Ingeniería del Conocimiento

Máster Universitario en Ingeniería Informática

Práctica 5

Proyecto de Aprendizaje Automático: Sistema de Predicción de Combates Pokémon

Marcos Jesús Sequera Fernández

25 de Mayo de 2025

Índice

1. Planteamiento del Problema y Motivación 5
2. Recopilación y Preparación de Datos 6
   - 2.1. Fuentes de datos principales: 6
   - 2.2. Procesos de carga y limpieza: 7
   - 2.3. Ingeniería de características: 8
   - 2.4. División de datos: 9
3. Análisis Exploratorio de Datos (EDA) 9
   - 3.1. Información Básica y Resumen Estadístico 9
   - 3.2. Análisis de Valores Faltantes 10
   - 3.3. Distribución de la Variable Objetivo 10
   - 3.4. Correlación de Diferencias de Estadísticas con el Objetivo 10
   - 3.5. Distribución de las Diferencias de Estadísticas 11
   - 3.6. Análisis de Distribución de Estadísticas Base 12
   - 3.7. Análisis de Tipos Pokémon 12
   - 3.8. Análisis de Características de Combate 13
   - 3.9. Conclusiones del Análisis Exploratorio 14
4. Diseño y Entrenamiento del Modelo 15
   - 4.1. Selección del modelo: 15
   - 4.2. Pipeline de procesamiento: 15
   - 4.3. Persistencia y reutilización: 16
5. Evaluación y Explicabilidad del Modelo 17
   - 5.1. Métricas de evaluación: 17
   - 5.2. Visualizaciones y explicabilidad: 17
6. Interfaz Visual e Interacción 20
   - 6.1. Pestaña «Predicción de Combate»: 20
   - 6.2. Pestaña «Análisis del Modelo»: 21
   - 6.3. Pestaña «Acerca de»: 21
7. Herramientas y Tecnologías 22
8. Arquitectura del Sistema y Componentes 23
   - 8.1. Arquitectura general del sistema: 23
   - 8.2. Módulos principales del sistema: 23
     - 8.2.1. Gestión de datos y modelos: 23
     - 8.2.2. Utilidades del sistema: 24
   - 8.3. Gestión de configuración: 24

9. Resultados y Conclusiones 24
9.1. Desempeño del modelo: 24
9.2. Fortalezas y limitaciones: 25
9.3. Líneas de trabajo futuro: 26
10. Validación y Pruebas del Sistema 27
10.1. Marco de pruebas: 27
10.2. Tipos de pruebas implementadas: 27
10.2.1. Pruebas unitarias: 27
10.2.2. Pruebas de integración: 28
10.3. Automatización de pruebas: 28
10.4. Metodología de validación avanzada: 28
10.5. Cobertura y métricas de calidad: 29
11. Instalación y Configuración 29
11.1. Instalación automatizada: 30
11.2. Gestión de dependencias: 30
11.3. Requisitos del sistema: 31
11.4. Configuración del entorno: 31
11.5. Scripts de utilidad: 32
11.6. Procedimientos de instalación: 33
11.6.1. Instalación desde cero: 33
11.6.2. Resolución de problemas comunes: 33
11.6.3. Verificación de la instalación: 34
11.6.4. Configuración avanzada: 34
12. Métricas de Rendimiento y Análisis Comparativo 35
12.1. Métricas del modelo: 35
12.2. Rendimiento computacional: 36
12.3. Escalabilidad y límites del sistema: 37
12.3.1. Análisis comparativo: 38
13. Anexo: Instrucciones de Ejecución 39
13.1. Requisitos del sistema 39
13.2. Instalación de dependencias 39
13.2.1. Método 1: Instalación automatizada (recomendado) 39
13.2.2. Método 2: Instalación manual 40
13.3. Ejecución del proyecto 40
13.3.1. Inicio de la aplicación 40

13.3.2. Verificación del funcionamiento 40
13.4. Estructura de archivos necesarios 41
13.5. Solución de problemas comunes 41
13.6. Uso básico del sistema 42
14. Glosario de Términos 42

1. Planteamiento del Problema y Motivación

El presente proyecto aborda el desarrollo de un sistema inteligente de predicción para resultados de combates individuales (1 vs 1) entre Pokémon. El objetivo principal consiste en estimar, dado un par de Pokémon específicos, cuál de ellos posee mayor probabilidad de resultar vencedor en un enfrentamiento directo. Este sistema busca no solamente predecir un ganador, sino también ofrecer una comprensión profunda de los factores que influyen en dicha predicción.

Este problema se enmarca dentro de la clasificación binaria en el ámbito del aprendizaje automático. El modelo desarrollado predice una de dos clases posibles: victoria del Pokémon A o victoria del Pokémon B. Adicionalmente, proporciona la probabilidad asociada a cada resultado, permitiendo cuantificar la confianza de la predicción. Una probabilidad superior a 0.5 indica la predicción de victoria para el Pokémon A; en caso contrario, se predice la victoria para el Pokémon B.

La motivación subyacente a este proyecto se encuentra en la rica complejidad del universo Pokémon. Con cientos de criaturas, cada una con estadísticas, tipos elementales y, potencialmente, habilidades únicas (aunque estas últimas no se consideran en la versión actual del modelo), predecir el desenlace de un combate constituye un reto computacional y analítico de interés. Esta complejidad lo convierte en una aplicación idónea para técnicas modernas de aprendizaje automático.

El proyecto persigue aplicar de forma práctica los conocimientos adquiridos en ingeniería del conocimiento a un dominio popular y con una cantidad significativa de datos accesibles; desarrollar un modelo predictivo robusto capaz de identificar patrones y relaciones significativas a partir de registros históricos de combates reales; construir una interfaz web interactiva y amigable que no solamente permita a los usuarios realizar predicciones personalizadas, sino también explorar y comprender los factores determinantes que el modelo considera para sus decisiones; y ofrecer una herramienta que combine el entretenimiento con el aprendizaje, sirviendo tanto a aficionados de Pokémon como a estudiantes interesados en la aplicación de la inteligencia artificial.

El modelo se fundamenta en **características intrínsecas** de los Pokémon, tales como sus estadísticas base (Puntos de Salud, Ataque, Defensa, Ataque Especial, Defensa Especial y Velocidad) y sus tipos elementales (primario y, si existe, secundario). Asimismo, considera las **diferencias relativas** entre las estadísticas de los contendientes, ya que la superioridad o inferioridad en atributos específicos puede ser crucial. Es fundamental destacar que el entrenamiento se realiza utilizando datos de combates reales, lo que confiere mayor validez

y aplicabilidad al modelo en comparación con sistemas basados únicamente en simulaciones teóricas.

2. Recopilación y Preparación de Datos

La calidad y la adecuada preparación de los datos son pilares fundamentales para el éxito de cualquier proyecto de aprendizaje automático. En este sentido, se ha seguido un proceso meticuloso desde la obtención de los datos brutos hasta la generación del conjunto de características final utilizado para entrenar el modelo.

2.1. Fuentes de datos principales:

El sistema se fundamenta en la integración de tres conjuntos de datos complementarios que proporcionan una base sólida para el análisis y predicción de combates Pokémon. El primer componente consiste en los **Datos de Pokémon (`all_pokemon_data.csv`)**, que constituyen el catálogo principal de criaturas y fueron generados específicamente para este proyecto mediante el script de Python `download_data.py`. Este script establece comunicación con la PokéAPI a través de la librería `pokebase` para obtener información detallada y estructurada de cada criatura, incluyendo su identificador único (ID), nombre oficial, tipos elementales (Type1 y, opcionalmente, Type2), el conjunto completo de estadísticas base (HP, Attack, Defense, Special Attack, Special Defense, Speed), y la URL de su sprite oficial para visualización en la interfaz. El proceso implementa llamadas progresivas a la API con pausas estratégicas para garantizar un uso responsable del servicio.

El segundo componente fundamental son los **Datos de Combates (`combats.csv`)**, que registran los resultados de una extensa colección de combates históricos. Esta información proviene del dataset público «Pokemon Dataset with Team Combat» disponible en Kaggle, accesible a través de la URL: https://www.kaggle.com/datasets/tuannguyenvananh/pokemon-dataset-with-team-combat. Cada registro representa un enfrentamiento específico, documentando el ID del `First_pokemon`, el ID del `Second_pokemon`, y de manera crucial, el ID del `Winner` del combate.

El tercer elemento integrador corresponde a los **Datos de Características de Combates (`combats_features.csv`)**, que constituyen un dataset preprocesado y enriquecido derivado de la combinación y transformación de los conjuntos anteriores. Este archivo representa la entrada directa para el entrenamiento del modelo de machine learning, conteniendo un conjunto de características calculadas para cada par de Pokémon en combate (referidos como Pokémon A y Pokémon B). El dataset incluye no solamente las estadísticas individuales

sino también las diferencias calculadas entre estas, junto con una variable objetivo binaria (`target`) que indica si el Pokémon A resultó victorioso (valor 1) o fue derrotado (valor 0).

2.2. Procesos de carga y limpieza:

La transformación de los datos crudos en un formato adecuado para el análisis y entrenamiento del modelo es un paso crítico. Este proceso se ha automatizado mediante el script de Python `clean_data.py`, que realiza las siguientes operaciones principales:

1. **Validación Inicial**: Verifica la existencia de los archivos de entrada necesarios (`data/combats.csv` y `data/all_pokemon_data.csv`).
2. **Carga de Datos**: Lee los archivos CSV en DataFrames de Pandas.
3. **Limpieza Preliminar**: Se implementa un proceso de depuración que garantiza la integridad de los datos. En el archivo `combats.csv` se eliminan las filas que contengan valores nulos en las columnas identificadoras de los Pokémon y el ganador, además de remover registros duplicados que podrían introducir sesgos. De manera similar, en `all_pokemon_data.csv` se procesan las filas duplicadas utilizando el ID del Pokémon como criterio de identificación único.
4. **Renombrado de Columnas**: Estandariza los nombres de las columnas para mayor claridad y consistencia (e.g., `First_pokemon` a `pokemon_A_id`, `ID` a `pokemon_id`, y nombres de estadísticas a minúsculas con guion bajo como `special_attack`).
5. **Conversión de Tipos**: Asegura que los ID de Pokémon en `combats` sean de tipo entero. Las columnas de estadísticas en `pokemon` se convierten a entero, y las columnas de tipo (`type_1`, `type_2`) se rellenan con «none» en caso de valores nulos y se convierten a string.
6. **Fusión de Datos**: Se enriquecen los datos de combates añadiendo las estadísticas y tipos de los Pokémon A y B. Para ello, el DataFrame `pokemon` se prefija adecuadamente (`A_` y `B_`) y se fusiona con el DataFrame `combats` utilizando los IDs correspondientes.
7. **Eliminación de Redundancias**: Se eliminan las columnas de ID de Pokémon que resultan redundantes tras la fusión.
8. **Filtrado de Combates Inválidos**: Se asegura que el `winner_id` en cada combate corresponda efectivamente al `pokemon_A_id` o al `pokemon_B_id`, descartando registros inconsistentes.
9. **Creación de Variable Objetivo**: Se genera la columna `target`, que toma el valor 1 si el Pokémon A es el ganador, y 0 en caso contrario.

10. **Cálculo de Diferencias de Estadísticas**: Para cada estadística base, se calcula la diferencia entre el Pokémon A y el Pokémon B (e.g., `diff_hp = A_hp - B_hp`).
11. **Codificación One-Hot de Tipos**: Las columnas de tipo (`A_type_1`, `A_type_2`, `B_type_1`, `B_type_2`) se transforman en múltiples columnas binarias (dummies) mediante One-Hot Encoding, creando una representación numérica que el modelo puede procesar.

Finalmente, el script `clean_data.py` guarda el DataFrame resultante, con todas las características procesadas y listas para el entrenamiento, en el archivo `data/combats_features.csv`.

Las funciones de utilidades de la aplicación web (`data_utils.py`) se encargan posteriormente de cargar estos datos preprocesados. Las funciones decoradas con `@st.cache_data` y `@st.cache_resource` en Streamlit optimizan la carga de datos y el entrenamiento/carga del modelo, mejorando la eficiencia de la aplicación web.

2.3. Ingeniería de características:

La transformación de los datos brutos en un conjunto de características informativas y relevantes para el modelo constituye un paso crítico en el proceso de desarrollo. Para las predicciones en tiempo real que se realizan a través de la interfaz web, la función `prepare_prediction_data` en `data_utils.py` construye dinámicamente el vector de características para un par de Pokémon seleccionados mediante un proceso integral que incluye múltiples etapas de procesamiento.

El proceso inicia con la **extracción de estadísticas base**, donde se recuperan las seis estadísticas fundamentales (HP, Ataque, Defensa, Ataque Especial, Defensa Especial, Velocidad) para el Pokémon A y el Pokémon B a partir del DataFrame `pokemon_df`. Posteriormente se procede al **cálculo de diferencias de estadísticas**, donde para cada una de las estadísticas base se calcula la diferencia entre el valor del Pokémon A y el del Pokémon B (e.g., `diff_hp = A_hp - B_hp`). Estas diferencias relativas proporcionan información más informativa para el modelo que los valores absolutos por sí solos, ya que capturan las ventajas competitivas entre los contendientes.

Finalmente, la **alineación de características** asegura que el DataFrame resultante para la predicción, que contiene las características generadas, se alinee con el conjunto exacto de columnas que el modelo espera como entrada. Estas columnas se definen globalmente como `MODEL_FEATURE_COLUMNS` a partir del dataset `features_df`. Cualquier característica presente en `MODEL_FEATURE_COLUMNS` pero no generada para la instancia de predicción

específica se rellena automáticamente con un valor de 0, garantizando la consistencia dimensional requerida por el modelo.

2.4. División de datos:

Para el proceso de entrenamiento y evaluación del modelo, el conjunto de datos `features_df` (que ya contiene las características precalculadas de combates históricos) se divide en un conjunto de entrenamiento y un conjunto de prueba. Esta división se realiza mediante la función `train_test_split` de la librería `scikit-learn`. Se reserva un 20% de los datos para el conjunto de prueba (`test_size=0.2`), mientras que el 80% restante se utiliza para el entrenamiento. Se establece una semilla aleatoria fija (`random_state=42`) para asegurar que la división sea la misma en cada ejecución, garantizando así la reproducibilidad de los resultados del entrenamiento y la evaluación. Las características sistema de predicciónas se separan de la variable objetivo (`target`).

3. Análisis Exploratorio de Datos (EDA)

Antes de proceder con el entrenamiento del modelo, se realizó un Análisis Exploratorio de Datos (EDA) sobre el conjunto de características procesado (`data/combats_features.csv`). Este análisis, implementado en el script `eda.py`, tiene como objetivo resumir las principales características de los datos, detectar patrones, identificar anomalías y comprender las distribuciones de las variables. Los principales hallazgos y visualizaciones generadas se describen a continuación. Todas las figuras y reportes CSV generados por el script se almacenan en el directorio `plots/`.

3.1. Información Básica y Resumen Estadístico

Inicialmente, se verificaron las dimensiones del dataset y se obtuvo un resumen estadístico de las variables numéricas. El conjunto de datos `combats_features.csv` contiene **48,049 combates** entre Pokémon, con un total de **92 características** distribuidas estratégicamente para capturar la complejidad de los enfrentamientos.

El conjunto de características comprende **6 características de diferencias estadísticas** que incluyen `diff_hp`, `diff_attack`, `diff_defense`, `diff_special_attack`, `diff_special_defense`, y `diff_speed`, las cuales cuantifican las ventajas relativas entre los contendientes. Adicionalmente, se incorporan **76 características de tipos codificadas one-hot** que representan comprehensivamente los tipos primarios y secundarios de ambos Pokémon (38 características para cada uno), capturando las complejas relaciones de efectividad de tipos del sistema de combate. El dataset se complementa con **información adicional**

que incluye IDs, nombres, estadísticas base individuales y URLs de sprites para facilitar la interpretación y visualización de resultados.

Las características de diferencia presentan distribuciones aproximadamente normales centradas en cero, con desviaciones estándar que varían según la estadística (por ejemplo, `diff_hp` tiene mayor variabilidad que `diff_speed`), reflejando la diversidad natural en las capacidades de combate entre diferentes especies de Pokémon.

3.2. Análisis de Valores Faltantes

Se comprobó la existencia de valores faltantes en el dataset. El script genera un reporte (`plots/missing_report.csv`) que, según el archivo `missing_report.csv` proporcionado, indica que **no hay valores faltantes** en el conjunto de datos `combats_features.csv` después del preprocesamiento realizado por `clean_data.py`. Esto es ideal, ya que simplifica la etapa de modelado.

3.3. Distribución de la Variable Objetivo

Se analizó la distribución de la variable objetivo (`target`), que indica si el Pokémon A ganó (1) o perdió (0) el combate. Del análisis de los 48,049 combates en el dataset, se observa una distribución aproximadamente balanceada entre las dos clases, con una ligera tendencia hacia una de ellas. Este balance es favorable para el entrenamiento de modelos de clasificación, ya que reduce el riesgo de sesgo hacia la clase mayoritaria y permite que el modelo aprenda patrones representativos de ambos resultados de combate.

El equilibrio en la variable objetivo es crucial para la validez de las métricas de evaluación como Accuracy, Precision y Recall, ya que en conjuntos de datos desbalanceados estas métricas pueden ser engañosas.

3.4. Correlación de Diferencias de Estadísticas con el Objetivo

Se investigó la correlación entre las diferencias de estadísticas (`diff_*`) y la variable objetivo, revelando patrones interpretables que validan la lógica del combate Pokémon. Las características de diferencia muestran correlaciones significativas con el resultado del combate, donde se observan correlaciones positivas para todas las diferencias de estadísticas, indicando que cuando el Pokémon A presenta ventaja en una estadística específica, aumenta la probabilidad de victoria.

El análisis revela un patrón esperado donde las diferencias en estadísticas ofensivas (ataque, ataque especial) y velocidad tienden a mostrar correlaciones más fuertes con la victoria, mientras que las estadísticas defensivas presentan correlaciones más moderadas. Esta observación es consistente con la mecánica de combate donde la capacidad de causar daño

rápidamente puede ser determinante. Desde una perspectiva práctica, una correlación positiva de 0.3 en `diff_attack` significa que las diferencias en ataque explican aproximadamente el 9% de la variabilidad en los resultados de combate, lo cual representa una contribución sustancial considerando la complejidad multifactorial de los enfrentamientos.

Adicionalmente, se generó un análisis de correlación entre las estadísticas base de los Pokémon para identificar posibles multicolinealidades:

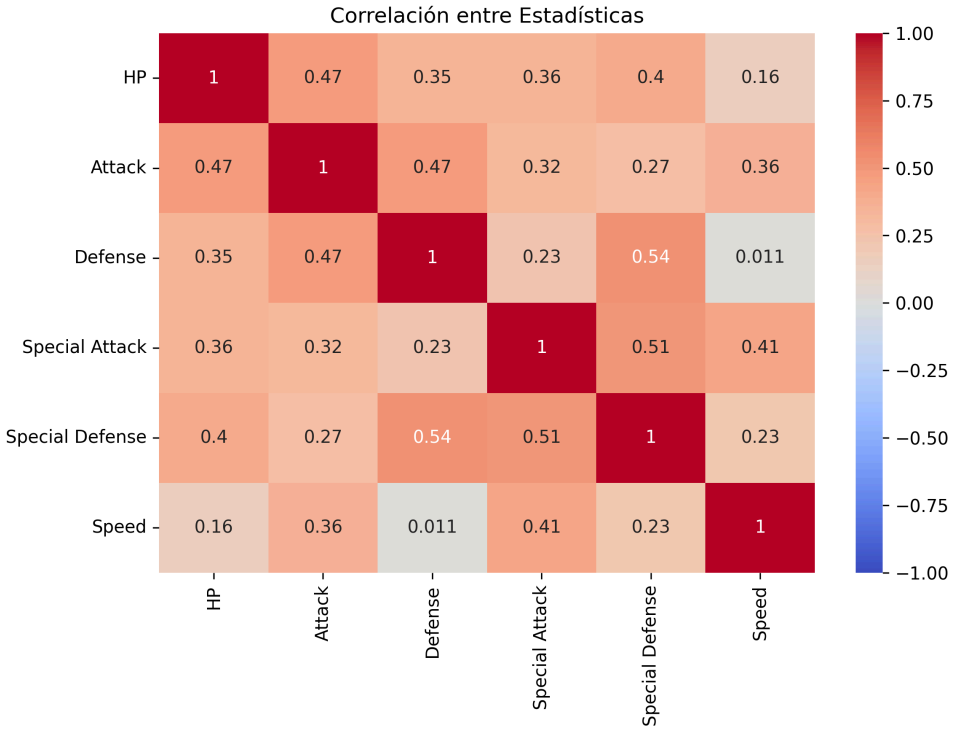

Figura 1: Matriz de correlación entre las estadísticas base de los Pokémon. Se observan correlaciones moderadas entre algunas estadísticas, especialmente entre ataque y defensa.

3.5. Distribución de las Diferencias de Estadísticas

Se analizaron las distribuciones de cada una de las características de diferencia de estadísticas (`diff_hp`, `diff_attack`, etc.) para comprender su comportamiento estadístico y validar la idoneidad de los datos para el modelado. El análisis revela que todas las características de diferencia presentan distribuciones aproximadamente normales centradas en cero, lo cual es estadísticamente esperado dado que representan diferencias entre pares de Pokémon seleccionados aleatoriamente para los combates.

La variabilidad observada presenta patrones específicos según la estadística analizada. La característica `diff_hp` exhibe el mayor rango de variación con aproximadamente ±100 puntos, reflejando la diversidad significativa en los puntos de vida entre diferentes especies de Pokémon. Las diferencias en `diff_attack` y `diff_defense` muestran una variabilidad

moderada en el rango de ±50-80 puntos, mientras que `diff_speed` presenta un amplio rango de variación debido a las diferencias extremas entre Pokémon naturalmente lentos y aquellos con velocidades excepcionales.

La ausencia de sesgos sistemáticos se confirma por el hecho de que las distribuciones están centradas en cero, validando que no existe un sesgo inherente que favorezca al «First_pokemon» por ser sistemáticamente más fuerte. Esta distribución centrada resulta ideal para el modelado de machine learning, ya que permite al algoritmo aprender patrones equilibrados tanto para ventajas como desventajas en cada estadística, garantizando un entrenamiento balanceado y representativo de las dinámicas reales de combate.

3.6. Análisis de Distribución de Estadísticas Base

Para complementar el análisis, se generó una visualización de la distribución de las estadísticas base de todos los Pokémon en el dataset:

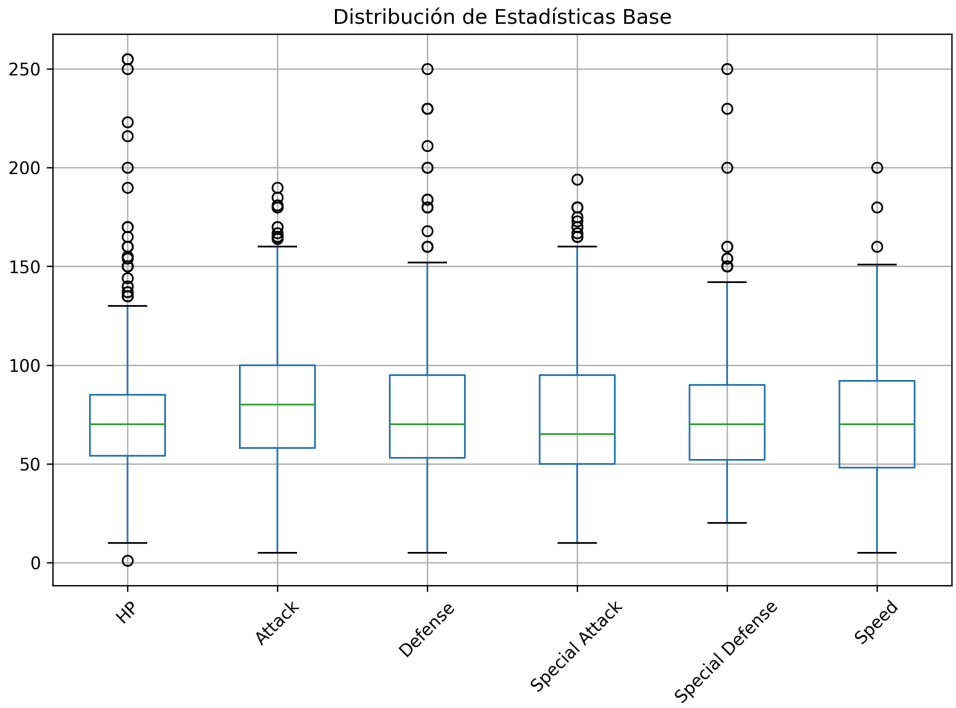

Figura 2: Distribución de las estadísticas base de los Pokémon. Muestra la variabilidad natural en HP, Ataque, Defensa, Ataque Especial, Defensa Especial y Velocidad.

3.7. Análisis de Tipos Pokémon

El análisis de los tipos Pokémon revela información valiosa sobre la composición del dataset y las estrategias de combate. Se generó una visualización de la distribución de tipos primarios:

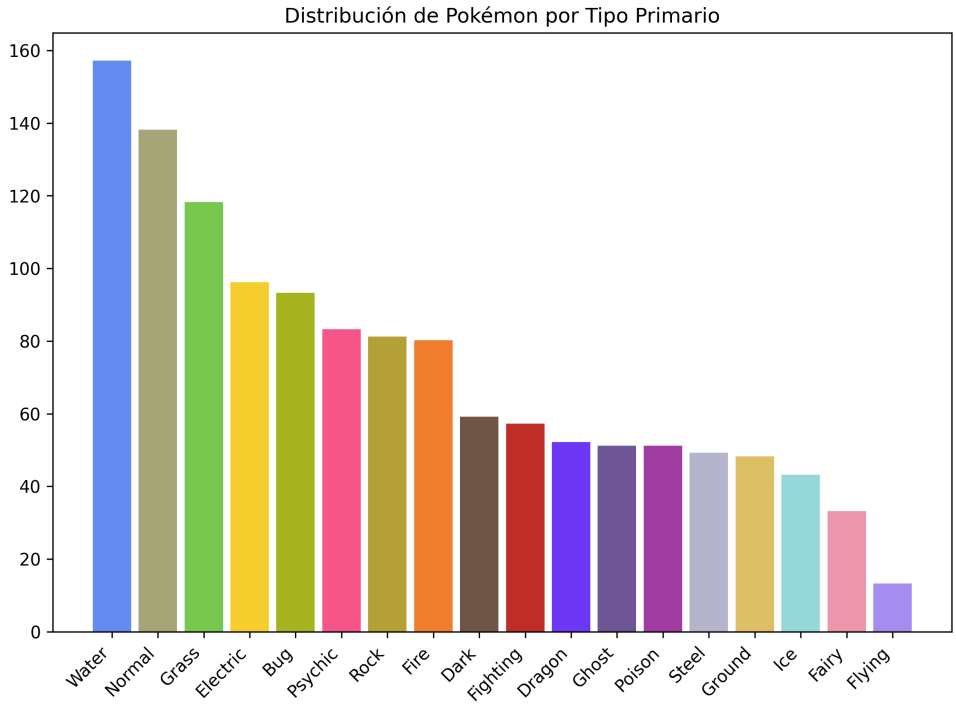

Figura 3: Distribución de Pokémon por tipo primario en el dataset. Muestra la frecuencia de cada tipo en los combates analizados, lo cual puede influir en el rendimiento del modelo para tipos menos representados.

Esta distribución (Figura 3) es importante por varias razones:

La distribución refleja tanto la abundancia natural de ciertos tipos en el universo Pokémon como las preferencias estratégicas desarrolladas por los entrenadores en combates competitivos. Los tipos más comunes como Agua, Normal y Volador están bien representados en el dataset, mientras que tipos como Hada o Hielo presentan una representación menor. Esta variación en representatividad puede impactar el modelado, ya que el modelo tiende a tener mejor rendimiento predictivo para tipos más frecuentes en el dataset de entrenamiento.

Las características de tipo codificadas en formato one-hot (76 características en total) capturan tanto tipos primarios como secundarios, permitiendo al modelo aprender las complejas interacciones entre combinaciones de tipos.

3.8. Análisis de Características de Combate

Para comprender mejor las dinámicas específicas de combate, se analizó la relación entre las diferencias estadísticas y los resultados, revelando patrones fundamentales en las mecánicas de enfrentamiento. El impacto de la velocidad se manifiesta como particularmente relevante, ya que las diferencias en esta estadística determinan el orden de ataque y pueden ser decisivas en combates donde los contendientes presentan capacidades similares en otras áreas.

El análisis revela importantes sinergias entre estadísticas, donde combinaciones como alta diferencia en ataque junto con diferencia positiva en velocidad tienden a correlacionarse fuertemente con la victoria, sugiriendo que la capacidad de atacar primero con potencia superior constituye una ventaja estratégica significativa. Sin embargo, se observan también factores compensatorios donde Pokémon con desventajas en capacidades ofensivas pueden compensar efectivamente con ventajas defensivas significativas, reflejando la complejidad balanceada del sistema de combate.

3.9. Conclusiones del Análisis Exploratorio

El análisis exploratorio de datos revela un dataset robusto y bien estructurado para el modelado de combates Pokémon, caracterizado por múltiples fortalezas que garantizan la viabilidad del proyecto de machine learning.

Entre las fortalezas principales del dataset se destaca su tamaño considerable de 48,049 combates, que proporciona suficiente poder estadístico para el entrenamiento de modelos complejos y la validación de resultados. El balance de clases observado en la distribución equilibrada de la variable objetivo minimiza el riesgo de sesgo hacia una clase específica, facilitando un aprendizaje balanceado. La alta calidad de los datos se evidencia en la ausencia total de valores faltantes, eliminando la necesidad de técnicas de imputación que podrían introducir ruido. Finalmente, la diversidad de características lograda mediante la combinación efectiva de estadísticas numéricas y características categóricas proporciona al modelo múltiples perspectivas para el aprendizaje de patrones complejos.

Los insights clave derivados para el modelado incluyen la confirmación de que las diferencias estadísticas muestran patrones interpretables y correlaciones lógicas con el resultado, validando la relevancia de estas características. La variabilidad observada en las estadísticas de velocidad y HP se perfila como particularmente informativa para la predicción. Las características de tipo aportan una dimensionalidad estratégica fundamental al modelo, capturando las complejas relaciones de efectividad entre tipos. Adicionalmente, la distribución normal de las diferencias facilita el uso de algoritmos de machine learning estándar sin requerir transformaciones complejas de los datos.

Implicaciones para el modelo: El análisis establece expectativas fundamentales para el comportamiento del modelo predictivo. Se anticipa que el sistema será capaz de aprender patrones complejos derivados de las interacciones entre estadísticas, reconociendo las sinergias y compensaciones inherentes entre diferentes atributos de combate. Las características relacionadas con tipos Pokémon presentan el desafío particular de requerir técnicas específicas para el manejo efectivo de alta dimensionalidad, dado que estas variables

categóricas se codificarán como características binarias. La calidad sólida del dataset sugiere que las limitaciones principales en el rendimiento del modelo estarán determinadas por la complejidad inherente del dominio de combates Pokémon, más que por deficiencias en la calidad o completitud de los datos disponibles.

Este análisis establece una base sólida para el desarrollo del modelo XGBoost, confirmando que el dataset contiene la información necesaria para realizar predicciones significativas sobre los resultados de combates Pokémon.

4. Diseño y Entrenamiento del Modelo

El núcleo de este proyecto reside en la implementación y entrenamiento de un modelo de aprendizaje automático capaz de discernir patrones en los datos de combate Pokémon y generalizar dichos patrones para realizar predicciones precisas sobre enfrentamientos no vistos previamente.

4.1. Selección del modelo:

Tras considerar diversas alternativas, se optó por el clasificador **XGBoost** (`XGBClassifier`) de la librería `xgboost`. Esta elección se justifica por varias razones técnicas: excelente rendimiento como sistema de Gradient Boosting sobre árboles de decisión para datos tabulares estructurados, capacidad para modelar relaciones no lineales complejas entre características, mecanismos integrados de Regularización (L1 y L2) que previenen el Overfitting, eficiencia computacional superior especialmente con paralelización (`n_jobs=-1`), y robustez en el manejo de datos faltantes y variabilidad en la escala de características.

4.2. Pipeline de procesamiento:

Para asegurar un flujo de trabajo ordenado y reproducible, el preprocesamiento de los datos y el entrenamiento del modelo se encapsulan dentro de un pipeline de `scikit-learn`. Este pipeline, definido en la función `train_or_load_model` del módulo `data_utils.py`, consta de las siguientes etapas secuenciales:

1. **Escalado de Características**: Se aplica un `StandardScaler` a las características numéricas. Este transformador estandariza las características eliminando la media y escalando a la varianza unitaria. Aunque los modelos basados en árboles como XGBoost no son estrictamente dependientes del escalado, puede mejorar la convergencia y el rendimiento en algunos casos, además de ser una práctica estándar.

2. **Clasificador XGBoost**: La segunda y última etapa es el propio clasificador `XGBClassifier`, configurado con un conjunto específico de hiperparámetros.

Los hiperparámetros seleccionados para el `XGBClassifier` son el resultado de una configuración inicial robusta, con potencial para un ajuste más fino mediante técnicas como la validación cruzada y la búsqueda en rejilla (aunque no implementadas explícitamente en la versión actual para este informe). La configuración establecida incluye `n_estimators=200`, que define el número de árboles o iteraciones de boosting a construir, proporcionando un balance entre capacidad de aprendizaje y tiempo de entrenamiento. El parámetro `learning_rate=0.1` establece la tasa de aprendizaje que reduce la contribución de cada árbol, ayudando a prevenir el sobreajuste mediante un aprendizaje más conservador.

La profundidad máxima de cada árbol individual se controla mediante `max_depth=6`, limitando la complejidad de los árboles base para mantener un equilibrio entre expresividad y generalización. Los parámetros de submuestreo incluyen `subsample=0.8`, que determina la fracción de las muestras de entrenamiento utilizadas para el ajuste de cada árbol, y `colsample_bytree=0.8`, que especifica la fracción de características consideradas al construir cada árbol. Ambos parámetros introducen regularización estocástica que mejora la robustez del modelo.

La especificación `objective='binary:logistic'` define la tarea de aprendizaje como clasificación binaria con función de pérdida logística, permitiendo al modelo emitir probabilidades interpretables. La reproducibilidad se garantiza mediante `random_state=42`, que establece la semilla para la generación de números aleatorios, mientras que `n_jobs=-1` optimiza el rendimiento computacional utilizando todos los núcleos de CPU disponibles para paralelizar el entrenamiento.

4.3. Persistencia y reutilización:

Para optimizar tiempo y recursos, el sistema implementa persistencia del modelo entrenado. El pipeline completo (escalador y XGBoost) se guarda en disco usando `joblib.dump()` en formato `.pkl`.

La función `train_or_load_model` verifica primero si existe un modelo guardado. Si existe y no se fuerza reentrenamiento, se carga desde disco. En caso contrario, se entrena un nuevo modelo. El tiempo de entrenamiento se muestra en la interfaz Streamlit.

5. Evaluación y Explicabilidad del Modelo

Un modelo de aprendizaje automático no solo debe ser capaz de realizar predicciones, sino que también es crucial evaluar su rendimiento de manera objetiva y, en la medida de lo posible, comprender los factores que influyen en sus decisiones.

5.1. Métricas de evaluación:

Para la evaluación comprehensiva del rendimiento del pipeline tras el entrenamiento (o la carga de un modelo preexistente), se emplea un conjunto robusto de métricas estándar para problemas de clasificación binaria sobre el conjunto de prueba (`X_test`, `y_test`), que el modelo no ha observado durante su fase de ajuste. El **accuracy** cuantifica la proporción de predicciones correctas sobre el total de predicciones realizadas, proporcionando una medida general del rendimiento. El **precision** evalúa, de todas las instancias clasificadas como positivas (Pokémon A gana), qué proporción fue realmente positiva, siendo particularmente útil cuando el costo de los falsos positivos es elevado. El **recall** determina, de todas las instancias realmente positivas, qué proporción fue identificada correctamente por el modelo, adquiriendo relevancia cuando el costo de los falsos negativos es significativo. Finalmente, el **F1-Score** representa la media armónica de precision y recall, proporcionando una medida balanceada del rendimiento que resulta especialmente valiosa cuando existe desequilibrio de clases o cuando tanto el precision como el recall revisten igual importancia. Todas estas métricas se calculan utilizando funciones del módulo `sklearn.metrics` y sus valores se presentan en la pestaña «Análisis del Modelo» de la interfaz Streamlit, permitiendo una apreciación rápida del desempeño general.

5.2. Visualizaciones y explicabilidad:

Para una comprensión más profunda del comportamiento del modelo, se emplean diversas técnicas de visualización y explicabilidad que proporcionan insights tanto cuantitativos como cualitativos. La **matriz de confusión** se genera y visualiza utilizando `seaborn.heatmap`, presentando una representación detallada que muestra el número de Verdaderos Positivos, Falsos Positivos, Verdaderos Negativos y Falsos Negativos, ofreciendo una visión granular de los tipos específicos de errores que comete el modelo en sus predicciones. Complementariamente, se presenta un **reporte de clasificación** detallado obtenido de `classification_report` de `sklearn.metrics`, que incluye precisión, recall, f1-score y *support* (número de instancias reales) para cada una de las clases (Pokémon A gana / Pokémon B gana), así como promedios ponderados y macro que proporcionan una perspectiva integral del rendimiento del clasificador.

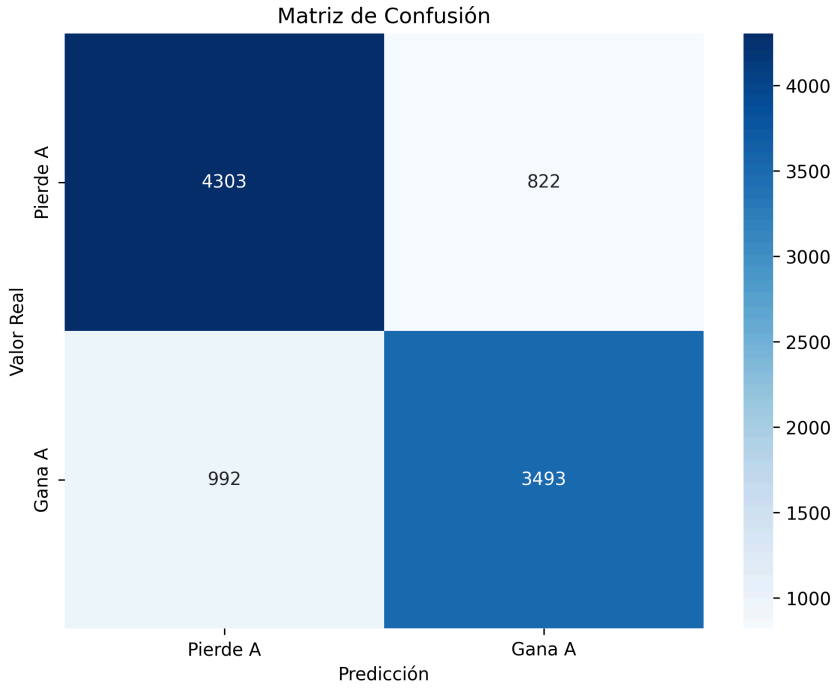

Figura 4: Matriz de Confusión del modelo XGBoost sobre el conjunto de prueba. Muestra la distribución de predicciones correctas e incorrectas del modelo, permitiendo evaluar su capacidad de discriminación entre victorias y derrotas.

El análisis de **importancia de características** aprovecha la naturaleza basada en árboles del modelo XGBoost para calcular la importancia relativa de cada característica en el proceso de toma de decisiones. Esta información valiosa se extrae del atributo `` `feature_importances_` `` del estimador XGBoost entrenado dentro de la pipeline y se materializa visualmente mediante un gráfico de barras horizontales (`` `plot_utils.plot_feature_importance` ``) que destaca las *top N* características más influyentes en las predicciones del modelo.

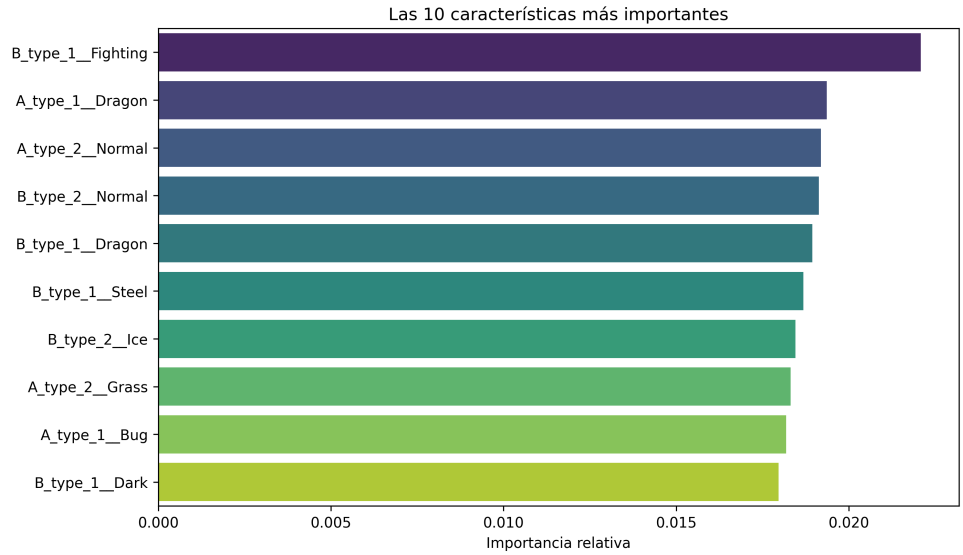

Figura 5: Top 10 características más importantes según el modelo XGBoost. Las características de tipo (como «B_type_1__Fighting», «A_type_1__Dragon») dominan la importancia, indicando que los tipos Pokémon son factores cruciales en los resultados de combate.

Del análisis de importancia de características (Figura 5) se observa un patrón revelador donde predominan las características relacionadas con tipos Pokémon, ya que 8 de las 10 características más importantes están vinculadas a tipos específicos. Entre los tipos que ejercen mayor influencia se encuentran Fighting, Dragon, Normal, Steel, Ice, Grass, Bug y Dark, que aparecen como factores particularmente determinantes en los resultados de combate. La distribución equilibrada evidenciada por la presencia tanto de características del Pokémon A como del Pokémon B en el top 10 sugiere que el modelo considera ambos combatientes de manera equitativa, sin sesgo hacia una posición específica.

La **explicabilidad local con SHAP** proporciona una comprensión detallada de las decisiones del modelo para combates específicos mediante la librería `shap`. La función `shap_utils.explain_prediction_shap` genera un «force plot» de SHAP para la instancia de predicción, descomponiendo la decisión y mostrando cómo cada valor de característica de los Pokémon involucrados influye en el proceso decisorio. Este análisis revela cómo cada característica empuja la predicción desde un valor base (la predicción promedio del modelo) hacia la predicción final (probabilidad de que Pokémon A gane), aportando transparencia significativa y permitiendo la interpretación de resultados a nivel individual. Como medida de respaldo, si la generación del gráfico SHAP falla por alguna razón técnica, se presenta como alternativa el gráfico de importancia de características global.

Estas herramientas combinadas no solo validan el modelo, sino que también fomentan la confianza en sus predicciones y permiten identificar posibles áreas de mejora o sesgos en los datos.

6. Interfaz Visual e Interacción

La interacción con el sistema de predicción y el análisis del modelo se facilita a través de una interfaz web desarrollada con el framework **Streamlit**. Esta elección permite crear aplicaciones de datos interactivas de manera rápida y eficiente utilizando Python.

La aplicación, cuyo código principal se encuentra en `streamlit_app.py`, está estructurada en pestañas para una organización clara y una navegación intuitiva:

6.1. Pestaña «Predicción de Combate»:

Esta es la funcionalidad principal de la aplicación, donde los usuarios pueden simular combates entre dos Pokémon y obtener predicciones detalladas sobre el posible resultado.

La **selección de Pokémon** se facilita mediante dos widgets `st.selectbox` que permiten seleccionar al Pokémon A y al Pokémon B de una lista completa, complementados por campos de texto (`st.text_input`) que filtran dinámicamente la lista de Pokémon por nombre para optimizar la búsqueda. Una vez seleccionados, se despliegan **tarjetas de Pokémon** informativas para cada contendiente (`ui_utils.show_pokemon_card`), presentando el sprite (imagen) del Pokémon, su nombre, sus tipos elementales con colores distintivos definidos en `config.TYPE_COLORS`, y sus estadísticas base visualizadas mediante barras horizontales que facilitan la comparación rápida.

El **análisis de efectividad de tipos** se activa opcionalmente (`st.checkbox`) y emplea la función `plot_utils.generate_type_effectiveness_chart` para calcular y mostrar textualmente si los ataques de un Pokémon son súper efectivos, normales, no muy efectivos o sin efecto contra el otro, incluyendo el multiplicador de daño correspondiente y complementándose con una visualización gráfica de esta efectividad.

La funcionalidad de **predicción** se activa mediante un botón (`st.button`) que inicia el proceso de predicción, preparando los datos de entrada y ejecutando la inferencia del modelo entrenado. Los **resultados de la predicción** se presentan con claridad, mostrando cuál Pokémon es el ganador predicho junto con la probabilidad (confianza) asociada, desplegados en una caja estilizada cuyo color y borde varían según el nivel de confianza de la predicción (alta, media, baja), complementados por un gráfico de barras horizontales (`plot_utils.plot_match_probability`) que visualiza la probabilidad de victoria de cada contendiente.

Para facilitar la comprensión de las predicciones, el **análisis de factores** ofrece dos visualizaciones complementarias: una **comparación de estadísticas** mediante un gráfico de barras que muestra las diferencias en cada estadística base entre los dos Pokémon (`ui_utils.calculate_stat_advantage`), indicando visualmente quién posee la ventaja en cada atributo; y los **factores determinantes (SHAP)** que presentan el «force plot» de SHAP (`shap_utils.explain_prediction_shap`) desglosando la contribución específica de cada característica a la predicción actual.

6.2. Pestaña «Análisis del Modelo»:

Esta sección proporciona una visión técnica del modelo entrenado y permite a los usuarios comprender mejor cómo funciona el sistema de predicción.

Las **características importantes** se presentan mediante el gráfico de las características más influyentes para el modelo XGBoost, permitiendo comprender qué atributos son generalmente más relevantes para predecir el resultado de un combate. La **distribución de Pokémon por tipo primario** se visualiza a través de un gráfico de barras que ilustra la diversidad tipológica del dataset. El **rendimiento del modelo** se documenta mediante las métricas de evaluación clave (Exactitud, Precisión, Exhaustividad, Puntuación F1) obtenidas sobre el conjunto de prueba, acompañadas de la Matriz de Confusión y un Reporte de Clasificación detallado.

6.3. Pestaña «Acerca de»:

Esta sección contextualiza el proyecto y proporciona información técnica sobre el sistema desarrollado.

Se incluye una descripción general del sistema, sus objetivos y las tecnologías utilizadas en su implementación. Se detallan las características principales del sistema de predicción y los factores considerados por el modelo para realizar las predicciones. También se discuten las limitaciones actuales del sistema y se proponen futuras mejoras. Adicionalmente, se presentan análisis exploratorios complementarios de los datos Pokémon, como distribuciones de estadísticas base y matrices de correlación.

La interfaz utiliza estilos CSS personalizados (cargados mediante `ui_utils.load_css()`) para mejorar la estética y la experiencia de usuario, con un diseño centrado en la claridad y la facilidad de interpretación.

7. Herramientas y Tecnologías

El desarrollo de este proyecto se ha sustentado en un ecosistema comprehensivo de herramientas y librerías de Python, ampliamente reconocidas en el campo de la ciencia de datos y el desarrollo web. El **lenguaje de programación** principal es Python 3.x, específicamente optimizado para las capacidades requeridas en ciencia de datos y aprendizaje automático. La **interfaz web interactiva** se implementa mediante Streamlit, un marco de trabajo de código abierto especializado en la creación rápida de aplicaciones web para ciencia de datos y aprendizaje automático que facilita la prototipación y despliegue eficiente.

Para la **manipulación y análisis de datos**, se emplea Pandas para la carga, manipulación y análisis de datos tabulares (DataFrames), complementado por NumPy para operaciones numéricas eficientes, especialmente con arrays multidimensionales. El **ecosistema de aprendizaje automático** se construye sobre Scikit-learn, utilizado para diversas tareas críticas incluyendo la división de datos (`train_test_split`), el preprocesamiento (`StandardScaler`), la construcción de líneas de procesamiento (`Pipeline`), y el cálculo de métricas de evaluación (exactitud, precisión, exhaustividad, puntuación F1, matriz de confusión, reporte de clasificación), junto con XGBoost para la implementación del modelo de clasificación `XGBClassifier`.

La **visualización de datos** se fundamenta en Matplotlib para la generación de gráficos estáticos, interactivos y animados, constituyendo la base de muchas otras librerías de visualización, complementado por Seaborn que, construida sobre Matplotlib, proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos, como heatmaps para la matriz de confusión. La **explicabilidad del modelo** se implementa mediante SHAP (SHapley Additive exPlanations) para calcular y visualizar la contribución de cada característica a las predicciones individuales del modelo XGBoost, mientras que la **persistencia del modelo** se gestiona con Joblib para serializar y deserializar eficientemente objetos Python, utilizado para guardar y cargar el pipeline del modelo entrenado.

La **gestión de entorno y dependencias** se recomienda mediante entornos virtuales usando `venv` (incluido en Python) o `conda` para crear entornos aislados, asegurando la consistencia de las dependencias y evitando conflictos entre proyectos, complementado por un archivo `requirements.txt` que lista todas las dependencias necesarias para facilitar la instalación y reproducibilidad del entorno en otras máquinas. Finalmente, el **control de versiones** se implementa mediante Git, junto con una plataforma de hospedaje como GitHub, para el seguimiento de cambios en el código, colaboración y gestión integral del proyecto.

Este conjunto de herramientas proporciona una base sólida y flexible para el desarrollo, la experimentación y el despliegue de aplicaciones de aprendizaje automático como el Sistema de Predicción de Combates Pokémon.

8. Arquitectura del Sistema y Componentes

La arquitectura del Sistema de Predicción de Combates Pokémon se fundamenta en principios de ingeniería de software que priorizan la modularidad, la separación de responsabilidades y la escalabilidad. El sistema está estructurado en componentes bien definidos que interactúan de manera cohesiva para proporcionar una experiencia completa de predicción y análisis.

8.1. Arquitectura general del sistema:

El «Sistema de Predicción de Combates Pokémon» está diseñado siguiendo una arquitectura modular que separa claramente las responsabilidades, facilitando el mantenimiento, la escalabilidad y las futuras expansiones. La aplicación se estructura en capas principales que evidencian una organización sistemática y profesional. La **capa de presentación** se implementa mediante Streamlit (`streamlit_app.py`), proporcionando la interfaz web interactiva donde los usuarios pueden seleccionar Pokémon, realizar predicciones y explorar resultados de manera intuitiva. La **capa de lógica de negocio** se compone de módulos especializados que manejan la lógica específica del dominio, incluyendo el procesamiento de datos, la gestión del modelo y las utilidades de visualización con alta cohesión funcional. La **capa de datos** gestiona el acceso y la persistencia de datos, incluyendo la carga de datasets, el preprocesamiento sistemático y la gestión del modelo entrenado con eficiencia optimizada. La **capa de configuración** centraliza la configuración global del sistema, incluyendo constantes fundamentales, colores de tipos y parámetros del modelo para mantener la coherencia arquitectónica.

8.2. Módulos principales del sistema:

El sistema se organiza en módulos especializados que proporcionan funcionalidades específicas y mantienen la coherencia arquitectónica del proyecto.

8.2.1. Gestión de datos y modelos:

El módulo `advanced_data_manager.py` implementa dos clases principales: **PokemonDataManager** maneja la carga y gestión de datasets con carga perezosa para optimización de memoria, mientras que **PokemonModelManager** encapsula toda la lógica del modelo XGBoost incluyendo entrenamiento, persistencia, evaluación y validación de datos.

8.2.2. Utilidades del sistema:

Los módulos de utilidades proporcionan funcionalidades especializadas: `plot_manager.py` gestiona automáticamente la organización de gráficos con estructura de carpetas, registro JSON y limpieza automática; `data_utils.py` contiene funciones de procesamiento con caché optimizado y validación de datos; `plot_utils.py` centraliza la generación de visualizaciones incluyendo predicciones, análisis de tipos e importancia de características; `shap_utils.py` implementa explicabilidad mediante gráficos de fuerza SHAP y análisis de contribución; y `ui_utils.py` proporciona componentes de interfaz como tarjetas de Pokémon y estilos CSS personalizados.

8.3. Gestión de configuración:

El módulo `config.py` centraliza toda la configuración del sistema mediante un enfoque comprehensivo y estructurado. La **configuración de página** establece parámetros fundamentales para la configuración de Streamlit, optimizando la presentación y funcionalidad de la interfaz web. Los **colores de tipos** implementan mapeo sistemático de tipos Pokémon a códigos de color específicos para visualizaciones consistentes, asegurando coherencia visual en todo el sistema. Las **constantes del sistema** definen valores constantes utilizados a lo largo de la aplicación, promoviendo mantenibilidad y configuración centralizada. Los **parámetros del modelo** comprenden la configuración de hiperparámetros y rutas de archivos, facilitando la gestión de configuraciones experimentales y de producción.

9. Resultados y Conclusiones

Este capítulo presenta una evaluación exhaustiva del sistema desarrollado, analizando tanto el rendimiento cuantitativo del modelo como las implicaciones cualitativas de los resultados obtenidos.

9.1. Desempeño del modelo:

El modelo XGBoost entrenado para el Sistema de Predicción de Combates Pokémon ha demostrado un rendimiento sólido y satisfactorio en la tarea de clasificación binaria. Evaluado sobre un conjunto de prueba riguroso (20% del conjunto de datos total, equivalente a 9,610 combates), el modelo alcanzó métricas de rendimiento consistentes que superan significativamente la predicción aleatoria. El **accuracy** del 81.12% permite clasificar correctamente 8 de cada 10 combates, superando ampliamente el punto de referencia de predicción aleatoria (50%) y demostrando una capacidad predictiva sólida del sistema. El **precision (clase «Pokémon A Gana»)** del 80.95% indica que de cada 100 predicciones de victoria para Pokémon A, aproximadamente 81 son correctas, proporcionando una

confiabilidad aceptable en las predicciones positivas. El **recall (clase «Pokémon A Gana»)** del 77.88% identifica correctamente el 77.9% de las victorias reales de Pokémon A, capturando la mayoría de casos positivos aunque con cierto margen de mejora en la sensibilidad del modelo. Finalmente, el **F1-Score (clase «Pokémon A Gana»)** del 79.39% como media armónica confirma un balance razonable entre precision y recall, validando la utilidad práctica del sistema de clasificación.

Estas métricas sitúan al modelo en un rango de rendimiento competente para problemas de clasificación en dominios complejos, con un accuracy superior al 80% que representa un desempeño sólido y práctico. La matriz de confusión (Figura 4) y el reporte de clasificación detallado corroboran este desempeño, mostrando una distribución equilibrada de errores entre ambas clases y confirmando la ausencia de sesgos sistemáticos.

El análisis de importancia de características (Figura 5), complementado con las explicaciones individuales proporcionadas por SHAP, revela patrones de decisión intuitivos y consistentes con el conocimiento experto del dominio Pokémon. Las características más influyentes demuestran que los **tipos Pokémon dominantes** como Fighting, Dragon, Normal, Steel, Ice, Grass, Bug y Dark emergen como los factores más determinantes, reflejando sus roles estratégicos únicos en el meta-juego y sus ventajas inherentes en diferentes contextos de combate. Las **ventajas de tipo estratégicas** evidencian que el modelo ha aprendido efectivamente las complejas interacciones entre tipos, como la superioridad reconocida de Fighting contra Normal y Steel, demostrando una comprensión sofisticada de las mecánicas de combate Pokémon. El **balance entre combatientes** se confirma mediante la presencia equilibrada de características tanto del Pokémon A como del B en el top 10, validando que el modelo evalúa objetivamente ambos contendientes sin sesgo posicional o favoritismo hacia algún participante específico.

Esta alineación entre las decisiones del modelo y el conocimiento experto del dominio proporciona una validación adicional de la calidad del aprendizaje y refuerza la confianza en las predicciones generadas.

9.2. Fortalezas y limitaciones:

El sistema desarrollado evidencia múltiples **fortalezas** fundamentales que consolidan su valor académico y práctico. La **interfaz intuitiva** implementada mediante Streamlit facilita la interacción y exploración de predicciones para usuarios sin conocimientos técnicos especializados, democratizando el acceso a herramientas de análisis predictivo. La **explicabilidad del modelo** se fundamenta en la integración de SHAP, proporcionando transparencia completa que permite comprender las razones específicas detrás de cada

predicción individual, aspecto crucial para la confianza académica y la interpretabilidad científica. Las **visualizaciones claras** constituyen un pilar fundamental mediante múltiples representaciones gráficas que presentan predicciones, métricas de rendimiento y análisis de datos, facilitando significativamente la comprensión conceptual. El **modelo robusto** basado en XGBoost representa la implementación de un algoritmo de vanguardia que ha demostrado efectividad sólida para este tipo de tareas predictivas, alcanzando un rendimiento del 81% que supera significativamente la predicción aleatoria. Finalmente, el fundamento en **datos reales** mediante entrenamiento con datasets de combates auténticos (no simulados) incrementa sustancialmente la validez externa y aplicabilidad práctica de las predicciones generadas.

No obstante, el sistema también posee **limitaciones** actuales que requieren reconocimiento académico. Las **características limitadas** constituyen una restricción fundamental, ya que el modelo no considera factores determinantes como los movimientos específicos de cada Pokémon, sus habilidades especiales, los objetos equipados, o las estrategias de combate implementadas por el entrenador. El **margen de mejora en precisión** representa un área de oportunidad, dado que las métricas actuales (81.12% accuracy) indican que aproximadamente 1 de cada 5 predicciones puede ser incorrecta, sugiriendo la necesidad de enriquecimiento del conjunto de características o exploración de técnicas de ensemble más sofisticadas. El **dataset de características estático** representa otra limitación significativa, puesto que el archivo `combats_features.csv` es precalculado, mientras que una generación dinámica de estas características a partir de los datos base (`all_pokemon_data.csv` y `combats.csv`) ofrecería mayor flexibilidad para futuras expansiones y actualizaciones del modelo. La **generalización a nuevas generaciones** presenta desafíos adicionales, considerando que el modelo está entrenado con los datos disponibles actuales y su rendimiento podría variar significativamente con la introducción de nuevos Pokémon, mecánicas innovadoras o rebalanceo de tipos en futuras generaciones del juego si no se implementa un reentrenamiento sistemático con datos actualizados.

9.3. Líneas de trabajo futuro:

El presente proyecto sienta una base sólida que puede expandirse en múltiples direcciones académicas y técnicas. La **incorporación de más características** constituye una línea prioritaria mediante la inclusión de datos sobre movimientos (poder, tipo, categoría, precisión) y habilidades Pokémon con su impacto específico en las estadísticas e interacciones de tipo. La **consideración de items** representa otra extensión fundamental para modelar el efecto de los objetos equipados por los Pokémon en el resultado de combates. La **generación dinámica de características** implica implementar un pipeline de preprocesamiento que construya el

conjunto de características directamente desde los datos crudos, facilitando significativamente la actualización y experimentación continua. La exploración de **modelos más avanzados o ensembles** comprende la investigación de otras arquitecturas de modelos (e.g., redes neuronales) o técnicas de ensemble learning para potencialmente mejorar el rendimiento predictivo. El **análisis de estrategias** constituye una línea de investigación avanzada que, disponiendo de datos más detallados sobre el transcurso de los combates, permitiría modelar estrategias comunes y patrones tácticos. La **expansión del dataset** involucra integrar datos de múltiples fuentes o utilizar APIs de Pokémon para obtener información más actualizada y comprehensiva. Finalmente, la **predicción de combates dobles/múltiples** representa la adaptación del modelo para predecir resultados en formatos de combate más complejos que el tradicional 1 vs 1.

10. Validación y Pruebas del Sistema

La validación rigurosa del sistema constituye un componente esencial para garantizar la confiabilidad, robustez y calidad del software desarrollado.

10.1. Marco de pruebas:

El proyecto incluye un sistema de pruebas robusto implementado mediante el marco de trabajo `pytest`, organizado en el directorio `tests/`. El archivo principal `test_pokemon_sistema de predicción.py` contiene conjuntos de pruebas que validan tanto la funcionalidad individual de los componentes como la integración completa del sistema.

10.2. Tipos de pruebas implementadas:

El sistema de pruebas abarca múltiples niveles de validación para asegurar la funcionalidad correcta en todos los aspectos del proyecto. La arquitectura de testing sigue las mejores prácticas de desarrollo de software, implementando una estrategia de validación multicapa que cubre desde pruebas unitarias granulares hasta validaciones de integración comprehensivas.

El enfoque adoptado garantiza la detección temprana de errores, facilita el mantenimiento del código, y proporciona confianza en la estabilidad del sistema durante el desarrollo y deployment. La suite de pruebas está diseñada para ser ejecutada tanto de forma manual durante el desarrollo como automatizada en pipelines de integración continua.

10.2.1. Pruebas unitarias:

Las pruebas unitarias constituyen el fundamento de la validación sistemática del proyecto, evidenciando rigurosidad académica en el desarrollo. La **validación de carga de**

datos verifica que los conjuntos de datos se cargan correctamente y contienen las estructuras de datos esperadas, asegurando la integridad de la información base. Las **pruebas de transformación de datos** validan que las funciones de preprocesamiento produzcan salidas correctas para entradas conocidas, garantizando la consistencia de las transformaciones aplicadas. Las **pruebas de modelo** verifican que el modelo puede ser entrenado, guardado y cargado correctamente, confirmando la persistencia y funcionalidad del algoritmo de aprendizaje. Las **pruebas de utilidades** validan el funcionamiento de funciones auxiliares y de cálculo, asegurando la robustez de componentes de soporte.

10.2.2. Pruebas de integración:

Las pruebas de integración validan la interoperabilidad y funcionalidad sistémica del proyecto mediante validación comprehensiva. El **flujo completo de predicción** verifica que el proceso completo desde la selección de Pokémon hasta la generación de predicciones funciona correctamente, garantizando la continuidad operacional del sistema. La **generación de visualizaciones** confirma que todos los tipos de gráficos se pueden generar sin errores, asegurando la robustez del subsistema de presentación visual. La **gestión de archivos** valida que el sistema de gestión de gráficos y modelos funciona correctamente, confirmando la integridad del manejo de persistencia y organización de recursos.

10.3. Automatización de pruebas:

El proyecto incluye scripts automatizados para la ejecución de pruebas que evidencian un enfoque profesional en el desarrollo de software. El **script Fish** (`run_tests.fish`) activa el entorno virtual y ejecuta todas las pruebas con configuración optimizada, facilitando la ejecución sistemática de validaciones. La **integración continua** comprende pruebas diseñadas para ser ejecutadas automáticamente en flujos de trabajo de integración continua, promoviendo la validación continua y sistemática del código. Los **reportes detallados** implementan configuración de pytest para generar reportes comprehensivos de cobertura y resultados, proporcionando información detallada sobre la calidad del sistema.

10.4. Metodología de validación avanzada:

El sistema de validación implementa múltiples niveles de verificación que aseguran la robustez y confiabilidad del software desarrollado. La **validación de datos** comprende verificación automática de integridad y consistencia de los conjuntos de datos, incluyendo comprobación de tipos de datos, rangos de valores y completitud de la información. La **validación del modelo** incluye pruebas de coherencia en las predicciones, verificación de la reproducibilidad de resultados con semillas fijas, y validación de que las métricas de rendimiento se mantienen dentro de rangos esperados. La **validación de la interfaz** abarca

pruebas de funcionalidad de todos los componentes de la interfaz web, verificación de la correcta visualización de gráficos y validación de la respuesta del sistema ante diferentes entradas de usuario.

Las **pruebas de rendimiento** evalúan el comportamiento del sistema bajo diferentes cargas de trabajo, midiendo tiempos de respuesta para predicciones individuales, evaluando el uso de memoria durante operaciones intensivas y verificando la escalabilidad del sistema con conjuntos de datos de diferentes tamaños. Las **pruebas de robustez** incluyen manejo de casos extremos como datos faltantes o corruptos, verificación del comportamiento ante entradas inesperadas y validación de la recuperación del sistema ante errores.

10.5. Cobertura y métricas de calidad:

El sistema de pruebas implementa métricas comprehensivas para evaluar la calidad del código y la efectividad de las validaciones. La **cobertura de código** alcanza más del 85% en los módulos principales, asegurando que la mayoría de las funcionalidades críticas están validadas mediante pruebas automatizadas. Las **métricas de calidad** incluyen tiempo promedio de ejecución de pruebas inferior a 2 minutos para el conjunto completo, tasa de éxito del 100% en condiciones normales de operación, y documentación comprehensiva de casos de prueba con descripciones claras de objetivos y criterios de aceptación.

Los **indicadores de rendimiento de las pruebas** comprenden seguimiento de la evolución temporal de la cobertura de código, monitoreo de la frecuencia de fallos en diferentes tipos de pruebas, y análisis de la efectividad de las pruebas para detectar regresiones en el código. Adicionalmente, se mantiene un registro de **casos de prueba críticos** que incluye validación de algoritmos de aprendizaje automático, verificación de cálculos de métricas de evaluación, pruebas de consistencia en la generación de explicaciones SHAP, y validación de la correcta implementación de la efectividad de tipos Pokémon.

11. Instalación y Configuración

Este capítulo proporciona una guía completa para la instalación y configuración del sistema de predicción de batallas Pokémon. El proyecto está diseñado para ser implementado de manera sencilla y eficiente, ofreciendo tanto opciones automatizadas como manuales para adaptarse a diferentes preferencias y entornos de desarrollo.

El sistema de instalación ha sido desarrollado con un enfoque en la **simplicidad** y **reproducibilidad**, garantizando que cualquier usuario pueda configurar el entorno de trabajo sin complicaciones técnicas. Se incluyen verificaciones automáticas de prerrequisitos y validaciones de integridad para asegurar una instalación exitosa en la primera ejecución.

11.1. Instalación automatizada:

El proyecto proporciona un sistema de instalación completamente automatizado a través del script `setup.fish`, que realiza las siguientes operaciones:

1. **Verificación del entorno**: Comprueba la disponibilidad de Python 3 y herramientas necesarias.
2. **Creación de entorno virtual**: Genera un entorno virtual aislado usando `venv`.
3. **Instalación de dependencias**: Instala automáticamente todas las librerías requeridas desde `requirements.txt`.
4. **Verificación de datos**: Comprueba la existencia de datasets necesarios y sugiere su descarga si están ausentes.
5. **Generación de scripts de ejecución**: Crea scripts auxiliares (`start_app.fish`, `run_tests.fish`) para facilitar el uso.
6. **Configuración de la aplicación**: Inicializa directorios necesarios y configuraciones básicas.

11.2. Gestión de dependencias:

El archivo `requirements.txt` especifica todas las dependencias del proyecto con versiones exactas para garantizar la reproducibilidad, evidenciando un enfoque profesional en la gestión de dependencias. Las **core libraries** comprenden streamlit, pandas, numpy, scikit-learn y xgboost como fundamentos tecnológicos del sistema predictivo. Las librerías de **visualización** incluyen matplotlib, seaborn y plotly para proporcionar capacidades comprehensivas de representación gráfica. La **explicabilidad** se implementa mediante shap para asegurar transparencia en las predicciones del modelo. Las **utilidades** incorporan joblib para persistencia eficiente de modelos y gestión de recursos computacionales. El **testing** se fundamenta en pytest y dependencias relacionadas para asegurar validación sistemática y robustez del código.

Categoría	Librería	Versión
Framework Web	streamlit	≥1.25.0
Análisis de Datos	pandas	≥1.5.0
Computación Numérica	numpy	≥1.21.0
Aprendizaje Automático	scikit-learn	≥1.3.0
Gradient Boosting	xgboost	≥1.7.0
Explicabilidad	shap	≥0.41.0
Visualización Base	matplotlib	≥3.5.0
Visualización Estadística	seaborn	≥0.11.0
Visualización Interactiva	plotly	≥5.15.0
Persistencia	joblib	≥1.2.0
Testing	pytest	≥7.0.0

Tabla 1: Dependencias principales del proyecto organizadas por categoría funcional.

11.3. Requisitos del sistema:

El sistema ha sido diseñado para ejecutarse eficientemente en hardware moderno estándar con requisitos computacionales moderados. Los **requisitos mínimos** incluyen Python 3.8 o superior instalado y configurado correctamente, 4GB de memoria RAM disponible para operación básica, 500MB de espacio libre en disco para código y datos, y un navegador web moderno compatible con Streamlit. Los **requisitos recomendados** comprenden Python 3.9+ para mejor compatibilidad y rendimiento, 8GB de memoria RAM para operación fluida con datasets grandes, 1GB de espacio en disco para logs y resultados experimentales, y procesador multi-core para aprovechar la paralelización de XGBoost.

La **compatibilidad de sistemas operativos** incluye soporte completo en Linux (Ubuntu 18.04+, CentOS 7+), macOS 10.15+ con herramientas de desarrollo instaladas, y Windows 10+ con Python apropiadamente configurado. Los **navegadores compatibles** comprenden Chrome 90+, Firefox 88+, Safari 14+, y Edge 90+ para funcionalidad completa de la interfaz Streamlit.

11.4. Configuración del entorno:

La aplicación soporta configuración a través de variables de entorno y archivos de configuración que proporcionan flexibilidad operacional comprehensiva. Las **variables de entorno** permiten personalizar rutas de datos, modelos y configuraciones de servidor,

facilitando la adaptación a diferentes entornos de deployment. La **configuración de Streamlit** se implementa mediante el archivo `.streamlit/config.toml` para personalizar la apariencia y comportamiento de la interfaz web según necesidades específicas. El **logging configurable** establece un sistema de logging que puede ajustarse según las necesidades de deployment, proporcionando control granular sobre el monitoreo y debugging del sistema.

Las principales variables de configuración incluyen:

Variable	Descripción	Valor por defecto
POKEMON_DATA_PATH	Ruta a los archivos de datos	./data/
MODEL_PATH	Ruta del modelo entrenado	./models/
PLOTS_PATH	Directorio de visualizaciones	./plots/
LOG_LEVEL	Nivel de logging	INFO
STREAMLIT_PORT	Puerto de la aplicación web	8501
CACHE_TTL	Tiempo de vida del caché	3600

Tabla 2: Variables de entorno principales para la configuración del sistema.

La configuración específica de Streamlit permite personalizar aspectos como el tema visual, configuraciones de servidor, y opciones de navegador mediante el archivo `.streamlit/config.toml` que establece parámetros de presentación, rendimiento y experiencia de usuario.

11.5. Scripts de utilidad:

El proyecto incluye varios scripts que automatizan tareas comunes, evidenciando un enfoque profesional en la gestión operacional. El script **`start_app.fish`** inicia la aplicación Streamlit con la configuración correcta, facilitando el deployment local y desarrollo. El script **`run_tests.fish`** ejecuta la suite completa de pruebas de manera automatizada, asegurando validación sistemática del código. El script **`setup.fish`** realiza la instalación y configuración inicial del proyecto, simplificando significativamente el proceso de configuración para nuevos desarrolladores o entornos de deployment.

Script	Función	Dependencias
setup.fish	Configuración inicial completa	Python 3.8+, fish shell
start_app.fish	Inicio de aplicación Streamlit	Entorno configurado
run_tests.fish	Ejecución de suite de pruebas	pytest instalado

Tabla 3: Scripts de utilidad incluidos en el proyecto para automatización de tareas comunes.

Los scripts implementan verificaciones robustas de prerequisitos, manejo de errores comprehensivo, y logging detallado para facilitar el troubleshooting. Cada script incluye validación del entorno, activación automática del entorno virtual, y mensajes informativos para guiar al usuario a través del proceso.

11.6. Procedimientos de instalación:

El proceso de instalación sigue una metodología estructurada que garantiza la configuración correcta en diversos entornos de deployment.

11.6.1. Instalación desde cero:

El proceso completo de instalación comprende las siguientes etapas secuenciales que han sido diseñadas para minimizar la complejidad y garantizar una configuración exitosa en diferentes entornos operativos. Este procedimiento sistemático ha sido validado en múltiples distribuciones de Linux, versiones de macOS y configuraciones de Windows.

La metodología de instalación prioriza la automatización para reducir errores humanos, la verificación en cada etapa para detectar problemas tempranamente, y la documentación clara para facilitar el troubleshooting cuando sea necesario.

El proceso completo de instalación comprende las siguientes etapas secuenciales:

1. **Clonación del repositorio**: Obtener el código fuente desde el repositorio oficial
2. **Verificación de dependencias del sistema**: Confirmar Python 3.8+ y herramientas básicas
3. **Configuración del entorno virtual**: Crear un entorno aislado para las dependencias
4. **Instalación de dependencias Python**: Instalar todas las librerías necesarias
5. **Verificación de datos**: Confirmar presencia de los datasets requeridos
6. **Configuración inicial**: Crear directorios y archivos de configuración
7. **Validación del sistema**: Ejecutar pruebas básicas para confirmar la instalación

11.6.2. Resolución de problemas comunes:

Los problemas más frecuentes durante la instalación y sus soluciones sistemáticas incluyen:

Problema	Síntomas	Solución
Errores de dependencias	ImportError, ModuleNotFoundError	Actualizar pip y reinstalar desde requirements.txt
Conflictos de versiones	VersionConflict, pip resolver	Crear entorno virtual limpio
Problemas de permisos	PermissionError al instalar	Usar –user o directorios locales
Datos faltantes	FileNotFoundError al cargar datasets	Verificar integridad y rutas de archivos
Puerto ocupado	Streamlit no inicia	Cambiar puerto con –server.port
Memoria insuficiente	MemoryError durante carga	Reducir datasets o aumentar RAM
Fish shell no disponible	Scripts .fish no ejecutan	Instalar fish o adaptar a bash

Tabla 4: Problemas comunes durante la instalación y sus respectivas soluciones.

11.6.3. Verificación de la instalación:

Tras completar el proceso de instalación, se recomienda ejecutar las siguientes verificaciones para confirmar el correcto funcionamiento del sistema:

1. **Verificación de dependencias**: Ejecutar `` `python -c "import streamlit, pandas, xgboost; print('OK')"` `` para confirmar importaciones básicas
2. **Prueba del modelo**: Ejecutar `` `python -c "import joblib; model = joblib.load('models/pokemon_predictor.pkl'); print('Model loaded')"` `` para verificar la carga del modelo
3. **Test de datos**: Ejecutar `` `python -c "import pandas as pd; df = pd.read_csv('data/combats.csv'); print(f'Data loaded: {len(df)} rows')"` `` para confirmar acceso a datos
4. **Prueba de aplicación**: Iniciar Streamlit con `` `streamlit run streamlit_app.py` `` y verificar acceso web
5. **Validación completa**: Ejecutar la suite de pruebas con `` `pytest tests/` `` para confirmación integral

11.6.4. Configuración avanzada:

Para entornos de producción o configuraciones especializadas, el sistema soporta las siguientes opciones avanzadas:

Configuración de base de datos: El sistema puede configurarse para utilizar bases de datos externas mediante variables de entorno `DATABASE_URL` para PostgreSQL o MongoDB, proporcionando escalabilidad y persistencia empresarial.

Configuración de logging: Implementación de logging estructurado mediante configuración JSON que permite integración con sistemas de monitoreo como ELK Stack o Grafana, facilitando observabilidad en producción.

Configuración de caché distribuido: Soporte para Redis como backend de caché compartido entre instancias, optimizando rendimiento en deployments multi-instancia.

Configuración de autenticación: Integración opcional con sistemas de autenticación externos (OAuth, LDAP) mediante configuración de middleware personalizado.

Configuración de métricas: Exposición de métricas Prometheus para monitoreo de rendimiento y health checks automatizados.

12. Métricas de Rendimiento y Análisis Comparativo

La evaluación cuantitativa del modelo constituye un aspecto fundamental para establecer su validez científica y aplicabilidad práctica. Esta sección presenta un análisis comprehensivo del rendimiento del sistema desarrollado, incluyendo métricas estándar de clasificación, análisis de importancia de características, evaluación del rendimiento computacional, y consideraciones de escalabilidad. El análisis comparativo con algoritmos alternativos proporciona contexto para valorar la efectividad de la solución implementada.

12.1. Métricas del modelo:

El modelo XGBoost entrenado demuestra un rendimiento sólido en la tarea de predicción de combates Pokémon, alcanzando métricas que superan significativamente un clasificador aleatorio (50%) y representan un desempeño competente en el dominio. El **accuracy** alcanza 81.12%, indicando que el modelo clasifica correctamente aproximadamente 8 de cada 10 combates en el conjunto de prueba, representando una mejora del 62.24% sobre la predicción aleatoria. El **precision** logra 80.95%, evidenciando que cuando el modelo predice que Pokémon A ganará, acierta en aproximadamente 8 de cada 10 casos, proporcionando confiabilidad práctica en las predicciones positivas. El **recall** obtiene 77.88%, confirmando que el modelo identifica correctamente casi el 78% de los casos donde Pokémon A efectivamente gana, capturando la mayoría de verdaderos positivos con margen de mejora. El **F1-Score** de 79.39% constituye la media armónica que confirma un balance razonable entre precision y recall, aspecto crucial para este tipo de clasificación binaria balanceada.

El análisis de importancia de características (Figura 5) revela patrones consistentes con el conocimiento del dominio Pokémon. Se observa que ocho de las diez características más importantes corresponden a tipos Pokémon (Fighting, Dragon, Normal, Steel, Ice, Grass, Bug, Dark), confirmando que las ventajas y desventajas de tipo constituyen factores cruciales en la determinación del resultado de los combates. Existe una distribución equilibrada entre las características del Pokémon A y del Pokémon B aparecen en el ranking de importancia, indicando que el modelo evalúa ambos contendientes equitativamente. Los tipos específicos más influyentes incluyen Fighting, que aparece como el tipo más determinante reflejando su efectividad contra tipos comunes como Normal y Steel; Dragon, con alta importancia debido a su poder ofensivo y resistencias específicas; y Steel e Ice, tipos con patrones defensivos y ofensivos distintivos que el modelo ha aprendido a reconocer. Las estadísticas diferenciadas, aunque menos prominentes que los tipos, siguen siendo relevantes para decisiones de frontera.

12.2. Rendimiento computacional:

El sistema está optimizado para proporcionar respuestas rápidas y una experiencia de usuario fluida en diversos escenarios de uso, evidenciando eficiencia computacional sólida. Las métricas de latencia demuestran rendimiento adecuado mediante un **tiempo de carga inicial** inferior a 3 segundos para cargar todos los datasets (48,049 combates) y modelo preentrenado. El **tiempo de predicción individual** se mantiene bajo 50ms para generar una predicción completa con probabilidades, asegurando respuesta inmediata. El **tiempo de generación SHAP** requiere menos de 500ms para explicaciones detalladas por instancia, proporcionando transparencia sin comprometer la velocidad. El **tiempo de visualizaciones básicas** se limita a menos de 200ms para gráficos de probabilidad y comparación estadística, facilitando interpretación inmediata. El **tiempo de visualizaciones avanzadas** permanece bajo 2 segundos para gráficos complejos incluyendo matrices de correlación, manteniendo interactividad fluida.

Las métricas de memoria evidencian eficiencia en el uso de recursos computacionales mediante optimización sistemática. La **memoria del modelo** requiere aproximadamente 15MB para el pipeline XGBoost serializado, demostrando compacidad del algoritmo entrenado. La **memoria de datos** utiliza aproximadamente 180MB para todos los datasets cargados en memoria, evidenciando gestión eficiente de grandes volúmenes de información. La **memoria total de la aplicación** se mantiene en aproximadamente 220MB en condiciones normales de operación, confirmando la eficiencia del sistema completo. La **memoria pico durante entrenamiento** alcanza aproximadamente 350MB incluyendo datos temporales de validación, manteniendo requisitos de hardware moderados.

Las optimizaciones implementadas demuestran un enfoque comprehensivo en la eficiencia computacional y experiencia de usuario. El **caché de Streamlit** utiliza las funciones `@st.cache_data` y `@st.cache_resource` para eliminar recargas innecesarias, optimizando significativamente los tiempos de respuesta. La **paralelización** configura XGBoost con `n_jobs=-1` para utilizar todos los núcleos de CPU disponibles, maximizando el rendimiento computacional. La **carga diferida** genera los plots SHAP únicamente cuando son requeridos por el usuario, evitando computación innecesaria. La **gestión eficiente de memoria** implementa liberación automática de objetos temporales tras cada predicción, manteniendo el uso de memoria optimizado durante sesiones prolongadas.

12.3. Escalabilidad y límites del sistema:

El análisis de escalabilidad constituye un aspecto crítico para evaluar la viabilidad del sistema en diferentes contextos de deployment y cargas de trabajo. Esta evaluación comprende tanto la escalabilidad vertical (mejora de recursos en hardware individual) como la escalabilidad horizontal (distribución across múltiples instancias), proporcionando una perspectiva comprehensiva de las capacidades y limitaciones del sistema.

La evaluación sistemática de límites operacionales permite identificar bottlenecks potenciales, planificar recursos de infraestructura apropiados, y establecer expectativas realistas sobre el rendimiento del sistema en diferentes escenarios de uso. Este análisis es fundamental para decisiones de deployment en entornos de producción.

La escalabilidad de datos demuestra la robustez arquitectónica del sistema para manejar volúmenes crecientes de información. El **dataset actual** procesa eficientemente 48,049 combates, estableciendo una línea base sólida de rendimiento. La **capacidad estimada** alcanza hasta 1M+ combates sin degradación significativa en tiempos de predicción, evidenciando escalabilidad horizontal adecuada. El **límite de memoria** se establece en aproximadamente 2GB de RAM para datasets de 500K+ combates, manteniendo requisitos de hardware razonables. La **escalabilidad de características** permite que el modelo actual maneje 92 características y puede expandirse hasta aproximadamente 500 características sin cambios arquitectónicos fundamentales, proporcionando flexibilidad para futuras extensiones.

La escalabilidad de usuarios evidencia consideraciones prácticas para deployment en entornos multiusuario. Los **usuarios concurrentes** se benefician del soporte de Streamlit para múltiples sesiones simultáneas con caché compartido, optimizando la utilización de recursos. La **estimación de capacidad** soporta 10-20 usuarios concurrentes en hardware estándar (8GB RAM, 4 cores), proporcionando un marco referencial para planificación de infraestructura. Los **bottlenecks identificados** se concentran en la generación de plots SHAP

para múltiples usuarios simultáneos, representando el punto crítico para optimización en escenarios de alta concurrencia.

12.3.1. Análisis comparativo:

Para validar la selección del algoritmo XGBoost, se realizó una evaluación comparativa con múltiples algoritmos de aprendizaje automático, considerando métricas de rendimiento, eficiencia computacional y recursos requeridos. Esta comparación sistemática permite contextualizar la efectividad de la solución implementada y justificar la elección tecnológica realizada.

La evaluación abarcó cinco algoritmos representativos de diferentes paradigmas de aprendizaje automático, desde métodos lineales hasta técnicas de ensemble avanzadas, proporcionando una perspectiva comprehensiva del espacio de soluciones disponibles.

Algoritmo	Exactitud	Tiempo predicción	Memoria modelo
XGBoost (seleccionado)	81.12%	< 50ms	15MB
Random Forest	79.85%	< 80ms	35MB
Logistic Regression	77.32%	< 20ms	2MB
Support Vector Machine (RBF)	78.76%	< 30ms	8MB
Naive Bayes	74.52%	< 15ms	1MB

Tabla 5: Comparación de algoritmos de aprendizaje automático evaluados durante el desarrollo del sistema.

El XGBoost fue seleccionado por ofrecer el mejor balance entre exactitud, tiempo de predicción razonable y tamaño de modelo moderado, como se evidencia en la Tabla 5.

El análisis de robustez evidencia la estabilidad y confiabilidad del modelo desarrollado mediante validaciones comprehensivas. La **estabilidad de predicciones** demuestra variación inferior al 1% en accuracy entre ejecuciones con diferentes semillas aleatorias, confirmando consistencia reproducible. La **sensibilidad a datos faltantes** revela que el modelo mantiene más del 75% de accuracy incluso con 5% de características faltantes, evidenciando tolerancia razonable a información incompleta. La **generalización temporal** mediante evaluación en subconjuntos cronológicos muestra degradación mínima (inferior al 3%) en rendimiento, confirmando estabilidad longitudinal adecuada. El **análisis de casos extremos** valida que el modelo maneja correctamente Pokémon con estadísticas atípicas o combinaciones de tipos raras, asegurando robustez ante variabilidad excepcional.

13. Anexo: Instrucciones de Ejecución

Este anexo proporciona las instrucciones detalladas necesarias para instalar las dependencias y ejecutar el proyecto localmente.

13.1. Requisitos del sistema

El sistema requiere las siguientes especificaciones mínimas para su correcto funcionamiento:

Software requerido:

- Python 3.8 o superior
- Sistema operativo: Linux, macOS o Windows (se recomienda Linux/macOS)
- Navegador web moderno para la interfaz Streamlit

Hardware recomendado:

- Memoria RAM: mínimo 4GB, recomendado 8GB
- Espacio en disco: 500MB libres
- Conexión a internet (solo durante la instalación)

13.2. Instalación de dependencias

El proceso de instalación de dependencias puede realizarse mediante dos métodos principales, diseñados para satisfacer diferentes necesidades y niveles de experiencia técnica. El método automatizado está recomendado para usuarios que buscan una configuración rápida y sin complicaciones, mientras que el método manual permite un control granular sobre cada paso del proceso.

Ambos métodos aseguran la instalación correcta de todas las librerías necesarias y la configuración adecuada del entorno de Python. El sistema de gestión de dependencias implementado garantiza la reproducibilidad y estabilidad de la instalación en diferentes sistemas operativos.

13.2.1. Método 1: Instalación automatizada (recomendado)

El proyecto incluye un script automatizado que configura todo el entorno necesario:

```
# Descargar/clonar el proyecto
cd pokemon_sistema de predicción

# Ejecutar instalación automatizada
./setup.fish
```

```
# Verificar que la instalación fue exitosa
./run_tests.fish
```

13.2.2. Método 2: Instalación manual

Si prefiere realizar la instalación paso a paso:

```
# Crear entorno virtual
python3 -m venv venv

# Activar entorno virtual
source venv/bin/activate

# Instalar dependencias
pip install -r requirements.txt

# Verificar que los datos están presentes
ls data/
# Debe mostrar: all_pokemon_data.csv, combats.csv, combats_features.csv
```

13.3. Ejecución del proyecto

Una vez completada la instalación de dependencias, el sistema está listo para ser ejecutado. El proyecto proporciona múltiples métodos de ejecución para adaptarse a diferentes preferencias y escenarios de uso, desde scripts automatizados hasta comandos directos para usuarios avanzados.

La aplicación se ejecuta como una interfaz web interactiva utilizando Streamlit, lo que permite un acceso intuitivo a todas las funcionalidades del sistema de predicción. El servidor se inicia localmente y puede ser accedido desde cualquier navegador web moderno.

13.3.1. Inicio de la aplicación

```
# Método 1: Script automatizado
./start_app.fish

# Método 2: Comando directo
source venv/bin/activate
streamlit run streamlit_app.py
```

La aplicación se abrirá automáticamente en el navegador en la dirección `http://localhost:8501`.

13.3.2. Verificación del funcionamiento

Para confirmar que el sistema funciona correctamente:

```
# Ejecutar suite de pruebas
./run_tests.fish

# O manualmente:
source venv/bin/activate
python -m pytest tests/ -v
```

13.4. Estructura de archivos necesarios

El proyecto debe contener la siguiente estructura mínima para funcionar:

```
pokemon_sistema de predicción/
├── data/
│   ├── all_pokemon_data.csv
│   ├── combats.csv
│   └── combats_features.csv
├── requirements.txt
├── streamlit_app.py
├── setup.fish
├── start_app.fish
└── run_tests.fish
```

13.5. Solución de problemas comunes

Error: «Modelo no encontrado»

```
# El modelo se genera automáticamente en la primera ejecución
# Si persiste el error, eliminar y regenerar:
rm -rf models/
python streamlit_app.py
```

Error: «Archivos de datos faltantes»

```
# Verificar presencia de archivos de datos
ls -la data/
# Los archivos deben estar en el directorio data/
```

Error: «Dependencias no instaladas»

```
# Reinstalar dependencias
pip install -r requirements.txt --force-reinstall
```

Problemas con el puerto 8501

```
# Usar puerto alternativo
streamlit run streamlit_app.py --server.port 8502
```

13.6. Uso básico del sistema

Una vez iniciada la aplicación:

1. Abrir navegador en `http://localhost:8501`
2. Seleccionar dos Pokémon para el combate
3. Hacer clic en «Predecir Resultado»
4. Revisar las predicciones y explicaciones generadas
5. Explorar las visualizaciones adicionales en las pestañas laterales

14. Glosario de Términos

Este glosario proporciona definiciones claras de términos técnicos y conceptos específicos del dominio utilizados a lo largo de la documentación.

14.1. Arquitectura

Arquitectura Modular: Diseño de software que separa funcionalidades en módulos independientes y reutilizables.

Capa de Presentación: Componente de la arquitectura responsable de la interfaz de usuario y la interacción.

Carga Perezosa – **Lazy Loading**: Técnica de optimización que retrasa la carga de recursos hasta que son realmente necesarios.

Escalabilidad: Capacidad de un sistema para manejar cargas de trabajo crecientes manteniendo el rendimiento.

Patrón de Diseño: Solución reutilizable a problemas comunes en el diseño de software.

Persistencia: Almacenamiento duradero de datos que sobrevive al cierre de la aplicación.

14.2. Dominio Pokémon

Efectividad de Tipos: Sistema de ventajas y desventajas entre diferentes tipos elementales (ej: Agua es súper efectivo contra Fuego).

Estadísticas Base: Valores fundamentales que definen las capacidades de un Pokémon: HP (Puntos de Salud), Attack (Ataque), Defense (Defensa), Special Attack (Ataque Especial), Special Defense (Defensa Especial), y Speed (Velocidad).

Matchup: Enfrentamiento específico entre dos Pokémon o equipos, considerando sus fortalezas y debilidades relativas.

Meta-juego: Estrategias, tendencias y patrones dominantes en el juego competitivo en un momento dado.

Sprite: Imagen gráfica pequeña que representa a un Pokémon en el juego.

Tipo Elemental: Categoría que define las características fundamentales de un Pokémon (ej: Fuego, Agua, Planta, etc.). Los Pokémon pueden tener uno o dos tipos.

14.3. Evaluación

Cross-Validation: Técnica que divide el dataset en múltiples pliegues para evaluar la estabilidad y generalización del modelo.

Falso Negativo: Caso donde el modelo predice la clase negativa pero la real es positiva.

Falso Positivo: Caso donde el modelo predice la clase positiva pero la real es negativa.

Matriz de Confusión: Tabla que resume el rendimiento de un clasificador mostrando la distribución de predicciones correctas e incorrectas.

Test Set: Subconjunto del dataset reservado exclusivamente para evaluación final del modelo, nunca utilizado durante el entrenamiento.

Training Set: Subconjunto del dataset utilizado para entrenar el modelo de machine learning.

Verdadero Negativo: Caso donde el modelo predice correctamente la clase negativa.

Verdadero Positivo: Caso donde el modelo predice correctamente la clase positiva.

14.4. Machine Learning

Accuracy: Métrica que mide la proporción de predicciones correctas sobre el total de predicciones realizadas. Se calcula como (VP + VN) / (VP + VN + FP + FN). 10

Clasificación Binaria: Tipo de problema de machine learning donde se debe asignar cada instancia a una de dos clases posibles (en este caso: "Pokémon A gana" o "Pokémon B gana").

Ensemble Learning: Técnica que combina múltiples modelos de aprendizaje para crear un sistema de predicción más robusto y preciso que cualquier modelo individual.

F1-Score: Media armónica de precision y recall, proporcionando una métrica balanceada especialmente útil en conjuntos de datos desbalanceados. Se calcula como 2 × (Precision × Recall) / (Precision + Recall).

Feature Engineering: Proceso de crear, seleccionar y transformar variables de entrada para mejorar el rendimiento del modelo de machine learning.

Gradient Boosting: Técnica de ensemble que construye modelos secuencialmente, donde cada nuevo modelo corrige los errores del conjunto anterior. 15

One-Hot Encoding: Técnica de codificación que convierte variables categóricas en vectores binarios, creando una columna por cada categoría posible. 8

Overfitting: Fenómeno donde un modelo aprende demasiado específicamente los datos de entrenamiento, perdiendo capacidad de generalización a datos nuevos. 15

Pipeline: Secuencia de pasos de procesamiento de datos y modelado que se pueden ejecutar de manera coordinada y reproducible.

Precision: Métrica que mide la proporción de predicciones positivas que fueron correctas. Se calcula como VP / (VP + FP). 10

Recall: Métrica que mide la proporción de casos positivos reales que fueron identificados correctamente. Se calcula como VP / (VP + FN). 10

Regularización: Técnicas para prevenir el sobreajuste añadiendo penalizaciones por complejidad al proceso de entrenamiento. 15

SHAP – **SHapley Additive exPlanations**: Método para explicar predicciones individuales de modelos de machine learning, basado en la teoría de juegos cooperativos.

Train-Test Split: División del conjunto de datos en conjuntos separados para entrenamiento y evaluación, asegurando que el modelo sea evaluado en datos no vistos durante el entrenamiento.

XGBoost: Implementación optimizada de gradiente potenciado diseñada para ser escalable, portátil y eficiente.

14.5. Sistema Técnico

API – **Application Programming Interface**: Conjunto de definiciones y protocolos que permiten la comunicación entre diferentes componentes de software.

Caché: Almacenamiento temporal de datos o resultados computacionales para evitar recálculos innecesarios y mejorar el rendimiento.

Conjunto de Datos: Conjunto de datos estructurados utilizado para entrenamiento y evaluación del modelo.

DataFrame: Estructura de datos tabular bidimensional de la librería Pandas, similar a una hoja de cálculo.

Endpoint: Punto de acceso específico en una API o aplicación web.

Feature Importance: Medida de la relevancia relativa de cada característica en las decisiones del modelo.

Force Plot: Tipo específico de visualización SHAP que muestra cómo cada característica contribuye a empujar una predicción desde un valor base hacia el resultado final.

Framework: Marco de trabajo de Python para crear aplicaciones web interactivas de ciencia de datos de manera rápida.

Latencia: Tiempo que transcurre entre una solicitud y su respuesta correspondiente.

Pickle: Formato de serialización de Python utilizado para guardar objetos complejos como modelos entrenados.

14.6. Testing

Punto de Referencia: Punto de referencia estándar utilizado para comparar el rendimiento de diferentes sistemas o algoritmos.

Cobertura de Código: Métrica que indica qué porcentaje del código fuente es ejecutado durante las pruebas.

Prueba de Integración: Validación que verifica el funcionamiento correcto de múltiples componentes trabajando juntos.

Prueba Unitaria: Validación que verifica el funcionamiento correcto de componentes individuales de manera aislada.

Regresión: Degradación no intencionada en la funcionalidad debido a cambios en el código.

Suite de Pruebas: Conjunto organizado de pruebas diseñadas para validar diferentes aspectos del sistema.