# Introducción y alcance

El objetivo principal de este proyecto es desarrollar un modelo de Machine Learning capaz de clasificar la intención de las frases que los usuarios expresan (por ejemplo, determinar si solicitan un crédito, hacen un reclamo, etc.) mediante el Chatbot Mari, y luego exponer el modelo a través de un servicio web, de manera que pueda ser consumido por aplicaciones externas como Botpress o similares.

Para lograrlo, se ha seguido un flujo que comienza con la recolección y preparación de los datos de texto (limpieza y normalización), continúa con el entrenamiento y validación del modelo (utilizando un pipeline de transformaciones y un algoritmo de clasificación), y culmina en la creación de un entorno virtual e implementación de un endpoint Flask que, ante un mensaje recibido, devuelve la categoría (intención) correspondiente junto con un porcentaje de confianza.

# Obtención y preparación de datos

## ***Fuente de Datos:***

Se contó con un histórico de conversaciones extraídas de la plataforma del Chatbot (Botpress), las cuales fueron descargadas y consolidadas en un archivo de Excel para su tratamiento inicial. Este archivo abarcó un total de 35,890 registros correspondientes a interacciones reales entre usuarios y el sistema, lo que permitió obtener un volumen representativo de distintas expresiones y solicitudes. De este modo, se dispuso de una base de datos suficiente para desarrollar el modelo de clasificación de intenciones.

## ***Descripción del Dataset***

* Estructura de la información (columnas):
  + **id:** Identificador único de cada registro (mensaje) en el dataset.
  + **createdAt:** Fecha y hora en que se generó el mensaje (registro temporal).
  + **conversationId:** Referencia a la conversación o sesión a la que pertenece el mensaje, permitiendo agruparlos en hilos.
  + **Input:** El texto o contenido real que el usuario escribió (la parte relevante para la clasificación).
  + **User\_ID:** Identificador del usuario que envió el mensaje (cuando está disponible).
  + **type:** Indica el tipo de emisor o la naturaleza del mensaje (por ejemplo, si es del usuario, del bot, un sistema, etc.).
  + **Categorizacion:** Columna que contiene la categoría asignada a ciertos registros para el entrenamiento inicial del modelo (no todos los registros fueron categorizados).
* Distribución de categorías (muestras etiquetadas)

La siguiente tabla refleja la cantidad de ejemplos que fueron etiquetados por categoría.

|  |  |
| --- | --- |
| Categoría | Recuentro |
| creditos | 461 |
| No\_interes\_campana | 316 |
| no\_clasificado | 280 |
| asesor | 210 |
| no\_pertenece | 199 |
| saludo | 185 |
| creditos\_asesor | 158 |
| aplicativo | 143 |
| reclamo | 138 |
| bloqueo\_tarjeta | 136 |
| ubicacion | 132 |
| apertura\_cuenta | 130 |
| ahorros | 127 |
| actualizacion\_datos | 120 |
| agradecimiento | 118 |
| pagos | 116 |
| cci | 113 |
| insultos | 88 |
| despedida | 81 |
| servicios | 61 |
| canales | 32 |
| campañas | 25 |
| giros | 10 |
| homebanking | 7 |
| carta\_adeudo | 5 |

Se inicio el modelo con un subconjunto de 3391 input categorizados, empleando estas muestras como una base para entrar la primera versión del modelo.

## ***Limpieza y preprocesamiento***

Para mejorar la calidad de los datos antes del entrenamiento, se realizó una etapa de limpieza y preprocesamiento que incluyó la eliminación de elementos considerados “ruido” en el texto, tales como emojis, números y signos de puntuación, ya que no aportaban valor semántico o dificultaban el análisis.

Asimismo, se revisaron los registros para descartar duplicados que pudieran distorsionar la frecuencia real de las expresiones y se validó la existencia de valores nulos, para asegurar que todos los ejemplos dispusieran de un contenido útil en la columna `Input`. Con estas acciones, se obtuvo un dataset más homogéneo y limpio, propiciando que el modelo de Machine Learning pudiera enfocarse en las palabras relevantes y mejorar así la clasificación de intenciones.

## ***División de conjuntos***

Para lograr una evaluación más confiable del rendimiento del modelo, se optó por %dividir el subconjunto etiquetado en tres partes: un 75% para el entrenamiento del modelo, 15% para el junto de pruebas y 10 % para validación.

De esta forma, el conjunto de validación (10%) permitió afinar hiperparámetros y realizar ajustes intermedios, mientras que el conjunto de pruebas (15%) se reservó para una medición final del desempeño que no influya en las decisiones de entrenamiento. Con ello, se garantizó un mejor control sobre el proceso de aprendizaje y se obtuvo una estimación más realista de cómo el modelo funcionará ante datos nuevos.

# Entrenamiento del modelo

Para el entrenamiento del modelo, se utilizó un pipeline para las fases de transformación y clasificación en una sola función.

* **Vectorización de texto:**

Se utilizó **TfidfVectorizer** para convertir cada oración en una representación numérica para que luego el modelo de clasificación lo pueda entender. Cada palabra o token se convierte en una columna y cada oración es una fila convirtiéndose en una matriz.

* **Clasificador principal:**

El clasificador seleccionado es **LogisticRegression**

* **Entrenamiento paso a paso:**

Se utiliza el pipeline con el porcentaje de los datos que ha sido seleccionado para el entrenamiento. Con la función predict, se obtendrá las predicciones del set seleccionado para validación. Se puede utilizar métricas como: accuracy, preccion y recall. En caso de que los resultados no son los esperados, se puede realizar ajustes en los parámetros hasta encontrar el balance deseado. Con el último modelo seleccionado, se realiza la predicción con el porcentaje seleccionado de test.

* **Guardado del modelo:**

Para utilizar el modelo en futuros escenarios, se requiere guardarlo, utilizando la librería “joblib” se crea una estructura donde tendrá los coeficientes aprendidos.

# Entorno virtual y dependencias

Para evitar conflictos entre diferencias entre las versiones se requiere crear un entorno virtual y posteriormente activarlo, en el cual se instalará todos los paquetes necesarios.

Dependencias:

Se deberán instalar las siguientes librerías clave: scikit-learn, Flask, joblib, numpy, pandas con sus respectivas versiones

Flask==2.2.3

scikit-learn==1.6.0

joblib==1.2.0

numpy==1.23.5

pandas==1.5.3

# Despliegue con Flask

Se creó un script en Python, donde se expondrá el endpoint (/predict) para determinar la categoría del input del cliente y el nivel de confianza en la predicción. Para ello, se utilizará el archivo que fue guardado en el punto 3 del entrenamiento del modelo.

# Diagrama

Diagrama

Descripción generada automáticamente

# Ejecución del proyecto para nuevos registros.

* **Actualización del Excel**

Para incrementar la cantidad de datos disponibles para el entrenamiento, es necesario incorporar inputs adicionales proporcionados por los usuarios. Estos datos deben ser registrados en la columna denominada "Categorización". Una vez completado este valor, se incrementará automáticamente el volumen de datos procesados para el entrenamiento, organizados según cada categoría correspondiente.

Tal como se ilustra en la imagen inferior, los inputs con valores en la columna "Categorización" serán integrados al dataset. Este dataset será posteriormente dividido en conjuntos para entrenamiento, validación y pruebas, optimizando así el proceso de aprendizaje del modelo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Ejecución del Código en Google Colab**

El desarrollo del código se encuentra en Google Colab. Para ejecutar el código correctamente, es necesario cargar el archivo actualizado mencionado en el punto anterior. Esto se logra añadiendo el archivo Excel en Colab, siguiendo los pasos detallados a continuación.

Texto

Descripción generada automáticamente con confianza baja

En la imagen superior se observa el código en Colab. En el panel lateral izquierdo, al seleccionar el ícono de la carpeta, se despliega una lista con todos los archivos cargados. Para subir un nuevo archivo Excel, se debe utilizar el ícono con una flecha hacia arriba.

Texto

Descripción generada automáticamente

Después de seleccionar el ícono de carga, se abrirá el explorador de archivos. Allí, se debe ubicar y seleccionar la ruta y el archivo que se desea cargar. Si la carga se realiza correctamente, el archivo aparecerá en la lista de archivos, tal como se muestra en la imagen superior.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Es necesario ejecutar todas las celdas del código para generar el archivo de entrenamiento actualizado con la nueva información del Excel cargado.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Al finalizar la ejecución de cada celda, se visualizará un check que confirmará su correcta ejecución. Una vez completado todo el proceso del código, se procederá a ejecutar la celda correspondiente al guardado, generando el archivo final denominado **("modelo\_entrenado.pkl").**

* **Despliegue**

En el código, es fundamental verificar las extensiones y versiones de las dependencias utilizadas. Mantenerlas actualizadas es crucial, ya que versiones desactualizadas podrían ocasionar errores durante la ejecución o afectar el rendimiento del sistema.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

El archivo “deploy.py” utiliza el modelo generado en Colab, el cual contiene todo el entrenamiento previamente realizado. Este archivo es clave para iniciar el proceso de clasificación, aplicando el modelo entrenado sobre nuevos datos de entrada.

Texto

Descripción generada automáticamente

En el mismo archivo se define el formato del resultado que será devuelto por el endpoint. Este incluirá tanto la clase predicha como el nivel de confianza asociado a dicha predicción, brindando información detallada y precisa para cada solicitud.

Texto

Descripción generada automáticamente

Para ejecutar el Script se deberá realizar lo siguiente:

* + Activar el entorno virtual mediante el comando: venv\Scripts\activate
  + Ejecutar el comando python deploy.py

Si todo está correcto entonces se visualizará como en la imagen superior.

* **Nuevos datos ingresados a través del Endpoint**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

  Descripción generada automáticamente

Al endpoint en ejecución se envía un texto en el cuerpo de la solicitud (Body), el cual representa el input del usuario. Al procesarse, el endpoint devuelve como resultado la predicción de la clase correspondiente junto con el nivel de confianza. Esto permite clasificar futuras entradas de manera precisa y eficiente.