Informe técnico — Detector de grooming.

1) Resumen ejecutivo — ¿qué hace este programa?

Este proyecto es una aplicación web sencilla (hecha con **Streamlit**) que recibe mensajes o conversaciones y usa un modelo de **aprendizaje automático** para decidir si cada texto tiene características asociadas a *grooming* (conducta de acercamiento y manipulación online hacia menores). El modelo es un **Naive Bayes multinomial** entrenado sobre representaciones TF-IDF del texto. La aplicación muestra una probabilidad de grooming por mensaje, puede analizar un solo mensaje o un archivo completo (.txt/.csv) y entrega tablas y contadores con los resultados.

2) Cómo funciona — explicación paso a paso en lenguaje natural

1. **Interfaz web (Streamlit):**
    - El usuario abre la app y elige entre: analizar un mensaje individual o subir una conversación completa.
    - Si analiza un mensaje, puede escribirlo en un chat; si sube un archivo, la app lee cada línea o la columna mensaje del CSV.
2. **Entrenamiento (al iniciar la app):**
    - Hay una función entrenar_modelo() que contiene una lista manual de 20 frases de ejemplo con etiquetas grooming / no grooming.
    - Se transforma ese texto con **TF-IDF** (vectores numéricos que representan la importancia de palabras) y se entrena un **Multinomial Naive Bayes** sobre esos vectores. El vectorizador y el modelo se devuelven y se reutilizan durante la sesión.
3. **Predicción:**
    - Para cada texto nuevo se transforma con el mismo TF-IDF y el modelo calcula:
        - **Predicción categórica:** grooming o no grooming.
        - **Probabilidad** asociada a la clase grooming (un número entre 0 y 1).

    - La app muestra un progreso visual y un mensaje (advertencia o confirmación). En análisis por archivo, devuelve una tabla con cada mensaje, su predicción y la probabilidad.

4. **Recursos:**
    - La interfaz añade enlaces para ayuda (por ejemplo, la Línea 102 de Argentina).

3) Componentes técnicos clave (breve, no intimidante)

- **Streamlit:** biblioteca para crear aplicaciones web interactivas en Python (maneja la UI).
- **TfidfVectorizer (sklearn):** convierte texto a números dando más peso a palabras distintivas.
- **MultinomialNB (Naive Bayes):** clasificador probabilístico simple, rápido y común para texto. Da probabilidades útiles pero tiene limitaciones si los datos son pocos o no representativos.
- **Pandas:** manejo de archivos CSV y presentación tabular de resultados.

4) Principales limitaciones y riesgos (es fundamental entender esto)

1. **Dataset extremadamente pequeño y artificial:** el modelo se entrena con sólo 20 frases creadas manualmente. Eso **no** es suficiente para aprender la variabilidad del lenguaje real. Resultado: **altas tasas de error**, especialmente falsos positivos y falsos negativos.
2. **Sesgo en ejemplos:** las frases de entrenamiento contienen patrones muy concretos (p. ej. “no le digas a nadie”) — el modelo aprende esos indicadores literales. Mensajes con intención dañina pero sin esas palabras pueden pasar desapercibidos.
3. **Preprocesamiento mínimo:** el código no normaliza (minusculas, lematización, eliminación de stopwords) ni limpia URLs, emojis, abreviaturas o errores ortográficos comunes en chats. Eso reduce la robustez frente a texto real.

4. **Ausencia de evaluación rigurosa:** no hay separación en datos de entrenamiento/validación ni métricas (precision/recall/F1). No sabemos qué tan fiable es la probabilidad que muestra la app.
5. **Privacidad y seguridad:** procesar conversaciones sensibles requiere medidas (encriptación, no almacenar datos si no es necesario, consentimiento y manejo legal). La app actual guarda mensajes en st.session_state (memoria de la sesión) y permite subir archivos, pero no muestra controles de privacidad.
6. **Responsabilidad y consecuencias:** una clasificación automática no debe usarse como único criterio para acciones legales o sanciones. Debe existir **revisión humana** y protocolos claros ante detecciones.

5) Recomendaciones prácticas e inmediatas (prioritarias)

Datos y modelo

- **Ampliar el dataset** con ejemplos reales y etiquetados por expertos: conversaciones reales anonimizadas o datasets públicos/colaborativos.
- **Dividir datos** en *train/validation/test* y medir **precision, recall y F1** — especialmente interesan *recall* para la clase grooming (queremos no perder casos reales) y *precision* para evitar falsas acusaciones.
- **Aumentar preprocesamiento**: minúsculas, normalizar acentos, eliminar URLs/mentions, tokenizar, opcionalmente lematizar, manejar emojis y jerga.
- **Probar modelos alternativos**: Logistic Regression, SVM, o modelos basados en embeddings (Word2Vec, FastText) o transformers (BERT) si hay datos suficientes.
- **Calibrar umbral:** no usar la etiqueta cruda del clasificador; usar la probabilidad y un umbral ajustado para balancear falsos positivos/negativos.

Producción y seguridad

- **Human-in-the-loop:** cualquier alerta automática debería revisarla una persona capacitada antes de tomar medidas.
- **Registro y trazabilidad:** guardar solo metadatos, no contenidos sin consentimiento; log de decisiones y probas.

- **Protección de datos:** cifrado en tránsito/almacenamiento y políticas claras de retención.
- **Notificaciones y protocolos:** definir el flujo (a quién alertar, cuándo escalar a autoridades) y mantener documentación legal y de privacidad.

Mejora de UX (interfaz)

- Mostrar **explicaciones** simples por predicción: cuáles palabras influyeron más (por ejemplo, mostrar las 3 palabras que más contribuyeron a la predicción) para dar contexto al usuario.
- Añadir **métricas** visibles (si hay conjuntos de prueba) y un aviso que recuerde que la herramienta no sustituye a un profesional.

6) Cambios de código sugeridos (fragmentos y apuntes rápidos)

1. **Preprocesamiento básico** antes de vectorizar:

```
import re

def limpiar(texto):

   texto = texto.lower()

   texto = re.sub(r"http\S+|www.\S+", " ", texto)      # eliminar URLs

   texto = re.sub(r"[^a-záéíóúüñ0-9\s]", " ", texto)  # quitar puntuación (con acentos)

   texto = re.sub(r"\s+", " ", texto).strip()

   return texto

# aplicar limpiar() a mensajes antes de vectorizar
```

2. **Guardar modelo** para producción:

```
import joblib

joblib.dump((vectorizador, modelo), "modelo_grooming.joblib")

# y luego cargar con joblib.load en la app desplegada
```

3. **Evaluación mínima**:

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

X = vectorizador.fit_transform(mensajes_limpios)

X_train, X_test, y_train, y_test = train_test_split(X, etiquetas, test_size=0.2,
random_state=42)

modelo.fit(X_train, y_train)

y_pred = modelo.predict(X_test)

print(classification_report(y_test, y_pred))
```

4. **Explicabilidad simple (palabras que influyen):**

```
def top_features_for_text(text, vectorizer, model, top_n=3):

  x = vectorizer.transform([text])

  feature_names = vectorizer.get_feature_names_out()

  log_probs = model.feature_log_prob_

  class_index = list(model.classes_).index("grooming")

  # aproximación: multiplicar tfidf * log_prob

  vals = (x.toarray()[0] * log_probs[class_index])

  top_idx = vals.argsort()[-top_n:][::-1]

  return [(feature_names[i], vals[i]) for i in top_idx]
```

7) Qué hacer ya mismo (acciones concretas y rápidas)

1. **No usar la app como única prueba.** Trátala como herramienta de triage.
2. **Agregar preprocesamiento** mínimo (ver fragmento arriba) y volver a entrenar.
3. **Recolectar y etiquetar más ejemplos** reales: al menos cientos de ejemplos por clase para un modelo básico fiable.

4. **Implementar revisión humana** antes de cualquier escalamiento.
5. **Añadir aviso legal / de privacidad** en la UI explicando limitaciones y qué se hará con los datos subidos.

8) Resumen final (en 2 frases)

La aplicación es una **prueba de concepto válida**: combina TF-IDF + Naive Bayes y una interfaz usable para detectar señales de grooming. Sin embargo, **hoy no es fiable** para decisiones automáticas por su entrenamiento limitado, falta de evaluación y riesgos de privacidad; antes de usarla en contextos reales es imprescindible ampliar datos, evaluar y añadir controles humanos y de seguridad.

Trabajo hecho por: Gil Lascano Lorenzo, Bayaslian Santiago, Buchholz Ariel, Núñez Mauro.