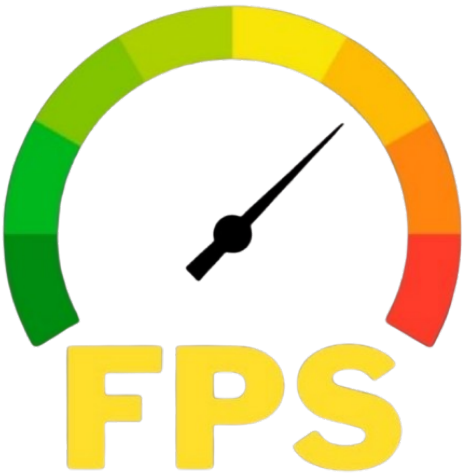
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APA

Predicción de los FPS en juegos modernos a partir de las características del hardware

Ismael El Basli y Jordi Muñoz



Q1 2024-25

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Índice

1. Descripción del trabajo y objetivos

Este trabajo se centra en la predicción de los FPS en juegos modernos a partir de las características del hardware, usando modelos de aprendizaje automático. El predecir FPS en base a características del hardware se trata de un problema real y complejo. Muchas veces, cuando uno quiere comprar una nueva CPU o GPU de cara a tener una mejor experiencia en videojuegos, debe ver el resultado que obtiene gente que ha probado el videojuego en cuestión usando ese hardware concreto, de cara a ver un aproximado de los FPS que obtendría. Si uno llega a obtener un buen modelo de aprendizaje automático para predecir los FPS, no seria necesaria la ejecución explícita de cara a saber los FPS aproximados que se obtendrían. Este problema nos parece realmente interesante, y al ser la variable objetivo una variable continua, estamos ante un problema de **regresión**.

Acerca de los objetivos del trabajo, podemos destacar los siguientes:

- Ser capaces de encontrar un problema complejo y real, con un dataset decente con el que trabajar, y estudiar posibles aproximaciones existentes de soluciones al problema.
- Analizar bien los datos y realizar el preprocesado pertinente.
- Entender bien como repartir nuestros datos en conjuntos de entrenamiento, validación y prueba.
- Usar modelos lineales y no lineales para resolver el problema, además de saber como se debe evaluar el rendimiento de éstos.
- Analizar los resultados y ser capaces de interpretarlos.

1.1. Breve descripción de los datos

Después de dedicar un tiempo a buscar un dataset que cumpliera con ciertas condiciones, hemos encontrado uno en Kaggle [1]. El dataset contiene mediciones reales de FPS de videojuegos modernos ejecutados en distintos ordenadores. Cada fila describe varios atributos del hardware del ordenador, así como características relacionadas con las configuraciones de resolución del videojuego. Un ordenador se caracteriza por detalles de su CPU y GPU. Los FPS son entonces la variable objetivo, que como mencionamos antes es continua y por lo tanto el problema es de regresión.

En nuestro dataset tenemos una combinación de variables numéricas y categóricas, que suman un total de 43 variables, sin contar la variable objetivo. Los datos no son sintéticos, y por este motivo dedicamos cierto tiempo al preprocesado ya que, por ejemplo, hay varias variables con valores perdidos. El dataset tiene un total de 24624 muestras, un numero suficiente para posiblemente obtener resultados decentes.

1.2. Breve descripción de los estudios previos

Después de dedicar cierto tiempo a buscar estudios previos sobre el conjunto de datos con el que trabajamos, no hemos encontrado nada. La fuente de la cual hemos obtenido los datos no adjunta ninguna referencia, y no hemos encontrado ningún estudio sobre el tema en internet. Lo único mínimamente relacionado son modelos de aprendizaje automático que buscan mejorar los FPS en videojuegos, como por ejemplo DLSS de NVIDIA [2] o XeSS de Intel [3]. Pero este no es el objetivo de nuestro trabajo, que se centra en predecir FPS en base al hardware.

2. Proceso de exploración de los datos

En esta sección se describe el preprocesado que hemos realizado a nuestros datos, así como cualquier decisión tomada que sea importante mencionar. Además, mencionamos brevemente una visualización de los datos en una dimensionalidad reducida. Cabe destacar que de forma previa al preprocesado realizado hemos hecho una exploración breve de las distribuciones de nuestros datos, pero no incluimos las imágenes porque el documento se haría demasiado extenso.

2.1. Tratamiento de valores perdidos

En nuestro dataset solo teníamos 4 atributos con valores faltantes. Podemos ver el porcentaje de valores faltantes en la siguiente tabla:

Atributo	**Porcentaje de valores faltantes**
gpunumberofexecutionunits	100.00 %
gpunumberofcomputeunits	77.78 %
cpudiesize	52.63 %
cpunumberoftransistors	52.63 %

Tabla 1: Columnas con valores faltantes y su porcentaje.

Dado que los porcentajes de valores faltantes son elevados (superiores al 50 %) imputar sus valores podría introducir ruido o sesgos en nuestros modelos al basarse en suposiciones o datos incompletos. Por este motivo hemos decidido eliminar las variables.

2.2. Tratamiento de valores anómalos

Dado que trabajamos con un dataset que dispone principalmente de variables que corresponden a componentes o características del hardware podemos comprobar con seguridad si los valores anómalos se deben a la naturalidad implícita de los datos o pueden ser considerados como errores que deberían ser eliminados o transformados.

Para detectar dichos valores hemos realizado un análisis univariante mediante *box-plots* para ver si hay ejemplos que se encuentran fuera del rango típico definido por

el rango intercuartílico (IQR). Después de observar los resultados obtenidos, hemos llegado a la conclusión que los valores que se salen del rango típico son correctos y se salen de lo habitual simplemente porque son atributos que corresponden a ordenadores con capacidades bastante superiores a la media.

2.3. Tratamiento de valores incoherentes o incorrectos

Como hemos mencionado previamente, debido a la naturaleza de nuestros nos es realmente fácil detectar si hay algún valor incoherente o incorrecto. Después de una exploración de los datos hemos visto que todos tienen sentido y son correctos. Por ejemplo, no tendría sentido el atributo `gpumemorysize` tenga un valor negativo, pero este tipo de ejemplos no aparecen en nuestros datos. Debido a esto, no ha sido necesario eliminar ningún valor.

2.4. Codificación de variables no continuas o no ordenadas

En esta sección describimos principalmente como hemos realizado la codificación de variables categóricas. Lo primero que hemos hecho ha sido detectar dichas variables y el núméor de categorías posibles, ya que es un valor clave que influye en la decisión de cómo codificar:

```
Columna: cpuname - Total de categorías: 19
Columna: cpumultiplierunlocked - Total de categorías: 2
Columna: gpuname - Total de categorías: 27
Columna: gpuarchitecture - Total de categorías: 6
Columna: gpubus.interface - Total de categorías: 2
Columna: gpudirectx - Total de categorías: 2
Columna: gpumemorytype - Total de categorías: 4
Columna: gpuopencl - Total de categorías: 2
Columna: gpuopengl - Total de categorías: 1
Columna: gpushadermodel - Total de categorías: 2
Columna: gpuvulkan - Total de categorías: 3
Columna: gamename - Total de categorías: 24
Columna: gamesetting - Total de categorías: 2
```

Figura 1: Variables categóricas y posibles categorías

Como se ve en la imagen, tenemos 13 variables categóricas, entre ellas 6 binarias, una con una única categoría posible y las 6 restantes con mas de 2 categorías posibles. Evidentemente eliminamos la variable `gpuopengl` con una única categoría posible, ya que no aporta información para la toma de decisión. Por otro lado, hemos convertido las variables categóricas binarias a numéricas con valores 0 o 1.

En cuanto a las variables categóricas con mas de 2 categorías posibles, hemos observado que había tres con un número de categorías igual o inferior a 6, por lo que hemos considerado viable realizar un *one-hot encoding*.

Finalmente, solo quedaba codificar 3 variables categóricas: `cpuname`, `gpuname` y `gamename`, las cuales tenían 19, 27 y 24 categorías, respectivamente. Debido al alto número de categorías posibles vimos inviable aplicar de forma sistemática *one-hot*

encoding, ya que se crearía un número excesivo de variables nuevas. Debido a que consideramos que estas variables eran muy importantes de cara a las predicciones decidimos observar los *boxplots* por cada variable frente la variable objetivo (FPS) para ver si podíamos sacar alguna conclusión que pudiese guiar nuestra decisión a la hora de codificar. A continuación podemos ver el `boxplots` de `gpuname`:

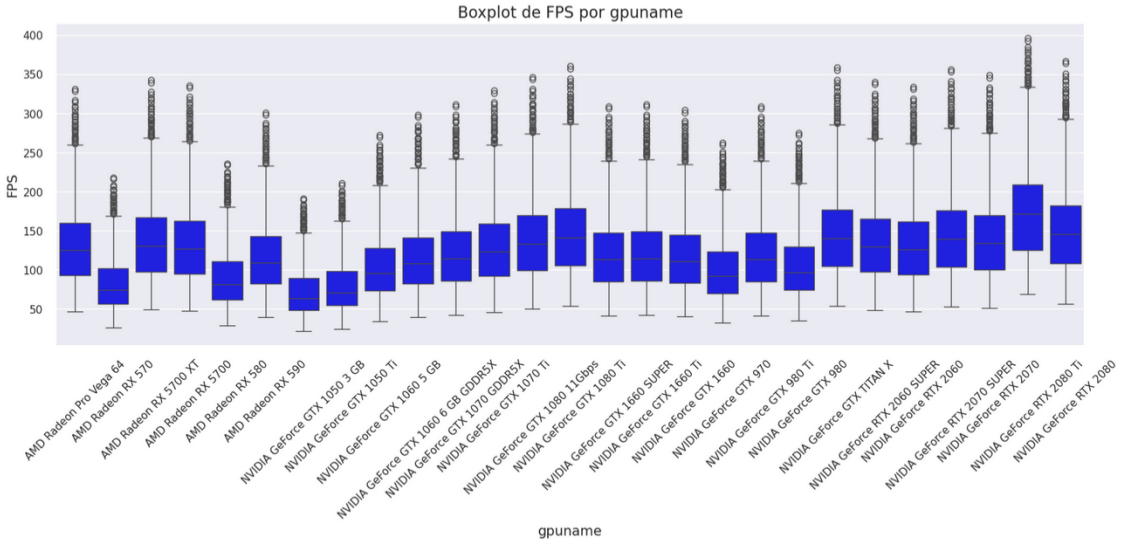


Figura 2: Boxplot de FPS por `gpuname`

No mostramos los otros dos, pero están disponibles en el notebook. Viendo los boxplots, vimos como estas variables eran informativas, ya que en función de la categoría que escojamos la distribución seria distinta. Barajamos varias opciones en esta codificación, como por ejemplo agrupar las gpu's en NVIDIA y AMD, y las cpu's en Intel y AMD, pero asumir que todas las GPU/CPU de una marca tienen el mismo impacto en los FPS ignora las diferencias de modelos específicos. Por ejemplo, una NVIDIA GTX 1050 rendiría peor que una NVIDIA RTX 3060, pero el modelo no captaría estas diferencias si solo usamos AMD/NVIDIA. También pensamos en usar *LabelEncoding*, pero no queríamos imponer una relación de orden artificial.

Después de pensarlo, hemos decidido usar ***Target Encoding***, una codificación basada en la media de FPS. Con esta codificación asignamos a cada categoría su promedio de FPS. Esto es eficiente en términos de dimensionalidad y nos permite capturar la relación entre la categoría y la variable objetivo

2.5. Eliminación de variables irrelevantes o redundantes

Como hemos visto previamente, hemos eliminado una variable categórica: `gpuopengl`, dado que era constante, así que luego hemos buscado entre las variables numéricas para ver si también encontrábamos alguna constante y efectivamente hemos encontrado dos, `cpubaseclock` y `gameresolution`, así que las hemos eliminado.

Seguidamente, dado que tras tratar las variables categóricas tenemos un número de variables bastante elevado, hemos procedido a eliminar variables altamente co-

rrelacionadas ya sea linealmente o aquellas que tengan una multicolinealidad muy alta, usando el VIF (Factor de la Inflación de la Varianza), que es una medida que cuantifica cuánto se inflan las varianzas de los coeficientes estimados en un modelo de regresión debido a la colinealidad entre las variables independientes.

Con la matriz de correlación hemos empezado eliminando (quedándonos con una de las dos) aquellas variables que tenían una correlación del 100 %, 6 en total.

Seguidamente veíamos que teníamos variables altamente correlacionadas, por ejemplo el grupo de las caches de la cpu y el numero de cores y threads o varias variables referentes a la gpu. Por eso hemos decidido establecer un threshold de +/-85 % para eliminar aquellas que lo superaban, siempre intentando coger las variables que tenían un número de parejas altamente correlacionadas mas alta (para eliminar el mínimo posible). En este paso hemos visto como nuestra hipótesis sobre las variables que teníamos intención de eliminar por criterio lógico, se ha cumplido en dos variables, sin embargo conservamos la varibale `cpuname` dado que al eliminar bastantes variables relativas a la cpu, vemos que con las restantes no tienen tanta correlación. Y por otra parte del grupo altamente correlacionado, nos hemos quedado con la cache de nivel 2 y el número de cores.

2.6. Creación de nuevas variables que puedan ser útiles

Dado que teniamos un número mas que considerable de variables útlies en nuestros datos, no hemos considerado añadir ninguna característica extra a los datos.

2.7. Transformación de las variables

Por lo que hace a la transformación de las variables no hemos considerado tampoco aplicar ninguna ya que nuestras variables no tienen ninguna asimetría grave y/o curtosis en los valores de los datos.

2.8. Conjuntos de datos

Como hemos visto y bien es sabido, muchas veces el preprocesado que uno cree que es el mejor, no lo es, es por eso que hemos decidido maximizar nuestras oportunidades de obtener un buen modelo creando un conjunto de datasets distintos pensando en el timepo de ejecución y en el criterio de selección de variables.

Para ello hemos creado los siguientes datasets:

- **Full-preprocessing**: Preprocesado completo de los datos.
- **Full-preprocessing con 50 % de instancias**: Por si observamos que tenemos demasiados datos.
- **Preprocessing antes de VIF**: Por si carecemos de sesgo; obtenemos resultados con el preprocesado completo.

- **Preprocessing antes de** ± 0.85: Por si carecemos de sesgo; obtenemos resultados con el preprocesado sin VIF.
- **One-hot encoding para todas las variables categóricas**: Por si vemos que el modelo inicial tiene margen en cuanto a tiempo de computación.
- **One-hot encoding para todas las variables categóricas con 50 % de instancias**.
- **One-hot encoding para todas las variables categóricas con 33 % de instancias**.

3. El protocolo de remuestreo

Dado que finalmente nuestro dataset principal obtenido post preprocesamiento tiene un número bastante considerable de instancias, hemos considerado realizar la partición de los datos en un 70 % datos de entrenamiento y un 30 % en datos de prueba. Por otra parte como tenemos un problema de regresión no debemos especificar ninguna estratificación al hacer la partición y finalmente para realizar la validación cruzada hemos fijado un número de *folds* de 5 para todos los modelos, de forma que así evitamos favorecer ciertos modelos. Por otro lado, es importante mencionar que fijamos una semilla para que las ejecuciones sean reproducibles.

Varios de los modelos que entrenamos tienen una serie de hiperparámetros ajustables. En esos casos hemos usados búsquedas exhaustivas o bayesianas dependiendo del potencial tiempo de entrenamiento. En casos donde se tarda poco hemos realizado una búsqueda exhaustiva mediante GridSearchCV, mientras que en casos donde se tarda mucho hemos optado por BayesianSearchCV. En las búsquedas bayesianas hemos fijado el número de iteraciones a 30 iteraciones, ya que después de algunas pruebas hemos visto que era suficiente. Cabe destacar que el espacio de hiperparámetros que definimos es en base a valores típicos que suelen dar buenos resultados. En relación a la métrica a optimizar usada en las búsquedas, hemos usado el error cuadrático medio (MSE).

Posteriormente al entrenamiento, siempre realizamos lo mismo por cada modelo:

- Mostramos los hiperparámetros que mejor resultado han ofrecido en la validación cruzada en train, y mostramos las métricas que corresponden.
- Analizamos el gráfico de los residuos para ver su distribución.
- Realizamos un plot de los resultados predichos frente los reales en la partición de test, para ver si el modelo no se ha sobreajustado.
- Realizamos una interpretación del modelo, viendo la contribución de las variables.
- Guardamos el modelo para posteriormente compararlo con los demás.

4. Resultados obtenidos con modelos lineales

Una vez hecho todo el preprocesado y análisis previo, procedemos a entrenar modelos lineales para tratar de resolver nuestro problema. Dado que en nuestro caso resolvemos una tarea de regresión, utilizaremos los modelos siguientes:

- Regresión lineal
- Regresión Ridge
- Regresión Lasso
- KNN
- SVM lineal
- SVM cuadrático

Sabemos que el enunciado solo pedía 3, pero hemos probado con mas porque queríamos explorar al máximo como se comportan nuestros datos con los distintos modelos. Por cada modelo, mostramos una tabla que muestra el R^2 obtenido tanto en la partición de train con validación cruzada como el obtenido en la partición de test, así como el MAE y MSE en test. Respecto la interpretabilidad, no mencionamos nada ya que hay una sección específicamente dedicada a ello. A parte de los resultados, indicamos cuál ha sido el mejor conjunto de parámetros.

4.1. Regresión lineal

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.948	0.947	151.6	8.53

Tabla 2: Resultados de la Regresión Lineal

Vemos como los resultados han sido muy buenos, pese a ser un modelo muy simple. El modelo claramente no se ha sobreajustado y generaliza bien. Aunque los resultados son buenos hay margen de mejora, ya que vemos como el MAE es de aproximadamente 8 FPS. Por otro lado, en este modelo no hay hiperparámetros que ajustar.

4.2. Regresión Ridge

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.948	0.947	151.6	8.53

Tabla 3: Resultados de la Regresión Ridge

Vemos como los resultados son idénticos a los de la regresión lineal simple, la regularización no ha permitido mejorar. En este caso sí que hay un parámetro ajustable, λ, que es el parámetro que controla cuánto penalizamos la magnitud de los coeficientes del modelo. El mejor valor de λ en la validación cruzada sobre train ha sido de 5.

4.3. Regresión Lasso

Los resultados obtenidos en esta regresión vuelven a ser idénticos a los dos anteriores, por lo que no mostramos la tabla de nuevo. Lo único que cabe destacar es que en este caso el mejor valor de λ es de 0.001, un valor distinto que el mejor para Ridge.

4.4. KNN

Para el KNN hemos empezado entrenando el modelo con los datos escaldaos con MinMax y haciendo una exploración tanto del número de vecinos como la función de peso en la predicción como de la métrica de distancia usada para el computo.

Incialmente hemos visto que el mejor resultado lo obteníamos con 3 vecinos, la métrica de chebyshev y la función de peso el inverso de la distancia. Sin embargo, al mostrar la evolución del rendimiento al variar el número de vecinos, hemos visto que por culpa de la función de peso, el modelo básicamente memorizaba los datos en su totalidad, así que hemos fijado la la función de peso a uniforme para volver a realizar la exploración.

Tras hacerlo hemos visto como ahora ya el modelo no memoriza los datos al 100 % y por lo tanto, tanto en el gráico de residuos como en el qq-plot, ya vemos unos resultados más aceptables, sin embargo vemos como los datos no siguen una distribucion muy gaussiana dado que la nube de puntos no tiene una forma muy elíptica y la forma del qq-plot no es una linea diagonal si no que tiene cierta curvatura en las dos lados de la campana.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.991	0.975	73.563	5.480

Tabla 4: Resultados del KNN

Vemos que con este modelo, con los parametros comentado previamente, obtenemos una $\mathbf{R^2}$ en el conjunto de entrenamiento de 99 % y 97.47 % en el de prueba, por lo tanto observamos un ligero sobreajuste, y que el MSE lo logramos reducir a 73 en este caso es decir se reduce mas de un 50 % respecto los modelos anteriores. Vemos por otra parte que las predicciones de error se ajustan mucho a la recta ideal.

4.5. SVM lineal

Para el SVM lineal, hemos realizado una exploración para jugar con el parámetro de regularización y la epsilon, obteniendo los mejores resultados con un valor de regularización equivalente a 1 y una epsilon de 5.

Al mostrar el grafico de residuos y el qq-plot, vemos como la forma es prácticamente idéntica a la obtenida con la Regressión lineal con y sin regularización, como era de esperar.

Si observamos la nube de puntos no es para nada gaussiana, tiene una forma que se asemeja a un "boomerangz la qq-plot presenta una curvatura considerable y en los extremos de las colas presenta anomalias mas notorias. Vemos a su vez que el grafico de prediccion de error sigue teniendo una forma parecida a una funcion logaritmica y como anteriormente no observamos sobreajuste en los datos observando el parámetro $\mathbf{R^2}$ en ambos conjuntos, sin embargo vemos como es el peor modelo obtenido hasta el momento.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.947	0.947	154.198	8.498

Tabla 5: Resultados del SVM Lineal

4.6. SVM cuadrático

Respecto al SVM cuadrático, al realizar la exploración, hemos visto que los mejores resultados han sido los obtenidos con un valor de regularización equivalente a 100 y una epsilon de 1.

Al ver el grafo de residuos vemos que la forma de los residous no se podría considerar que se asemeja mucho a una forma elíptica y presenta algun patron en forma diagonal tanto en la parte positiva de los residuos como en la negativa. Respecto al qq-plot vemos como pese a observar curvatura, como nos adelantaba el gráfico de residuos, vemos como las colas estan bastante mas suavizadas pero la curvatura es algo mas notoria. Respecto al grafico de prediccion de errores vemos como se ajusta casi a la perfeccion con el ajuste idílico.

Si comparamos la métrica $\mathbf{R^2}$ vemos como no hay una sobreespecializacion muy grande, de hecho solo hay una diferencia de 0.12 % y por otra parte vemos como el MSE se reduce hasta un valor de 6.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.998	0.998	6.092	1.485

Tabla 6: Resultados del SVM Cuadrático

5. Resultados obtenidos con modelos no lineales

Como modelos no lineales hemos decidido probar con los siguientes:

- SVM con kernel RBF
- Random Forest
- MLP
- Gradient Boosting
- Combinación de modelos: Voting y Stacking

Mostramos la tabla con métricas igual que en la secció anterior, además de mencionar el mejor conjunto de hiperparámetros.

5.1. SVM con kernel RBF

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
1.0	1.0	0.303	0.408

Tabla 7: Resultados de la SVM con kernel RBF

Este modelo ha sido el modelo base que nos ha dado un mejor rendimiento. Vemos como el MAE obtenido en test es de aproximadamente 0.4 FPS, y el R^2 en la partición de train con validación cruzada es de 1. Por otro lado, cabe mencionar que aun haber usado una búsqueda bayesiana el entrenamiento ha sido bastante costoso en cuanto al tiempo de ejecución. En este caso, se exploraron los hiperparámetros C y γ, claves para el equilibrio entre el ajuste y la capacidad de generalización del modelo. El mejor conjunto encontrado fue $C = 331.13$, que controla la penalización por errores, y $\gamma =$ 'auto', que ajusta la influencia de cada muestra. Esta configuración logró un balance óptimo entre precisión y generalización. Igualmente, es importante mencionar que había otros conjuntos de hiperparámetros que proporcionaban un resultado similar.

5.2. Random Forest

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
1.0	0.998	4.945	1.402

Tabla 8: Resultados del Random Forest

Se ve claramente como Random Forest es un buen modelo que tampoco se ha sobreajustado. Este modelo ha sido con diferencia el que nos ha costado mas tiempo entrenar, debido a nuestra avaricia a la hora de definir el espacio de hiperparámetros en el que buscar con la búsqueda bayesiana. Para este modelo, se exploraron

hiperparámetros clave como el criterio de división, la profundidad máxima del árbol, el número mínimo de muestras por hoja y la cantidad de árboles en el bosque. El mejor conjunto encontrado fue: criterion = 'squared_error', max_depth = None, min_samples_leaf = 1 y n_estimators = 200. Esta configuración optimizó el balance entre sobreajuste y generalización.

5.3. MLP

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
1.0	1.0	1.115	0.724

Tabla 9: Resultados del MLP

De nuevo, MLP es un modelo que nos ha dado resultados muy buenos, y sin sobreajuste. En este caso, se exploraron hiperparámetros esenciales como la función de activación, el tamaño de las capas ocultas y la tasa de aprendizaje inicial. El mejor conjunto encontrado fue: activation = 'relu', hidden_layer_sizes = 200 y learning_rate_init = 0.01. Esta configuración permitió un entrenamiento eficiente con una buena capacidad de generalización.

5.4. Gradient Boosting

En la sección de Gradient Boosting, inicialmente quieramos probar con distintos modelos LightGBM, CatBoost y XGBoost. Sin embargo, finalmente hemos decidido usar XGBoost, dado que nuestro conjunto de datos no era tan grande como para que viesemos una mejora de rendimiento con LightGBM respecto los XGBoost y por otra el CatBoost nos hacia trabajar con otro conjunto de datos preprocesados y como fuimos viendo que el conjunto preprocesado incial ya obtenia resultados muy buenos, decidimos mantener coherencia y cohesión en cuanto al conjunto de datos para comparar modelos.

Para aplicar XGBoost hemos hecho una búsqueda exploratoria en cuanto a la profundiad màxima, la cantidad de estimadores (árboles) y la tasa de aprendizaje.

Tras aplicar la búsqueda, hemos obtenido una R^2 de 0.99965 un MSE de 1.007 y un MAE de 0.727, lo que nos deja a este modelo como el segundo mejor clasificado por debajo del SVM RBF, respecto el gráfico de residuos observamos una nube de puntos bastante éliptica y respecto el qq-plot, vemos como en el conjunto de entrenamiento observamos una linealidad bastante mas aceptable que con el conjunto de prueba. Y finalmente vemos comparando la R^2 del conjunto de entrenamiento 0.99978, con la del conjunto de prueba, 0.99965, que no hay una sobreespecialización notoria.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
0.99978	0.99965	1.008	0.727

Tabla 10: Resultados del XGBoost

5.5. Combinación de modelos

Finalmente para conlcuir con los modelos no lineales, hemos decidido implementar tanto un modelo de voting y un modelo de stacking. Para ambos hemos decidido usar los 3 mejores modelos, el SVM con kernel RBF, el modelo MLP y el XGboost.

5.5.1. Voting

Tras mostrar el grafo de residuos y el qq-plot, vemos como la métrica R^2 obtenida es realmente alta, se redonde a 1 pero realmente es de 0.99988. Vemos como el grafico de residuos tiene una nube de puntos parecida a una elipse con mas anchura que altura y que hay algunos outliers y en el qq-plot vemos como la distribución no es muy gaussiana, presenciamos cierta curvatura en ambas colas.

Vemos por otra parte que el MSE sube 0.033 puntos respecto al mejor modelo previo (SVM RBF), sin embargo el MAE se reduce en 0.002 puntos.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
1.0	1.0	0.336	0.406

Tabla 11: Resultados del Voting

5.5.2. Stacking

Por lo que hace al modelo de stacking, hemos probado con dos meta-modelos distintos y hemos visto que el mejor era con una Regresión lineal, con un grafo de residuos con un poco mas de errores en los valores mas altos de FPS, pero con una nube de puntos bastante elíptica y el qq-plot vemos como se ajusta bastante mas a una recta dado que las curvarturas presenciadas en los previos modelos aparecen bastante mas suavizadas.

Para finalizar, vemos como en referencia a las mètricas, obtenemos una reducción de 0.078 puntos en el MSE y de 0.058 en el MAE, resultando en el mejor modelo con una métrica R^2 de 0.99992.

R^2 **TrainCV**	R^2 **Test**	**MSE Test**	**MAE Test**
1.0	1.0	0.228	0.350

Tabla 12: Resultados del Stacking

6. Comparativa de los resultados

A continuación mostramos una tabla con las métricas de todos los modelos:
Observamos que los modelos lineales como intuíamos al ajustar una regresion linear simple para el paso previo a la elaboración de la práctica, han sido los que han obtenido peores resultados, dado que son más simples y funcionan bien en datos donde

Modelo	R^2 TrainCV	R^2 Test	MSE Test	MAE Test
Regresión lineal	0.948	0.947	151.6	8.5
Regresión Ridge	0.948	0.947	151.6	8.5
Regresión Lasso	0.948	0.947	151.6	8.5
KNN	0.991	0.975	73.5	5.5
SVM lineal	0.947	0.947	154.2	8.5
SVM cuadrático	0.998	0.998	6.1	1.5
SVM RBF	1.0	1.0	0.3	0.4
Random Forest	1.0	0.998	4.9	1.4
MLP	1.0	1.0	1.1	0.7
Gradient Boosting	1.0	1.0	1.0	0.7
Voting	1.0	1.0	0.3	0.4
Stacking	1.0	1.0	0.2	0.3

Tabla 13: Resultados finales

la relación entre las variables es esencialmente lineal. Sin embargo, hemos visto al analizar la matriz de correlaciones y el análisis de multicolinealidad con VIF en el preprocesado de los datos, que nuestras características dependian mucho las unas de las otras.

Los modelos no lineales, como el SVM con kernel RBF, el XGBoost o los modelos de aprendizaje profundo como el MLP, son capaces de capturar patrones más complejos y ofrecen una mayor precisión, aunque generalmente requieren más recursos computacionales y tiempo de entrenamiento.

Los modelos lineales requerian de unos escasos segundos de ajuste, sin embargo modelos como los SVM requerían una media de 1 minuto (con nuestro HW) por modelo, lo que ya implicaba un coste razonable al realizar la exploración de hiperparámetros y por otra parte el Random Forest ha sido sin duda el modelo que más tiempo requería, dado que los árboles son independientes y grandes y no tienen optimización interna ni por regularización ni por gradiente.

7. Justificación del modelo escogido

Como hemos visto, a sorpresa de nuestras expectativas iniciales y gracias a un procesado acertado, hemos conseguido obtener una variedad de modelos con muy buen rendimiento. A pesar que los modelos lineales son muy buenos, los modelos no lineales han sido capaces de mejorar aun mas los resultados obtenidos.
Dado que no hemos presenciado señales de sobre ajuste en ningún modelo, hemos decidido elegir la combinación de SVM con RBF, MLP y XGBoost mediante *Stacking*. Esto es debido a que es relativamente rápido y además es el que es mas preciso. Combina los mejores modelos base que hemos obtenido. Dado que en general tenemos muy buenos modelos, uno podría escoger el que quiera en función de sus prioridades. Por ejemplo, si la interpretabilidad es crucial, se podria optar por al-

gun modelo lineal.

Así que vemos que nuestro modelo definitivo, es capaz de predecir los FPS con un MAE de aproximadamente 0.3 FPS y un R^2 en el conjunto de test de 0.99992, así como un R^2 de 1 en la partición de train con validación cruzada.

8. Interpretación de los modelos

En nuestro caso, en todos los modelos, tanto lineales como no lineales se coincide en que el atributo mas importante se trata de `gamename_encoded`, con diferencia. Cabe recordar que esta variable se trataba de una categórica que decidimos codificar con *Target encoding*. Los resultados tan buenos que hemos obtenido nos han permitido confirmar que esa decisión fue muy acertada. Además, todos los modelos lineales también coinciden en que los siguientes atributos más importantes para las predicciones son `cpuname_encoded` y `gamesetting_encoded`, los cuales también eran variables categóricas que fueron codificadas con *Target encoding*. A parte de estos atributos, luego solo reciben importancia a considerar ciertas variables relacionadas con el hardware de la CPU. Sin embargo, en los modelos no lineales, `cpuname_encoded` y `gamesetting_encoded` suelen aparecer como no tan importantes en comparación a ciertas características del hardware de la GPU, aunque todo depende del modelo.

Cabe destacar que en los modelos lineales hemos realizado la interpretación simplemente mirando los pesos asignados a los atributos, mientras que en los modelos no lineales nos hemos fijado en la permutation_importance en la partición de test.

En conclusión, lo mas importante para nuestro problema ha sido con diferencia el juego probado. A modo de ejemplo, a continuación se ve el gráfico de la permutation_importance del modelo que finalmente hemos escogido (stacking):

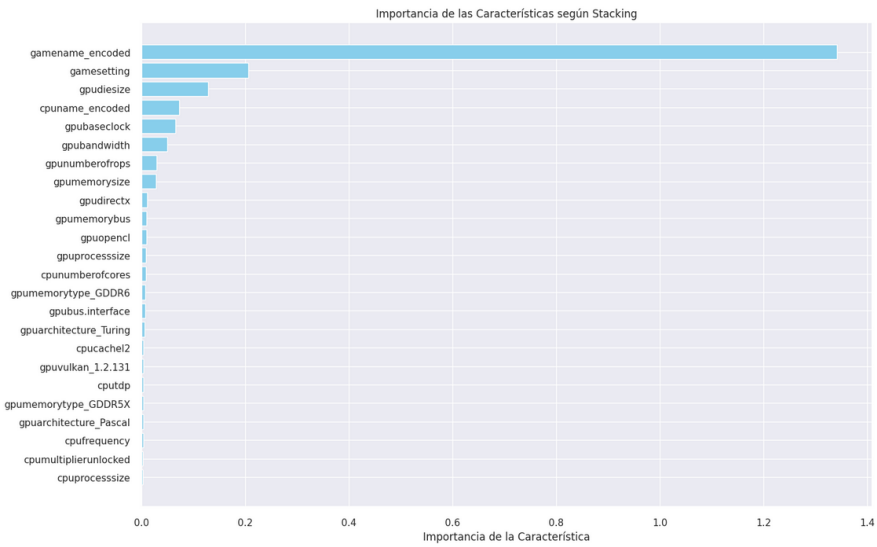


Figura 3: Interpretabilidad del modelo escogido

9. Conclusiones

9.1. Autoevaluación de éxitos y fracasos

Podemos decir que hemos logrado con éxito nuestros objetivos. Hemos sido capaces de obtener modelos muy buenos para predecir FPS, que generalizan muy bien con datos nuevos y tienen un margen de error muy pequeño. Aun así, nada hubiera sido posible sin un buen preprocesado de los datos, por lo que también estamos muy satisfechos del trabajo realizado en esa parte del proyecto. Por otro lado, debemos mencionar algo importante: un prerequisito del proyecto era que el problema a resolver sea lo suficientemente complejo como para que no se obtenga un acierto o R^2 casi perfectos usando una regresión lineal, y aparentemente nuestro problema parecía muy complejo con un preprocesado inicial muy básico, ya que obteníamos resultados pésimos con dicha regresión. Sin embargo, al realizar un preprocesado mas trabajado hemos observado que el problema era mas sencillo (en el sentido de la facilidad de encontrar buenos modelos) de lo que pensábamos. Evidentemente, cambiar de problema no era una opción viable después del tiempo dedicado al preprocesado y la falta de tiempo. Así que hemos aprendido algo importante: el preprocesado es tan o mas importante que el entrenamiento de los modelos y que la calidad y la cantidad de los datos.

9.2. Conclusiones científicas y personales

En conclusión, hemos visto como el videojuego en si tiene un peso muy importante a la hora de predecir los FPS que se obtendrían, de hecho es el atributo mas importante en todos los modelos. Eso nos hace pensar que las personas que programan videojuegos deben ser muy cuidadosos con las decisiones que tomen, ya que código poco optimizado podría llevar a problemas con los FPS, y en consecuencia empeoraría la experiencia de juego (como por ejemplo ARK Survival Evolved). Por otro lado, a nivel personal consideramos que hemos consolidado muchos de los conocimientos de aprendizaje automático aprendidos durante el curso.

Una posible extensión del proyecto seria tratar de aumentar el número de videojuegos que tienen en cuenta los modelos, pero eso implicaría obtener muchos mas datos sobre ejecuciones en otros juegos. Al final, los datos son el cuello de botella del aprendizaje automático. Por otro lado, hemos visto que el juego tiene un peso muy importante en la predicción de los FPS, por lo que tratar de tener un modelo donde no se especifique el juego seria una mala idea.

9.3. Experimentos futuros

En el preprocesado elaboramos un conjunto de datasets adelantándonos a posibles errores/problemas futuros, pero como hemos visto que el preprocesado principal ha permitido con creces predecir el objetivo en un tiempo computacional razonable, ya no hemos realizado pruebas con los demás datasets. En próximos experimentos se podría realizar un análisis de como de efectivo ha resultado el preprocesado com-

pleto, entrenando modelos con datasets con un nivel de preprocesado más bajo y comparar resultados. También nos hubiese gustado evaluar el rendimiento con CatBoost, dado que es un algoritmo muy usado en competiciones de Kaggle que obtiene resultados muy buenos en ciertos dominios.

Referencias

[1] Ulrik Thyge Pedersen (2023) *FPS Benchmark Dataset*, Kaggle. Disponible en: https://www.kaggle.com/datasets/ulrikthygepedersen/fps-benchmark

[2] NVIDIA Corporation (2024) *NVIDIA DLSS: Deep Learning Super Sampling*, NVIDIA GeForce. Disponible en: https://www.nvidia.com/es-es/geforce/technologies/dlss/

[3] Intel Corporation (2024) *Intel XeSS: AI-Enhanced Upscaling Technology*, Intel Arc. Disponible en: https://www.intel.la/content/www/xl/es/products/docs/discrete-gpus/arc/technology/xess.html

[4] Pedregosa, F. et al. (2011) *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research. Disponible en: https://scikit-learn.org/

[5] Chen, T. y Guestrin, C. (2016) *XGBoost: A Scalable Tree Boosting System*. Disponible en: https://xgboost.ai/

[6] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[7] Leslie Lamport (1994) *LATEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.