Deep Learning

A01703572

Luis Arturo Rendón Iñarritu

14 – Noviembre - 2024

1. Datos

Para esta actividad se eligió el dataset de “Game recommendations on Steam” el cual trae 3 CSV.

Autor: Anton Kozyriev

Última actualización: Hace 3 meses

https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam

El primer CSV contiene la información de los juegos alojados en Steam (Plataforma de distribución de videojuegos digitales). Contiene el nombre, fecha de lanzamiento, id, compatibilidad con plataformas, etc. El segundo es el de usuarios el cual es el más pesado de todos con 2.02 GB y contiene las reviews de los juegos con columnas como horas jugadas, si los usuarios encuentran el review útil, etc. Y por último un csv con los usuarios que dejaron las reviews, CSV que no se utilizó para el trabajo en cuestión.

2. Experimento (Problemática)

Con la información antes mencionada y siendo yo un apasionado usuario de los videojuegos me llamó mucho la atención el dataset en cuestión. Como al comprar juegos, al menos en mi caso, siempre me fijo en las reviews y estas impactan mucho en si adquiero el juego o no decidí crear un modelo el cual pudiera predecir en que categoría cae un juego (Overwhelmingly Negative, Mostly Negative, Negative, Mixed, Positive, Mostly Positive y Overwhelmingly Positive).

3. Hardware

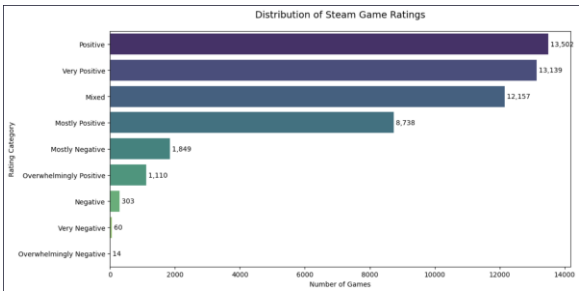
Empecé mi proyecto usando TensorFlow, seré breve, lo odiaré hasta el último de mis días por los conflictos de versiones, drivers, etc. Pytorch salvó mi vida y trabaja con mis drivers sin conflicto alguno.

4. ETL

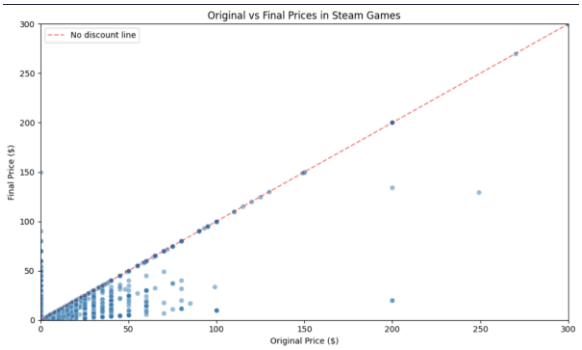
Es curioso que en el csv de juegos tengamos 50,872 registros, pero en el de usuarios 14,306,064 jugadores y en el de reviews 41,154,794 registros. Para el problema nada más usé el de reviews (obvio) y el de games para corroborar que cada review si corresponde a un juego dentro del registro de Seam.

Primero, se ve que casi todos los juegos caen del lado positivo, lo que me lleva a deducir que los usuarios se sienten

inclinados a dejar reviews cuando les gusta el juego, cuando no es mejor un refund o solamente no jugarlo.



Y con respecto a los precios de los juegos también se encontraron juegos con precios exorbitantemente altos.



Rondan los 300 dls. Por ejemplo, el juego Out of the Box:

app_id	2599300
title	🧠 OUT OF THE BOX
date_release	2023-10-24 00:00:00
win	True
mac	True
linux	True
rating	Very Positive
positive_ratio	100
user_reviews	7494460
price_final	299.99
price_original	299.99
discount	90.0
steam_deck	True
days_in_market	9986
avg_hours	487.0
median_hours	511.5
std_hours	495.120177
recommendation_ratio	1.0
avg_helpful	268.285714
avg_funny	266.285714

No solo muestra costar 300 dls, también aparece en los registros 3 veces con nombres diferentes. Mi conclusión es que los desarrolladores subieron el juego múltiples veces para hacer pruebas, esto pasa con 121 juegos. Al ser tan pocos registros se decidió removerlos.

De hecho, al comprobar el juego directo en Steam encontré solo uno con un precio de $150 pesos mexicanos.

Por último, juntamos ambos df en uno y definimos las featurs que usará el modelo para predecir la calificación final.

5. Embeddings

Para los embeddings requeridos para el proyecto decidí usar el nombre del juego. Primero lo limpiamos y normalizamos bajando todo a minúsculas y quitando caracteres especiales.

Luego aplicamos un tokenizador bastante simple con 10,000 palabras y <OOV> para las palabras desconocidas. Cuenta las palabras más comunes y le da un identificados a cada palabra.

También para cada palabra de ser más corta que el padding establecido la rellenamos con 0 y si es más larga la cortamos.

6. Modelo 1

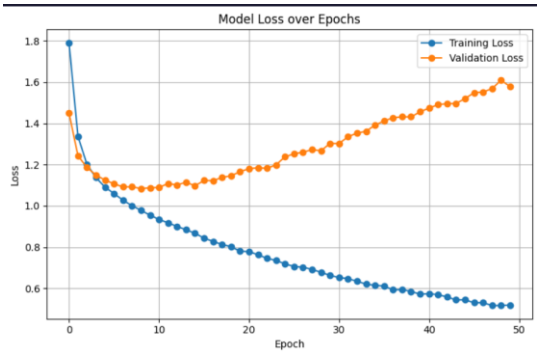
Para el primer modelo tomé el 60% de los datps para train, 20% para validation y 20% para test.

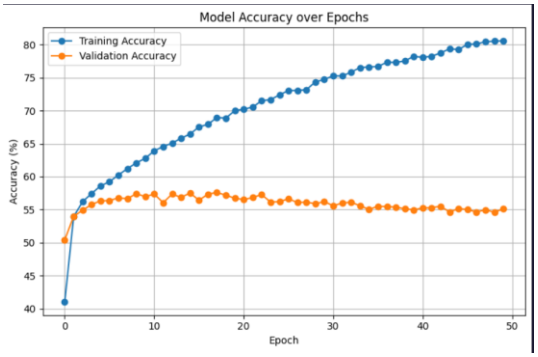
Usamos un batch size de 500 reviews y entrenó por 50 épocas sin parar.

Primero las características numéricas pasan por una capa que aprende patrones con los números y se normalizan para tener los valores en rangos manejables.

Ya después se combinan las capas en el modelo general, texto + números. Son ocho capas en total, usan ReLU para capturar relaciones no lineales y usan dropout para evitar overfitting. También normalizan los datos para aprender mejor.

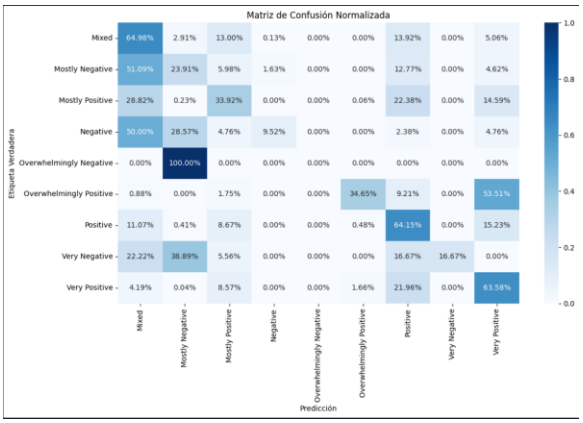
Ahora con respecto a los resultados del primer modelo:





Test Accuracy: 56.22%

Se ve claramente el OVERFITTING del modelo. El train llega a más del 80% de accuracy y con un loss menor a 0.5, pero la validation se queda con un accuracy del 55% y un loss que va bajando hasta 1.6.



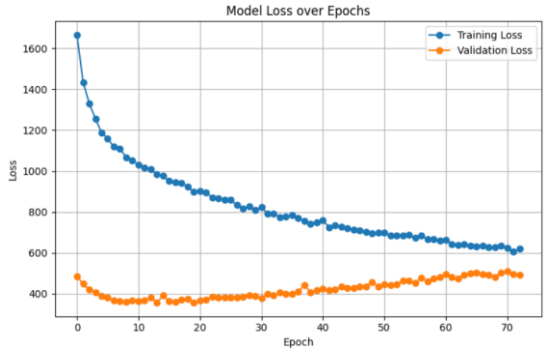
Ahora en base a la matriz de confusión se ve un conflicto en la identificación de casos negativos, esto se debería a la falta de ejemplos en el dataset. Por ejemplo ”overwhelmingly negative" solo aparece en 14 casos, llevando al modelo a desecharlo por completo y solo usar otras categorías como ”Mostly negative”, y dando como resultado un fracaso del 100% con esta categoría. Las categorías que tienen más presencia aparecen con más éxito como lo sería ”Very positive”, ”Positive” y ”Mixed”. En estas 3 categorías en específico presenta un porcentaje de éxito superior al 60%.

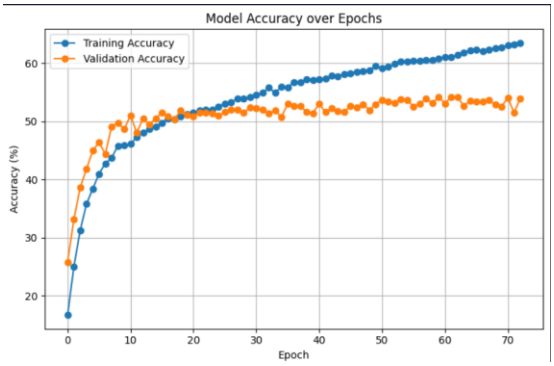
7. Segundo modelo

Para el segundo modelo decidí hacer bastantes cambios debido a varios descubrimientos con el primero.

a. Se hizo modelo más simple para que no cayera en overfitting tan rápido. Más que retirar capas simplifiqué las mismas, los embeddings usan una dimensión menor, las capas intermedias son más peque
b. Además de hacerlo más simple también le di un freno, tiene un contador el cual lleva las épocas en las que no encuentra mejoría y para el entrenamiento de llegar a un cierto número de épocas malas.
c. Bajo el batch size de 400 a 40 para que aprenda con más precisión debido al ruido extra y a la falta de datos por vuelta.
d. Metemos residual block para permitir que el gradiente fluya entre capas. De esta manera no llega de manera insignificante a las primeras. (Esto lo hice antes de volver el modelo más simple, la verdad con tan pocas capas no debe hacer mucha diferencia).
e. Aumenté las épocas a 400 para darle al tiempo que mejore todo lo posible y se detenga solo si las cosas van mal.

Resultados

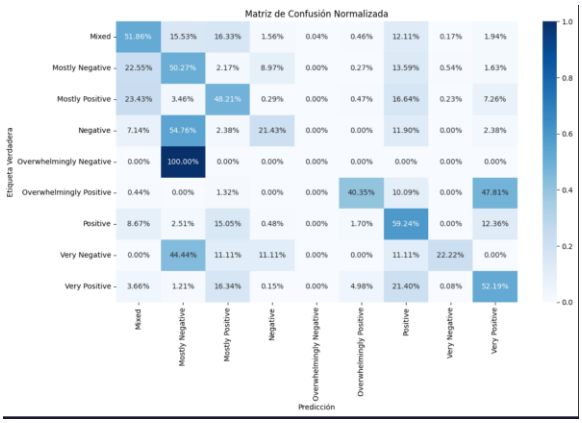




Test Accuracy: 54.90%

Se puede ver una mejoría considerable durante el entrenamiento, no me refiero obtener mejores predicciones, si no que el train, validation y test se mantienen muy cercanos durante el entrenamiento incluso por más épocas que el primer modelo.

A pesar de darle 400 épocas siempre se detiene cerca de la 70 debido a que empieza a sobreajustar.



Ahora basándonos en la matriz de confusión del segundo modelo podemos ver que es sumamente parecida a la original. Al final pudimos evitar el sobreajuste, sin embargo la falta de datos en las categorías negativas, especialmente en la peor de todas, tiende a apostarle a ”Mostly negative”. Para evitar estos conflictos habría que considerar eliminar las categorías que no presentan relevancia como ”Overwhelmingly negative”.

8. Conclusiones

Conseguí mejorar el aspecto del overfitting entre los modelos, incluso con más épocas usadas en el segundo no presenta un sobreajuste excesito y cuando este empieza se detiene solo.

Después de pelearme un rato descubrí la causa de que el modelo no suba más allá del 50%-60% (lo cual me hizo sentir sumamente menso). Tengo 13,502

juegos con reviews positivas, 13,139 con reviews muy positivas, etc. Y las negativas rondan entre las 1,849 y las 14 .___. Esta distribución de reviews está causando que el modelo aprenda y se vaya mucho más por el lado de darle a los juegos reviews positivas. Cuando un juego debería de ser negativo o mayormente negativo aun así la apuesta a que será positivo es más segura. Digamos que me preguntan a mí que diga que tan bien calificado está un juego dependiendo vairas cosas, de saber que casi todo está en positivo por todo Steam también le apostaría a que la gente le dio positivo.

Mi descubrimiento más grande es este: o casi todos los juegos de Steam son realmente buenos, o los usuarios no quieren dar reviews malos por la cuestión que sea.

9. Anexo

Después de platicar con el profesor, me comentó que la plataforma de Steam filtra los juegos para tener cierto estándar de calidad. De esta manera se explica que casi todos los datos tengan tendencias positivas, además de que los mismos desarrolladores pueden dar de baja el juego en caso de caer bajo críticas extremadamente malas.