

Práctica 3:

Modelos

neuronales para

minería de

opinión

Equipo: 1

Miembros: Yijun Wang, Gonzalo García

Contenido

EJERCICIO 1: Clasificación de reseñas con embeddings estáticos y redes neuronales	3
1. Embeddings de documento	3
2. Clasificación con red neuronal feedforward (FNN)	4
2.1. Arquitectura	4
2.2. Búsqueda de hiperparámetros	4
2.3. Entrenamiento final y evaluación	5
2. Análisis de resultados de clasificación – Modelo RNN/LSTM.....	6
2.1. Preparación de secuencias.....	6
2.2. Arquitectura del modelo LSTM.....	7
2.3. Resultados de clasificación	7
EJERCICIO 2: Clasificación con fine-tuning de BERT	8
1. Preparación de los datos.....	8
2. Tokenización y representación de entrada	9
3. Arquitectura del modelo.....	9
4. Configuración de entrenamiento	10
5. Evolución del entrenamiento	10
6. Resultados finales en test.....	10
Informe de clasificación.....	11
Matriz de confusión	11
Análisis de resultados.....	11
7. Conclusión del ejercicio.....	11
EJERCICIO 3. Generación de resumen de opinión sobre las reseñas de un juego	12

EJERCICIO 1: Clasificación de reseñas con embeddings estáticos y redes neuronales

1. Embeddings de documento

Para el **Ejercicio 1** se utilizaron **embeddings estáticos de palabra** preentrenados con el modelo **Word2Vec Google News** de 300 dimensiones, cargado mediante la librería *gensim* (GoogleNews-vectors-negative300.bin.gz).

A partir del corpus equilibrado de reseñas (corpus_features_balanced.json), donde cada reseña está representada como una lista de oraciones tokenizadas (tokens_by_sentence), se construyeron los embeddings de documento en dos pasos:

1. Embedding de oración

Para cada oración se calcularon los vectores de todas las palabras presentes en el vocabulario del modelo Word2Vec y se obtuvo su **media aritmética**.

- Si una palabra no estaba en el vocabulario, se ignoraba.
- Si una oración no contenía ninguna palabra reconocida, se asignó un vector nulo de dimensión 300.

2. Embedding de reseña (documento)

Para cada reseña se agregaron los embeddings de todas sus oraciones, de nuevo mediante la **media aritmética**, obteniendo así un único vector denso de dimensión 300 por reseña.

- Si ninguna oración de la reseña generaba un embedding válido, se asignó un vector nulo de 300 dimensiones.

Con este procedimiento se obtuvo una matriz de características X de tamaño **(4464, 300)**, es decir, 4464 reseñas representadas en un espacio de 300 dimensiones, y un vector de etiquetas y de longitud 4464. Posteriormente se realizó una partición **estratificada** en entrenamiento y prueba usando `train_test_split` con `test_size = 0.2` y `random_state = 42`, resultando:

- X_{train} : **(3571, 300)**
- X_{test} : **(893, 300)**

Las etiquetas de opinión se codificaron numéricamente como:

- $0 \rightarrow \text{negative}$
- $1 \rightarrow \text{neutral}$
- $2 \rightarrow \text{positive}$

La distribución de clases en el conjunto de entrenamiento quedó prácticamente balanceada, con **1191 reseñas negativas, 1190 neutras y 1190 positivas**.

2. Clasificación con red neuronal feedforward (FNN)

Para la clasificación de polaridad a partir de los embeddings de documento se diseñó una **red neuronal feedforward (FNN)** implementada en **PyTorch**, tomando los vectores de 300 dimensiones como entrada.

2.1. Arquitectura

La arquitectura final seleccionada fue la siguiente:

- Capa de entrada:
 - Dimensión de entrada: **300**.
- Capa oculta 1:
 - Capa lineal `Linear(300, 64)`
 - Activación **ReLU**
 - Capa de *dropout* con probabilidad `drop1 = 0.0888` (aprox.).
- Capa oculta 2:
 - Capa lineal `Linear(64, 64)`
 - Activación **ReLU**
 - Capa de *dropout* con probabilidad `drop2 = 0.1403` (aprox.).
- Capa oculta 3:
 - Capa lineal `Linear(64, 64)`
 - Activación **ReLU**.
- Capa de salida:
 - Capa lineal `Linear(64, 3)` que produce los **logits** para las tres clases (*negative, neutral, positive*).
 - No se aplica *softmax* explícito, ya que la función de pérdida `CrossEntropyLoss` lo incorpora internamente.

Todas las capas se agrupan en un módulo `nn.Sequential` dentro de la clase `Classifier`.

2.2. Búsqueda de hiperparámetros

Se utilizó **HyperOpt** (búsqueda bayesiana TPE) para ajustar algunos hiperparámetros clave:

- **Tasa de aprendizaje (lr):** distribución log-uniforme en el rango $[e^{-9}, e^{-3}]$.
- **Tamaño de la capa oculta (hidden_dim):** valores candidatos $\{64, 128, 256, 300\}$.
- **Dropout 1 (drop1):** uniforme en $[0.0, 0.5]$.
- **Dropout 2 (drop2):** uniforme en $[0.0, 0.5]$.

Para cada combinación propuesta, se entrenó el modelo durante **40 épocas** sobre el conjunto de entrenamiento completo, usando:

- Optimizador **AdamW**.
- Función de pérdida **CrossEntropyLoss**.

El criterio de selección fue la **pérdida en el conjunto de prueba** (`val_loss`). Tras 20 evaluaciones, la mejor configuración encontrada fue:

- `hidden_dim = 64`
- `drop1 ≈ 0.0888`
- `drop2 ≈ 0.1403`
- `lr $\approx 3.80 \cdot 10^{-3}$`

```
Best hyperparameters found:
{'drop1': np.float64(0.08880583923696656), 'drop2': np.float64(0.1402821366795367), 'hidden_dim': np.int64(64), 'lr': np.float64(0.0037972717108499822)}
```

2.3. Entrenamiento final y evaluación

Con estos hiperparámetros se entrenó el modelo final durante **80 épocas** sobre el conjunto de entrenamiento. Al finalizar, se evaluó en el conjunto de prueba (`X_test_t`, `y_test_t`) obteniendo las predicciones mediante `argmax` sobre los logits y calculando:

- **Accuracy en test**
- **F1 ponderado (weighted)**
- **Informe de clasificación** por clase (precisión, *recall*, F1)
- **Matriz de confusión** para las tres clases.

El modelo feedforward final entrenado con embeddings estáticos Word2Vec alcanzó un rendimiento moderado en el conjunto de prueba. Concretamente, se obtuvo:

- **Accuracy en test:** 0.6327
- **F1 ponderado (weighted):** 0.6288

```
... Accuracy: 0.6326987681970885
    F1 weighted: 0.6287812135015587
```

Estas cifras muestran que el modelo es capaz de generalizar parcialmente a ejemplos no vistos, aunque con margen de mejora respecto a modelos más complejos que se evaluarán posteriormente (LSTM y BERT).

A continuación se analizan las métricas por clase:

Classification report:				
	precision	recall	f1-score	support
negative	0.66	0.69	0.68	297
neutral	0.52	0.46	0.49	298
positive	0.69	0.74	0.72	298
accuracy			0.63	893
macro avg	0.63	0.63	0.63	893
weighted avg	0.63	0.63	0.63	893

Las clases **negative** y **positive** muestran un rendimiento claramente superior a la clase **neutral**, algo habitual en tareas de análisis de sentimientos: la clase neutral tiende a presentar fronteras difusas y menor coherencia semántica, lo que dificulta su separación lineal o poco profunda en espacios de embedding promedio.

Matriz de confusión

```
Confusion matrix:  
[[206  70  21]  
 [ 84 138  76]  
 [ 20  57 221]]
```

- La clase **positive** es la mejor identificada (221 aciertos, recall 0.74).
- La clase **negative** también se reconoce de forma razonable (206 aciertos, recall 0.69).
- La clase **neutral** presenta la mayor confusión, especialmente con positive y negative, confirmando su menor separabilidad con este tipo de representación.

Interpretación

Estos resultados indican que los embeddings de documento generados mediante promedios de Word2Vec aportan una información semántica útil, pero insuficiente para capturar matices más sutiles de opinión. La arquitectura FNN, al no modelar estructura secuencial ni dependencias contextuales, encuentra dificultades especialmente en la detección de neutralidad.

En comparación con los algoritmos clásicos de la Práctica 1 (como SVM o Logistic Regression), se espera que el rendimiento de esta FNN sea similar o ligeramente superior, pero aún lejos del obtenido mediante modelos más avanzados basados en transformadores (*fine-tuning* de BERT).

2. Análisis de resultados de clasificación – Modelo RNN/LSTM

Para evaluar una arquitectura capaz de modelar dependencias secuenciales, se entrenó una **red LSTM bidireccional** utilizando como entrada secuencias de embeddings Word2Vec sin promediar. Cada reseña se transformó en una secuencia de hasta **50 tokens**, truncada o rellenada con ceros según fuera necesario. Cada token se representó mediante el embedding de 300 dimensiones del modelo Word2Vec Google News.

2.1. Preparación de secuencias

- Se extrajeron todos los tokens de la reseña a partir de `tokens_by_sentence`.
- Para cada token presente en el vocabulario de Word2Vec, se recuperó su embedding.
- Las secuencias se truncaron a **longitud máxima 50** o se rellenaron con vectores nulos en caso de menor longitud.
- El resultado final fue un tensor `X_lstm` de forma **(4464, 50, 300)**.

Posteriormente, se realizó una partición estratificada en entrenamiento y prueba:

- `X_lstm_train`: (3571, 50, 300)

- `X_lstm_test: (893, 50, 300)`

Las etiquetas se codificaron según:

0 = negative, 1 = neutral, 2 = positive.

2.2. Arquitectura del modelo LSTM

El modelo empleado fue:

- **LSTM bidireccional** (`bidirectional=True`)
- **1 capa LSTM** (`lstm_layers=1`)
- **Hidden size = 128**
- **Dropout = 0.2** (inactivo al tener una única capa)
- Concatenación de los estados finales *forward* y *backward* → vector de 256 dimensiones
- **Capa final lineal:** `Linear(256, 3)`
- La salida son **logits**, procesados con `CrossEntropyLoss`

El entrenamiento se realizó durante **12 épocas**, con:

- Optimizador: **AdamW**
- Tasa de aprendizaje: **$1e-3$**

La evolución del *loss* indicó una estabilidad moderada, descendiendo desde 1.10 hasta aproximadamente 1.05.

2.3. Resultados de clasificación

Los resultados obtenidos en el conjunto de prueba fueron:

- **Accuracy:** 0.5162
- **F1 ponderado (weighted):** 0.5143

```
Accuracy: 0.5162374020156775
F1 weighted: 0.5142908530352828
```

Informe de clasificación

Classification report:				
	precision	recall	f1-score	support
negative	0.52	0.49	0.51	297
neutral	0.47	0.44	0.46	298
positive	0.55	0.61	0.58	298
accuracy			0.52	893
macro avg	0.51	0.52	0.51	893
weighted avg	0.51	0.52	0.51	893

Matriz de confusión

```
Confusion matrix:  
[[146  81  70]  
 [ 86 132  80]  
 [ 48  67 183]]
```

Interpretación

El rendimiento de la LSTM fue **significativamente inferior** al obtenido por la FNN basada en embeddings promedio (accuracy ≈ 0.63). Las principales razones son:

1. **Las secuencias con embeddings estáticos suelen ser muy ruidosas**
El modelo recibe secuencias directas de Word2Vec sin integración contextual, lo cual dificulta que el LSTM capture dependencias útiles, especialmente con una longitud máxima reducida a 50 tokens.
2. **LSTM superficial** (solo una capa, sin mecanismos adicionales como atención).
Esto limita la capacidad de capturar patrones de sentimiento más complejos.
3. **Clase neutral problemática**
Igual que en el FNN, la clase neutral tiene recall y F1 bajos, reflejando que incluso con información secuencial el límite entre positivo y negativo no queda bien representado por Word2Vec.
4. **El LSTM no supera al FNN**
Esto es coherente con el hecho de que:
 - Los embeddings estáticos no contienen información contextual.
 - La FNN trabaja sobre un embedding ya condensado y menos ruidoso (promedio de todas las oraciones).
 - El LSTM recibe secuencias largas con variabilidad alta, lo que complica su optimización.

En conjunto, la arquitectura LSTM no aportó una mejora respecto al FNN, y su rendimiento quedó en torno al **0.51–0.52 de F1**, destacando únicamente un mejor recall en la clase positiva.

EJERCICIO 2: Clasificación con fine-tuning de BERT

1. Preparación de los datos

Se utilizó el corpus equilibrado de reseñas (corpus_features_balanced.json), extrayendo para cada reseña el texto limpio (text_clean) o, en su defecto, el texto original (text_raw). El conjunto total contiene **4464 reseñas**, etiquetadas en la práctica 2 con tres clases de polaridad:

- *negative*
- *neutral*
- *positive*

Las etiquetas se codificaron numéricamente como:

- $0 \rightarrow \text{negative}$
- $1 \rightarrow \text{neutral}$
- $2 \rightarrow \text{positive}$

Posteriormente, se realizó una división **estratificada** del conjunto de datos:

- **Entrenamiento:** 3571 reseñas (80%)
- **Test:** 893 reseñas (20%)

De este modo se garantiza que la distribución de clases se mantiene constante entre ambos subconjuntos.

2. Tokenización y representación de entrada

Para la tokenización se utilizó el tokenizador oficial de **BERT base uncased** (bert-base-uncased). Cada reseña se transformó en una secuencia de tokens siguiendo el formato estándar del modelo:

- Se añadieron los tokens especiales [CLS] y [SEP].
- Se fijó una longitud máxima de secuencia $\text{MAX_LEN} = 128$.
- Las secuencias más largas se truncaron y las más cortas se rellenaron con *padding*.
- Se generó la correspondiente **máscara de atención** (attention_mask) para distinguir tokens reales de padding.

Cada ejemplo de entrada quedó representado por:

- input_ids
- attention_mask
- target (etiqueta de clase)

3. Arquitectura del modelo

El modelo empleado consta de dos partes:

1. **Encoder BERT**
 - Modelo base: bert-base-uncased.
 - Dimensión del embedding contextual: 768.
 - Se utiliza la salida pooler_output, asociada al token [CLS], como representación global de la reseña.
2. **Cabeza de clasificación**
 - Capa *Dropout* con probabilidad 0.3 para reducir sobreajuste.
 - Capa lineal final $\text{Linear}(768, 3)$ que proyecta a las tres clases de polaridad.

El modelo devuelve **logits**, sin aplicar *softmax*, ya que la función de pérdida lo gestiona internamente.

4. Configuración de entrenamiento

El entrenamiento se realizó ajustando **todos los parámetros de BERT** (fine-tuning completo), con la siguiente configuración:

- Número de épocas: **3**
- Tamaño de batch: **16**
- Optimizador: **AdamW**
- Tasa de aprendizaje: **$2 \cdot 10^{-5}$**
- Función de pérdida: **CrossEntropyLoss**
- Scheduler: **Linear learning rate scheduler** sin warm-up
- Clipping del gradiente: norma máxima 1.0

Durante el entrenamiento se monitorizó el rendimiento sobre el conjunto de validación (test) y se guardó el modelo con mejor accuracy.

5. Evolución del entrenamiento

La evolución del entrenamiento mostró una mejora progresiva y estable:

Época	Train accuracy	Train loss	Val accuracy	Val loss
1	0.555	0.8959	0.675	0.7016
2	0.750	0.5950	0.700	0.6642
3	0.841	0.4185	0.712	0.6773

Se observa que el modelo aprende rápidamente durante las primeras épocas, alcanzando un rendimiento estable sin signos claros de sobreajuste severo.

6. Resultados finales en test

Una vez cargado el mejor modelo entrenado, se evaluó sobre el conjunto de test completo, obteniendo los siguientes resultados:

- **Accuracy:** 0.7122
- **F1 macro:** 0.7125
- **F1 weighted:** 0.7125

```
• Accuracy: 0.7122060470324748
  F1 macro: 0.7125321994207185
  F1 weighted: 0.7124840209703995
```

Informe de clasificación

Classification report:				
	precision	recall	f1-score	support
negative	0.77	0.74	0.76	297
neutral	0.60	0.61	0.61	298
positive	0.77	0.78	0.78	298
accuracy			0.71	893
macro avg	0.71	0.71	0.71	893
weighted avg	0.71	0.71	0.71	893

Matriz de confusión

Confusion matrix:				
[[221	63	13]		
[59	182	57]		
[8	57	233]]		

Análisis de resultados

El modelo BERT obtiene el **mejor rendimiento de todos los modelos evaluados en la práctica**, superando tanto a la red FNN basada en promedios de Word2Vec como al modelo LSTM.

Los principales aspectos destacables son:

- Mejora clara en la **clase neutral**, tradicionalmente la más difícil de clasificar, gracias a la información contextual que aporta BERT.
- Alto rendimiento en las clases *positive* y *negative*, con F1 cercanos a 0.78.
- Reducción significativa de confusiones extremas entre clases opuestas (negative vs positive).

Estos resultados confirman la ventaja de los **embeddings contextualizados** frente a representaciones estáticas, incluso con un número reducido de épocas de entrenamiento.

7. Conclusión del ejercicio

El ajuste fino de BERT demuestra ser una solución muy eficaz para la clasificación de sentimientos en reseñas largas y con lenguaje natural complejo. A pesar del mayor coste computacional, el incremento de rendimiento justifica su uso frente a modelos neuronales más simples, cumpliendo plenamente los objetivos planteados en el ejercicio 2 de la práctica.

EJERCICIO 3. Generación de resumen de opinión sobre las reseñas de un juego

Para el último ejercicio de la práctica hemos decidido abordar la tarea **a)** de minería de opinión, cuyo objetivo es obtener una opinión general o un resumen a través de *prompting* que recoja los puntos principales de las críticas y, si es el caso, las opiniones enfrentadas sobre un juego en particular, a partir de un conjunto de sus reseñas.

Para conseguir dicho objetivo, usaremos como LLM el modelo de Gemini disponible en la versión gratis de su API, ***gemini-2.5-flash***. La estructura de *prompting* que vamos a emplear es la siguiente: en primer lugar, definimos en el *prompt* el formato de objeto que recibe el modelo, en este caso, un JSON que contiene múltiples reseñas para múltiples juegos distintos, y que debe escoger aquellas que tienen el identificador que hemos seleccionado. Después redactamos en el *prompt* que, de las reseñas del juego dado, debe seleccionar hasta 5 características positivas y hasta 5 características negativas, y expresar dichas características en forma de *bullet points*. Además, el programa debe detectar si hay opiniones enfrentadas y una vez que ha recogido toda la información determinar si el sentimiento general del juego es positivo, neutro o negativo.

Para que el *prompt* refleje todos los requisitos que necesitamos del modelo, seguimos la siguiente estructura:

1. Formato de los datos de entrada e información general
2. Tareas para realizar por parte del agente LLM.
3. Formato de salida de la información que necesitamos.

Un ejemplo de *prompt* es el siguiente:

"""

You are given a JSON object containing multiple reviews for a game (game_id: {game_id}).

Analyze all reviews and produce a concise summary.

Tasks:

- 1) List up to 5 common positive points (bullet list)
- 2) List up to 5 common negative points (bullet list)
- 3) Contrary opinions between positive points and negative points (if any)
- 4) Overall sentiment: Positive, Neutral, or Negative

Output in plain text:

Game: {game_id}

Summary:

- Positive: ...

- Negative: ...

- Contrary Opinions: ...

Overall Sentiment: ...

JSON reviews:

{reviews_str}

""

La salida del *prompt* es la siguiente:

Game: 16398 Summary:

- Positive:
 - Can be fun or enjoyable for very young children.
 - Helps children learn numbers, card values, and basic gaming concepts (like taking turns).
 - Uses a standard deck of cards, making it widely accessible and free.
 - Evokes nostalgia and childhood memories for some players.
 - The "war" mechanic (tie-breakers) can create moments of excitement or anticipation for kids.
- Negative:
 - Critically lacks player agency, strategy, or meaningful decisions, being almost entirely luck-driven or predetermined.
 - Games are frequently too long or of indeterminate length, often feeling endless and tedious.
 - Widely considered boring, dull, and pointless by adult or experienced players.
 - Many reviewers argue it doesn't qualify as a "real game" due to the absence of player choice.
 - Not enjoyable or suitable for adults, often described as a "waste of time" for older players.
- Contrary Opinions:
 - **Fun vs. Boring:** Many reviewers consider the game fun for very young children or for nostalgic reasons, while adults overwhelmingly describe it as boring, tedious, or even "torturous."
 - **Length:** Some find the extended "war" sequences (ties) exciting, while the majority view the indeterminate and excessive game length as a significant negative point that contributes to its tediousness.
 - **Is it a game?:** Some defend it as a basic game or valuable learning activity for children, but a strong contingent argues it fails to meet the definition of a game due to the absolute lack of player decisions. Overall Sentiment: Negative

Overall Sentiment: Negative

Para comprobar si la salida del *prompt* era cierta y válida, hemos comprobado un gran número de reseñas del juego 16398. Estas son algunas de las reseñas que confirman que la salida del *prompt* es acertada. En cuanto a puntos negativos:

```
"text_raw": "Actually pretty fun. Lasts too long",
```

```
-Indeterminate length, a game can end in 5 minutes or an hour.",
```

```
"text_clean": "Complete luck",
```

```
"text_clean": "It's a coinflip but with cards
```

```
but in the end its just a 50 50 if you win or lose.
```

```
no thought to the actual fun minimal or challenge nonexistent it plays itself
```

En cuanto a puntos positivos:

```
more often than not it is a fun way to spend some time with friends or family.
```

```
"Got a dinosaur theme deck to play with my daughter, we play some variants."  
"Is some fun to be had as she s not quite ready for mtg yet."
```

```
Grew up playing this (one of the earliest games and few card games I played when young).  
wine3".
```

El agente LLM le ha dado a este juego una valoración general **negativa**. La puntuación media del juego es de **2.4**, por lo tanto, el agente está en lo cierto.

En resumen, trabajar con un *prompt* con la estructura definida anteriormente arroja buenos resultados sobre los juegos guardados. Si se necesitasen entradas o salidas distintas, o incluso tareas distintas, bastaría con cambiar la parte del *prompt* correspondiente siguiendo las mismas normas que hemos seguido para formar el *prompt* que son: que el todo el *prompt* sea claro, que sea breve y que facilite al agente LLM la mayor cantidad de información posible para evitar posibles invenciones.