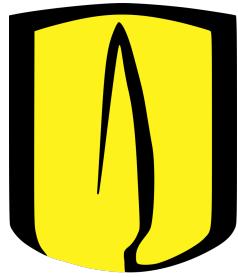


**UNIVERSIDAD DE LOS ANDES**  
**DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y**  
**COMPUTACIÓN**



**PROYECTO 1 - CLASIFICACIÓN DE OBJETIVOS DE DESARROLLO  
SOSTENIBLE (ODS)**

**ISIS 33301 – INTELIGENCIA DE NEGOCIOS**

**Grupo 14**

María Alejandra Angulo Mejía 202121458

Laura Sofía Murcia Carreño 202123423

David Tobón Molina 202123099

**2025-20**

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Trabajo en Grupo</b>	<b>2</b>
<b>3. Estructura del Repositorio</b>	<b>2</b>
3.1. Carpetas . . . . .	2
3.2. Archivos . . . . .	3
<b>4. Desarrollo</b>	<b>3</b>
4.1. Aumentación de datos y reentrenamiento del modelo . . . . .	3
4.2. Proceso de automatización . . . . .	4
4.3. Construcción del API . . . . .	4
4.4. Primer endpoint . . . . .	5
4.5. Segundo endpoint . . . . .	5
4.6. Métodos de reentrenamiento . . . . .	5
4.6.1. Método 1: Reentrenamiento completo . . . . .	5
4.6.2. Método 2: Reentrenamiento incremental . . . . .	6
4.6.3. Método 3: Ventanas de tiempo . . . . .	6
4.6.4. Método 4: Reentrenamiento completo con ajuste de peso . . . . .	6
4.7. Desarrollo de la aplicación . . . . .	6
4.7.1. Integración en la organización . . . . .	6
4.7.2. Recursos y tecnologías requeridas . . . . .	7
<b>5. Conclusiones</b>	<b>7</b>
<b>6. Vídeo de sustentación</b>	<b>7</b>
<b>7. Reflexión de grupo</b>	<b>7</b>
7.1. Distribución de la carga . . . . .	8
<b>8. Referencias</b>	<b>8</b>

## 1. Introducción

En esta segunda etapa del proyecto desarrollado junto al Fondo de Población de las Naciones Unidas (UNFPA), el trabajo se centró en mejorar el modelo analítico encargado de clasificar opiniones ciudadanas según su relación con los Objetivos de Desarrollo Sostenible (ODS) 1, 3 y 4. Luego de crear el modelo base en la etapa anterior, el propósito principal fue reentrenarlo utilizando datos aumentados mediante técnicas de prompting, y desarrollar una aplicación funcional que permita su uso por parte de diferentes tipos de usuarios.

Desde el rol de científico de datos, se aplicaron estrategias de aumentación textual para ampliar y diversificar el conjunto de entrenamiento. Esto permitió que el modelo aprendiera a reconocer mejor distintas formas de expresión y mejorara su capacidad de generalización. Además, se comparó su rendimiento con el modelo original, evaluando métricas como la precisión, el recall y el F1-score para determinar los avances obtenidos.

Por su parte, el ingeniero de datos organizó el proceso de despliegue del modelo, incluyendo la estructura del código y el pipeline para facilitar su seguimiento. Finalmente, el ingeniero de software integró el modelo en una aplicación web, a través de la cual los usuarios pueden ingresar textos, visualizar los resultados de clasificación y aportar nuevos datos para seguir mejorando el modelo.

En conjunto, esta etapa permitió pasar de un desarrollo experimental a una solución funcional, capaz de analizar textos reales y apoyar la toma de decisiones en el marco de la Agenda 2030.

## 2. Trabajo en Grupo

**Líder de Proyecto y Negocio:** Maria Alejandra Angulo Mejia. Encargada de definir las fechas de reuniones, pre-entregables del grupo e interceder por la carga equitativa y la buena convivencia dentro del grupo. Asimismo, se encarga de subir la entrega y revisar la completitud del mismo. Bajo esta misma tarea, es responsable de resolver el problema identificado y ver que el desarrollo de la tarea está alineado con las necesidades del negocio.

**Ingeniera de datos:** Maria Alejandra Angulo Mejia. Encargado de gestionar los datos que se van a usar en el proyecto y la asignación de tareas relacionadas. Es la encargada del modelo analítico desarrollado. Adicionalmente, supervisa el correcto tratamiento de los datos al implementar procesos de aumentación y reentrenamiento.

**Ingeniera de software responsable del diseño de la aplicación y resultados:** Laura Sofia Murcia Carreño. Encargada de desarrollar el diseño software de la aplicación. Esto incluye procesos de automatización, desarrollo de la aplicación, análisis de resultados y discusión del proyecto.

**Ingeniero de software responsable de desarrollar la aplicación final:** David Tobon Molina. Encargado de desarrollo software de la aplicación en cuanto a su despliegue y desarrollo web. Adicionalmente, distribuye y realiza tareas asociadas al desarrollo de una aplicación completa y funcional.

## 3. Estructura del Repositorio

A continuación se detalla la estructura del repositorio con el fin de que este sea fácilmente navegable. Este se puede encontrar en el siguiente url: [Repositorio](#)

### 3.1. Carpetas

- **datos:** contiene los archivos de datos de la entrega pasada
- **datos\_pruebas\_sinteticos:** contiene los archivos con los datos sintéticos adicionales para pruebas. Incluye datos etiquetados y no etiquetados.
- **df\_retrain:** contiene los archivos con los datos que se utilizan para el reentrenamiento.
- **fastAPI:** contiene los archivos de definición y funcionamiento del API.
- **frontend:** contiene los archivos de la interfaz web de la aplicación.

### 3.2. Archivos

- `proyecto_etapa1.ipynb`: archivo previo con el desarrollo del modelo presentado en la etapa 1
- `proyecto_etapa2_aug.ipynb`: archivo detallado del proceso de aumentación de datos. Cuenta con las métricas del modelo de la etapa 1, la evaluación del modelo de la etapa 1 con el dataset entregado para esta entrega, una nueva iteración del modelo entrenado con la concatenación de datasets, una segunda iteración del modelo con aumento de los datos de entrenamiento para balancear las clases y una comparación entre el modelo sin aumentación y el modelo con aumentación.
- `pipeline`: notebook donde se detalla el desarrollo del pipeline para automatizar el proceso de entrenamiento y uso del modelo.
- `WordTokenizer`: archivo para almacenar la clase especial utilizada dentro del pipeline.
- `pipeline.cloudpkl`: archivo generado sobre el pipeline del modelo.
- `proyecto_generar_pruebas`: archivo donde se hace la generación de datos con IA gen para probar el modelo. Incluye tanto los datos etiquetados, usando como referencia los datasets de entregas pasadas como datos no etiquetados. Estos son usados para mirar como funciona la aplicación con datos que no ha visto antes.
- `docker_compose`: archivo para el despliegue de los contenedores.

## 4. Desarrollo

### 4.1. Aumentación de datos y reentrenamiento del modelo

Todo el proceso puede ser encontrado en el notebook: [Notebook Aumentación de datos](#).

Se evidencia la necesidad de aumentar los datos en el modelo al probar con los datos dados para la entrega 2 y ver una desmejora en sus métricas. Para poder generar una aumentación de los datos se utilizan dos estrategias principales. La primera fue combinar el dataset de la entrega 1 y de la entrega 2. Sobre este se genera el split de train, val y test. Es importante que se generara el split en este punto, ya que como se va a balancear las clases haciendo aumentación de datos con inteligencia artificial generativa, se desea evitar sesgos en los conjuntos de val y test que permiten evaluar correctamente el desempeño del modelo. Este dataset cuenta con los datos limpios, asegurando la calidad de los datos, pero sin tokenizar.

Ya con el split del nuevo dataset concatenado, se procede a reentrenar el modelo desde 0. El primer paso es tokenizar los textos con la clase `WordTokenizerTransformer` para tokenizar y facilitar más adelante el proceso de automatización. Después se genera una instancia de `CountVectorizer(ngram_range=(1,1))` para poder vectorizar los datos de train y val. De la misma manera, se genera una nueva instancia del modelo `LogisticRegression(max_iter=1000, random_state=42)` para después entrenar el modelo de concatenación y evaluarlo con los datos de validación.

Ya con esos resultados, se realiza a hacer la aumentación de datos con IA generativa sobre el conjunto de entrenamiento para balancear las clases. Para esto se llama la llave de OpenAI, se calcula cuántos ejemplos se necesitan por clase para balancear el dataset y se crea un dataframe de train extendido con el prompt:

`Genera {n_to_generate} opiniones ciudadanas breves (1{2 oraciones) en español (Colombia), realistas y respetuosas, sobre problemáticas locales que correspondan al ODS {clase}.`

Requisitos:

- TODAS deben pertenecer exclusivamente al ODS {clase}.
- Varía zona (urbano/rural), actores e instituciones; evita datos personales.
- Mantén neutralidad política.
- Devuelve SOLO JSON: lista de objetos con llaves "textos" (string) y "labels" (entero {clase}).

Ejemplos de inspiración:

```
{chr(10).join(f"- {s}" for s in seeds) if seeds else "- (sin ejemplos de contexto)"}  
3
```

Con los datos ya generados en el nuevo dataframe, se guardan en un Excel en la ruta `df_retrain/datos_retrain.xlsx` para no tener que volver a generar los datos cuando se necesiten nuevamente.

Ya con el nuevo conjunto de entrenamiento, se procede nuevamente a instanciar un tokenizador, un vectorizador y un modelo para entrenar (con los datos nuevos de train). Deben ser nuevas instancias (tanto el vectorizador de train como el de val, así sean los mismos datos) para evitar filtraciones entre modelos. Al entrenar y evaluar los datos con el conjunto de validación, se ve el correcto funcionamiento del modelo por lo que se realiza la comparación entre modelos con los datos de test que no han sido evaluados por ningún modelo hasta el momento.

Al hacer la comparación se encuentran los resultados presentados en la siguiente tabla:

Clase / Métrica	Precisión (sin aumento)	Recall (sin aumento)	F1-Score (sin aumento)	Precisión (con aumento)	Recall (con aumento)	F1-Score (con aumento)	Soporte
ODS 1	0.98	0.94	0.96	0.98	0.96	0.97	51
ODS 3	0.99	0.98	0.98	0.96	0.98	0.97	97
ODS 4	0.96	0.99	0.98	0.97	0.96	0.97	105
Exactitud	—	—	0.98	—	—	0.97	253
Promedio macro	0.98	0.97	0.97	0.97	0.97	0.97	253
Promedio ponderado	0.98	0.98	0.98	0.97	0.97	0.97	253

Al analizar los modelos, se observa un comportamiento muy similar entre ellos. La principal diferencia radica en que, sin aumento de datos, el modelo basado únicamente en la concatenación presenta un mejor desempeño en los ODS 3 y ODS 4, con un f1-score de 0.98 frente a 0.97 en el modelo con datos aumentados. No obstante, se evidencia una mejora en el f1-score del ODS 1, que pasa de 0.96 en el modelo sin aumentación a 0.97 en el modelo con datos generados, destacándose especialmente el incremento en el recall, que aumenta de 0.94 a 0.96.

En términos generales, debido a la marcada diferencia entre las clases, las mejoras entre modelos son mínimas. Sin embargo, la aplicación de técnicas de aumentación permitió una mejora notable en la clasificación del ODS 1, uno de los principales objetivos del ajuste, dado que representaba una de las debilidades del modelo inicial.

## 4.2. Proceso de automatización

Para automatizar el proceso de entrenamiento y predicciones del modelo, se implementó una pipeline para manejar el proceso. Este pipeline se encarga de tokenizar los textos, generar los features con un `CountVectorizer` definido con unigramas debido a los resultados de la etapa previa de este proyecto, y generar el modelo de clasificación a partir de una regresión logística con 1000 iteraciones como máximo definido. No se incluyen etapas adicionales de procesamiento de datos ya que se parte del análisis del dataset con el que se trabaja y se confirma que no se cuentan con problemas de calidad que se deban arreglar dentro del pipeline.

Esta pipeline se realiza usando `cloudpickle` debido a las facilidades que permite para usar transformaciones específicas y poder exportarlas dentro del archivo del modelo. El pipeline que se implementa se encuentra definido dentro del archivo `pipeline.ipynb` y hace uso de la clase adicional definida en el archivo `WordTokenizer.py`.

Dentro de `WordTokenizer.py` se define la clase `WordTokenizerTransformer` que hace uso de `nltk` para eliminar ciertos caracteres especiales de los textos y tokenizarlos de manera adecuada. Esto es útil para la implementación ya que hace que el modelo no se vea sesgado por stop words y reconozca patrones más fácilmente.

Finalmente, para entrenar el modelo el pipeline recibe como entrada los datos X, y. En donde X hace referencia a la columna de textos y y hace referencia a los labels asociados a cada texto. Con el fin de poder obtener métricas sobre este modelo al entrenarlo, se genera la división de datos entre train y test antes de entrenar el modelo. Con los datos de test se obtienen las métricas correspondientes. Todo este proceso más detallado se puede ver dentro del archivo `pipeline.ipynb`.

## 4.3. Construcción del API

Se utilizó FastAPI para este numeral. Con esto, dentro de la carpeta FastAPI del proyecto se encuentra el código que incluye la definición de los 2 endpoints requeridos. Dentro de `main.py` se definen los endpoints de la aplicación. En `DataModel` se definen las estructuras que reciben las peticiones y en `PredictionModel` se define el llamado al modelo entrenado a partir del uso del archivo generado con `cloudpickle`.

Para poder utilizar el API es necesario ubicarse sobre la carpeta correspondiente y utilizar el comando:

```
uvicorn main:app --reload
```

#### 4.4. Primer endpoint

El primer endpoint se encuentra con la ruta `predict`. Este primer endpoint es de tipo post y se utiliza para generar predicciones sobre una o múltiples instancias recibidas por parámetros. Este endpoint usa un `DataModel` definido que permite el ingreso de los textos como una lista de string en formato JSON. Haciendo uso del modelo, utiliza el método `predict_proba` con el cual se devuelven las probabilidades de que el dato pertenezca a cada una de las clases. Estos resultados se procesan y se devuelven con el formato JSON:

```
[  
  {  
    "prediction": int,  
    "probabilities": {  
      "class_id": float  
    }  
  }  
]
```

#### 4.5. Segundo endpoint

El segundo endpoint se encuentra con la ruta `retrain`. Dentro de este endpoint se genera el reentrenamiento del modelo. Este usa un `DataModel` que incluye una lista de textos y una lista de labels que le corresponden a cada texto. Con esto, y haciendo uso de los datos originales se aplica el método de reentrenamiento seleccionado. Al aplicar el reentrenamiento se generan las métricas del modelo y se devuelve un JSON con la estructura:

```
{  
  "precision": float,  
  "recall": float,  
  "f1_score": float,  
  "classes": [int],  
  "precision_per_class": [float],  
  "recall_per_class": [float],  
  "confusion_matrix": [[int]],  
  "model_timestamp": string,  
  "model_saved_as": string  
}
```

### 4.6. Métodos de reentrenamiento

A continuación, se revisan diferentes métodos de reentrenamiento analizando sus ventajas y desventajas y decidiendo cuál implementar dentro del modelo final.

#### 4.6.1. Método 1: Reentrenamiento completo

Este método consiste en tomar las nuevas instancias y reentrenar todo el modelo al unir los sets de datos. Entre las ventajas de este método se incluye que permite un aprendizaje completo del modelo, lo que lo puede hacer más robusto frente a diferentes casos. Sin embargo, entre sus desventajas se resalta que requiere más tiempo y tiene un costo computacional más elevado, lo que dependiendo de la aplicación puede no ser deseado.

#### 4.6.2. Método 2: Reentrenamiento incremental

Algunos tipos de modelos permiten un reentrenamiento parcial en donde se procesa sobre los datos nuevos nada más. Esto resulta útil para modelos que no tengan los recursos para soportar un entrenamiento completo, o requieran aplicaciones más rápidas en tiempo. Sin embargo, una de las desventajas de este método es que no todos los modelos lo soportan. En particular, `Logistic Regression` no soporta este tipo de reentrenamiento a diferencia de modelos como `SGDClassifier` y `PassiveAggressiveClassifier` lo cual lo hace incompatible para este caso.

#### 4.6.3. Método 3: Ventanas de tiempo

En algunos contextos resulta útil e interesante utilizar un reentrenamiento con ventanas de tiempo. Esto significa que se van eliminando los datos más viejos y se mantienen los nuevos. Esto tiene sentido para problemas en los que el comportamiento de las distribuciones cambia y se presentan situaciones cambiantes con el tiempo que pueden no ser representadas correctamente a partir de los datos nuevos. En este problema en particular, siempre se desea clasificar los datos entre 3 ODS que no cambian de definición. Esto hace que el uso de ventanas de tiempo no sea necesariamente el más adecuado. Es así como una de las ventajas de este método es que permite adaptarse mejor a los nuevos datos y darles más relevancia, lo que ayuda cuando cambian los patrones a lo largo del tiempo. Sin embargo, una de las desventajas de este método es que si los patrones no cambian, se puede llegar a perder información útil para el desempeño del modelo.

#### 4.6.4. Método 4: Reentrenamiento completo con ajuste de peso

Finalmente, se propone nuevamente el uso de reentrenamiento completo del modelo pero se usa un ajuste de pesos dentro del modelo. La ventaja de esto es que además de usar todos los datos, ajusta el peso de clases para responder frente a casos en los que se genere desbalanceo de clases. Esto puede ayudar a que si los datos nuevos generan un desbalance, el modelo compense este comportamiento automáticamente. Sin embargo, las desventajas de la implementación original se mantienen ya que al ser un reentrenamiento completo es computacionalmente complejo.

Al implementar los métodos se obtuvieron las siguientes métricas:

Método	F1-Score	Precision	Recall
1	95.7 %	95.7 %	95.8 %
2	94.7 %	94.8 %	94.7 %
3	94.7 %	94.7 %	94.7 %
4	95.7 %	95.7 %	95.8 %

A partir de las métricas obtenidas se elige la opción de reentrenamiento completo con ajuste de pesos. A simple vista no se detectan diferencias entre el método 1 y 4, sin embargo, la elección se justifica al comparar el caso en el que los datos están más desbalanceados. Como se puede ver dentro del notebook las métricas del método 1 cuando se genera el desbalance tienen un peor desempeño. Es por esto que dentro del endpoint esta es la solución que se implementa. Es de mencionar, adicionalmente, que al reentrenar el modelo se actualiza el archivo en el que se guardan los datos con los que se entrena y se actualiza así mismo el modelo con el que se generan las predicciones, guardando la versión como la más reciente para trabajar con ella. Esto hace que sea persistente la aplicación.

### 4.7. Desarrollo de la aplicación

*Para desplegar la aplicación y probar su funcionamiento, por favor remitirse al [README.md](#) del repositorio.*

#### 4.7.1. Integración en la organización

**Usuario/rol de la organización que va a utilizar la aplicación:** La aplicación puede ser utilizada por un usuario del departamento de analítica de datos que tenga la tarea de limpiar y etiquetar datos de opiniones de ODS 1, 3 y 4 en Colombia. Para esto utilizaría la pantalla de predicción,

la cual etiqueta grandes cantidades de opiniones con el último modelo entrenado. De todos modos, se recomienda que el usuario que tenga acceso a la aplicación tenga un rol de mayor experticia en el negocio, ya que la aplicación también permite el reentrenamiento del modelo con visualización de métricas. Este usuario debe tener el conocimiento suficiente para evaluar el desempeño del modelo y tener el poder y los datos para efectuar la acción de reentrenar el modelo de clasificación.

**Importancia para el negocio:** La aplicación puede apoyar en el proceso de etiquetación de opiniones de ODS para un futuro análisis que requiera los datos ya etiquetados. El proceso de negocio de etiquetado de opiniones será beneficiado por el ahorro de tiempo de la labor manual equivalente. Este rol ahora puede usar su tiempo y energía no en leer y etiquetar cada opinión, sino en analizar el desempeño del modelo e identificar rápidamente aquellas pocas opiniones de las cuales el modelo no tiene tanta seguridad en su clasificación. Además, se puede enfocar en otros procesos de negocio como el análisis del contenido de las opiniones ya segmentadas por ODS gracias a la etiquetación automática hecha por nuestro sistema.

**Riesgos:** Los principales riesgos del uso de la aplicación son que los datos que se utilicen en el reentrenamiento tengan características diferentes a los datos utilizados previamente, por lo que el desempeño del nuevo modelo se desplome y se requiera que el equipo de analítica haga un rollback del modelo para utilizar la versión anterior y deba entrenar un modelo nuevo para clasificar las nuevas opiniones que causaron la disminución en el desempeño del modelo actual.

#### 4.7.2. Recursos y tecnologías requeridas

Los recursos informáticos requeridos para el sistema son:

- Un servidor Linux con CPU, mínimo 16 GB de memoria RAM y espacio en disco de mínimo 64 GB.

#### Tech Stack:

- Despliegue: Docker
- Backend: Python 3.12, FastAPI, Scikit-learn, NLTK, cloudpickle
- Frontend: JavaScript, React

### 5. Conclusiones

A partir del desarrollo de este proyecto se pudo generar una aplicación web funcional, desplegada en Docker para clasificación de comentarios de texto sobre los ODS 1, 3 y 4, y reentrenamiento de modelos. Por medio de esta, se pretende impulsar los objetivos del negocio dando una herramienta completa, intuitiva y visualmente atractiva que permitirá fácilmente cumplir con 2 funciones claves. La primera función consiste en la predicción del ODS asociado a uno o múltiples comentarios. La segunda función consiste en el reentrenamiento del modelo a partir de un archivo con múltiples datos.

En el contexto colombiano, esto es útil como una herramienta de análisis que automatiza procesos de clasificación y puede repercutir positivamente en la generación de estrategias y análisis de percepción asociado al desarrollo de los ODS en Colombia. A partir del entendimiento de los comentarios del público general, se puede entender en qué acciones se debe priorizar la acción y encaminar la gestión, de forma que sea más eficiente, apropiada y contundente.

### 6. Video de sustentación

[Video de sustentación](#)

### 7. Reflexión de grupo

A continuación se incluye la reflexión sobre el trabajo de cada integrante y el uso de herramientas para el desarrollo de este.

**Líder de proyecto e Ingeniera de datos:** Maria Alejandra Angulo Mejia

Se encargó de realizar la evaluación del modelo anterior y la aumentación de datos para balancear las clases, así como generar los archivos de prueba para la prueba de la aplicación, así como en parte de la documentación del proyecto. Para esto se invirtieron 7h distribuidas en 3 días. En cuanto el uso de inteligencia artificial generativa se utilizó para generar los datos sintéticos, garantizando que tomara de referencia los datos dados por el equipo de Inteligencia de Negocio, y corrección de estilo cuando fue necesario.

**Ingeniera de software responsable del diseño de la aplicación y resultados:** Laura Sofia Murcia Carreño.

Se encargó del desarrollo de la API, automatización de procesos con pipelines, investigación de métodos de reentrenamiento y documentación de resultados generales. Para esto se invirtieron alrededor de 8 horas distribuidas en 3 días. En cuanto al uso de inteligencia artificial generativa, su uso fue mínimo. En donde las principales funciones para las que se utilizó incluyen corrección de estilo y depuración de código.

**Ingeniero de software responsable del desarrollo de la aplicación web, integración con el backend y despliegue del sistema:** David Tobón Molina.

Se encargó del desarrollo de la API para mejorar la integración con el frontend y con los objetivos analíticos y de negocio. Además, se encargó del diseño y desarrollo de la interfaz web y el despliegue del sistema en contenedores Docker. Para esto invirtió alrededor de 8 horas distribuidas a lo largo de 2 días. Durante el desarrollo de sus tareas utilizó la Inteligencia Artificial Generativa para mejorar el diseño de la interfaz del frontend y que fuera visualmente organizada y limpia.

## 7.1. Distribución de la carga

Integrante	Carga
Maria Alejandra Angulo Mejia	33.33 %
David Tobón Molina	33.33 %
Laura Sofia Murcia Carreño	33.33 %

## 8. Referencias

1. “LogisticRegression”. scikit-learn. Accedido el 13 de octubre de 2025. Disponible en: [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
2. M. Gusarova. “How to improve logistic regression in imbalanced data with class weights”. Medium. Disponible en: <https://medium.com/@data.science.enthusiast/how-to-improve-logistic-regression-in-imbalanced-data-with-class-weights-1693719136aa>
3. M. Mohapatra. “Retrain or Refresh? Updating Machine Learning Models the Right Way”. Medium. Disponible en: <https://mhmohapatra.medium.com/retrain-or-refresh-updating-machine-learning-models-the-right-way-820098a9a3b2>
4. M. Mohapatra. “Taking the Next Step: Improvising and Retraining Your First ML Model in Python”. Medium. Disponible en: <https://mhmohapatra.medium.com/taking-the-next-step-improvising-and-retraining-your-first-ml-model-in-python-41a14ec96f1f>
5. “Model Retraining: Why & How to Retrain ML Models?” AIMultiple. Disponible en: <https://research.aimultiple.com/model-retraining/>