Memoria del Proyecto

Predicción del nivel de 'engagement' turístico a partir de imágenes y datos tabulares
Primer modelo de Deep Learning con datos de Artgonuts

Enfoque y estrategia

Mi objetivo principal al abordar esta práctica fue construir un modelo funcional, aunque sencillo en un primer momento, y posteriormente, si disponía de tiempo, aumentar progresivamente su complejidad. Para ello, comencé con una exploración inicial de los datos, tanto tabulares como visuales.

Exploración y preparación de los datos

A nivel tabular, llevé a cabo un análisis exploratorio básico: detección y tratamiento de valores nulos, visualización de distribuciones, y codificación de variables categóricas. En paralelo, inspeccioné las imágenes, observando que algunas presentaban niveles atípicos de saturación y contraste. Por ello, decidí aplicar técnicas de data augmentation para aumentar la diversidad del conjunto de entrenamiento.

A continuación, construí una métrica de engagement personalizada. Basándome en el análisis de correlaciones, propuse la siguiente fórmula:

$$\text{engagement} = 0.8 \times \text{likes} + \text{bookmarks} + 0.2 \times \text{visits} - 0.5 \times \text{dislikes}$$

Posteriormente, transformé esta métrica en tres clases balanceadas para representar distintos niveles de relevancia de las atracciones turísticas.

Eliminé las variables textuales (no procesadas en este proyecto) y las que formaban parte de la fórmula de engagement para evitar fugas de información.

División de los datos y organización del proyecto

Dividí el conjunto en entrenamiento (80 %), validación (20 % del entrenamiento) y test (20 %). En un primer momento omití el uso de semillas, pero lo corregí posteriormente en el notebook de optimización para garantizar la reproducibilidad, siguiendo las buenas prácticas del curso.

En esta fase, decidí organizar el proyecto en notebooks independientes por fases (EDA, modelado, etc.) y guardar funciones reutilizables en un script `utils.py`. Sin embargo, me encontré con varios problemas de sincronización con Google Colab, incluso tras clonar el repositorio y subir archivos desde local. La única solución funcional fue montar Google Drive y acceder a una copia manual del archivo `utils.py`. Esta complicación me hizo descartar la estructura modular temporalmente y continuar el desarrollo en la rama principal (`main`).

Arquitectura del modelo

Mi primer modelo fue una red híbrida: utilicé una CNN preentrenada para procesar imágenes (descongelando únicamente la última capa) combinada con una red fully connected para los datos tabulares. Ambas salidas se concatenaban y pasaban por un clasificador final, siguiendo una arquitectura sugerida en las sesiones prácticas.

Amplié ligeramente esta estructura añadiendo dos capas adicionales tanto en el modelo de características tabulares como en el clasificador. Desarrollé el código de entrenamiento inicial con esta versión para evaluar su rendimiento.

Optimización con Optuna

Comencé ajustando hiperparámetros manualmente, de forma exploratoria, pero decidí avanzar hacia una optimización sistemática con Optuna. Creé una nueva rama del proyecto (`optimización`) y adaptando el código para hacerlo más parametrizable y reproducible (incluyendo semillas y control de configuraciones).

Dado que era el último día que podía dedicar al proyecto, opté por simplificar la arquitectura y reducir el espacio de búsqueda para garantizar un resultado funcional. El nuevo modelo incluía una sola capa oculta para los datos tabulares y una capa para el clasificador, ambas con normalización y dropout. Optimicé el número de neuronas, tasa de dropout, learning rate y batch size. Debido a la falta de GPU en Colab, trabajé con CPU, lo que limitó la capacidad de experimentar con configuraciones más complejas.

Evaluación de resultados

El mejor modelo alcanzó una precisión del 80 % en validación, aunque no se observó una mejora clara tras ese punto. Entrené el modelo final desde cero con los mejores parámetros encontrados. Para evaluar su robustez, comparé dos enfoques:

1. Entrenamiento con los conjuntos train y validation separados.
2. Entrenamiento unificado (train + validation).

El segundo enfoque no mejoró significativamente la precisión sobre el conjunto de test. En el entrenamiento separado, observé signos de sobreajuste a partir de la época 5 (divergencia entre curvas de entrenamiento y validación), que podrían haberse mitigado con early stopping, como apliqué en un notebook previo (`optimization_trial1`). El entrenamiento con los datos combinados produjo una curva más suave, pero no comparable en términos de validación.

Conclusiones y pasos futuros

Tengo la impresión de que la primera arquitectura explorada ofrecía un mayor potencial, pero los problemas técnicos con Colab y la frustración derivada afectaron algunas de mis decisiones posteriores. Aun así, me siento satisfecha de haber llegado a un modelo funcional, superando dificultades técnicas con creatividad y resiliencia.

Para seguir avanzando, me gustaría:

- Reintroducir una o dos capas en el modelo de características tabulares y en el clasificador.
- Explorar más ampliamente los rangos de hiperparámetros como dropout, learning rate, weight decay y función de activación.
- Analizar a fondo los casos de error del modelo para identificar patrones y posibles mejoras.

Quiero destacar que el asistente Gemini integrado en Colab fue de gran ayuda para depurar errores y acelerar el desarrollo. Sin esta herramienta, probablemente no habría podido completar la práctica a tiempo.

En resumen, ha sido una experiencia muy didáctica y gratificante, especialmente al ver el modelo funcionando como se esperaba.