict() for item in data])
    X = df_historical.drop(columns=["ID","Label"])
    y = df_historical["Label"]
    X_processed = pipeline[:-1].transform(X)
    X_train, X_val, y_train, y_val = train_test_split(X_processed, y, test_size=0.2, random_state=42)
    new_model = GradientBoostingClassifier(n_estimators=500, max_depth=5, criterion="friedman_mse")
    new_model.fit(X_train, y_train)

    y_train_pred = new_model.predict(X_train)
    train_f1 = f1_score(y_train, y_train_pred, average="weighted")
    train_recall = recall_score(y_train, y_train_pred, average="weighted")
    train_precision = precision_score(y_train, y_train_pred, average="weighted")
    train_accuracy = accuracy_score(y_train, y_train_pred)
```

Ilustración 18 EndPoint reentrenamiento, entrenamiento del nuevo modelo

```python
y_train_pred = new_model.predict(X_train)
train_f1 = f1_score(y_train, y_train_pred, average="weighted")
train_recall = recall_score(y_train, y_train_pred, average="weighted")
train_precision = precision_score(y_train, y_train_pred, average="weighted")
train_accuracy = accuracy_score(y_train, y_train_pred)

y_val_pred = new_model.predict(X_val)
val_f1 = f1_score(y_val, y_val_pred, average="weighted")
val_recall = recall_score(y_val, y_val_pred, average="weighted")
val_precision = precision_score(y_val, y_val_pred, average="weighted")
val_accuracy = accuracy_score(y_val, y_val_pred)
```

Ilustración 19 Generación de métricas de interés para el EndPoint de reentrenamiento

```python
pipeline.steps[-1] = ("classification", new_model)
dump(pipeline, "pipelineRetrain.joblib")
dump(new_model, "modelRetrain.joblib")
return {
    "Training Metrics": {
        "F1": train_f1,
        "Recall": train_recall,
        "Precision": train_precision,
        "Accuracy": train_accuracy
    },
    "Validation Metrics": {
        "F1": val_f1,
        "Recall": val_recall,
        "Precision": val_precision,
        "Accuracy": val_accuracy
    }
}
```

Ilustración 20 Respuesta dada por el EndPoint de reentrenamiento

1.6.3.1. Método de reentrenamiento

Según Géron (2022), el reentrenamiento puede abordarse desde diferentes perspectivas. En el reentrenamiento desde cero, el modelo se vuelve a entrenar utilizando la totalidad de los datos disponibles, combinando tanto los registros históricos como los nuevos. Esta estrategia permite aprovechar al máximo la información acumulada, eliminando posibles sesgos o errores que puedan haberse generado en versiones anteriores del modelo. Sin embargo, una limitación importante de este método es el elevado costo computacional y el tiempo requerido para procesar grandes volúmenes de datos. Por otro lado, el entrenamiento incremental o aprendizaje en línea consiste en actualizar el modelo existente con nuevos datos sin desechar completamente el conocimiento previo. Esta aproximación resulta eficiente en términos de recursos y permite adaptarse rápidamente a cambios en la distribución de los datos, aunque puede generar problemas de acumulación de errores si no se gestiona cuidadosamente. Finalmente, el *transfer learning* o ajuste fino se basa en reutilizar un modelo preentrenado en un amplio conjunto de datos y posteriormente afinarlo con datos específicos de la nueva tarea. Este método reduce considerablemente el tiempo de entrenamiento y es especialmente útil cuando se dispone de pocos datos nuevos, aunque su eficacia puede verse limitada si existe una diferencia significativa entre el dominio original y el de la tarea a resolver.

En este proyecto se evaluaron las tres estrategias y, tomando en cuenta que el modelo utilizado (GradientBoostingClassifier) no soporta el aprendizaje incremental y dada la disponibilidad de una cantidad considerable de datos históricos y nuevos, se optó por implementar el reentrenamiento desde cero. Esta decisión permite aprovechar de forma integral toda la información disponible, lo cual es esencial para lograr un modelo robusto y de alta precisión en la detección de desinformación política. Al elegir esta estrategia, se garantiza que el modelo se actualice de manera completa, eliminando la dependencia de versiones previas que pudieran estar sesgadas o contener errores acumulados.

2. Desarrollo de la aplicación y justificación

2.1 Usuarios y roles

La aplicación desarrollada está pensada para ser utilizada por editores de medios digitales, periodistas, equipos de verificación de datos (*fact-checkers*) y analistas políticos o de contenido, tanto en organizaciones de medios de comunicación como en entidades gubernamentales encargadas de combatir la desinformación. Estos usuarios desempeñan un papel clave en la curaduría de contenidos informativos que se difunden masivamente en canales públicos y privados, y requieren herramientas confiables para evaluar la veracidad de las noticias antes de su publicación o análisis.

En contextos institucionales, también puede ser útil para áreas de comunicaciones institucionales, oficinas de transparencia o análisis político, que buscan comprender cómo circulan narrativas falsas y prevenir su impacto en la opinión pública. Por otro lado, en un nivel más técnico, usuarios como científicos de datos o ingenieros de datos pueden hacer uso del componente de reentrenamiento para actualizar el modelo y adaptarlo a nuevas tendencias de desinformación.

2.2. Conexión de la aplicación con el proceso de negocio apoyado

La aplicación se integra directamente con el proceso de negocio de verificación automatizada de contenido digital, permitiendo reducir la carga operativa de revisar manualmente cada noticia que se publica o analiza. Esto es especialmente relevante en organizaciones donde el volumen de información es alto y el tiempo para la validación es limitado. La herramienta automatiza el análisis y la clasificación de noticias, y permite a los usuarios ingresar información individual o por lotes (en CSV), recibiendo predicciones acompañadas de probabilidades. Además, la funcionalidad de reentrenamiento del modelo permite actualizar la lógica de clasificación con base en nuevos datos verificados, cerrando el ciclo de mejora continua del sistema.

2.3. Importancia de la existencia de la aplicación

En un entorno donde la desinformación política circula a gran velocidad y escala, contar con una herramienta automatizada que clasifique el contenido como falso o veraz permite reducir el impacto de noticias maliciosas, mejorar la calidad del debate público y fortalecer la confianza en los canales informativos. La aplicación disminuye la carga sobre los analistas humanos, agiliza procesos de curaduría editorial y brinda soporte técnico para procesos internos de control de contenido. Desde un enfoque institucional, esta aplicación puede contribuir al desarrollo de políticas públicas de comunicación responsable, transparencia informativa y educación digital, al ofrecer una solución concreta basada en analítica de textos.

2.4. Elaboración y descripción de la aplicación

La aplicación cuenta con una *landing* page informativa, donde se presenta el propósito del sistema, su impacto en la lucha contra la desinformación y nubes de palabras generadas a partir del análisis exploratorio de más de 57.000 noticias en español. Esta visualización permite al usuario entender las diferencias léxicas entre noticias verídicas y falsas, y crea conciencia sobre patrones comunes en la desinformación.

Desde la parte superior de la página, el usuario puede acceder a tres funcionalidades principales:

- **Predicción individual de noticias:** En esta sección, el usuario puede ingresar manualmente una noticia, completando campos como fecha, ID, título y cuerpo del texto. Al hacer clic en “Clasificar”, la aplicación devuelve si la noticia es falsa o no, junto con la probabilidad asociada a cada predicción. Esta funcionalidad es útil para analistas que trabajan con casos puntuales o material en tiempo real.
- **Predicción por lotes:** Esta pestaña permite al usuario subir un archivo CSV con múltiples noticias, las cuales son procesadas en conjunto. La aplicación devuelve un archivo con las predicciones y probabilidades para cada una, facilitando así análisis masivos y reportes en tiempo reducido.
- **Reentrenamiento del modelo:** Dirigida a usuarios con perfil técnico o experto, esta sección permite subir un conjunto de noticias etiquetadas para actualizar el modelo. Se ejecuta un nuevo

entrenamiento desde cero, se actualiza el *pipeline* y se presentan métricas clave como precisión, *recall* y F1-score del modelo actualizado. Esta funcionalidad es especialmente útil para ajustar el sistema ante cambios en la narrativa política o la aparición de nuevos patrones de desinformación.

2.5. Recursos requeridos

Para entrenar, ejecutar y desplegar el modelo analítico junto con la aplicación, se requieren ciertos recursos informáticos tanto a nivel de *hardware* como de *software*. Desde el punto de vista técnico, se necesita un equipo con al menos 4 núcleos de procesamiento, 8 GB de memoria RAM y espacio suficiente en disco para almacenar archivos de entrada y salida generados por el sistema. Aunque el dataset original (fake_news_spanish.csv) utilizado en las etapas de análisis y entrenamiento ocupa aproximadamente 20 MB, su procesamiento y transformación durante el *pipeline* pueden generar estructuras intermedias más pesadas, especialmente al aplicar técnicas como TF-IDF. Por lo tanto, se recomienda contar con un entorno que permita manipular eficientemente datos textuales de mediano tamaño.

En cuanto al software, se utiliza Python 3.11 y bibliotecas como scikit-learn, pandas, joblib, nltk y FastAPI para la construcción del modelo, el preprocesamiento y el desarrollo de la API. La aplicación se despliega con Uvicorn como servidor ASGI (*Asynchronous Server Gateway Interface),* y puede ejecutarse localmente o ser alojada en plataformas en la nube como AWS o GCP. El pipeline y el modelo son persistidos mediante *joblib*, lo que permite cargarlos fácilmente al momento de realizar predicciones o ejecutar un proceso de reentrenamiento.

2.6. Integración organizacional y riesgos

Desde el punto de vista organizacional, la aplicación puede convertirse en una herramienta clave para procesos como curaduría editorial, monitoreo de contenido digital y verificación de información. Está orientada a entidades que manejan grandes volúmenes de noticias o información sensible, como medios de comunicación, instituciones gubernamentales u organizaciones civiles interesadas en combatir la desinformación política. A través de su API REST, la solución puede integrarse con sistemas internos de análisis o publicación de contenido, automatizando la clasificación de noticias antes de su distribución. También puede usarse de forma independiente, por equipos editoriales, o como una herramienta educativa de libre acceso para fomentar el pensamiento crítico.

Sin embargo, su implementación conlleva algunos riesgos importantes. Las predicciones generadas por el modelo se basan en estimaciones probabilísticas, por lo que deben ser interpretadas como aproximaciones y no como verdades absolutas. Esto implica el riesgo de malinterpretaciones, especialmente si el usuario no está familiarizado con los principios del aprendizaje automático. Además, si el conjunto de entrenamiento presenta sesgos (por ejemplo, en la selección de fuentes o temas), el modelo puede heredar dichas tendencias, afectando la imparcialidad de las predicciones.

Otro riesgo clave es el uso inapropiado de la funcionalidad de reentrenamiento. Si los usuarios cargan archivos sin validación, con etiquetas incorrectas o datos mal estructurados, el modelo puede deteriorarse significativamente en términos de precisión. Asimismo, al ser un sistema automatizado, existe la posibilidad de obtener falsos positivos o falsos negativos, lo cual puede tener implicaciones serias si las decisiones tomadas con base en los resultados del modelo no son posteriormente validadas por un experto.

Como equipo desarrollador, recomendamos que antes de utilizar la aplicación, el usuario lea la *landing page* para comprender el propósito del sistema y hacer un uso responsable de los resultados. En futuras versiones se podrían incluir advertencias más visibles sobre la interpretación de las predicciones y restricciones en funciones críticas como el reentrenamiento. Además, sugerimos evaluar periódicamente el desempeño del modelo y documentar los cambios realizados para mantener la confiabilidad del sistema.

3. Resultados

En el video se presentan cuatro acciones concretas que el usuario puede realizar como resultado de su interacción con la aplicación. En cada caso, el modelo analítico desarrollado aporta directamente a la ejecución de estas acciones mediante la clasificación automática de noticias y el cálculo de probabilidades asociadas:

1. **Evitar la difusión de desinformación:** El usuario puede ingresar manualmente una noticia y obtener la predicción del modelo junto con su probabilidad. Si el sistema indica una alta probabilidad de que el contenido sea falso, el usuario puede decidir no compartir esa información. El modelo aporta al entregar una clasificación respaldada por una métrica probabilística, que permite tomar decisiones informadas y responsables.

2. **Mejorar la calidad informativa en medios u organizaciones:** Al permitir la carga de archivos con múltiples noticias, la aplicación ayuda a verificar de forma masiva contenidos antes de ser publicados. El modelo, al clasificar automáticamente cada entrada, aporta eficiencia en entornos donde se manejan grandes volúmenes de información y se requiere rapidez sin sacrificar precisión.
3. **Actualizar el sistema frente a nuevas formas de desinformación:** A través del módulo de reentrenamiento, el usuario puede subir noticias etiquetadas con datos recientes. El modelo se entrena nuevamente y devuelve métricas que evalúan su nuevo desempeño. Esto permite mantener la relevancia y exactitud del clasificador, ajustándolo a cambios en el lenguaje o en el contexto político actual.
4. **Evaluar la confiabilidad del modelo tras cada actualización:** Luego de cada reentrenamiento, la aplicación muestra métricas clave como precisión, *recall* y F1-score. Esta información le permite al usuario decidir si continuar utilizando la nueva versión del modelo o no. Así, el modelo aporta no solo predicciones, sino también transparencia y trazabilidad sobre su propio desempeño.

3.1. Pruebas de Usuarios

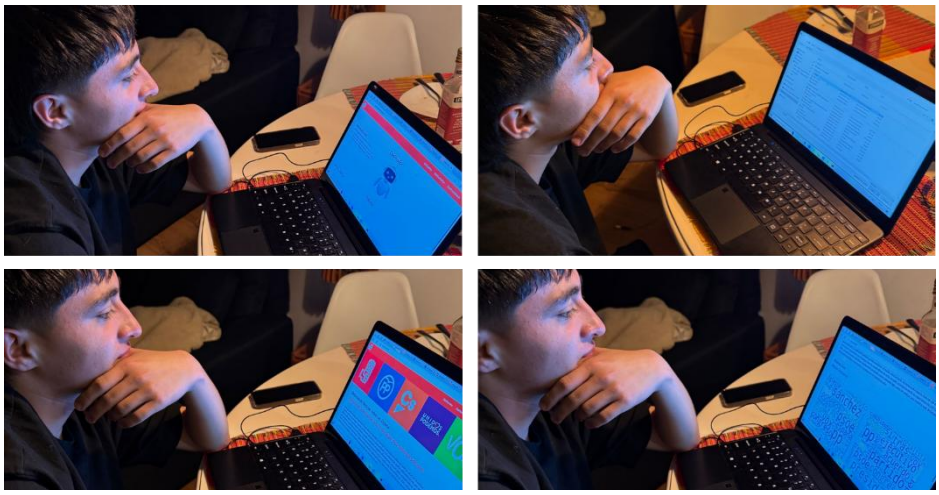


Ilustración 21 Usuario utilizandojn la aplicación

En las imágenes se puede observar cómo un usuario de redes sociales, incluso si no pertenece al público objetivo principal de la aplicación, puede hacer uso de ella para realizar una verificación independiente de noticias. La interfaz intuitiva y fácil de usar le permite evaluar rápidamente la veracidad de la información sin necesidad de conocimientos técnicos avanzados. Esto demuestra que la aplicación no solo es una herramienta especializada para periodistas, analistas o fact-checkers, sino que también puede empoderar a cualquier persona interesada en combatir la desinformación en su día a día.

4. Trabajo en equipo

4.1. Roles y tareas realizadas por cada integrante

Para la implementación del proyecto, el equipo se organizó en tres roles principales: **líder de proyecto**, **líder de datos y responsable de los resultados (Ingeniero de Datos)** y **líder de software responsable de desarrollar la aplicación final**. Esta estructura permitió una distribución equitativa de las responsabilidades, asegurando que cada integrante contribuyera de manera efectiva al desarrollo del modelo de clasificación de noticias falsas. Se llevaron a cabo reuniones estratégicas para definir los objetivos, asignar actividades y hacer seguimiento al avance del trabajo, lo que favoreció la comunicación y una integración adecuada de cada parte del proyecto.

Integrante 1: Natalia Villegas Calerón (Líder del Proyecto)
Contribución: 33.3 / 100.
Horas dedicadas: 20
Tareas:

- Liderar la gestión del proyecto, coordinando la planificación general, la asignación equitativa de tareas y la toma de decisiones clave.
- Establecer y comunicar las fechas de reuniones, pre-entregables y entrega final del grupo.
- Verificar el cumplimiento de los criterios establecidos en la rúbrica y garantizar la calidad de los entregables.
- Redactar y estructurar el documento principal del informe, integrando los aportes de los demás integrantes.
- Describir los perfiles de usuario de la aplicación y elaborar las diapositivas correspondientes con enfoque en el impacto del sistema.
- Corregir los errores detectados en la entrega anterior, particularmente en el pipeline, para asegurar su funcionamiento adecuado dentro de la API y la aplicación web.

Integrante 2: Carol Sofía Florido Castro (Líder de datos y responsable de los resultados – Ingeniera de datos)
Contribución: 33.3 / 100.
Horas dedicadas: 20
Tareas:

- Diseñar e implementar la API REST, integrando dos endpoints: uno para predicción y otro para reentrenamiento, conforme a los requerimientos técnicos del proyecto.
- Adecuar los datos procesados de la iteración anterior para que fueran compatibles con la estructura de la API y la lógica del modelo actual.
- Definir y documentar tres enfoques diferentes de reentrenamiento, evaluando sus ventajas y desventajas, y justificar la metodología seleccionada para esta iteración.
- Implementar el backend de la aplicación web utilizando un framework adecuado, asegurando la conexión fluida con el modelo y el pipeline.
- Realizar pruebas funcionales del backend y documentar su correcto desempeño

Integrante 3: Juan Martín Vásquez Cristancho (Líder de Software responsable de desarrollar la Aplicación final)
Contribución: 33.3 / 100.
Horas dedicadas: 20
Tareas:

- Liderar el diseño funcional y visual de la aplicación web, definiendo la experiencia de usuario (UX) y la interfaz gráfica (UI).
- Desarrollar el frontend de la aplicación, asegurando una experiencia de usuario intuitiva para el ingreso de datos y la visualización de resultados.

- Coordinar con el backend para integrar adecuadamente los endpoints de predicción y reentrenamiento dentro de la interfaz web.
- Ejecutar pruebas funcionales y de usabilidad sobre el frontend, verificando la correcta comunicación con la API y el comportamiento de la interfaz ante distintos escenarios.
- Participar activamente en la integración final de la solución, contribuyendo al despliegue funcional de la aplicación.

4.2. Reuniones de grupo

Reunión de lanzamiento y planeación
Fecha: 18 de marzo de 2025
Integrantes: Todos
Resumen: En esta reunión inicial se definieron los roles de cada integrante del grupo, asegurando un balance entre responsabilidades técnicas y de gestión. Se discutió la forma de trabajo, se identificaron las fortalezas individuales y se establecieron los primeros entregables internos. Además, se realizó una lluvia de ideas sobre cómo abordar el desarrollo de la API, la construcción de la aplicación web y la estrategia de integración del modelo analítico. También se acordaron los medios de comunicación oficiales (canales digitales) y el cronograma tentativo de reuniones de seguimiento y entrega final.

Reunión de seguimiento
Fecha: 22 de marzo de 2025
Integrantes: Todos
Resumen: En esta sesión se revisaron los avances generales del proyecto, incluyendo el diseño y la implementación de la arquitectura de la aplicación. Se discutieron las propuestas de cada integrante en relación con el *backend*, *frontend* y los *endpoints* de la API, especialmente en lo referente a la predicción y al reentrenamiento del modelo. Se establecieron acuerdos técnicos sobre el uso de *frameworks*, la estructura del pipeline y el flujo de datos entre componentes. Además, se abordaron aspectos relacionados con la experiencia de usuario, pruebas funcionales, validaciones de resultados y ajustes en la interfaz. La reunión permitió consolidar el trabajo hecho hasta el momento y definir las últimas acciones necesarias para completar la entrega final de forma exitosa.

Notificaciones de avance
Fecha: 26 de marzo de 2025
Integrantes: Todos
Resumen: A través del canal oficial del grupo, cada integrante compartió los avances individuales relacionados con la estructura de los *endpoints*, el diseño del *frontend*, la automatización del *pipeline* y el manejo del repositorio. Esta comunicación permitió validar la alineación técnica del desarrollo con los objetivos del proyecto, aclarar dudas y coordinar tareas interdependientes. Posteriormente, se realizaron validaciones cruzadas entre *backend* y *frontend*, confirmando el correcto funcionamiento de la API tanto para predicciones como para el reentrenamiento del modelo. Esta instancia fue clave para ajustar los últimos detalles técnicos antes de la entrega final.

Reunión de finalización
Fecha: 29 de marzo de 2025
Integrantes: Todos
Resumen: Se consolidaron los entregables del proyecto: la documentación, el video de presentación, el código de la API y la aplicación web. El equipo evaluó la efectividad de la distribución de tareas y destacó la buena comunicación como un factor clave del éxito. También se reflexionó sobre las lecciones aprendidas y se identificaron puntos de mejora para futuras fases, como iniciar las tareas técnicas con mayor anticipación y utilizar herramientas de control de versiones más robustas para evitar conflictos en el desarrollo colaborativo.

4.3. Retos enfrentados en el proyecto

Uno de los principales retos que enfrentó el grupo fue la falta de experiencia previa con el *framework* Flask y el desarrollo de una API REST funcional. Esto generó una curva de aprendizaje considerable y ralentizó el proceso de implementación, obligando al equipo a recurrir a múltiples ciclos de prueba y error para entender el comportamiento esperado de los *endpoints* y su integración con el modelo. Para superar esta dificultad, se realizaron búsquedas constantes de documentación y ejemplos, y se dividió el trabajo para que cada integrante se familiarizara con partes específicas del flujo. Adicionalmente, surgieron múltiples problemas de compatibilidad de versiones, especialmente al compartir archivos como el modelo serializado (*.joblib*) entre diferentes equipos. Estos modelos, al ser ejecutados en entornos distintos, presentaban errores o incompatibilidades. La forma de mitigar este problema fue que cada integrante ejecutara localmente su propio entorno con las versiones exactas requeridas, evitando así conflictos derivados del traspaso de archivos.

Otro desafío fue la adecuación de los resultados de la entrega anterior, en particular la preparación de datos y la estructuración del *pipeline* para que funcionara correctamente dentro del flujo automatizado de la API. También resultó complejo lograr una integración estable entre el *backend* y el *frontend*, dado que cada componente debía comunicarse de forma precisa, respetando los formatos esperados en las solicitudes y respuestas. Finalmente, los tiempos de entrega y la coincidencia con otras responsabilidades académicas requirieron una gestión eficiente y una comunicación constante para asegurar que cada integrante cumpliera con sus tareas a tiempo y con la calidad requerida.

Referencias

Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow : concepts, tools, and techniques to build intelligent systems (Third edition). O'Reilly Media, Inc.