Reporte de implementación de knn, regresión lineal y naive bayes para PREDICCIÓN DE GANAR O PERDER UNA PARTIDA DE VIDEOJUEGOS (septiembre de 2025)

A. Jaramillo, Barón (A01029079), P. Mauri, Martínez (A01029143) y R. Calvo, Pérez (A01028889).

Resumen - Este reporte registra el proceso que se llevó a cabo para realizar la implementación de las 3 modelos de machine learning de 3 algoritmos de clasificación binaria vistos en clase, en un contexto a elegir por el equipo de trabajo de alumnos, el cual generó por medio de google-forms; el dataset relacionado con el contexto elegido. Se explicará el contexto elegido, la teoría de los algoritmos de clasificación, explicación del proceso de desarrollo, resultados y conclusiones.

Índice de términos - Backend, data-set, frontend, KNN (K nearest neighbours), lambda, main script, naive bayes, overfitting, regresión logística, scikit-learn, SUB-TASKS, TASKS, tipo de regulación & underfitting.

I. INTRODUCCIÓN

El contexto elegido por los alumnos fue:

Predicción del resultado de una partida de videojuegos (ganar o perder) en función de las circunstancias actuales del jugador. La problemática que buscamos atender se relaciona con el impacto que tienen los hábitos de sueño en el rendimiento de los adolescentes. Con frecuencia, los jugadores sacrifican horas de descanso por continuar jugando, lo que se traduce en sesiones poco productivas y en un desgaste físico y mental. Nuestra propuesta busca ofrecer una herramienta que permita al jugador anticipar si, dadas sus condiciones actuales (como nivel de cansancio o falta de sueño), tendrá una mala sesión de juego. De esta forma, podrá tomar decisiones más saludables, como detenerse a descansar y retomar la actividad en mejores condiciones. Además, el proyecto también tiene aplicaciones en el ámbito profesional de los e-sports. Para coaches, esta herramienta puede convertirse en un apoyo que favorezca una mejor gestión de los horarios de entrenamiento, evitando prácticas poco eficientes y priorizando el descanso de los jugadores. Así, se fomenta un rendimiento más consistente y competitivo en los torneos.

II. PROCEDIMIENTO PARA EL ENVÍO DEL TRABAJO

A. Etapa de brainstorm

En nuestro caso, tuvimos dos ideas antes del contexto del data-set antes de decidirnos por la predicción de resultados de partidas de videojuegos, los cuales fueron: Asignación de carro ideal según las características dadas por el usuario. Y estimación de precio para vender un carro usado según las características de este. Sin embargo, ambas ideas debieron ser descartadas por complicaciones en la recopilación del data-set.

B. Etapa de organización

Tras haber elegido el contexto de la problemática a resolver, se creó un repositorio en GitHub para llevar a cabo la organización del desarrollo de la solución por medio de issues, se crearon 4 TASKS principales; una para el desarrollo de los algoritmos de

clasificación y preparación de datos (backend), otro para el diseño de la interfaz de usuario (frontend), otro para realizar la conexión entre la interfaz de usuario y los algoritmos, y por último; la documentación, reporte y presentación del proyecto. Para cada TASK se crearon sus respectivas SUB-TASKS, como el diseño del data-set y procesamiento de este, el desarrollo de cada algoritmo individualmente y agrupación de estos, testeo, arreglo de bugs, documentación, nuevas features, entre otros. El link del repositorio de github es el siguiente: https://github.com/andresjaramillo7/WinLosePredictor.git.

C. Etapa de desarrollo

BACKEND:

Primero se desarrollaron los 3 modelos de machine learning utilizando scikit-learn en su respectivo script (knn.py, logistic_regresion.py y naive_bayes.py), usando como métricas de rendimiento accuracy, precision, recall, f1 y ROC/AUC, priorizando las mejores calificaciones en presicion por las preferencias del contexto.

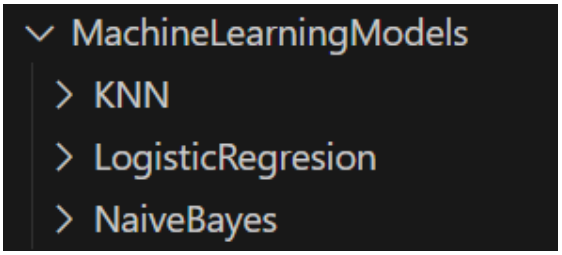

Fig. 1. Directorio de Machine Learning Models

También se implementó k-fold cross validation en cada modelo, sin embargo, se comentó esa sección por los malos resultados que dieron los modelos con la implementación. Tras haber implementado correctamente todos los modelos se realizaron pruebas en knn y regresión logística, para ver qué parámetros darían los mejores resultados. En el caso de knn, se realizó un notebook .ipynb para experimentar entre varios valores para k y fórmulas de medición de distancias. Mientras que en el caso de regresión logística se varió los parámetros lambda y tipo de regulación de manera manual. Posteriormente, se adaptaron los 3 modelos para poder importarlos en un 'main script' con el objetivo de correr desde el mismo script los tres modelos a la vez con el mismo data-set.

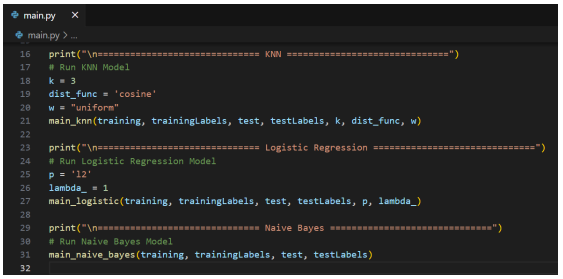

Fig. 2. Llamando los 3 modelos en un script

De forma paralela al desarrollo de los 3 modelos, se desarrolló el script (data_preparation.py) el cual tiene la función de leer el archivo .csv en donde se guarda el data-set recopilado via google forms, guardarlos en dataframes, limpiar los datos, aplicar one-hot encoding, asignar 1s y 0s a las etiquetas de 'Gané' y 'Perdí', y separar el data-set 80% de entrenamiento y 20% de prueba. Para posteriormente guardarlos en 'training', 'trainingLabels', 'test' y 'testLabels' e importar los data frames en el 'main script' mencionado anteriormente.

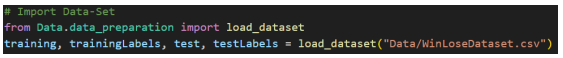

Fig. 3. Importando data-set en main_script.py

Con el objetivo de realizar un análisis de los datos recopilados, se creó otro notebook .ipynb, el cual lee el archivo .csv de las muestras y genera gráficas de tipo pie, mostrando las diferencias en el número de respuestas para cada pregunta de la encuesta. Este análisis nos ayudó mucho a la hora de tratar de comprender el comportamiento de los modelos al hacer pruebas con estos.

Para este momento del desarrollo, el directorio dedicado al "backend" del proyecto se veía de esta manera:

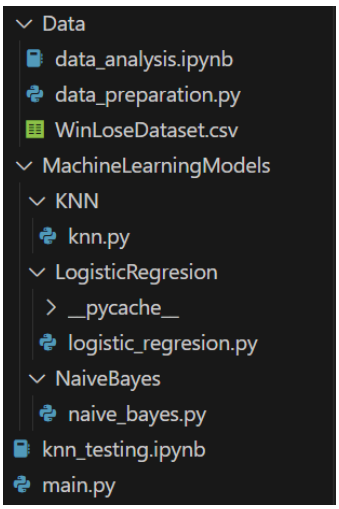

Fig. 4. Primer directorio de backend

FRONTEND:

Tras completar los elementos fundamentales del backend, se implementó el frontend como una interfaz para capturar una nueva muestra y mostrar la probabilidad de ganar o perder una partida según los modelos de aprendizaje automático desarrollados. El objetivo fue ofrecer un flujo claro en donde el usuario responde un conjunto corto de preguntas y recibe, en la misma vista, las estimaciones de cada modelo.

Se utilizó React 18 con Vite por su arranque rápido y su soporte para módulos ES. La aplicación se estructuró en un único componente raíz 'App.jsx' que mantiene el estado del formulario y el renderizado de resultados. Se tomó esta decisión ya que reduce la complejidad y facilita la trazabilidad entre entradas y salidas.

El formulario se construyó con entradas controladas mediante 'useState'. Se definieron dos objetos estáticos que son las opciones, que concentra los catálogos para cada grupo de radios y selects, e 'initial_vals', que fija el estado inicial. Cada input actualiza el estado con handlers simples, evitando desalineaciones entre la UI y los datos.

Para robustez se activó una validación mínima en el navegador con 'required' sobre los campos críticos. La semántica HTML usa 'fieldset', 'legend' y 'label' para agrupar opciones, lo que mejora accesibilidad y navegación por teclado. Esta base permite escalar luego a validaciones más estrictas sin reescribir la estructura.

El envío del formulario dispara una función 'onSubmit' que empaqueta el estado actual y solicita el cálculo al backend. La respuesta se deserializa en un objeto 'result' con las probabilidades por modelo. Este objeto se almacena en el estado y habilita la vista de resultados.

El renderizado es condicional. Mientras no exista 'result', se muestra el formulario completo. Una vez recibido, el formulario se oculta y se presenta una tarjeta de resultados. Este patrón elimina cambios de página, acorta el ciclo de retroalimentación y minimiza distracciones.

INTERCAMBIO DE DATOS:

Tras haber desarrollado tanto el 'backend' como el 'frontend', se procedió a realizar la conexión entre ambas secciones, logrando la comunicación en tiempo real entre ambas partes se implementaron 'WebSockets', lo que permitió que el 'frontend' recibiera de manera inmediata las respuestas enviadas por el 'backend'. Cada vez que el 'frontend' llama al 'socket', este invoca directamente a nuestras funciones de predicción en los algoritmos desarrollados (KNN, regresión logística y Naive Bayes).

El resultado de estas funciones se organiza en un diccionario, donde se almacenan las probabilidades y salidas correspondientes de cada modelo. Este diccionario es enviado nuevamente al 'frontend', que lo procesa y muestra de forma instantánea en la interfaz, sin necesidad de recargar la página.

Este mecanismo asegura una interacción fluida con los algoritmos, ya que la comunicación es inmediata y los resultados se reflejan en tiempo real, mejorando significativamente la experiencia de usuario.

En el backend se implementó un script en Python que contiene una función central encargada de recibir los datos de entrada en forma de lista. Estos datos pasan primero por un proceso de preprocesamiento, donde se

transforman y normalizan las características para garantizar que sean compatibles con los modelos previamente entrenados.
Una vez preparados, los datos se envían a los modelos de clasificación KNN, regresión logística y Naive Bayes que fueron entrenados con anterioridad y posteriormente serializados mediante la librería joblib. Gracias a esta serialización, los modelos quedaron almacenados en el directorio artifacts/, lo que permite cargarlos en el backend sin necesidad de volver a entrenarlos, reduciendo tiempo de cómputo y mejorando la eficiencia del sistema.
Posteriormente, cada modelo recibe los datos ya procesados y devuelve su predicción en forma de probabilidades asociadas a las clases 1 (ganar) y 0 (perder). Estas salidas se agrupan en un diccionario que incluye tanto las probabilidades como las decisiones binarias generadas por cada modelo.
Este diccionario es finalmente transmitido al frontend a través de la conexión en tiempo real habilitada con WebSockets, asegurando que los resultados se reflejen de manera inmediata en la interfaz sin necesidad de recargar la página.

D. Etapa de testeo

Para el testing nos enfocamos principalmente en el modelo de KNN y Regresión logística.
Iniciando con KNN, lo que se buscó fue maximizar la precisión, para esto se realizaron pruebas, se entrenó el modelo con Ks con números impares del 3 al 41 y esto con distintas funciones de distancia (euclidiana, manhattan y coseno). Obteniendo como función ganadora la función coseno con una k de 17.

```
Metric: euclidean | Best k: 19 | Precision: 80.00%
Metric: manhattan | Best k: 11 | Precision: 73.33%
Metric: cosine    | Best k: 17 | Precision: 81.82%

>>> Best metric: cosine | Best k: 17 | Precision: 81.82%
```

Fig. 5. Resultados de KNN, mejor métrica y ‘k’

Para tener una mejor visualización tenemos la siguiente gráfica que muestra K vs métricas de distancia:

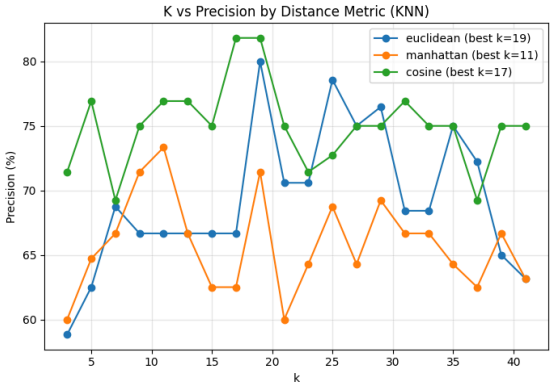

Fig. 6. Gráfica de resultados de experimentos con knn

Después también se realizaron pruebas con la regresión logística, aquí se buscó aplicar la regularización. Para scikit-learn hay dos tipos de regularización, nosotros aplicamos la de Ridge porque vimos que era la más parecida a la que vimos en clase. Realmente no hubieron casi cambios positivos, si poniéndole un valor de lambda de ‘1.0’ no hubieron cambios, al utilizar ‘0.01’ tuvimos resultados negativos al igual que si utilizamos un valor alto como ‘100’, lambdas intermedios a estos no surtieron ningún efecto en las métricas:

```
============================== Logistic Regression ==============================
--- Applying Logistic Regression using scikit-learn ---
Training Logistic Regression...
Testing Logistic Regression...
--- Logistic Regression Metrics (lambda = 0.01) | Ridge (L2) ---
Model accuracy: 0.6296296296296297
Model precision: 0.6086956521739131
Model recall: 0.9333333333333333
Model F1 score: 0.7368421052631579
Model ROC / AUC: 0.5916666666666667
Model saved to backend/artifacts/logreg_model.joblib
```

Fig. 7. Print de resultados de regularización con lambda ‘0.01’

Como podemos visualizar hizo un overfitting al tener un recall casi del 100%.

```
============================== Logistic Regression ==============================
--- Applying Logistic Regression using scikit-learn ---
Training Logistic Regression...
Testing Logistic Regression...
--- Logistic Regression Metrics (lambda = 100) | Ridge (L2) ---
Model accuracy: 0.6296296296296297
Model precision: 0.6923076923076923
Model recall: 0.6
Model F1 score: 0.6428571428571429
Model ROC / AUC: 0.6333333333333334
Model saved to backend/artifacts/logreg_model.joblib
```

Fig. 8. Print de resultados de regularización con lambda ‘0.01’

Por último, en la fase de pruebas decidimos dejar por defecto Naive Bayes porque nuestro dataset tiene correlación y Naive Bayes no trabaja bien con este tipo de datasets, pero nos interesaba conocer los resultados que podía llegar a tener, debido a esto tuvo un bajo

rendimiento en comparación con los demás modelos.

III. CORRER PROYECTO:

PARA CORRER CÓDIGO:

- Clona el repositorio en tu directorio local.
- Abre dos terminales en el directorio del repositorio.
- En la Terminal 1, ejecuta: cd '.\frontend'.
- En la Terminal 2, déjalo en el directorio del repositorio.

- Para ejecutar los tres modelos de entrenamiento y pruebas con el conjunto de datos, ejecuta el siguiente comando en la Terminal 2:
    - python.exe "[local_directory]/WinLosePredictor/backend/main.py".

- Para ejecutar la aplicación React, ejecuta los siguientes comandos en la Terminal 1:
    - Ejecuta "npm i" para instalar todas las dependencias.
    - Ejecuta "pip install socketio", la herramienta que permite guardar los modelos de entrenamiento para nuevas pruebas.
    - Ejecuta "npm run dev" para inicializar la aplicación en el host local.

CÓDIGO CORRIENDO:

Primero se corre el entrenamiento y prueba de los 3 modelos de machine learning:

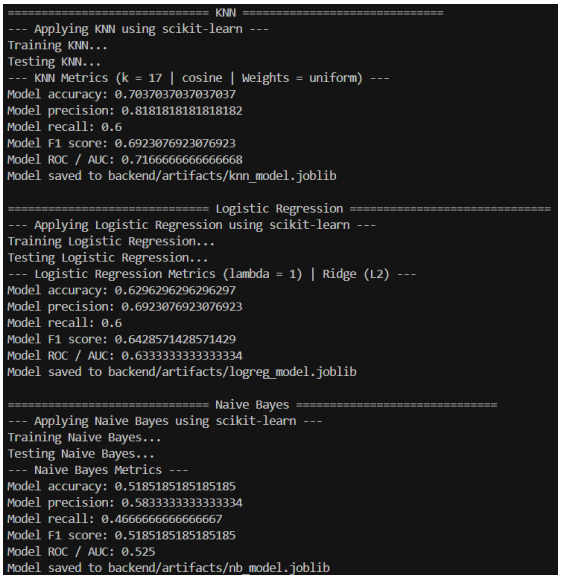

Fig. 9. Entrenando y probando modelos

Tras haber entrenado los modelos, podemos ponerlos a prueba con nuevos datos a través del 'frontend':

Fig. 10. Primer mitad de cuestionario en frontend

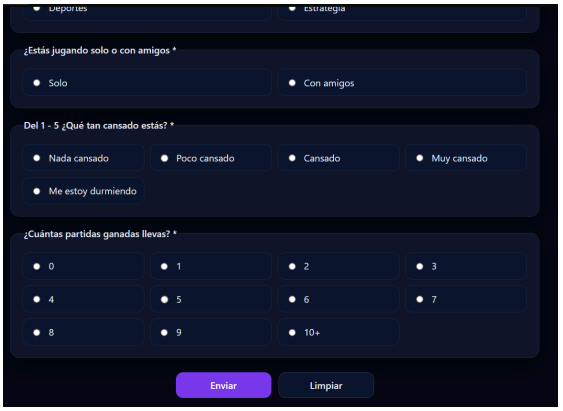

Fig. 11. Segunda mitad de cuestionario en frontend

Tras haber seleccionado todas las respuestas en el cuestionario, podemos ver de cuanto es la

probabilidad de ganar o perder según cada modelo:

Fig. 12. Resultados de prueba en frontend

Antes de mandar nuestras respuestas del cuestionario, tenemos la opción de limpiar nuestras respuestas para cambiarlas, y no se podrán enviar hasta que todas sean contestadas. Mientras que tras mostrar los resultados de una prueba, podemos realizar una nueva predicción.

IV. LA MATEMÁTICA

A. Algoritmo Knn:

El algoritmo de los k vecinos más cercanos (KNN) es un método de aprendizaje supervisado no muy simple pero eficaz. Se utiliza principalmente para problemas de clasificación y regresión. Consiste en plasmar las muestras de entrenamiento, para que cuando llegue una muestra de prueba se haga lo siguiente: calcular la distancia entre la nueva muestra de prueba y cada una de las muestras de entrenamiento de forma individual. Tras haber calculado todas las distancias, se seleccionan las ‘k’ más cercanas y se calcula la salida moda entre esas muestras para asignarla a la muestra de prueba.

Hay varias fórmulas de calcular las distancias entre las muestras para escoger, en nuestro caso experimentamos entre la euclidiana, la manhattan y la coseno para ver cuál sería la óptima.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Ec. 1. Distancia euclidiana

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

Ec. 2. Distancia manhattan

$$d(A,B) = \frac{A \cdot B}{||A|| ||B||} = \frac{\sum_{I=1}^{N} A_i B_i}{\sqrt{\sum_{I=1}^{N} A^2{}_i} \sqrt{\sum_{I=1}^{N} B^2{}_i}}$$

Ec. n. Distancia coseno

Tras la experimentación, nosotros concluimos que la distancia coseno era la óptima para usar con nuestro dataset.

B. Regresión logística:

La regresión logística funciona estimando la probabilidad de que una instancia de datos pertenezca a una clase en particular. Utilizando la función sigmoidal para transformar el resultado de una combinación lineal de las características de entrada. El algoritmo consiste en tomar todas las características de la muestra y multiplicarlas por un conjunto de pesos ‘theta’. El resultado de la combinación entre los valores de ‘theta’ y las características de la muestra, se le aplica la función sigmoidal, lo cual da un valor entre 0 y 1. Ahora, para determinar si la muestra pertenece a clase A o clase B, establece un umbral de decisión (generalmente 0.5), de esta manera, si el resultado está por debajo del umbral está en una clase, y si está por arriba se ubica en la otra clase.

$$h_\theta(x) = g(\theta^T x)$$

Ec. 3. Hipótesis de regresión logística

$$g(z) = \frac{1}{1+e^{-z}}$$

Ec. 4. Función sigmoidal

Por otro lado, la regularización en la regresión logísticaes una técnica utilizada para evitar el 'overfitting' en el entrenamiento del modelo. Consiste en añadir un término extra en la función de costo de la regresión la cual tiene el propósito de ajustar el valor del conjunto 'theta' cuando se equivoca. El parámetro lambda es el encargado de controlar la regularización.

$$J(\theta) = -\frac{1}{M}\sum_{i=1}^{M}[y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})$$
$$log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{i=1}^{n}\theta_j^2$$

Ec. 5. Ecuación de costo con regularización

$$\frac{\lambda}{2m}\sum_{i=1}^{n}\theta_j^2$$

Ec. 6. Fórmula de término de regularización

Si el valor λ es muy pequeño o igual a 0, se corre el riesgo de 'overfitting'. Mientras que si el valor de λ es muy grande, se corre el riesgo de 'underfitting'.

C. Naive Bayes:

El algoritmo Naive Bayes es un clasificador probabilístico basado en el Teorema de Bayes, el cual describe la probabilidad de que ocurra un evento dado cierto conjunto de evidencias. Su característica principal es el supuesto de independencia condicional entre las variables predictoras, es decir, se asume que cada atributo contribuye de manera independiente a la probabilidad del resultado, aunque en la práctica esto rara vez se cumple de forma estricta.
El modelo calcula la probabilidad de que una muestra pertenezca a una clase C_k dado un conjunto de características:

$$X = (x_1, x_2, x_n)$$

Ec. 7. Matriz de muestras de datos

usando la siguiente expresión:

$$P(C_k|X) = \frac{P(X|C_k)*P(C_k)}{P(X)}$$

Ec. 8. Calcular la probabilidad de X para estar en C_k

Donde:

- $P(C_k)$ es la probabilidad previa de la clase C_k
- $P(X|C_k)$ es la probabilidad de observar las características X dado que la clase es C_k
- $P(X)$ es la probabilidad total de observar las características X

El clasificador selecciona finalmente la clase con la mayor probabilidad posterior:

$$C = argmax * P(C_k)\prod_{i=1}^{n}P(x_i|C_k)$$

Ec. 9. Fórmula para clasificar con la C más probable.

Esto nos va a dar como resultado la probabilidad en un valor en un rango entre 0 y 1. En donde un valor cercano a 1 indica que la muestra tiene una alta probabilidad de pertenecer a esa clase, mientras que un valor cercano a 0 indica que la probabilidad es muy baja. En la práctica, el algoritmo calcula una probabilidad para cada clase posible y la suma de todas ellas es igual a 1, y nosotros decidiremos en base a lo que se busque, con qué probabilidad nos vamos a quedar.

D. K-fold cross validation:

K-Fold Cross Validation es una técnica de validación utilizada en machine learning para evaluar el rendimiento de un modelo de forma más confiable. Su objetivo es aprovechar al máximo los datos disponibles, evitando que los resultados dependan demasiado de una única partición entre entrenamiento y prueba.
El procedimiento consiste en dividir el conjunto de datos en K partes o "folds" de tamaño similar. Luego, el modelo se entrena K veces, cada vez utilizando K-1 folds para entrenamiento y dejando el fold restante para validación. Al finalizar, se promedian los

resultados obtenidos en cada iteración, lo que proporciona una estimación más estable y representativa del desempeño del modelo.
De manera general, el proceso se puede describir así:

1. Dividir el dataset en K subconjuntos (folds).
2. Para cada iteración i:
    a. Entrenar el modelo con K-1 subconjuntos.
    b. Validar el modelo con el subconjunto restante.
3. Guardar la métrica de evaluación (por ejemplo, accuracy, F1-score, recall, etc.).
4. Calcular el promedio de las métricas obtenidas en las K iteraciones.

Matemáticamente, si M_i representa la métrica obtenida en la iteración i, el resultado final se expresa como:

$$M = \frac{1}{K}\sum_{i=1}^{K} M_i$$

Ec. 10. Fórmula para k-folds cross validation

Esta técnica reduce el riesgo de overfitting o underfitting, ya que el modelo se evalúa con diferentes particiones de datos y no depende de un único conjunto de validación. En la práctica, valores comunes de K suelen ser 5 o 10, aunque pueden ajustarse según el tamaño del dataset.
Gracias a este enfoque, se obtiene una visión más realista del rendimiento del modelo y se aprovechan de manera más eficiente los datos disponibles.

E. One-hot encoding:

En el preprocesamiento de nuestro ‘dataset’ fue necesario transformar las variables cualitativas a un formato numérico para que los algoritmos de machine learning pudieran utilizarlas correctamente. Para la mayoría de las variables se aplicó la técnica de One Hot Encoding, lo que significa que cada categoría se representó como una columna independiente con valores binarios. De esta manera se evitó imponer un orden artificial entre categorías que no lo tienen, como ocurre en tipo de juego o forma de jugar.
La única excepción fue la variable horas jugadas, en la cual aplicamos una codificación numérica ordinal asignando valores del 0 al 4, donde cada número corresponde a un rango específico de tiempo de juego. Este enfoque resultó adecuado porque en este caso sí existe una relación de orden entre las categorías (más horas jugadas implica mayor tiempo invertido).
Con estas transformaciones, todas las variables quedaron expresadas en forma cuantitativa, lo que permite que los algoritmos implementados puedan procesar los datos de una manera eficiente, sin perder el significado de las categorías originales.

V. CONCLUSIÓN

CONCLUSIONES PERSONALES

ANDRÉS:
Siento que este proyecto aborda una problemática muy real. A mí me gustan mucho los videojuegos y más de una vez me he quedado jugando hasta tarde, para después arrepentirme porque en una partida competitiva no se rinde igual con el cansancio de la madrugada. Por eso me parece valioso trabajar en algo que pueda ayudar a tomar mejores decisiones y a cuidar el descanso.
También me dio mucha satisfacción aplicar lo que he estado aprendiendo en la concentración de IA Avanzada en un tema que me interesa personalmente. No solo fue poner en práctica lo técnico, sino verlo reflejado en una situación que conozco y con la que me identifico.
En cuanto a los datos, sé que hay un sesgo porque los obtuvimos de personas cercanas al entorno del equipo. Aun así, son datos totalmente reales y sirvieron como base para el modelo. En un futuro me gustaría mejorar esa parte, segmentando mejor a quiénes se les envía el cuestionario y llegando a más

personas con distintas experiencias. Sé que es un reto, pero también sería la manera de trabajar con un set de datos más variado y obtener resultados más completos.

PEDRO:
Me gustó mucho trabajar con mis compañeros en este proyecto. La organización de trabajo que tuvimos fue muy buena, todos tuvimos tareas fundamentales para el desarrollo de nuestro trabajo, y todos trabajamos de manera equitativa. Me gustó haber implementado diversos temas y conceptos vistos en la clase y que los hayamos aplicado a un caso real de nuestra elección, esto nos enseña que lo visto en clase nos es muy útil para desarrollar proyectos en contextos que se no hacen familiares, de esta manera, nuestra lógica de programación mejora para aplicarlo en más contextos. El haber elegido nosotros nuestra problemática y el haber recolectado nosotros nuestras muestras de datos, nos hizo darnos una perspectiva más personal y divertida para realizar la actividad. Aprendí mucho durante todo el primer periodo del curso y durante el desarrollo de nuestro proyecto, lo disfruté y considero que hicimos un excelente trabajo aún teniendo algunas fallas menores en el dataset.

RICARDO:
A lo largo del desarrollo de este proyecto aprendimos el proceso completo que implica construir un sistema de aprendizaje automático, desde la creación y análisis del ‘dataset’, hasta la implementación del ‘backend’ y ‘frontend’ conectados mediante WebSockets.
En primer lugar, comprendimos que la construcción de un ‘dataset’ no es fácil pues se requiere identificar variables relevantes, recolectar suficientes muestras y cuidar el balance de las clases para evitar sesgos que afecten el desempeño de los modelos. Nuestro análisis mostró que la predominancia de ciertas categorías, como el tipo de juego o la forma de jugar, puede condicionar los resultados y reducir la capacidad de generalización.
También descubrimos implicaciones prácticas de integrar modelos de machine learning en una aplicación real. Fue necesario diseñar un ‘backend’ capaz de ejecutar algoritmos como KNN, regresión logística y Naive Bayes, y al mismo tiempo implementar una interfaz en el frontend que recibiera y mostrará los resultados de manera inmediata. El uso de sockets permitió garantizar una comunicación dinámica y en tiempo real, lo que enriqueció la experiencia de usuario.
Finalmente, este proyecto nos permitió reflexionar sobre la importancia de evaluar no sólo la exactitud de los modelos, sino también las condiciones del dataset, las limitaciones de cada algoritmo y los posibles sesgos que surgen de los datos.

APÉNDICE

Imágenes:

- Fig. 12. Resultados de prueba en frontend.

Ecuaciones:

- Ec. 1. Distancia euclidiana.
- Ec. 2. Distancia manhattan.
- Ec. 3. Hipótesis de regresión logística.
- Ec. 4. Función sigmoidal.
- Ec. 5. Ecuación de costo con regularización.
- Ec. 6. Fórmula de término de regularización.
- Ec. 7. Matriz de muestras de datos.
- Ec. 8. Calcular la probabilidad de X para estar en Ck.
- Ec. 9. Fórmula para clasificar con la C más probable.
- Ec. 10. Fórmula para k-folds cross validation.

RECONOCIMIENTO

Los tres autores del proyecto quieren agradecer al profesor Víctor Manuel de la Cueva Hernández por haber estado al pendiente cuando el equipo de trabajo tuvo dudas acerca de los requerimientos de la entrega.

Así como agradecer al profesor Esteban Castillo Juarez, profesor Gualberto Aguilar Torres y de nuevo al profesor Víctor Manuel de la Cueva Hernández, por repartir los conocimientos básicos de los conceptos utilizados en la solución realizada.

REFERENCES

[1] Jaramillo Barón, A., Mauri Martínez, P., & Calvo, Pérez, R., (2025, 16 septiembre). WinLosePredictor. GitHub. https://github.com/andresjaramillo7/WinLosePredictor.git

[2] E. C. Juarez, “Clasificación con k-Nearest Neighbors (vecinos más cercanos)” (s. f.). Recuperado de: https://experiencia21.tec.mx/courses/623988/files/244401393?wrap=1, pp. 8-10.

[3] E. C. Juarez, “Clasificación con regresión logística” (s. f.). Recuperado de: https://experiencia21.tec.mx/courses/623988/files/244401451?wrap=1, pp. 11-13.

[4] E. C. Juarez, “Clasificador Naive Bayes” (s. f.). Recuperado de: https://experiencia21.tec.mx/courses/623988/files/244401459?wrap=1, pp. 13-18.