Reporte CRISP-DM – Detección de Spam en SMS
Grupo #2

1. Business Understanding
   - El objetivo del proyecto es proteger a los usuarios identificando SMS fraudulentos mediante clasificadores ML y DL, con un criterio de éxito ≥95 % de exactitud.
   - La motivación incluye reducir fraudes, mejorar la experiencia del usuario y ahorrar costos operativos en la infraestructura de mensajería.

2. Data Understanding
   - Se utiliza spam_dataset_es.csv, una versión traducida al español con 5 572 mensajes (13.41 % spam) cargada desde disco y normalizada a las columnas Target y Texto.
   - Durante la ejecución de la pipeline se validó el balance de clases (4 825 ham / 747 spam) para ajustar acciones posteriores de preparación.

3. Data Preparation
   - Se definió un limpiador que corrige encoding, normaliza Unicode, anonimiza URLs, filtra caracteres, pasa a minúsculas, tokeniza con NLTK y elimina stopwords en español.
   - Los textos limpios alimentan un TfidfVectorizer limitado a 5 000 términos y luego se dividen en train/test con estratificación; SMOTE se aplica sobre train para mitigar el desbalance.
   - Se entrena un primer baseline SVM lineal alcanzando 0.982 de accuracy y f1=0.93 para spam sobre el split clásico de sklearn, superando el umbral de negocio de 95 %.

4. Modeling
   - Se agrega un FFNN (MLPClassifier con una capa oculta de 128 neuronas) sobre las mismas features TF-IDF para medir la ganancia de un modelo no lineal.
   - Paralelamente se implementa un pipeline completo para un LSTM bidireccional: tokenización propia, vocabulario de 10k palabras, padding a 100 tokens, Dataset personalizado y dataloaders para train/val/test.
   - El entrenamiento del LSTM (3.65 M parámetros) se ejecuta por una época con Adam. Aunque alcanza 0.90 de accuracy en train y 0.97 en validación, el rendimiento en test cae (accuracy 0.145) por sesgo del dataset y falta de ajuste de umbral, revelando la necesidad de mayor regularización/entrenamiento (src/mlops_crispdm/modeling.py (lines 284-333)).

5. Evaluation
   - Se unificó un test set (836 ejemplos) para comparar modelos bajo las mismas instancias. Resultados:

Modelo	Accuracy	Precision	Recall	F1-Score
SVM (Baseline)	0.9880	0.9397	0.9732	0.9561
FFNN (Baseline)	0.9809	0.8934	0.9732	0.9316
LSTM	0.1447	0.1345	0.9911	0.2369

   - Se realizó un análisis cualitativo de errores para el LSTM, listando falsos positivos y negativos que evidencian sobreajuste a patrones sentimentales o conversacionales. Esto sirve para priorizar mejoras en la fase de modelado.

6. Deployment
   - Se empaqueta una función predict_with_lstm que transforma texto crudo a secuencias y ejecuta inferencia en el modelo entrenado, mostrando probabilidades para frases nuevas de negocio.
   - El CLI mlops_crispdm.cli ejecuta toda la pipeline secuencialmente, lo que facilita integrarlo en CI/CD (el proyecto ya incluye publicación en PyPI/Docker y workflows de GitHub Actions, descritos en README.md).

Conclusiones, Escalabilidad y Próximos Pasos

Conclusiones generales
Los baselines clásicos (SVM y FFNN) ya superan el criterio de éxito (>98 % accuracy) con un pipeline reproducible de limpieza, vectorización y balanceo. El enfoque LSTM necesita reevaluarse debido a su pobre generalización actual; sin ajustes, no aporta valor frente a los modelos ligeros.

Escalabilidad técnica
Para sostener el crecimiento del proyecto, es crucial desacoplar el preprocesamiento (limpieza + TF-IDF) en artefactos serializables, versionar datasets traducidos y exponer los modelos mediante endpoints o microservicios. El actual LSTM opera en CPU y con un vocabulario fijo; para soportar catálogos multilingües o volúmenes mayores conviene migrar a embeddings pre-entrenados (p. ej., BETO, DistilBERT) que se beneficien de transfer learning sin necesidad de vocabularios hechos a mano.

Próximos pasos recomendados
1. Ajustar y recalibrar el LSTM (más épocas, scheduler, pos_weight, threshold tuning) o reemplazarlo por modelos transformers finos para español; añadir validaciones cruzadas y pruebas unitarias de preprocesamiento.
2. Persistir vectorizer, svm y mlp con joblib, exponerlos vía FastAPI/Gradio y automatizar su despliegue usando las imágenes Docker ya definidas.
3. Implementar monitoreo de drift y alertas (p. ej., comparar tasas de spam reales vs. esperadas), junto con pipelines de retraining orquestados (Airflow/Prefect) que consuman nuevos datos etiquetados.
4. Ampliar el dataset con mensajes recientes y multi-región para robustecer la detección de nuevas variantes de spam y evaluar escalabilidad horizontal (sharding por región, colas de inferencia, etc.).