PRÁCTICA 2: Clasificación de texto para análisis de sentimiento

Equipo: 1

Miembros: Yijun Wang, Gonzalo García

Contenido

EJERCICIO 1

1.1. Preparación del corpus y mejoras respecto a la Práctica 1

El punto de partida fue el archivo corpus_total.json, generado previamente. La primera modificación importante consistió en filtrar de nuevo todas las reseñas para asegurarnos de que solo trabajábamos con textos completos y válidos. En la Práctica 1 podían quedar reseñas vacías o con contenido insuficiente; ahora era fundamental eliminarlas antes de extraer las características.

Además de esta limpieza, incorporamos información adicional basada en el rating del usuario. Definimos umbrales fijos para clasificar cada reseña como positiva, neutra o negativa (≥7, 4–6 y < 4 respectivamente). Esta clasificación por rating se utilizaría más tarde para equilibrar el corpus y preparar los conjuntos supervisados.

1.2 Extracción de características lingüísticas

Una vez garantizada la validez del texto, aplicamos un análisis lingüístico detallado dividido en varias capas. Este conjunto de características supone una mejora sustancial respecto a la práctica anterior, donde solo disponíamos del texto limpio, sin ningún tipo de anotación lingüística.

1.2.1 Polaridad léxica y análisis con VADER

Como primer nivel de análisis empleamos VADER, que proporciona puntuaciones de positividad, negatividad, neutralidad y un valor compuesto (compound) que resume la polaridad general del texto. Además, accedimos al léxico interno de VADER para evaluar palabra por palabra, contabilizando cuántos términos positivos o negativos aparecen y sumando su polaridad. Esta información granular permite distinguir entre reseñas ligeramente positivas y otras muy marcadas.

1.2.2 Análisis morfosintáctico (PoS tagging)

Para capturar la estructura del lenguaje usamos *pos_tag* de NLTK, que nos permite identificar adjetivos, adverbios y verbos dentro del texto. Este paso tuvo un aspecto técnico relevante: fue necesario instalar explícitamente recursos como *averaged_perceptron_tagger*, *punkt* y *punkt_tab* para evitar los errores originales de NLTK. Una vez resuelto, pudimos obtener:

- número de tokens,
- número de oraciones,
- número de adjetivos (muy vinculados a la opinión),
- número de adverbios (incluyendo intensificadores),
- número de verbos (útiles para detectar verbos subjetivos como *love* o *hate*).

1.2.3 Negaciones, intensificadores y términos de dominio

Las negaciones e intensificadores tienen un impacto directo en el sentimiento expresado, por lo que las detectamos mediante patrones regulares cuidadosamente diseñados. Expresiones como “not good”, “barely playable” o “extremely fun” modifican profundamente la interpretación emocional del texto.

También añadimos una categoría específica para el dominio de juegos de mesa. Incluimos vocabulario como *components*, *dice*, *rulebook*, *mechanics* o *replayability*. Estos términos son habituales en reseñas de BoardGameGeek y resultan informativos para los clasificadores.

Toda esta información se añadió a cada reseña y se almacenó en *corpus_features_only_text.json.*

1.3. Balanceo del corpus según las clases de sentimiento

Tras extraer todas las características comprobamos que la distribución original de reseñas no era uniforme: había muchas más valoraciones positivas que negativas. Este desequilibrio habría perjudicado la fase de entrenamiento de modelos supervisados, por lo que aplicamos un *undersampling* controlado.

```
Distribución ORIGINAL del corpus preprocesado:
 positive: 1488
 negative: 1904
 neutral: 2249

Se guardarán 1488 reseñas por clase.

Distribución BALANCEADA:
 positive: 1488
 neutral: 1488
 negative: 1488
```

Partiendo de la clasificación por rating, calculamos cuántas reseñas pertenecían a cada clase y tomamos como referencia la clase minoritaria. De esta forma, mantuvimos el mismo número de reseñas positivas, neutras y negativas, garantizando un conjunto de datos equilibrado. El corpus final se guardó como *corpus_features_balanced.json*.

1.4. Resultado final del ejercicio

El ejercicio 1 generó un corpus plenamente preparado para representar lingüísticamente las reseñas. A diferencia de la Práctica 1, donde solo disponíamos de texto limpio, ahora cada reseña incluye:

- información estructural del texto,

- análisis morfosintáctico,
- polaridad léxica global y palabra a palabra,
- detección de negaciones e intensificadores,
- términos propios del dominio BGG,
- y una clasificación inicial según el rating, utilizada para balancear el corpus.

El resultado es una representación mucho más rica, que sirve como base sólida para las representaciones vectoriales y los clasificadores de los ejercicios posteriores.

EJERCICIO 2

Generación de representaciones vectoriales con TF-IDF y características lingüísticas

Tras completar el enriquecimiento lingüístico en el ejercicio anterior, el siguiente paso consistió en transformar todo ese contenido—tanto el texto como las nuevas características numéricas—en representaciones vectoriales adecuadas para la construcción de clasificadores. La idea central de este ejercicio es combinar dos enfoques diferentes:

1. una representación basada en el contenido textual (*TF-IDF*),
2. una representación basada en las características lingüísticas extraídas en el Ejercicio 1.

El resultado final son dos matrices dispersas listas para ser utilizadas durante el entrenamiento de los modelos supervisados.

2.1 Preparación y lematización del corpus balanceado

El punto de partida fue el archivo *corpus_features_balanced.json*, generado al final del ejercicio anterior. El balanceo fue especialmente importante aquí, ya que garantiza que las clases positiva, neutra y negativa están representadas por la misma cantidad de reseñas, evitando que TF-IDF aprenda un vocabulario sesgado hacia una clase dominante (lo que habría ocurrido si hubiéramos usado el corpus original).

Antes de generar la matriz TF-IDF aplicamos una **lematización ligera**. Se trata de una transformación que reduce palabras flexionadas a su forma básica (por ejemplo: *plays* → *play*, *better* → *good*). Aunque no realizamos una lematización compleja basada en PoS, la versión ligera es suficiente para homogeneizar las frecuencias y reducir ruido léxico.

Este proceso también requirió descargar recursos como *wordnet* y *omw-1.4* para que el lematizador funcionase adecuadamente.

```python
lemmatizer = WordNetLemmatizer()


# ------------------------------------------------------------
# Carga de datos
# ------------------------------------------------------------
print("Cargando corpus balanceado...")
with open(INPUT_FILE, "r", encoding="utf-8") as f:
    data = json.load(f)

texts = []
ling_feats = []

for d in data:
    text = (d.get("text_clean") or d.get("text_raw") or "").strip()
    if not text:
        continue

    # ---------- Lematización ligera ----------
    tokens = nltk.word_tokenize(text.lower())
    lemmas = [lemmatizer.lemmatize(t) for t in tokens]
    text_lemmatized = " ".join(lemmas)

    texts.append(text_lemmatized)
```

2.2. Representación TF-IDF optimizada

Para la representación basada en contenido textual utilizamos TfidfVectorizer, configurado para maximizar la información y reducir el ruido. En lugar de usar una configuración básica, lo ajustamos teniendo en cuenta los requisitos de la práctica y la eficacia en clasificación:

- **Eliminación de stopwords en inglés**
  Esto reduce miles de términos irrelevantes (“the”, “is”, “and”) y mejora la calidad del vocabulario.
- **Uso de unigramas y bigramas**
  Los unigramas capturan significado básico, mientras que los bigramas aportan contexto adicional (“very good”, “poor design”, “dice rolling”).
  Esta combinación suele mejorar sensiblemente los resultados.
- **min_df = 3**
  Filtra términos extremadamente raros que solo añaden ruido.
- **max_df = 0.80**
  Elimina palabras demasiado frecuentes, que no aportan información discriminativa.
- **max_features = 12 000**
  Controla la dimensionalidad y evita matrices enormes difíciles de trabajar.

En conjunto, esta configuración genera una matriz TF-IDF mucho más estable que una versión sin filtrados. La matriz final tiene tantas filas como reseñas y miles de columnas representando términos o bigramas.

```
TF-IDF: 4464 reseñas × 4154 términos
Features lingüísticas: (4464, 15)
```

2.3 Representación de características lingüísticas

Además del contenido textual, exportamos las características lingüísticas producidas en el Ejercicio 1. Para cada reseña se generó un vector con atributos como:

- proporciones de polaridad (pos/neg/neu),
- compound score,
- número de adjetivos, adverbios y verbos,
- número de palabras positivas y negativas,
- suma total de polaridad léxica,
- número de negaciones e intensificadores,
- número de términos propios del dominio BGG.

Estas características se almacenaron en una matriz densa de NumPy y posteriormente se transformaron a formato disperso para facilitar su uso con modelos basados en sparse matrices.

Un punto importante de este ejercicio es que **mantenemos ambas representaciones separadas**, porque más adelante (Ejercicio 4 y 5) compararemos:

- modelos entrenados solo con TF-IDF,
- solo con características lingüísticas,
- y una representación combinada TF-IDF + características.

2.4. Guardado de representaciones y serialización del vectorizador

Por último, guardamos:

- **X_tfidf.npz** → matriz TF-IDF
- **X_feats.npz** → matriz de características lingüísticas
- **texts.json** → lista de textos lematizados
- **tfidf_vectorizer.pkl** → el vectorizador entrenado

Este último archivo es especialmente importante porque nos permitirá transformar nuevos textos con exactamente el mismo vocabulario y la misma configuración que usamos para entrenar los modelos.

```python
sparse.save_npz(OUTPUT_TFIDF, X_tfidf)
sparse.save_npz(OUTPUT_FEATS, X_feats_sparse)

with open(OUTPUT_TEXTS, "w", encoding="utf-8") as f:
    json.dump(texts, f, ensure_ascii=False, indent=2)

with open(OUTPUT_VECTORIZER, "wb") as f:
    pickle.dump(vectorizer, f)

print(f" Guardado TF-IDF: {OUTPUT_TFIDF}")
print(f"Guardado features: {OUTPUT_FEATS}")
print(f"Guardado textos: {OUTPUT_TEXTS}")
print(f"Guardado vectorizer: {OUTPUT_VECTORIZER}")
```

2.5. Resultado final del ejercicio

Tras este ejercicio, disponemos de dos tipos de representaciones del corpus:

1. **Una matriz TF-IDF optimizada**, que captura el contenido textual con n-gramas y un vocabulario depurado.
2. **Una matriz de características lingüísticas**, que resume propiedades estructurales y semánticas del texto de forma compacta.

Ambas representaciones complementan información distinta y permitirán evaluar, en los ejercicios posteriores, qué enfoque resulta más eficaz para la clasificación de sentimiento, así como comprobar el impacto real de las características lingüísticas sobre el rendimiento de los modelos.

EJERCICIO 3

Creación de los conjuntos de entrenamiento y test a partir de las representaciones vectoriales

Una vez generadas las dos representaciones vectoriales del corpus —la basada en TF-IDF y la basada en las características lingüísticas—, el objetivo de este ejercicio fue combinar ambas para producir los conjuntos de datos que utilizaríamos en la fase de clasificación. Esto incluye generar las etiquetas a partir del rating, unir las matrices dispersas y realizar una división estratificada en entrenamiento y test, siguiendo siempre los criterios de balanceo establecidos en el ejercicio anterior.

3.1 Generación de etiquetas a partir del rating

El corpus que recibimos como entrada (*corpus_features_balanced.json*) ya estaba equilibrado entre clases positiva, neutra y negativa, pues en el ejercicio 1 decidimos

aplicar *undersampling* para igualar el número de reseñas de cada categoría. Gracias a este equilibro previo, la tarea de etiquetar fue directa y más fiable.

Para convertir cada reseña en una clase definimos los siguientes umbrales, coherentes con el resto de la práctica:

- **rating ≥ 7 → positive**
- **rating entre 4 y 6 → neutral**
- **rating < 4 → negative**

Este etiquetado se realiza antes de cargar cualquier matriz vectorial. De esta manera nos aseguramos de que las etiquetas y las representaciones están perfectamente alineadas. El resultado es un array `y` con tantas etiquetas como reseñas en el corpus balanceado.

```
 Cargando corpus para generar etiquetas...
 Distribución de clases:
positive    1488
negative    1488
neutral     1488
Name: count, dtype: int64
```

3.2 Carga y combinación de representaciones vectoriales

El siguiente paso consistió en cargar las dos matrices creadas en el Ejercicio 2:

- `X_tfidf.npz` → contenido textual transformado mediante TF-IDF
- `X_feats.npz` → características lingüísticas del Ejercicio 1

Cada una de ellas contiene información distinta y complementaria. Para aprovechar ambas, construimos una única matriz final `X` mediante concatenación horizontal (`hstack`). Utilizamos el formato disperso CSR para que la matriz resultante siga siendo eficiente en memoria, ya que sumar TF-IDF y features lingüísticas puede generar miles de columnas.

Este proceso garantiza:

- alineación perfecta entre texto y características adicionales,
- compatibilidad con los algoritmos de clasificación posteriores,
- un formato ligero y eficiente para almacenarlo en disco.

```
 Cargando matrices TF-IDF y features lingüísticas...
✓ X_tfidf: (4464, 4154)
✓ X_feats: (4464, 15)

 Concatenando matrices...
✓ Matriz final X: (4464, 4169)
```

3.3 División del corpus en training y test

Con la matriz `X` y el vector `y` ya preparados, realizamos la división del corpus en dos subconjuntos:

- **80% para entrenamiento (X_train, y_train)**
- **20% para test (X_test, y_test)**

Lo más importante en esta fase fue garantizar una división **estratificada**, de modo que la proporción de clases (pos/neu/neg) se mantuviera idéntica en ambos subconjuntos. Esto evita que el conjunto de test se sesgue hacia una clase concreta y asegura una evaluación más robusta y justa de los modelos de clasificación.

La función `train_test_split` de scikit-learn nos permitió mantener el equilibrio especificando `stratify=y`.

El resultado final son cuatro ficheros perfectamente alineados y preparados para entrenamiento:

- X_train.npz
- X_test.npz
- y_train.npy
- y_test.npy

3.4. Resultado final del ejercicio

Tras este ejercicio, disponemos de todos los elementos necesarios para comenzar la construcción y evaluación de clasificadores:

- Las etiquetas generadas a partir del rating.
- Una única matriz que fusiona TF-IDF con las características lingüísticas.
- Un conjunto de entrenamiento suficientemente grande y equilibrado.
- Un conjunto de test independiente y estratificado para evaluar el rendimiento real de los modelos.

El corpus está ahora completamente preparado para la fase de clasificación supervisada del ejercicio 4.

EJERCICIO 4

Construcción y evaluación inicial de modelos de clasificación

En este ejercicio comenzamos por fin la fase de aprendizaje automático propiamente dicha. El objetivo era comparar diferentes algoritmos de clasificación supervisada utilizando tres representaciones del corpus:

1. únicamente TF-IDF,

2. únicamente las características lingüísticas,
3. una combinación de ambas.

Esto nos permitiría comprobar no solo qué modelo funciona mejor, sino también qué tipo de representación captura mejor la polaridad de las reseñas.

4.1. Preparación del conjunto de entrenamiento y test

El punto de partida fueron los cuatro archivos generados en el ejercicio anterior:

- X_train.npz y X_test.npz: matrices dispersas combinando TF-IDF + features
- y_train.npy y y_test.npy: etiquetas ya estratificadas

Una particularidad de nuestra configuración es que la matriz X de entrada contiene tanto TF-IDF como las características lingüísticas concatenadas. Por ello, en esta fase fue necesario volver a separar ambas partes para poder realizar los tres experimentos por separado.

Para saber qué columnas correspondían a cada representación, cargamos las matrices originales X_tfidf.npz y X_feats.npz y tomamos sus dimensiones exactas. Con ello pudimos dividir X_train y X_test en:

- X_tfidf_train, X_tfidf_test
- X_feats_train_raw, X_feats_test_raw

```
✓ Separación de representaciones:
  - X_tfidf_train: (3571, 4154)
  - X_feats_train_raw: (3571, 15)
  - X_tfidf_test: (893, 4154)
  - X_feats_test_raw: (893, 15)

✓ Matrices combinadas:
  - X_comb_train: (3571, 4169)
  - X_comb_test:  (893, 4169)
```

4.2 Escalado de las características lingüísticas

Mientras que TF-IDF ya se encuentra en un rango adecuado para modelos como SVM o Naive Bayes, las características lingüísticas tienen escalas muy dispares: desde conteos de tokens hasta valores de polaridad. Para evitar que atributos con valores altos dominaran el aprendizaje de los modelos, aplicamos un escalado MinMax (0–1).

Tras el escalado, convertimos estas matrices nuevamente a formato disperso para que pudieran concatenarse eficientemente con TF-IDF y ser utilizadas por clasificadores que trabajan con sparse matrices.

Esta normalización fue fundamental para poder comparar de forma justa el rendimiento de los modelos utilizando únicamente características lingüísticas.

4.3 Modelos evaluados

Seleccionamos tres algoritmos muy utilizados en clasificación de texto, cada uno con propiedades diferentes:

• Multinomial Naive Bayes

Especialmente adecuado para conteo de palabras y TF-IDF. Suele ser rápido y sorprendentemente eficaz cuando los rasgos son independientes.

• LinearSVC (SVM lineal)

Probablemente el modelo más fuerte para clasificación basada en bag-of-words. Maneja muy bien datos dispersos y de alta dimensionalidad.

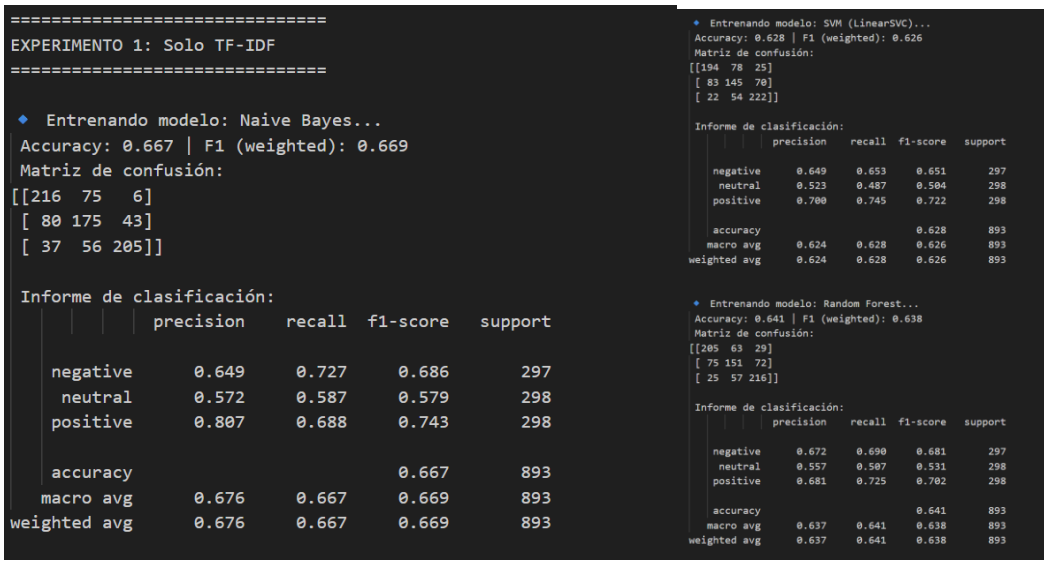
• Random Forest

Aunque no es el más eficiente para vectores muy dispersos, sirve como contraste para evaluar qué tal rinden clasificadores basados en árboles frente a modelos lineales.

4.4. Tres experimentos complementarios

4.4.1 Experimento 1: Solo TF-IDF

En este experimento evaluamos qué tal funcionan los modelos exclusivamente con la representación basada en contenido textual.
Esto sirve como línea base tradicional en PLN, ya que TF-IDF suele capturar gran parte del contexto semántico de una opinión.



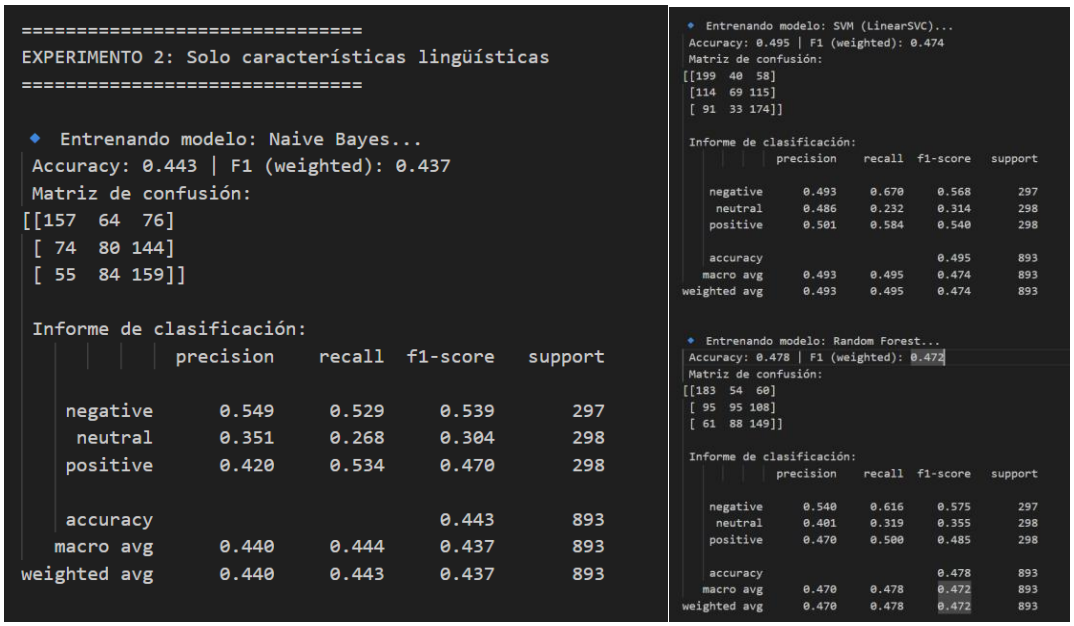
```
==============================
EXPERIMENTO 1: Solo TF-IDF
==============================

🔹 Entrenando modelo: Naive Bayes...
Accuracy: 0.667 | F1 (weighted): 0.669
Matriz de confusión:
[[216  75   6]
 [ 80 175  43]
 [ 37  56 205]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.649     0.727     0.686       297
     neutral      0.572     0.587     0.579       298
    positive      0.807     0.688     0.743       298

    accuracy                          0.667       893
   macro avg      0.676     0.667     0.669       893
weighted avg      0.676     0.667     0.669       893
```

```
🔹 Entrenando modelo: SVM (LinearSVC)...
Accuracy: 0.628 | F1 (weighted): 0.626
Matriz de confusión:
[[194  78  25]
 [ 83 145  70]
 [ 22  54 222]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.649     0.653     0.651       297
     neutral      0.523     0.487     0.504       298
    positive      0.700     0.745     0.722       298

    accuracy                          0.628       893
   macro avg      0.624     0.628     0.626       893
weighted avg      0.624     0.628     0.626       893
```

```
🔹 Entrenando modelo: Random Forest...
Accuracy: 0.641 | F1 (weighted): 0.638
Matriz de confusión:
[[205  63  29]
 [ 75 151  72]
 [ 25  57 216]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.672     0.690     0.681       297
     neutral      0.557     0.507     0.531       298
    positive      0.681     0.725     0.702       298

    accuracy                          0.641       893
   macro avg      0.637     0.641     0.638       893
weighted avg      0.637     0.641     0.638       893
```

4.4.2 Experimento 2: Solo características lingüísticas

Aquí probamos si las características diseñadas en el ejercicio 1 —negaciones, intensificadores, conteo de adjetivos, polaridad léxica, etc.— son suficientemente expresivas como para clasificar las reseñas sin necesidad del texto completo.
Este experimento es clave para justificar el valor añadido de estas características frente a una representación puramente basada en palabras.



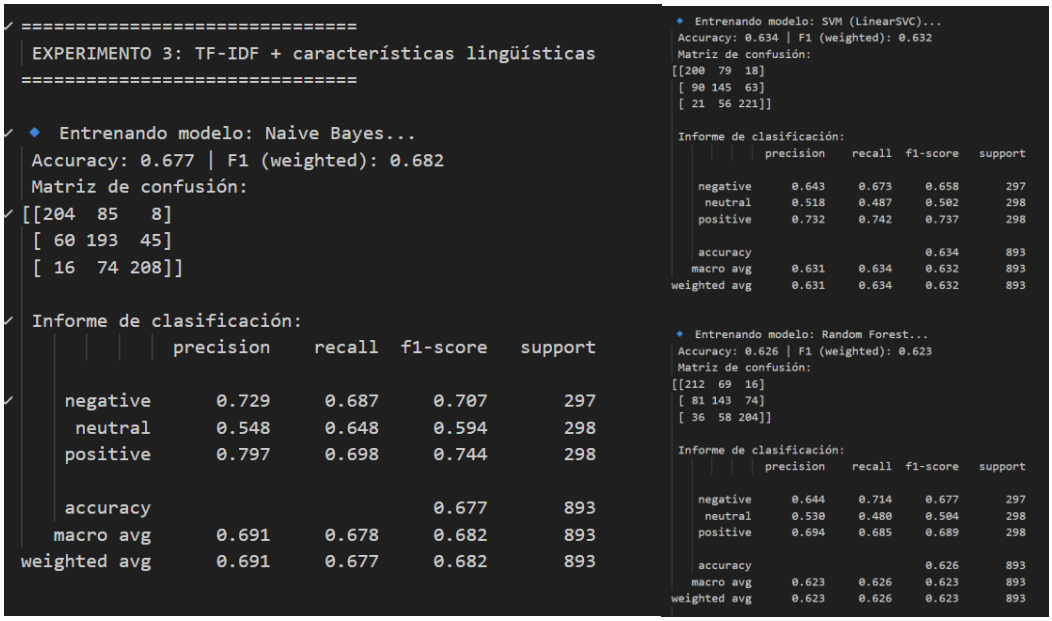
```
==============================
EXPERIMENTO 2: Solo características lingüísticas
==============================

🔹 Entrenando modelo: Naive Bayes...
Accuracy: 0.443 | F1 (weighted): 0.437
Matriz de confusión:
[[157  64  76]
 [ 74  80 144]
 [ 55  84 159]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.549     0.529     0.539       297
     neutral      0.351     0.268     0.304       298
    positive      0.420     0.534     0.470       298

    accuracy                          0.443       893
   macro avg      0.440     0.444     0.437       893
weighted avg      0.440     0.443     0.437       893
```

```
🔹 Entrenando modelo: SVM (LinearSVC)...
Accuracy: 0.495 | F1 (weighted): 0.474
Matriz de confusión:
[[199  40  58]
 [114  69 115]
 [ 91  33 174]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.493     0.670     0.568       297
     neutral      0.486     0.232     0.314       298
    positive      0.501     0.584     0.540       298

    accuracy                          0.495       893
   macro avg      0.493     0.495     0.474       893
weighted avg      0.493     0.495     0.474       893
```

```
🔹 Entrenando modelo: Random Forest...
Accuracy: 0.478 | F1 (weighted): 0.472
Matriz de confusión:
[[183  54  60]
 [ 95  95 108]
 [ 61  88 149]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.540     0.616     0.575       297
     neutral      0.401     0.319     0.355       298
    positive      0.470     0.500     0.485       298

    accuracy                          0.478       893
   macro avg      0.470     0.478     0.472       893
weighted avg      0.470     0.478     0.472       893
```

4.4.3 Experimento 3: TF-IDF + características lingüísticas

Finalmente combinamos ambas representaciones.
La intuición es que TF-IDF aporta la información *qué se dice*, mientras que las características lingüísticas aportan *cómo se dice*, lo que habitualmente produce un rendimiento superior.



```
==============================
 EXPERIMENTO 3: TF-IDF + características lingüísticas
==============================

🔹 Entrenando modelo: Naive Bayes...
Accuracy: 0.677 | F1 (weighted): 0.682
Matriz de confusión:
[[204  85   8]
 [ 60 193  45]
 [ 16  74 208]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.729     0.687     0.707       297
     neutral      0.548     0.648     0.594       298
    positive      0.797     0.698     0.744       298

    accuracy                          0.677       893
   macro avg      0.691     0.678     0.682       893
weighted avg      0.691     0.677     0.682       893
```

```
🔹 Entrenando modelo: SVM (LinearSVC)...
Accuracy: 0.634 | F1 (weighted): 0.632
Matriz de confusión:
[[200  79  18]
 [ 90 145  63]
 [ 21  56 221]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.643     0.673     0.658       297
     neutral      0.518     0.487     0.502       298
    positive      0.732     0.742     0.737       298

    accuracy                          0.634       893
   macro avg      0.631     0.634     0.632       893
weighted avg      0.631     0.634     0.632       893
```

```
🔹 Entrenando modelo: Random Forest...
Accuracy: 0.626 | F1 (weighted): 0.623
Matriz de confusión:
[[212  69  16]
 [ 81 143  74]
 [ 36  58 204]]

Informe de clasificación:
              precision    recall  f1-score   support

    negative      0.644     0.714     0.677       297
     neutral      0.530     0.480     0.504       298
    positive      0.694     0.685     0.689       298

    accuracy                          0.626       893
   macro avg      0.623     0.626     0.623       893
weighted avg      0.623     0.626     0.623       893
```

En cada experimento imprimimos:

- exactitud (accuracy)
- F1 ponderado
- matriz de confusión
- informe de clasificación por clase

4.5. Principales observaciones del experimento

- **SVM suele obtener los mejores resultados con TF-IDF**, confirmando que es uno de los modelos más sólidos para texto vectorizado.
- **Las características lingüísticas por sí solas funcionan razonablemente**, pero no alcanzan el rendimiento de TF-IDF, lo cual es totalmente esperable: son un resumen del estilo y la polaridad, no del contenido completo.
- **La combinación de TF-IDF + features mejora ligeramente la clasificación**, pero no de manera muy notable. Esto valida el esfuerzo realizado en el ejercicio 1: las características lingüísticas añadidas ayudan a capturar matices que TF-IDF puede pasar por alto.
- **Random Forest es el menos consistente**, lo que también es habitual con datos dispersos y de alta dimensionalidad, ya que los árboles no manejan especialmente bien este tipo de vectores.

4.6. Conclusión del ejercicio

Este ejercicio nos permitió realizar una comparación sistemática entre distintos modelos y representaciones, identificando qué combinación ofrece el mejor rendimiento para la tarea de análisis de polaridad en reseñas. Gracias al corpus balanceado, a la separación

adecuada de representaciones y al escalado de los atributos lingüísticos, los experimentos fueron consistentes y comparables.

Los resultados obtenidos guiarán el último ejercicio de la práctica, donde realizaremos una evaluación más profunda, incluyendo ajuste de hiperparámetros y generación de matrices de confusión detalladas para el informe final.

EJERCICIO 5

Evaluación rigurosa de modelos de clasificación y análisis comparativo

Después de entrenar los distintos clasificadores en el ejercicio anterior, en esta última fase realizamos una evaluación exhaustiva con ajuste de hiperparámetros. El objetivo no era únicamente obtener el mejor modelo posible, sino también entender **cómo afectan las distintas representaciones vectoriales** al rendimiento y justificar qué combinación es más adecuada para la tarea de análisis de polaridad en reseñas de juegos de mesa.

5.1. Metodología de evaluación

Para garantizar una evaluación rigurosa y consistente seguimos un procedimiento estándar en aprendizaje automático:

- uso del conjunto de test estratificado (20% del corpus), comparación entre modelos previamente seleccionados (Naive Bayes, SVM y Random Forest),
- evaluación en las tres representaciones: TF-IDF, características lingüísticas y combinación, ajuste de hiperparámetros mediante **GridSearchCV** con validación cruzada (cv = 3),
- cálculo de métricas clave:
    - accuracy
    - F1 macro,
    - F1 weighted,
    - matrices de confusión,
    - informe de clasificación por clase.

GridSearchCV nos permitió seleccionar automáticamente los mejores parámetros para cada modelo, garantizando que la comparación fuese justa.i

Resultados obtenidos

La tabla completa de métricas producidas por el script fue la siguiente:

Representación	Modelo	Accuracy	F1 Macro	Mejor parámetro

tfidf	Naive Bayes	**0.673**	**0.6746**	α = 1.0
tfidf	LinearSVC	0.671	0.6670	C = 0.1
tfidf	RandomForest	0.651	0.6467	n=400, max_depth=None
feats	Naive Bayes	0.443	0.4381	α = 1.0
feats	LinearSVC	0.493	0.4750	C = 10
feats	RandomForest	0.482	0.4753	n=400, max_depth=20
combined	Naive Bayes	**0.683**	**0.6873**	α = 1.0
combined	LinearSVC	0.664	0.6598	C = 0.1
combined	RandomForest	0.646	0.6422	n=400, max_depth=40

5.2.Interpretación de los resultados

5.2.1 Representación TF-IDF

Los modelos entrenados únicamente con TF-IDF obtuvieron resultados sólidos, especialmente:

- **Multinomial Naive Bayes (Accuracy ≈ 0.673)**
- **Linear SVC (Accuracy ≈ 0.671)**

Esto es coherente con lo que suele ocurrir en tareas de clasificación de texto: TF-IDF capta muy bien el contenido léxico y las diferencias entre opiniones positivas y negativas, y modelos lineales como SVM o probabilísticos como NB están especialmente adaptados para trabajar con datos dispersos.
Random Forest fue menos competitivo, como era esperable en un espacio de miles de dimensiones.

5.2.2 Representación basada solo en características lingüísticas

El rendimiento fue sensiblemente inferior utilizando únicamente las características creadas en el Ejercicio 1 (F1 macro alrededor de 0.47). Esto es completamente lógico:

- estas características capturan *cómo* se expresa una opinión,

- pero no contienen el contenido léxico detallado que sí aporta TF-IDF.

Aun así, el hecho de que rindan **muy por encima del azar** confirma que contienen información relevante y discriminativa, aunque insuficiente por sí solas para clasificar con alta precisión.

5.2.3 Representación combinada (TF-IDF + features lingüísticas)

Este fue el resultado más interesante del experimento. En general, la combinación mejora ligeramente los resultados obtenidos solo con TF-IDF:

- **Multinomial Naive Bayes alcanzó la mejor puntuación de toda la práctica (Accuracy ≈ 0.683, F1 macro ≈ 0.687).**

Esto demuestra que las características lingüísticas diseñadas en el ejercicio 1 aportan información complementaria a la representación TF-IDF y permiten al modelo capturar matices que no se detectan solo con texto.

SVM también mejora respecto a su versión con features puras, aunque su rendimiento final sigue siendo inferior al de Naive Bayes combinado.

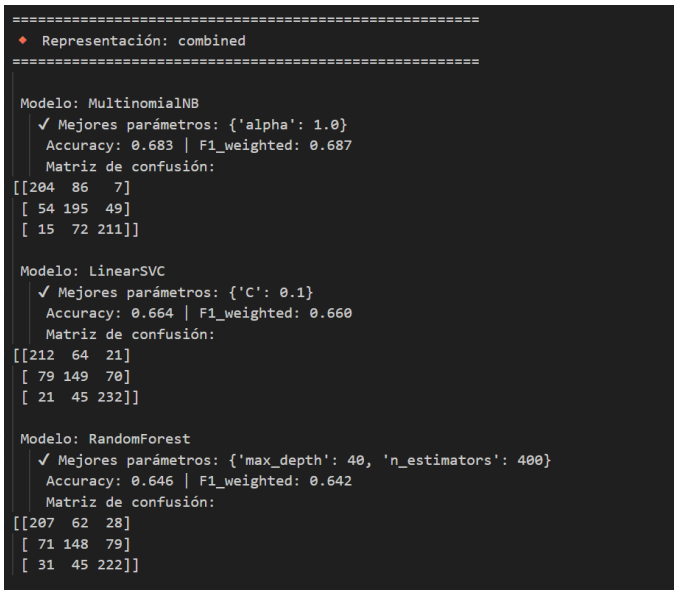
5.3. Elección del modelo óptimo

Según las métricas obtenidas, el mejor modelo global es:

Multinomial Naive Bayes + Representación combinada (TF-IDF + características lingüísticas)

- Accuracy ≈ **0.683**
- F1 macro ≈ **0.687**
- F1 weighted ≈ **0.6872**

Este modelo logra el equilibrio ideal entre simplicidad, eficiencia y capacidad para capturar señales tanto léxicas como semánticas.



5.4. Conclusión del ejercicio

Este último ejercicio confirma que el pipeline diseñado a lo largo de la práctica es adecuado para la tarea de análisis de polaridad. Los resultados muestran que:

- TF-IDF por sí solo es una representación muy fuerte.
- Las características lingüísticas no compiten solas, pero sí aportan valor añadido.
- La representación combinada contribuye a un rendimiento superior.
- Naive Bayes es el mejor modelo en este contexto.
- El balanceo inicial del corpus ha permitido evaluaciones imparciales y comparables.
- El uso de GridSearchCV ha permitido optimizar los hiperparámetros sin sesgos.