

Procesamiento del Lenguaje Natural

Clasificación de textos para análisis de sentimiento

Autores: Adam Maltoni
Ibón de Mingo

Clase: 4º ciencia de datos
Grupo 03
Asignatura: PLN



Abstract

Esta práctica desarrolla un pipeline completo para la clasificación de sentimiento en reseñas de BoardGameGeek. Comienza con una fase de ingeniería de características donde se extraen atributos lingüísticos avanzados (polaridad léxica, negaciones, modificadores de intensidad y vocabulario específico). Después se generan siete representaciones vectoriales diferentes, basadas en n-gramas TF-IDF, en rasgos lingüísticos y en combinaciones de ambos, optimizando su almacenamiento para evitar problemas de memoria. El corpus se etiqueta en tres clases, se balancea por submuestreo y se divide en conjuntos estratificados de entrenamiento, validación y test. A continuación se entrena un conjunto de modelos en paralelo mediante *GridSearchCV*, cubriendo cuatro algoritmos de ML y siete representaciones. Finalmente, se evalúan exhaustivamente los modelos seleccionados sobre el conjunto de test, produciendo un análisis comparativo y resultados que permiten identificar la mejor combinación modelo-representación.

Keywords: Análisis de Sentimiento, Clasificación de Texto, Sklearn, NLTK, Feature Engineering, SVM, Random Forest, Naive Bayes.

Índice

Abstract.....	2
Índice.....	3
1 Introducción.....	4
2 Preprocesamiento y construcción y tratamiento de los conjuntos de datos.....	5
3 Features lingüísticas utilizadas para la representación vectorial..	7
4 Experimentos realizados para evaluar los modelos sobre diferentes representaciones vectoriales.....	9
5 Comparación y análisis de los resultados de la evaluación de modelos.....	11
6 Discusión y dificultades encontradas.....	13
7 Conclusiones y trabajo futuro.....	14
8 Reproducibilidad y anexos.....	15
8.1 Requisitos e instalación.....	15
8.2 Ejecución base.....	15
9 Referencias.....	16
10 Anexos.....	17

1 Introducción

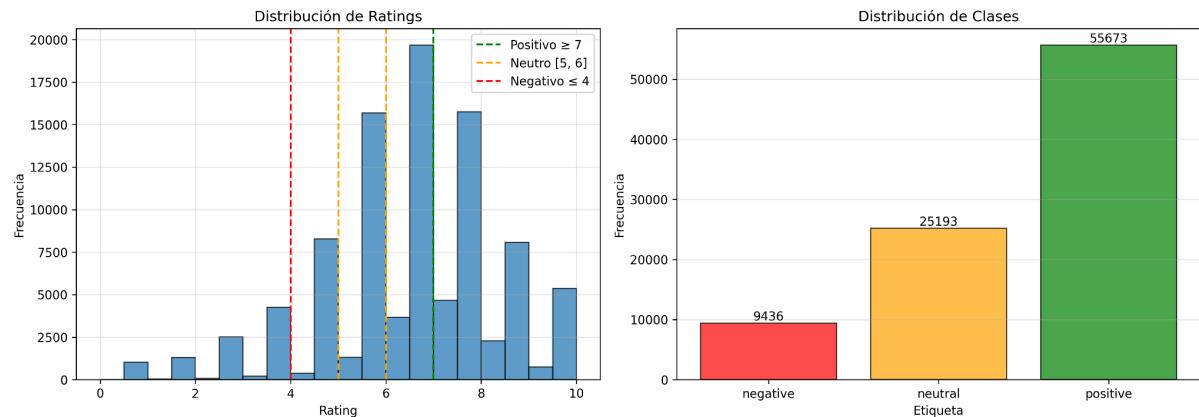
El objetivo de esta práctica es aplicar técnicas de PLN para construir un sistema que prediga la polaridad de reseñas en BoardGameGeek. Más que resolver una tarea concreta, el proyecto introduce las fases clave de un pipeline real de análisis de sentimiento: extracción de rasgos lingüísticos, generación de representaciones vectoriales, preparación del corpus y evaluación comparativa de modelos supervisados. Esta estructura permite comprender cómo se diseñan y validan sistemas fiables de clasificación textual en aplicaciones prácticas.

2 Preprocesamiento y construcción y tratamiento de los conjuntos de datos

2.1 Definición de polaridad de las reseñas

El primer paso del E3 consiste en la definición de las clases de polaridad. Para ello, la función `rating_to_label` del script del ejercicio 3 asigna una etiqueta categórica a cada reseña basándose en su *rating* numérico (de 1 a 10). Siguiendo la recomendación del enunciado, se definieron tres clases: **Positiva** (*ratings* ≥ 7), **Neutra** (*ratings* entre 5 y 6, ambos inclusive) y **Negativa** (*ratings* ≤ 4). Las reseñas con *ratings* fuera de estos rangos (p.ej., 4.5) o sin *rating* fueron descartadas del conjunto de datos.

Este etiquetado reveló un fuerte desbalanceo de clases, con una mayoría de reseñas positivas, tal y como se muestra en el siguiente gráfico.



2.2 Balanceo de clases

Para tratar el desbalanceo de clases, obligatorio según el enunciado, la función `balance_dataset` aplica una estrategia de **submuestreo (undersampling)**. Este método identifica la clase minoritaria (en nuestro caso, 'negativa') y reduce aleatoriamente las otras dos clases (positiva y neutra) hasta igualar el número de muestras de la minoritaria. Esto asegura un conjunto de datos final perfectamente balanceado (33% por cada clase), previniendo que los modelos aprendan un sesgo hacia la clase mayoritaria.

Una vez balanceados los datos, la función `create_train_val_test_split` partitiona el conjunto. Se reserva un **20%** para el **test final** (datos que los

modelos nunca verán hasta el E5). El 80% restante se divide a su vez, destinando un **10% para validación** (usado en el E4 para el ajuste de hiperparámetros) y un **70% para entrenamiento**. Es crucial destacar que este particionado se realiza de forma **estratificada** (`stratify=True`), asegurando que la proporción 33-33-33 de clases se mantenga en los tres conjuntos (train, val y test).

3 Features lingüísticas utilizadas para la representación vectorial

Para construir los clasificadores, se generaron dos tipos principales de representaciones vectoriales, que se utilizaron tanto de forma independiente como combinada, dando lugar a 7 conjuntos de características.

1. Representación Léxica (N-Gramas TF-IDF): Esta representación captura las características superficiales y contextuales del texto. Se implementó usando `TfidfVectorizer` de Scikit-learn, aplicando primero un preprocesamiento de lematización (con `WordNetLemmatizer`) y eliminación de *stopwords*. Para controlar la alta dimensionalidad, se restringió el vocabulario a las 5000 características más relevantes (`max_features=5000`), se aplicó un escalado logarítmico (`sublinear_tf=True`) y se ignoraron términos que aparecían en menos de 2 documentos. Se generaron tres variantes, incluyendo de una a tres longitudes de cadena: `ngram_unigram`, `ngram_unigram_bigram`, `ngram_unigram_bigram_trigram`.

2. Representación Lingüística (Sentimiento): Esta es una representación de baja dimensionalidad (29 características) que captura información semántica de alto nivel, diseñada específicamente para el análisis de opinión. Las características, extraídas en el Ejercicio 1, se agrupan en:

- **Polaridad (VADER manual):** 4 *scores* (positivo, negativo, neutro y *compound*) basados en nuestros léxicos de dominio.
- **Polaridad (SentiWordNet):** 5 *scores* (promedio positivo/negativo, máximo positivo/negativo, y conteo de palabras de sentimiento).
- **Morfología (PoS):** 3 *scores* (conteo de adjetivos, adverbios y verbos).
- **Negaciones:** 3 *scores* (conteo de palabras de negación, conteo de sentimientos negados).
- **Modificadores:** 4 *scores* (conteo de intensificadores y atenuadores, y su ratio).
- **Vocabulario de Dominio:** 7 *scores* (conteo de términos por categoría: mecánicas, componentes, reglas, experiencia; conteo total de dominio, ratio y co-ocurrencias con sentimiento).
- **Metadatos:** 3 *scores* (conteo de *tokens*, conteo de frases, longitud media de palabra). Este vector de características fue normalizado usando `StandardScaler` para asegurar que todas las *features* tuvieran una media de 0 y una desviación estándar de 1.

3. Representaciones Combinadas: Se generaron tres representaciones más: `combined_unigram_sentiment`, `combined_unigram_bigram_sentiment` y, por último `combined_unigram_trigram_sentiment` concatenando horizontalmente

(`np.hstack`) las representaciones léxicas y lingüísticas. El objetivo es proporcionar a los modelos tanto el contexto de las palabras (TF-IDF) como las características semánticas de la opinión.

4 Experimentos realizados para evaluar los modelos sobre diferentes representaciones vectoriales

Se quiere evaluar rigurosamente el impacto de las diferentes representaciones vectoriales y los distintos algoritmos de clasificación.

Conjunto de datos: El *pipeline* del Ejercicio 3 generó 7 conjuntos de datos, uno por cada representación vectorial. Todos los conjuntos fueron balanceados por submuestreo (conteniendo un número idéntico de reseñas por clase) y particionados de forma estratificada en Train (70%), Validation (10%) y Test (20%).

Proceso de entrenamiento y optimización (ej.4): Se definió un *pool* de 28 experimentos (7 representaciones x 4 modelos). Los modelos seleccionados fueron:

1. **Multinomial Naive Bayes**
2. **SVM (kernel lineal)**
3. **SVM (kernel RBF)**
4. **Random Forest**

Para cada experimento, se realizó un ajuste de hiperparámetros por *GridSearchCV* utilizando los conjuntos de train y val, con una validación cruzada de 5 *folds* y optimizando para la métrica **F1-score**. Se presentan en la siguiente tabla los hiperparámetros óptimos obtenidos:

Table 1: Parámetros Óptimos Encontrados por Modelo y Representación (E4)

Representación	Multinomial NB	SVM (Linear)	SVM (RBF)	Random Forest
combined unigram bigram sentiment	alpha=1.0, fit prior=True	C=0.1, class weight=None	C=1, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
combined unigram bigram trigram sentiment	alpha=1.0, fit prior=True	C=1, class weight=None	C=1, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
combined unigram sentiment	alpha=1.0, fit prior=True	C=0.1, class weight=None	C=10, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
ngram unigram	alpha=1.0, fit prior=True	C=0.1, class weight=None	C=1, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
ngram unigram bigram	alpha=1.0, fit prior=True	C=0.1, class weight=None	C=1, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
ngram unigram bigram trigram	alpha=1.0, fit prior=True	C=1, class weight=None	C=1, gamma='scale'	max depth=None, min samples leaf=1, min samples split=5, n estimators=100
sentiment	alpha=0.5, fit prior=True	C=1, class weight=None	C=10, gamma=0.01	max depth=None, min samples leaf=4, min samples split=2, n estimators=100

Proceso de Evaluación Final (E5): Los 28 modelos óptimos resultantes del E4 se evaluaron contra el conjunto de **Test** (datos 100% nuevos para los modelos). Se recopilaron las siguientes métricas de evaluación para la comparativa final: **Accuracy**, **Precision**, **Recall** y **F1-score**. Adicionalmente, se generó una **Matriz de Confusión** para cada uno de los 28 modelos para un análisis visual del error.

Las matrices de confusión obtenidas para cada método se pueden consultar en la sección de Anexos de esta memoria.

5 Comparación y análisis de los resultados de la evaluación de modelos

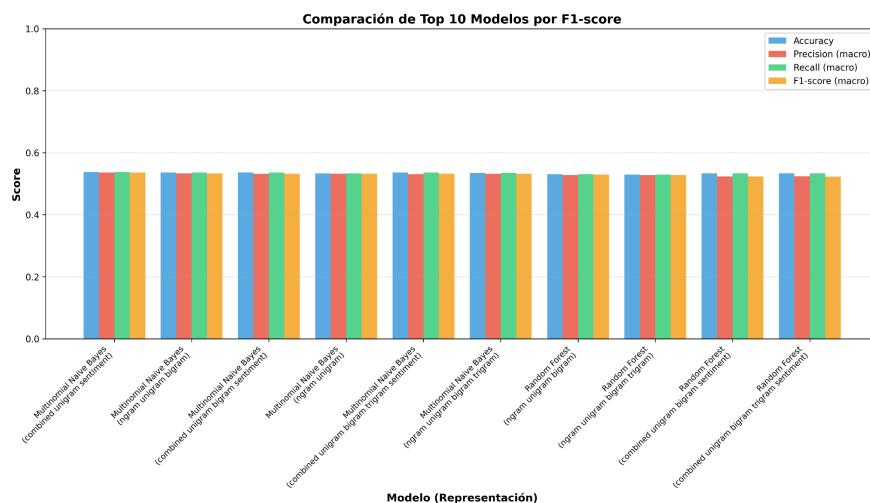
Nota: Los resultados presentados se basan en un corpus de aproximadamente 95,000 reseñas (tras el filtrado del E1 y antes del balanceo).

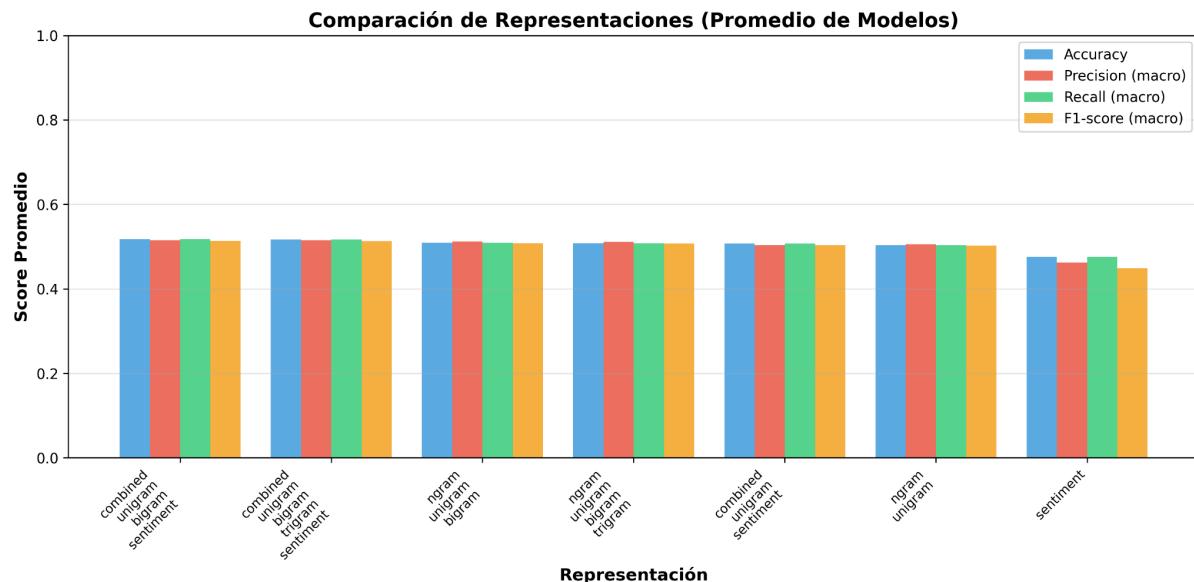
Todos los resultados de la evaluación de los 28 modelos en el conjunto de *test* se encuentran en el directorio */evaluacion/*, principalmente en el archivo *comparison_table.csv*. A continuación, se presenta un extracto de una tabla comparativa con los 10 mejores modelos encontrados, ordenados por su F1-score en el conjunto de *test*.

Table 1: Top 10 modelos por F1-score en conjunto de test

Representación	Modelo	Accuracy	Precision (macro)	Recall (macro)	F1-score (macro)	CV Score	Val Score
combined unigram sentiment	MNBayes	0.538	0.536	0.538	0.536	0.527	0.520
ngram unigram bigram	MNBayes	0.536	0.533	0.536	0.533	0.532	0.532
combined unigram bigram sentiment	MNBayes	0.536	0.532	0.536	0.533	0.531	0.527
ngram unigram	MNBayes	0.533	0.533	0.533	0.532	0.528	0.527
combined unigram bigram trigram sentiment	MNBayes	0.536	0.531	0.536	0.532	0.530	0.526
ngram unigram bigram trigram	MNBayes	0.535	0.532	0.535	0.532	0.530	0.529
ngram unigram bigram	RForest	0.531	0.528	0.531	0.530	0.516	0.529
ngram unigram bigram trigram	RForest	0.530	0.528	0.530	0.529	0.516	0.527
combined unigram bigram sentiment	RForest	0.534	0.524	0.534	0.524	0.511	0.526
combined unigram bigram trigram sentiment	RForest	0.534	0.524	0.534	0.523	0.509	0.533

Como puede observarse, la mejor performance se ve determinada, de acuerdo con nuestros resultados, casi siempre por modelo en vez de por representación, aunque cabe destacar que obtenemos una variabilidad bajísima para accuracias y métricas de evaluación de modelos, lo que parece sugerir que el rendimiento que obtenemos es bastante independiente de ambos factores, como puede verse a continuación:





De estas gráficas, se puede suponer que no hay ganancia significativa en considerar N-gramas de mayor orden, por lo que tendría sentido recurrir a los unigramas (palabras) al ser los menos costosos de computar. Sin embargo, si vemos que los N-gramas tienen valor, puesto que, a pesar de la baja variabilidad, las configuraciones que solo consideran la representación por sentimiento obtienen un rendimiento más débil, confirmando que esa representación de cadenas de palabras es valiosa para un aprendizaje de las características lingüísticas.

El modelo ganador, Multinomial Naive Bayes, resultó ser también el más rápido con diferencia a la hora de entrenarse y de realizar predicciones. Por otro lado, las SVMs mostraron un rendimiento temporal muy pobre, debido a su alta complejidad de entrenamiento para la búsqueda de la frontera de decisión.

Analizando las matrices de confusión (disponibles en *Anexos*), vemos que, como norma general, se observa que nuestros modelos son muy **precisos identificando reseñas negativas y positivas**. Sin embargo, su principal punto de fallo es la clase 'neutral', la cual es incapaz de distinguir y que confunde de forma casi idéntica con 'negativa' y 'positiva' en la gran mayoría de casos, suponemos que por estar 'al medio' en el rango de *ratings*.

6 Discusión y dificultades encontradas

Durante el desarrollo de la práctica, surgieron tres dificultades principales.

En el ejercicio 2, el diseño inicial de nuestro programa intentaba generar las 7 representaciones vectoriales y mantenerlas en memoria simultáneamente para su procesamiento, y escribirlas al final. Dado el tamaño del corpus (casi 100k reseñas) y la alta dimensionalidad de los n-gramas (5000+ características), esto agotaba la RAM de nuestro sistema (`MemoryError`). La solución fue rediseñar `pln_p2_7461_03_e2.py` para operar secuencialmente: el script ahora genera una representación, la guarda en disco (en formato `.npy`, usando `float32` para optimizar espacio) y libera la memoria antes de comenzar con la siguiente.

El entrenamiento de los 28 modelos presentó un cuello de botella severo. Los modelos SVM, en particular el de kernel RBF, mostraban tiempos de entrenamiento de varias horas por cada tarea. Además, se obtenían `ConvergenceWarning`, es decir, que en las SVMs ni siquiera se llegaba a la solución tras tanto tiempo. Para resolver esto, decidimos usar los datos con `Standard Scaler`, para acelerar la convergencia. Para la SVM con kernel RBF, también tuvimos que recurrir a subsamplear datos para el `GridSearch`, para poder ‘estimar’ unos parámetros óptimos con menos datos, y luego entrenar todos los datos con ese modelo.

También en el ejercicio 4, intentamos parallelizar los modelos, y que cada modelo paralelizara su `GridSearch`. Esto resultó en demasiados procesos, colapsando el ordenador. El diseño tuvo que refactorizarse un *pool* de tareas no bloqueante: se crea una lista única de 28 tareas (7 reps x 4 modelos) y un número fijo de *workers* (`joblib.Parallel`) las consume. Esto aseguró que los modelos rápidos (sobretodo MN Naive Bayes) se completaran sin esperar a los lentos (SVMs) y que la CPU se utilizara al máximo (8 hilos en nuestro ordenador) sin sobrecargarse.

7 Conclusiones y trabajo futuro

Hemos conseguido cumplir los objetivos de la práctica, implementando un *pipeline* end-to-end de análisis de sentimiento (contando con la Práctica 1 también, desde la extracción de datos hasta la evaluación de modelos pasando por muchas fases intermedias de preprocesamiento, representación y entrenamiento de modelos). Se trajeron características lingüísticas complejas, se generaron 7 representaciones vectoriales (contando con todas las combinaciones de representaciones), y se entrenaron y optimizaron 28 modelos de clasificación.

A pesar de esto, el mejor modelo (Multinomial Naive Bayes) con la representación de unígrafo + sentimiento, alcanzó un F1-score de ~0.54, pudiendo mejorarse mucho como línea futura.

Sería interesante explorar técnicas de balanceo más avanzadas que la simple subrepresentación para preservar más datos de las clases mayoritarias. También podría establecerse un preprocesamiento de mayor calidad (al etiquetar las reseñas como positivas, negativas o neutras, realmente no tiene sentido que consideremos un 7 como “igual de positivo” que un 10, por ejemplo). La parte buena es que este *pipeline* de modelos “clásicos” (SVM, RF, NB) puede servir como un buen *baseline* contra el cual comparar arquitecturas más modernas, potentes y de “estado del arte” (como redes neuronales modernas tipo RNNs o Transformers).

8 Reproducibilidad y anexos

8.1 Requisitos e instalación

Los *requirements* del proyecto, a nivel de librerías, están especificados en el repositorio, dentro del archivo `requirements.txt`.

Para reproducir resultados, solo es necesario crear un nuevo **virtual environment** de **python**, e instalar las librerías y versiones especificadas, para posteriormente poder ejecutar los archivos del proyecto sin problemas de dependencias.

8.2 Ejecución base

En la propia raíz del proyecto, se proporciona un script '`run_all.bat`' que permite directamente ejecutar todas las partes del desarrollo realizado en serie. Para entornos Linux, basta con copiar su contenido a un script `.sh`.

9 Referencias

[1] **BoardGameGeek Ratings Pages** – Sección de reseñas públicas por juego.
Disponible en: <https://boardgamegeek.com/boardgame/>

[2] **NLTK: Natural Language Toolkit** – *Proyecto de procesamiento de lenguaje natural en Python.* Disponible en: <https://www.nltk.org/>

[3] **Scikit-learn: Machine Learning in Python** – Biblioteca para minería de datos y análisis de datos. Disponible en: <https://scikit-learn.org/>

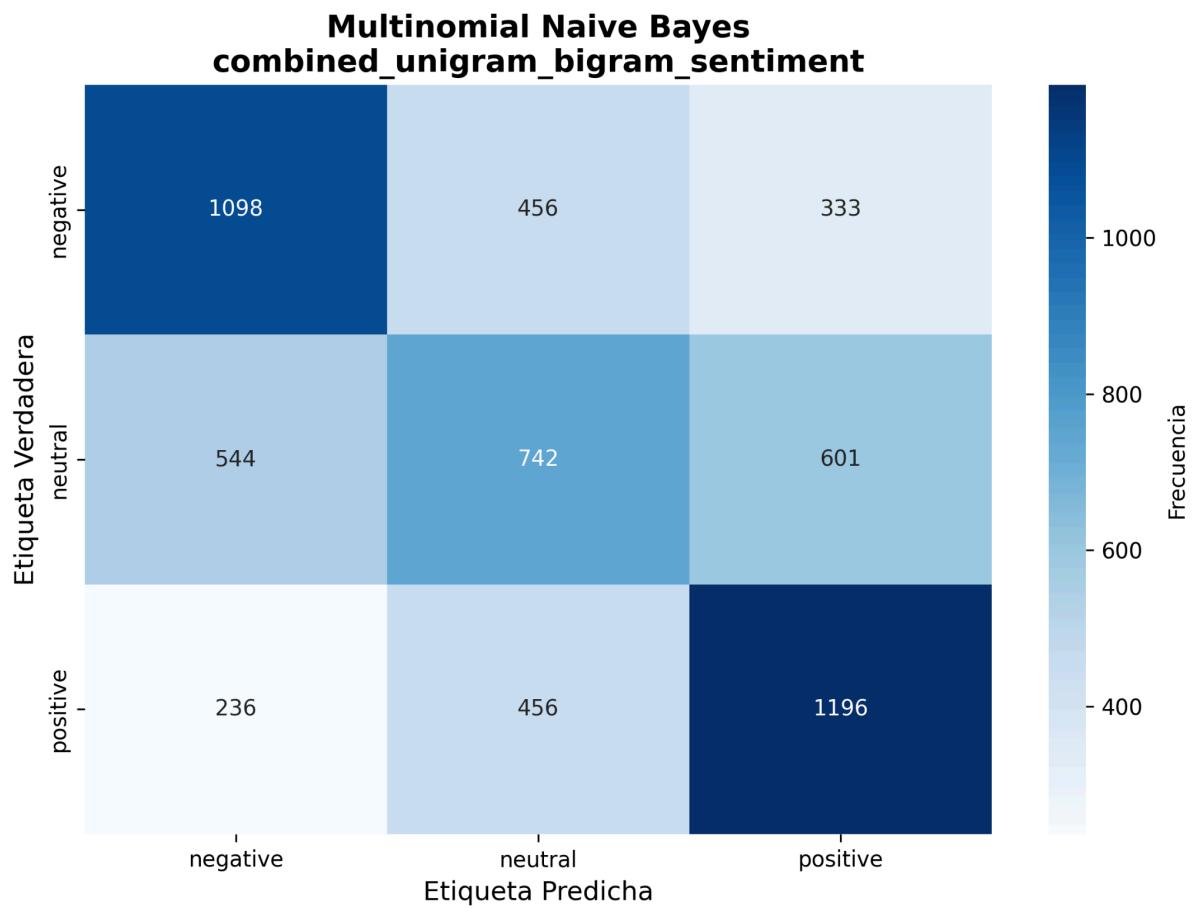
[4] **Pandas: Python Data Analysis Library** – Biblioteca para la manipulación y análisis de datos. Disponible en: <https://pandas.pydata.org/>

[5] **Matplotlib: Visualization with Python** – Biblioteca para la generación de gráficos y visualizaciones. Disponible en: <https://matplotlib.org/>

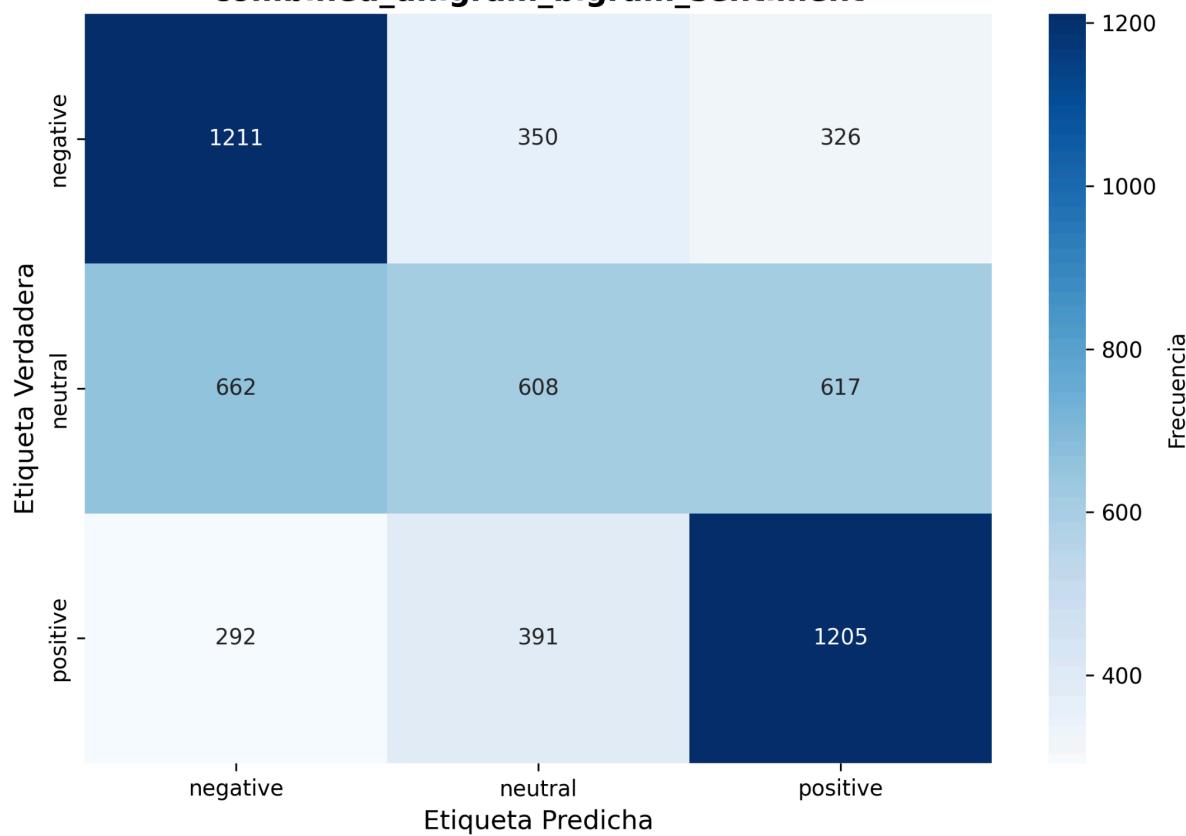
[6] **(Material de la Asignatura, tanto de teoría como de prácticas).** Procesamiento de Lenguaje Natural, Grado en Ciencia e Ingeniería de Datos, Universidad Autónoma de Madrid, 2025.

10 Anexos

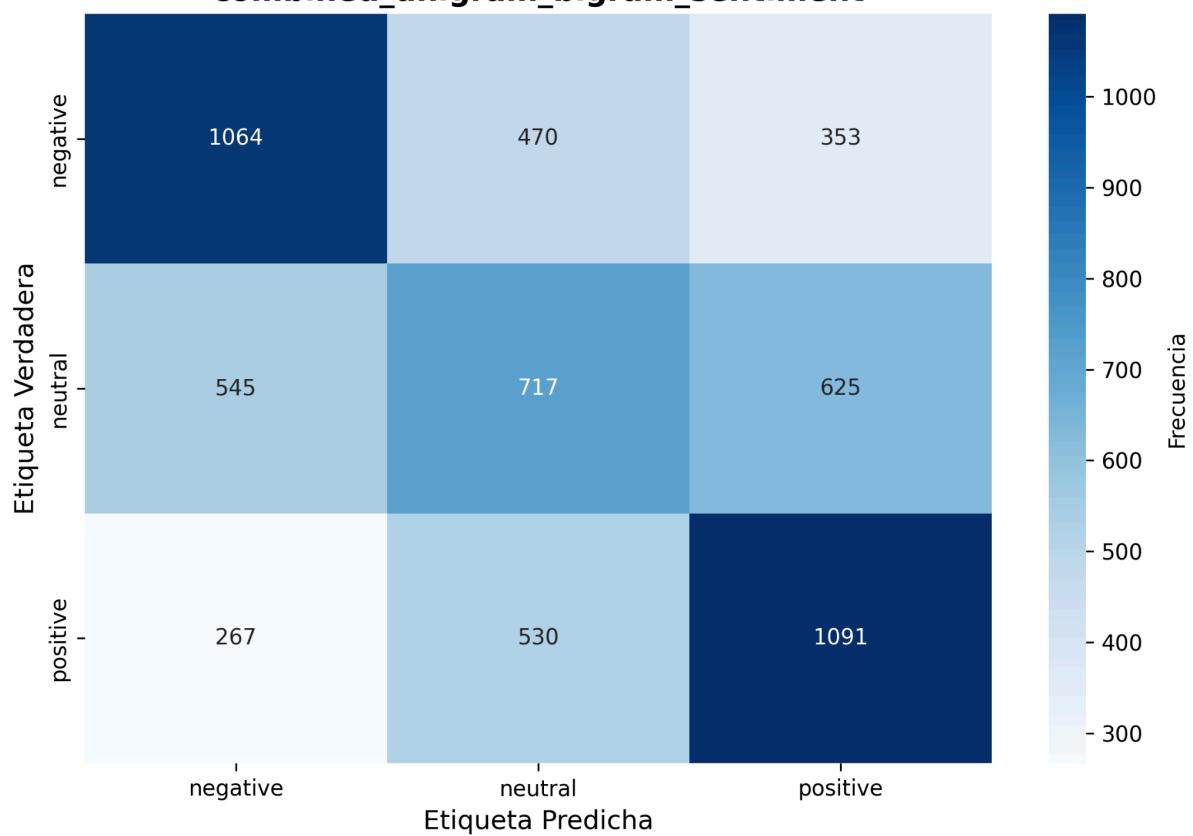
Confusion matrix por modelo y representación



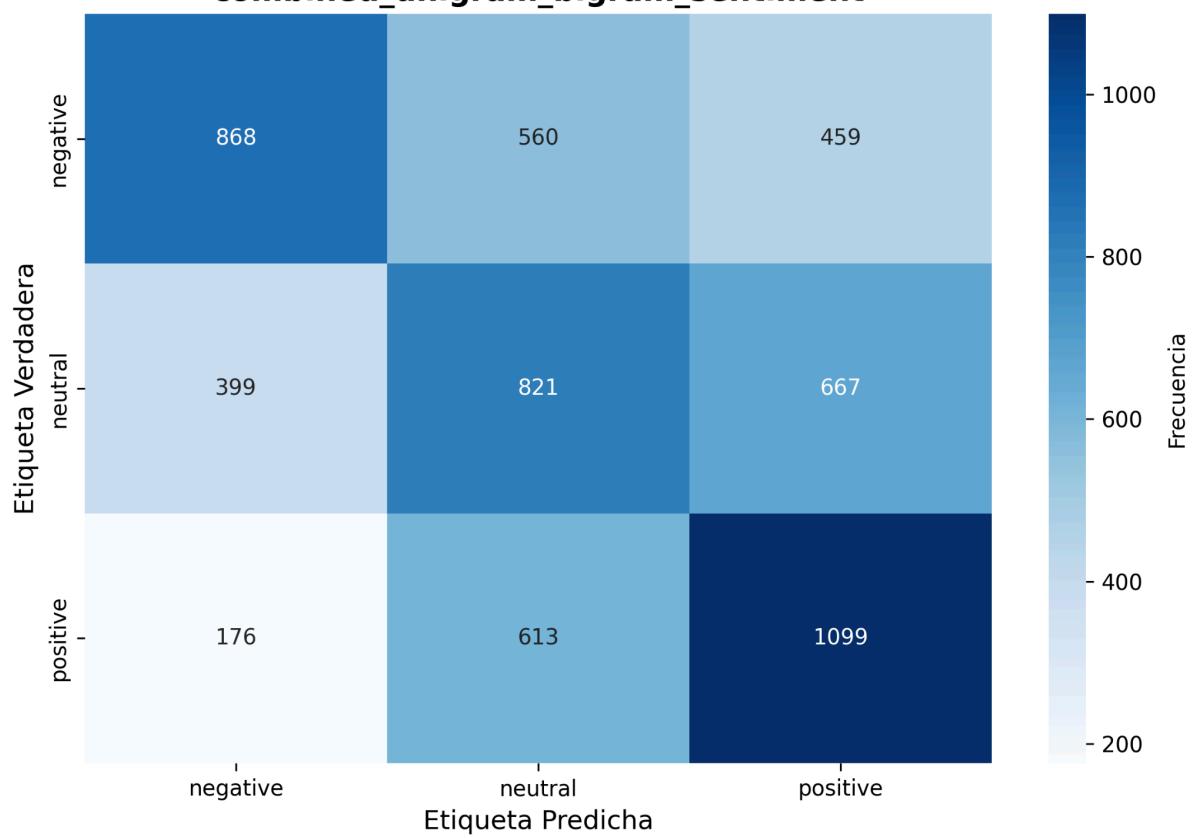
**Random Forest
combined_unigram_bigram_sentiment**



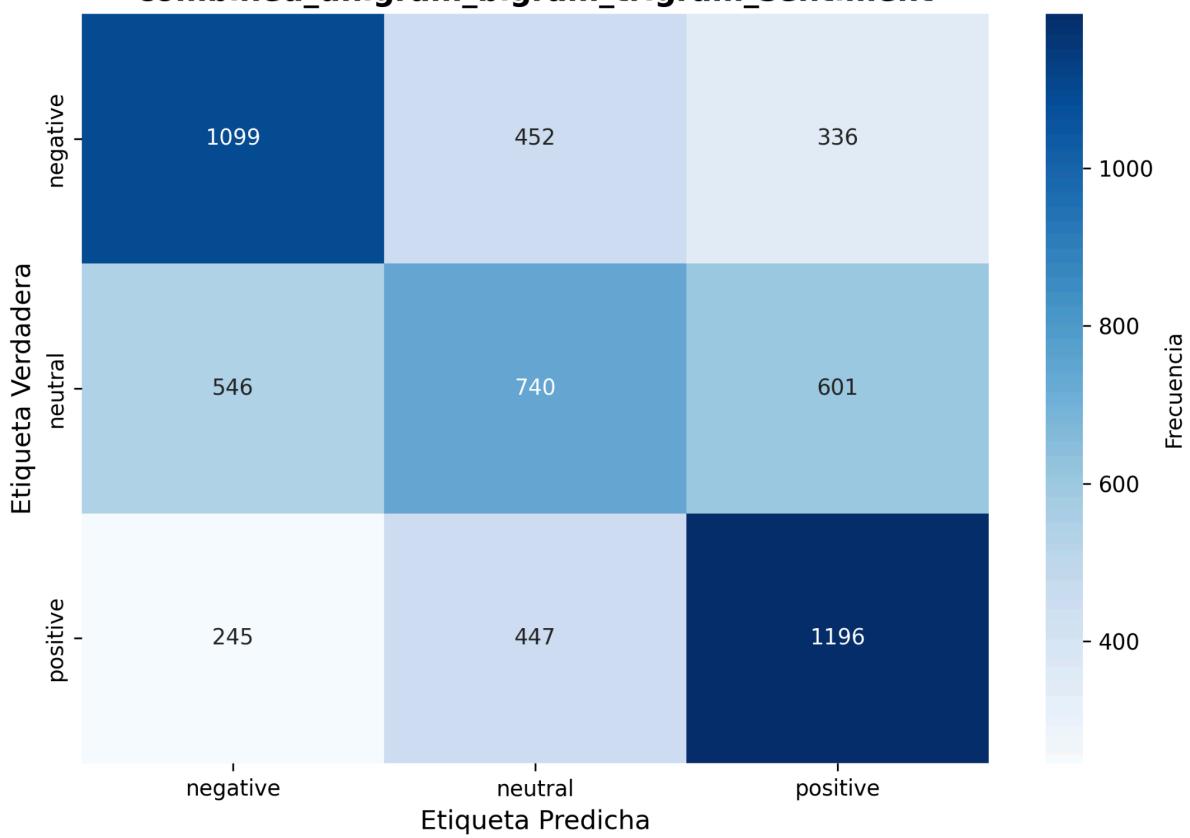
SVM (LinearSVC)
combined_unigram_bigram_sentiment



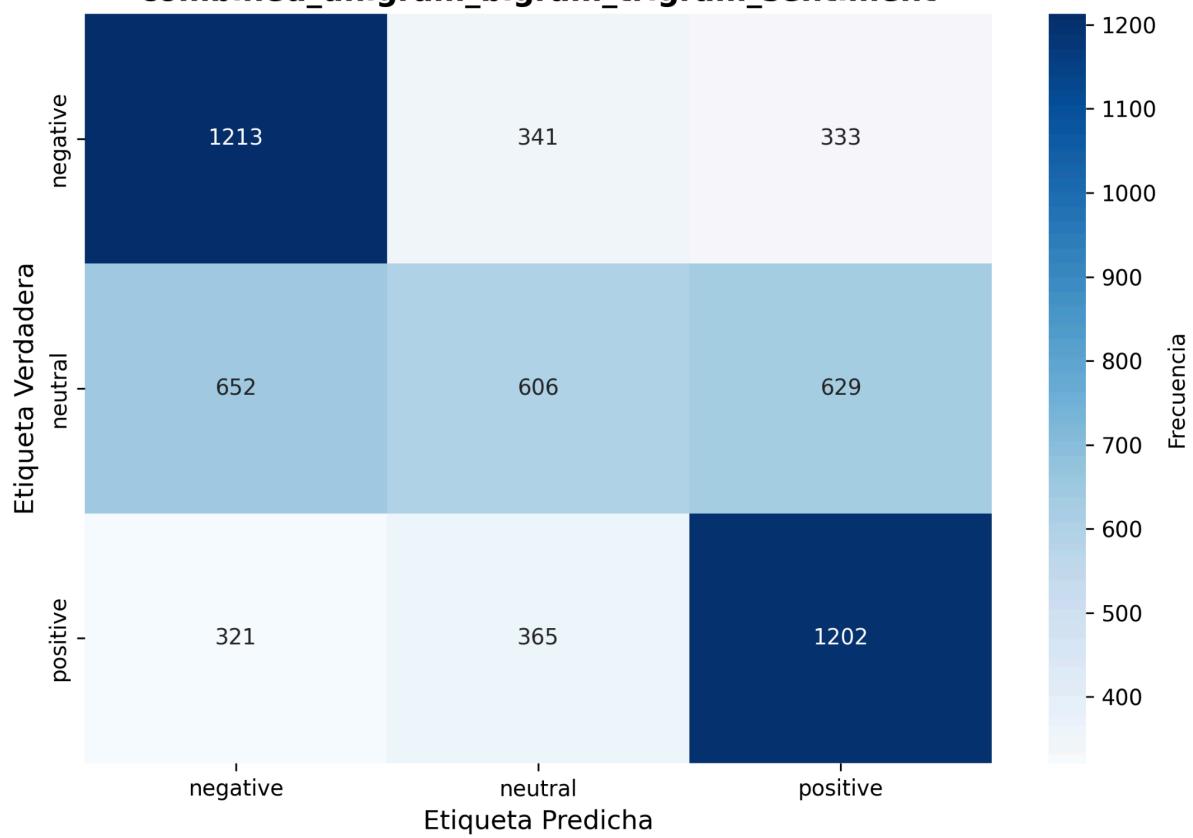
SVM (RBF)
combined_unigram_bigram_sentiment



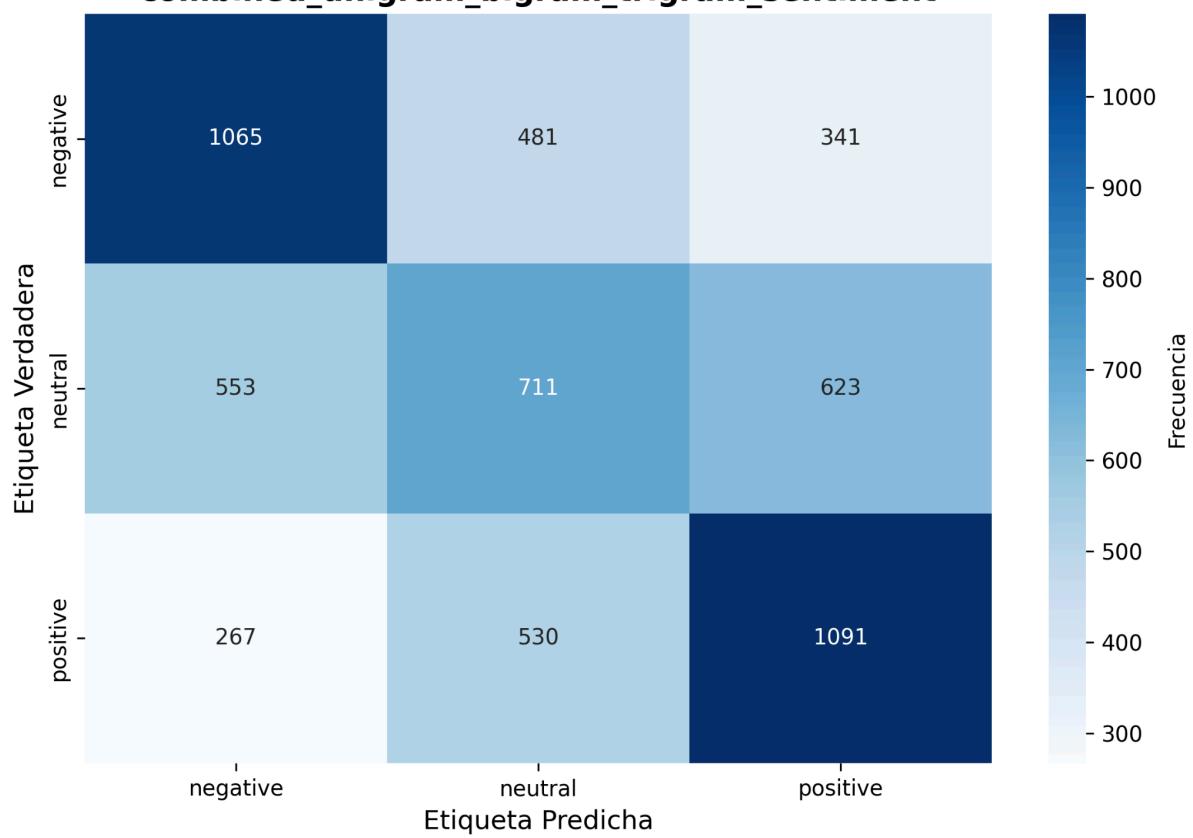
Multinomial Naive Bayes
combined_unigram_bigram_trigram_sentiment



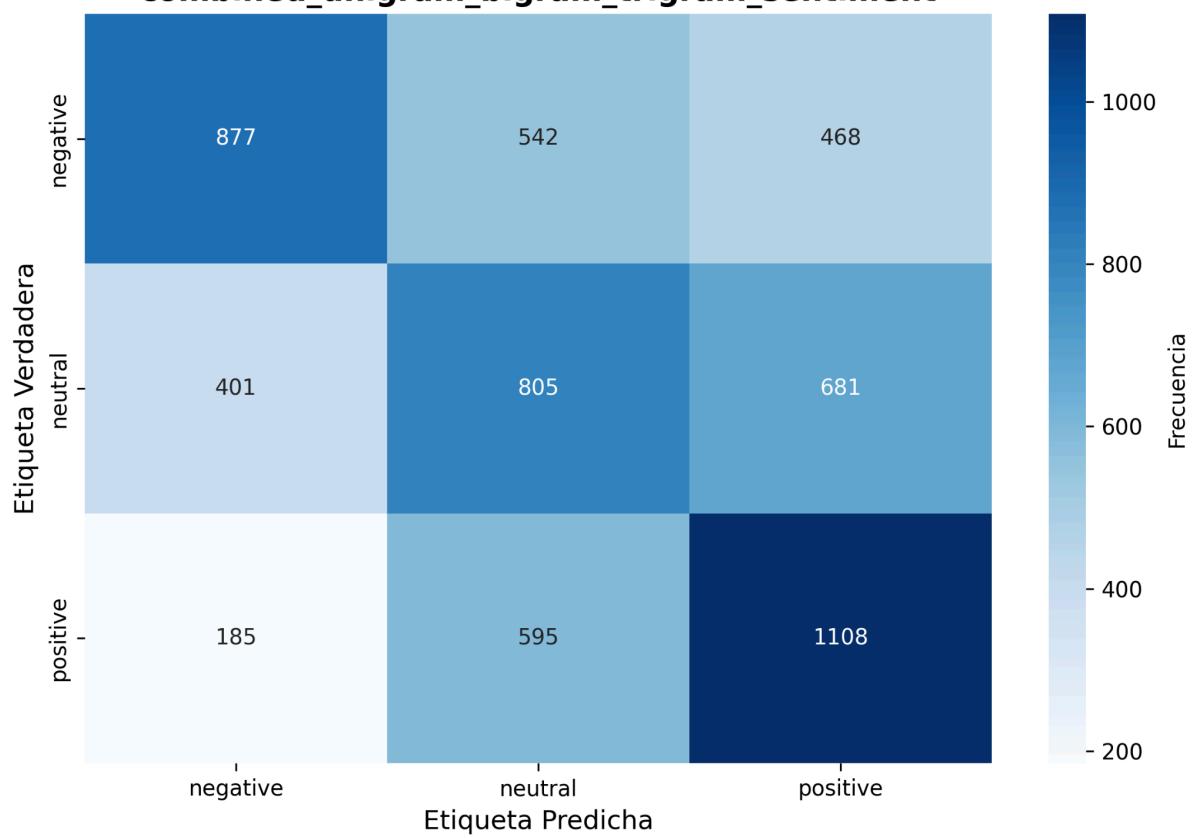
Random Forest
combined_unigram_bigram_trigram_sentiment



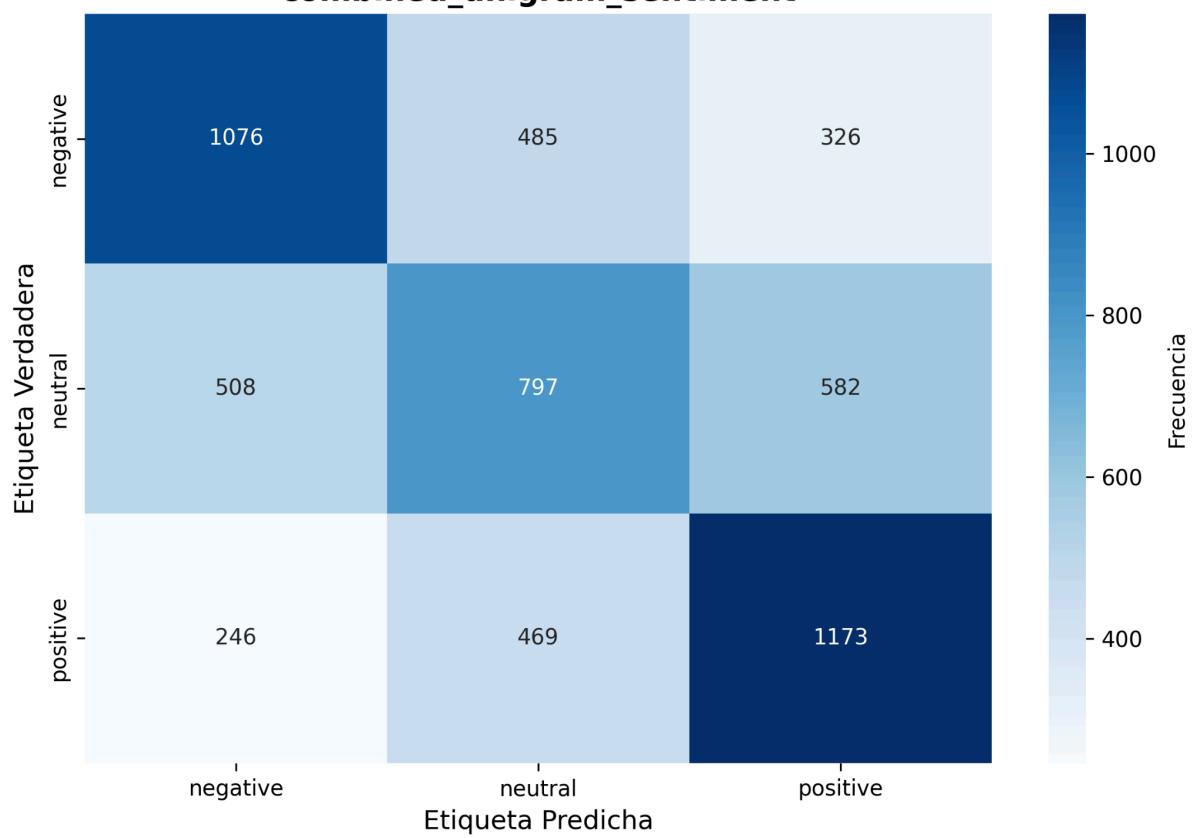
SVM (LinearSVC)
combined_unigram_bigram_trigram_sentiment

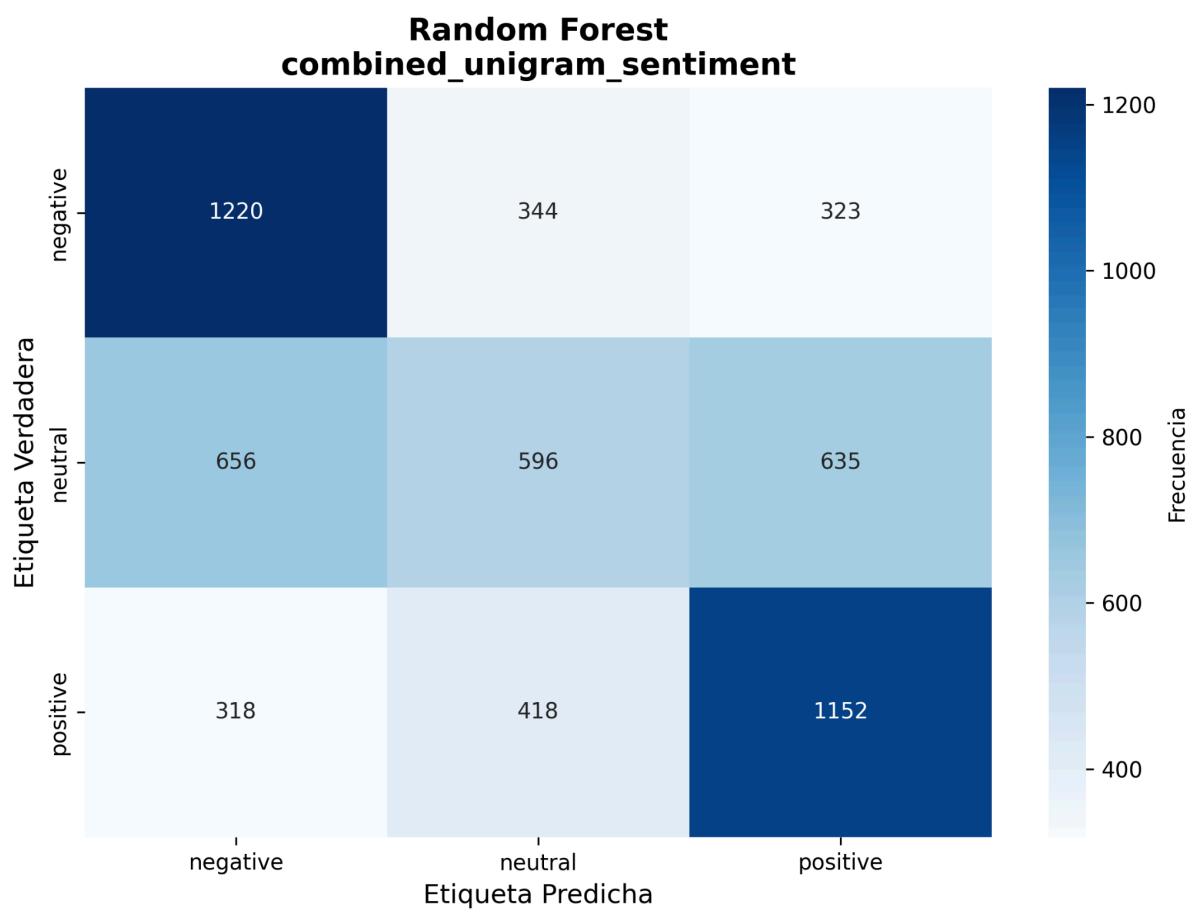


SVM (RBF)
combined_unigram_bigram_trigram_sentiment

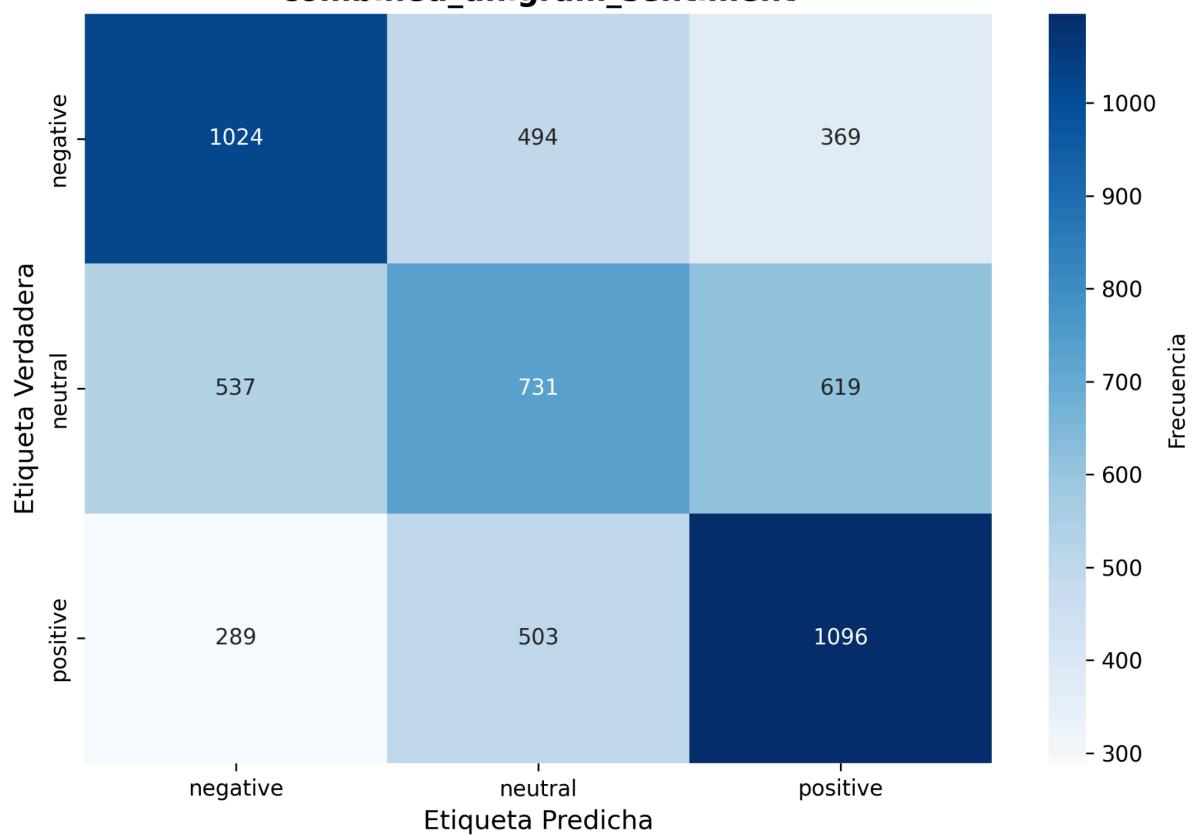


**Multinomial Naive Bayes
combined_unigram_sentiment**

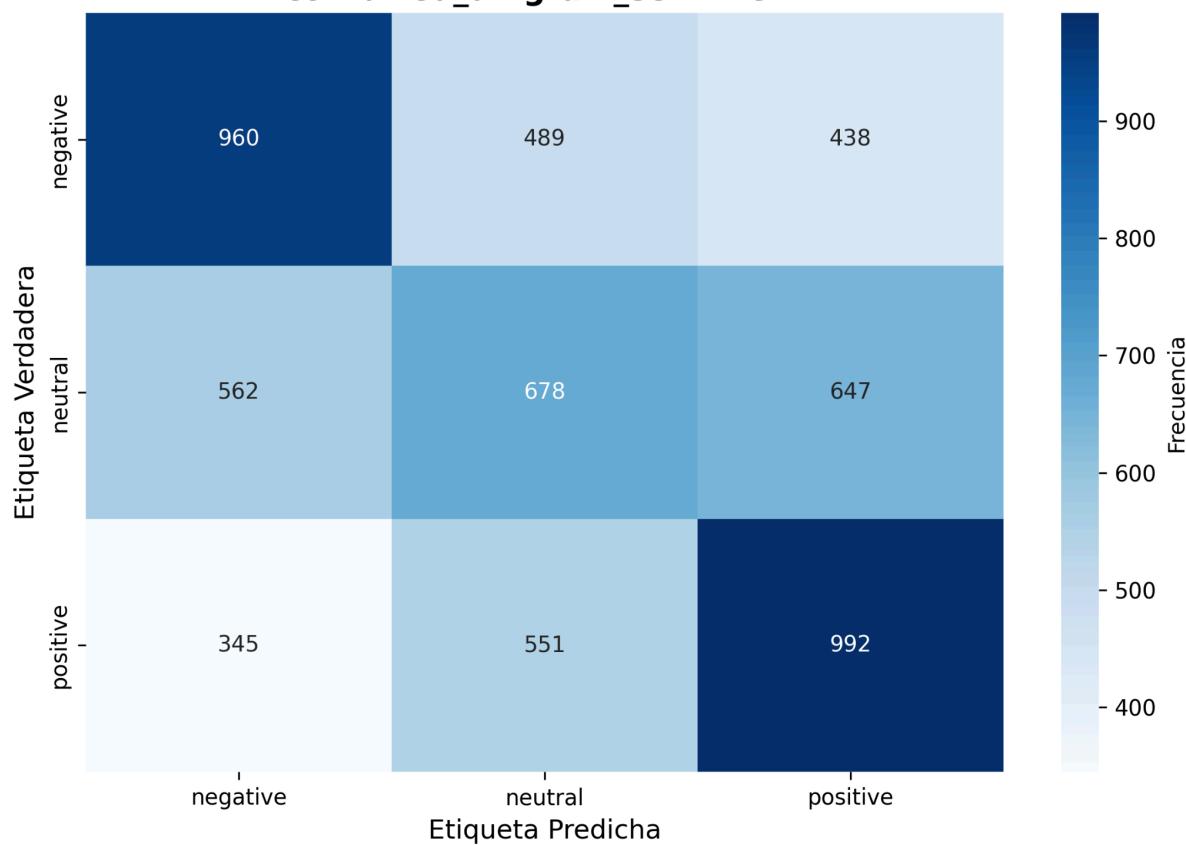




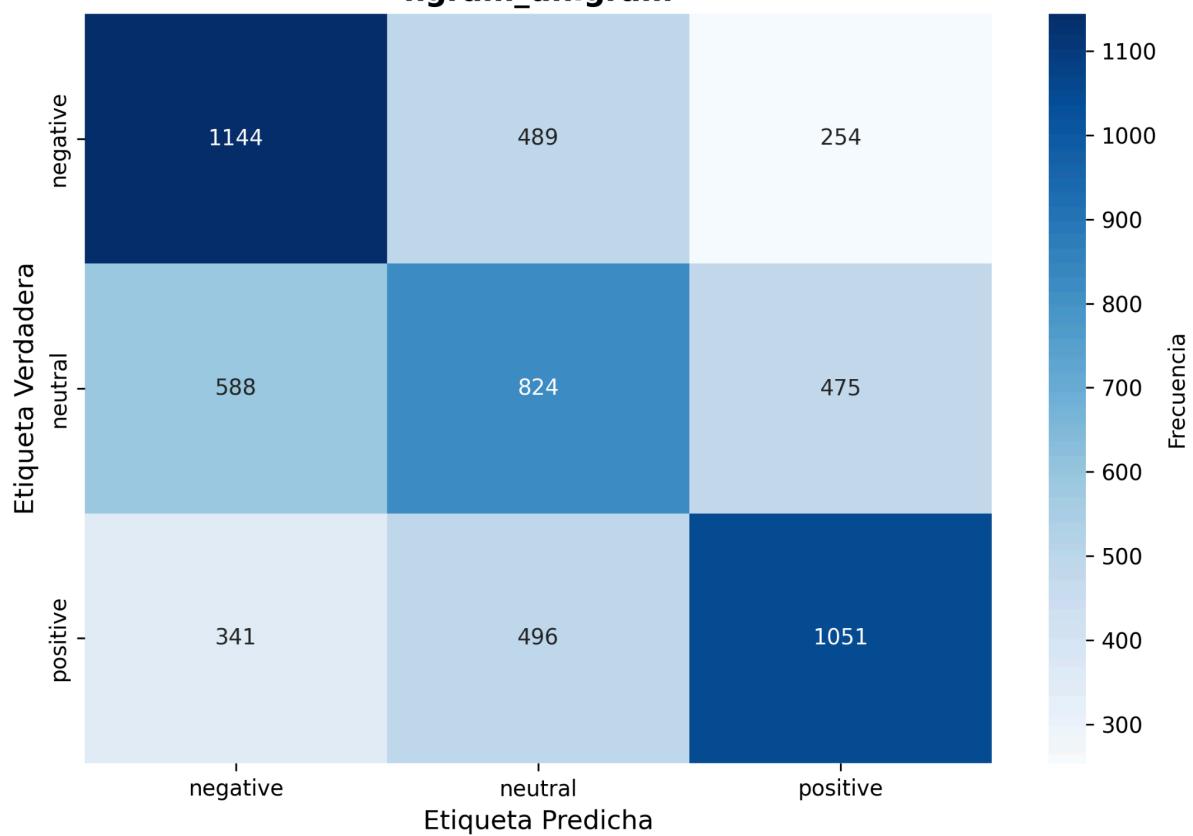
SVM (LinearSVC)
combined_unigram_sentiment



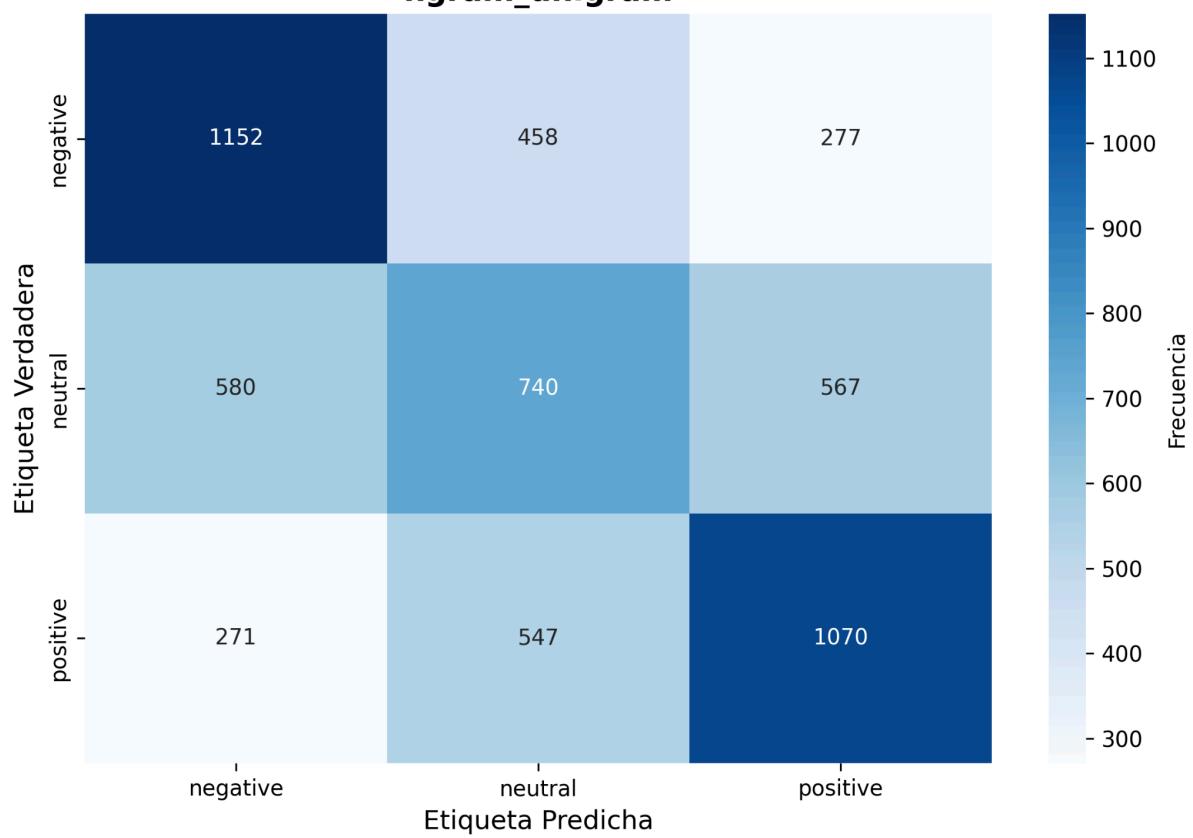
SVM (RBF)
combined_unigram_sentiment



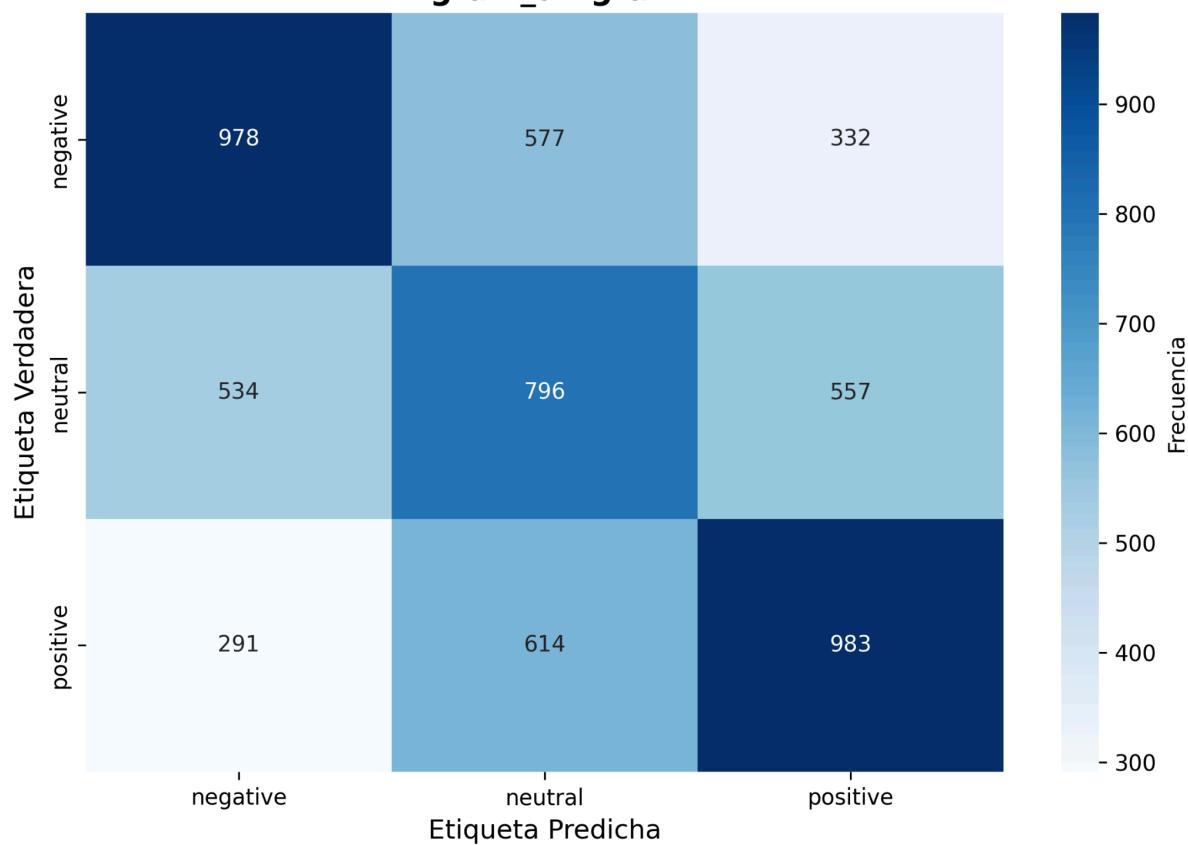
Multinomial Naive Bayes ngram_unigram



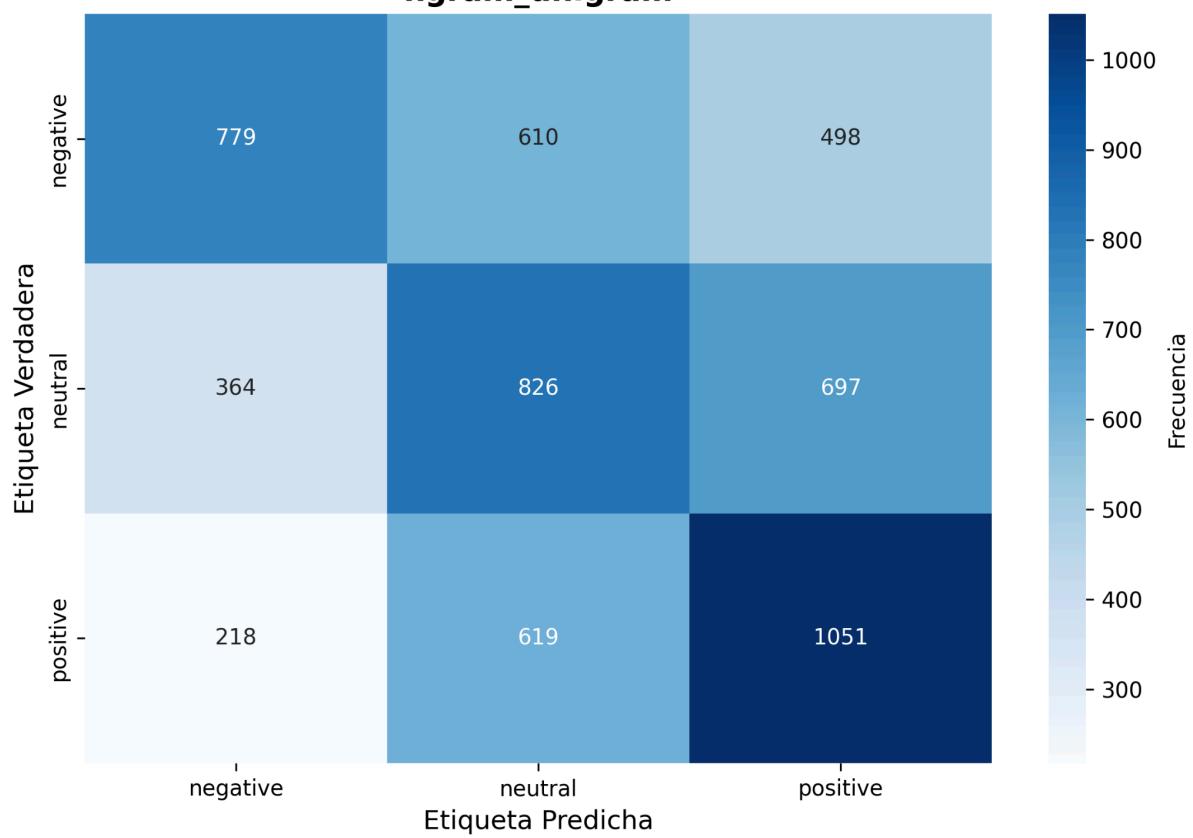
Random Forest ngram_unigram



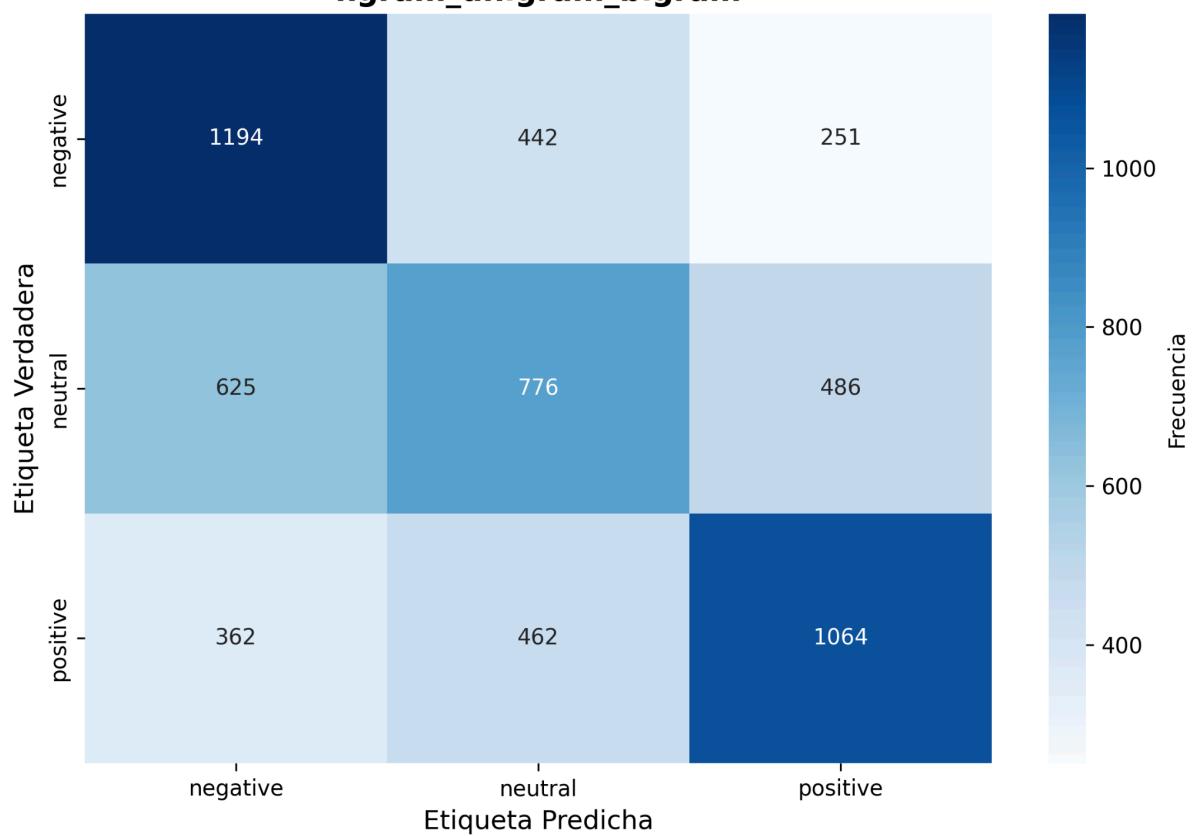
SVM (LinearSVC) ngram_unigram



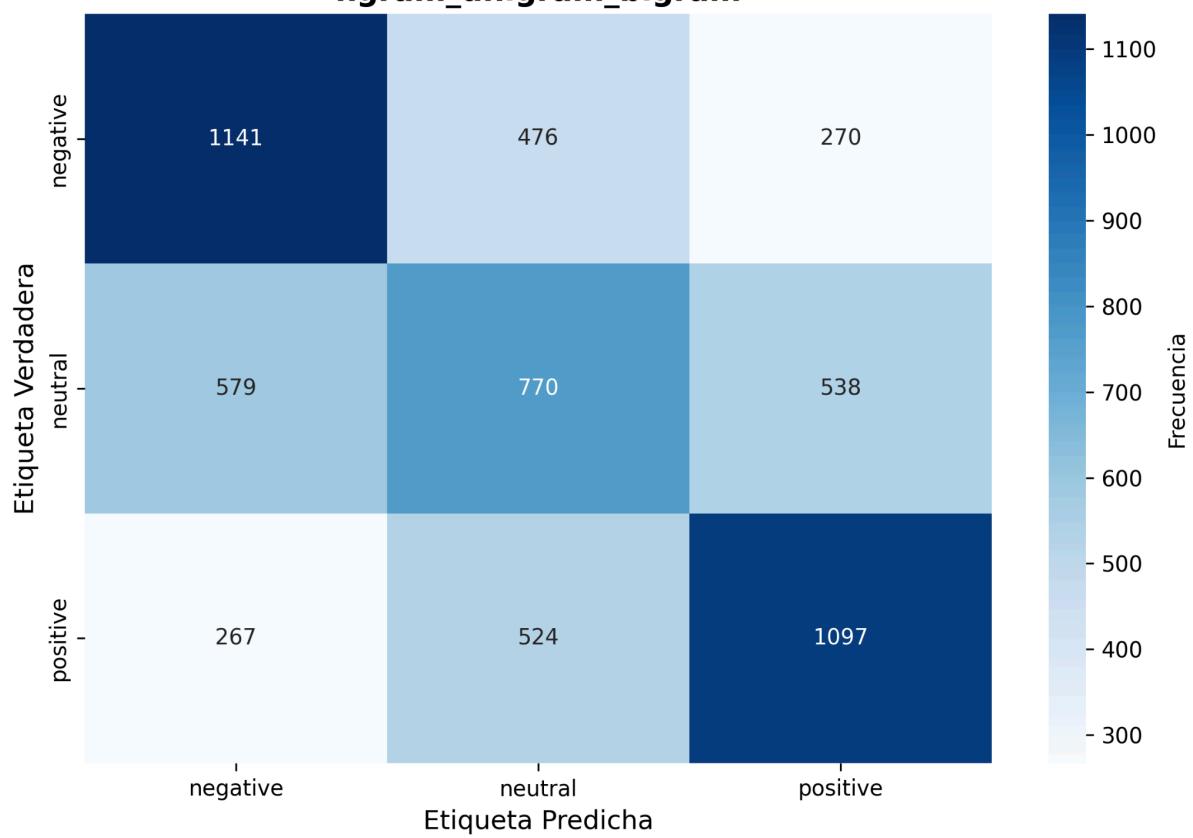
SVM (RBF)
ngram_unigram



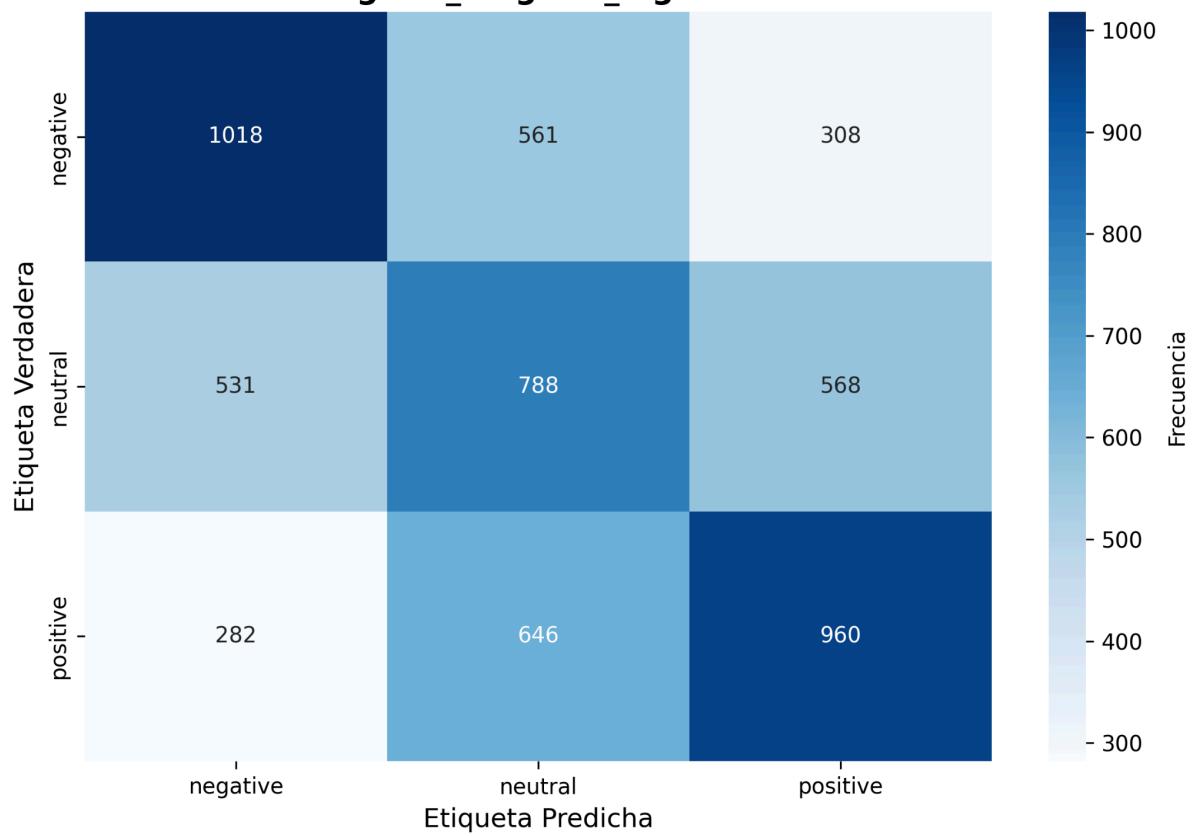
Multinomial Naive Bayes ngram_unigram_bigram



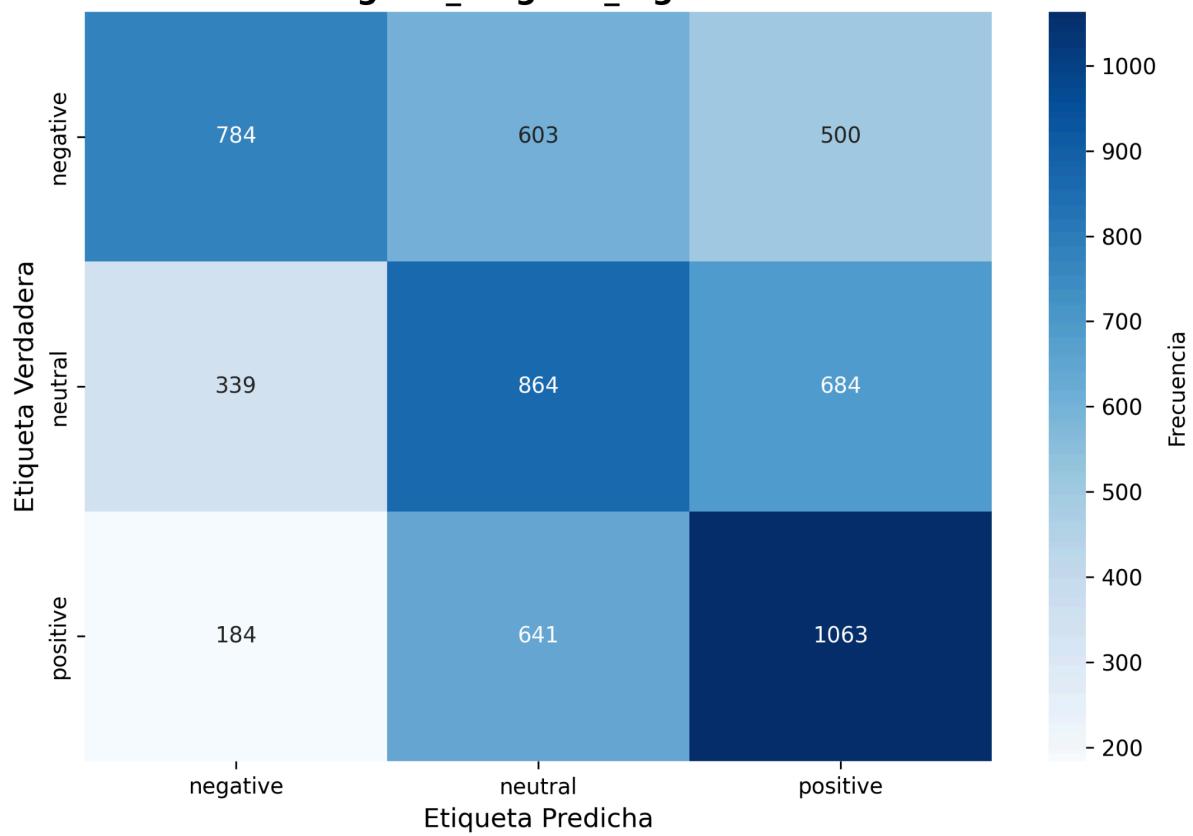
Random Forest
ngram_unigram_bigram



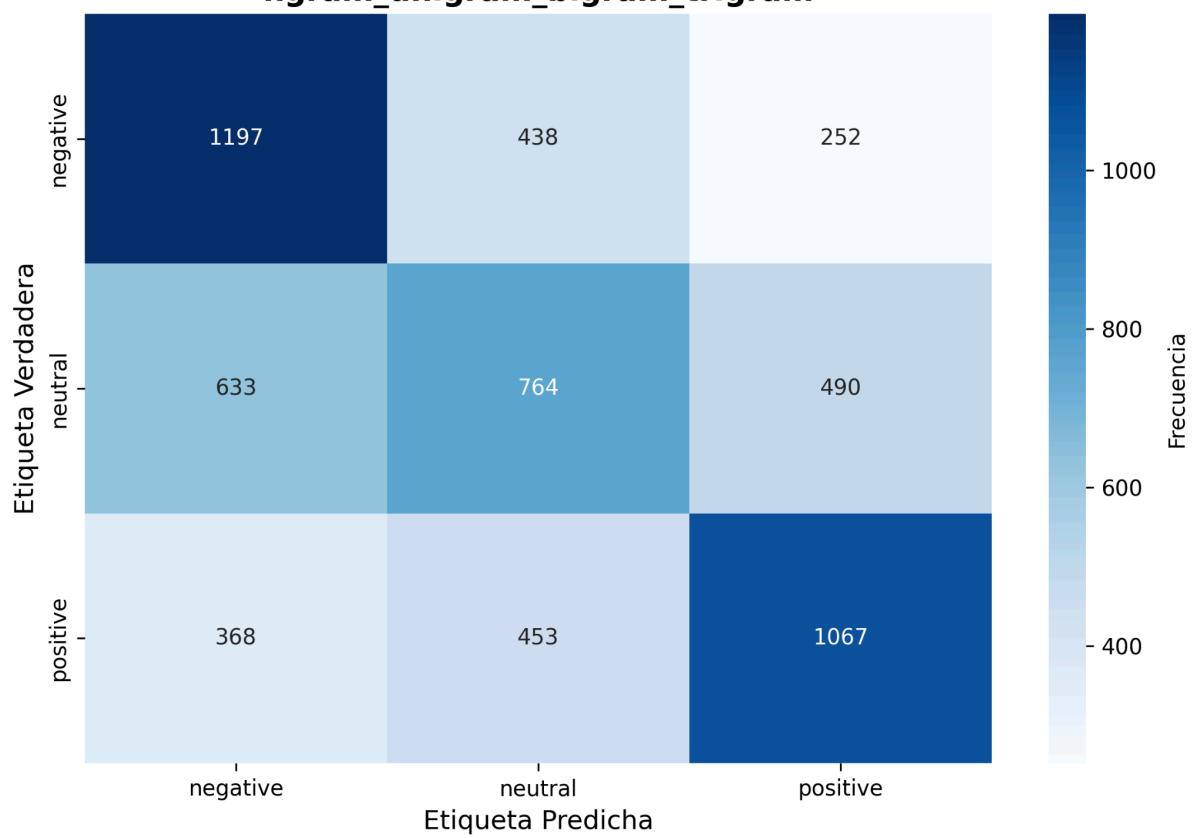
SVM (LinearSVC)
ngram_unigram_bigram



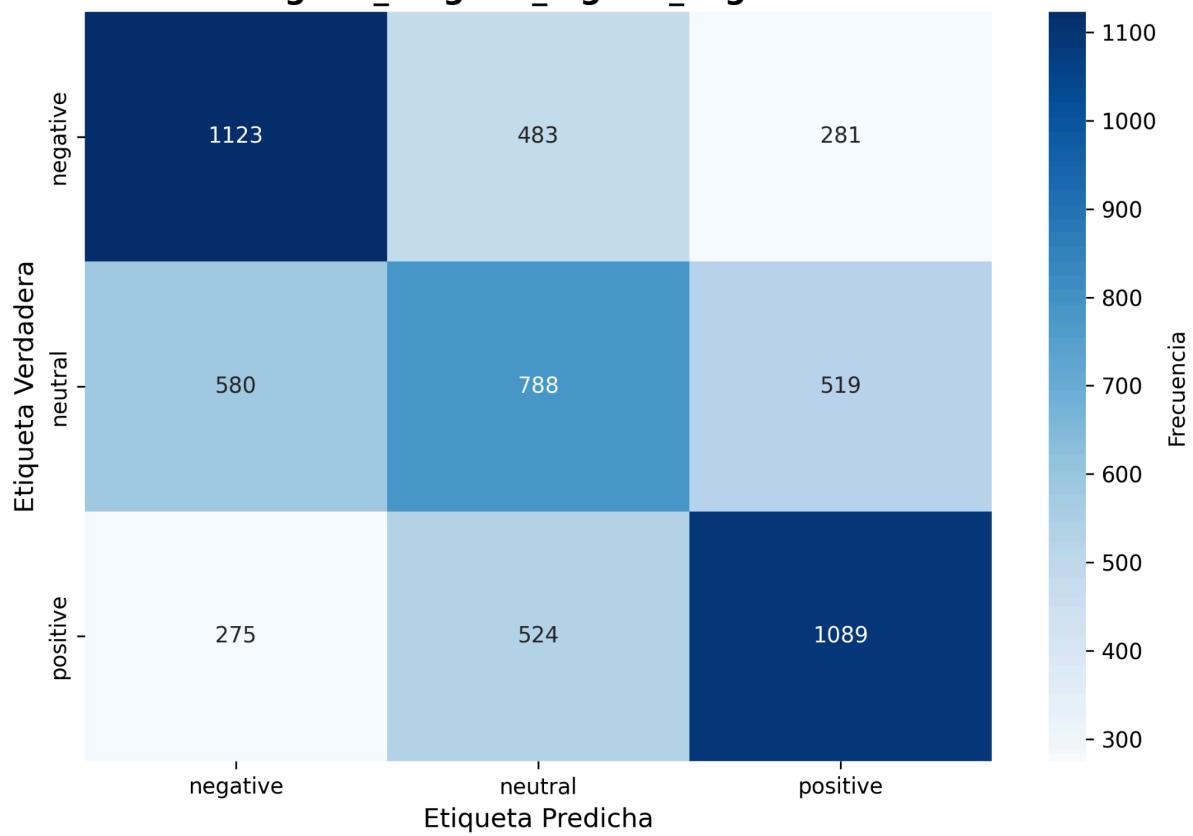
SVM (RBF)
ngram_unigram_bigram



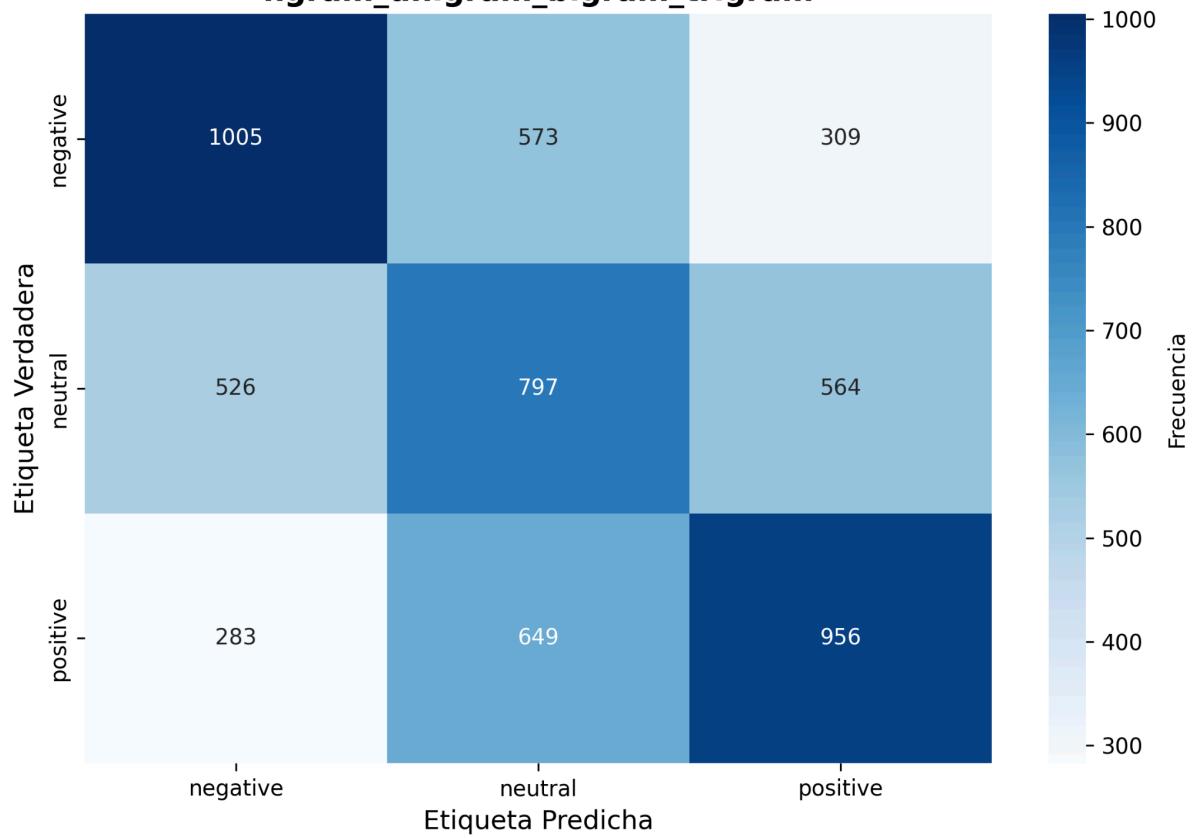
Multinomial Naive Bayes
ngram_unigram_bigram_trigram



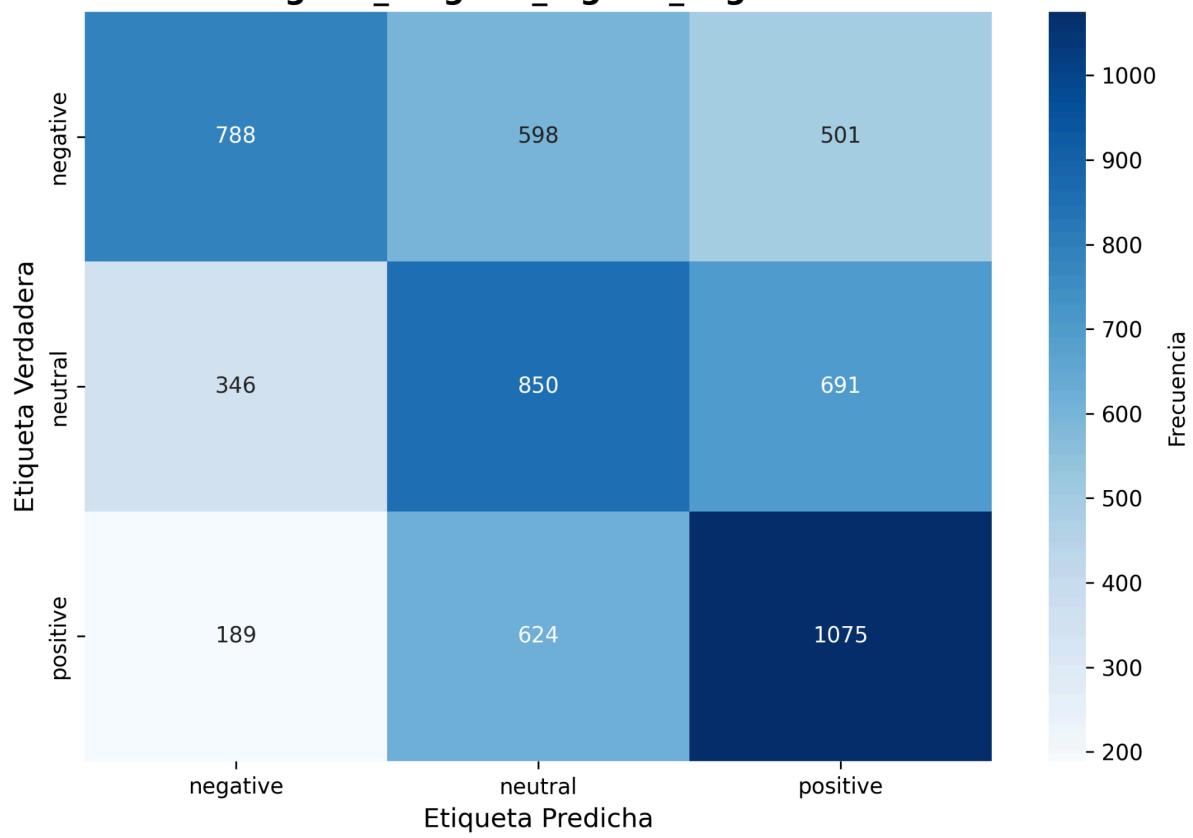
Random Forest
ngram_unigram_bigram_trigram



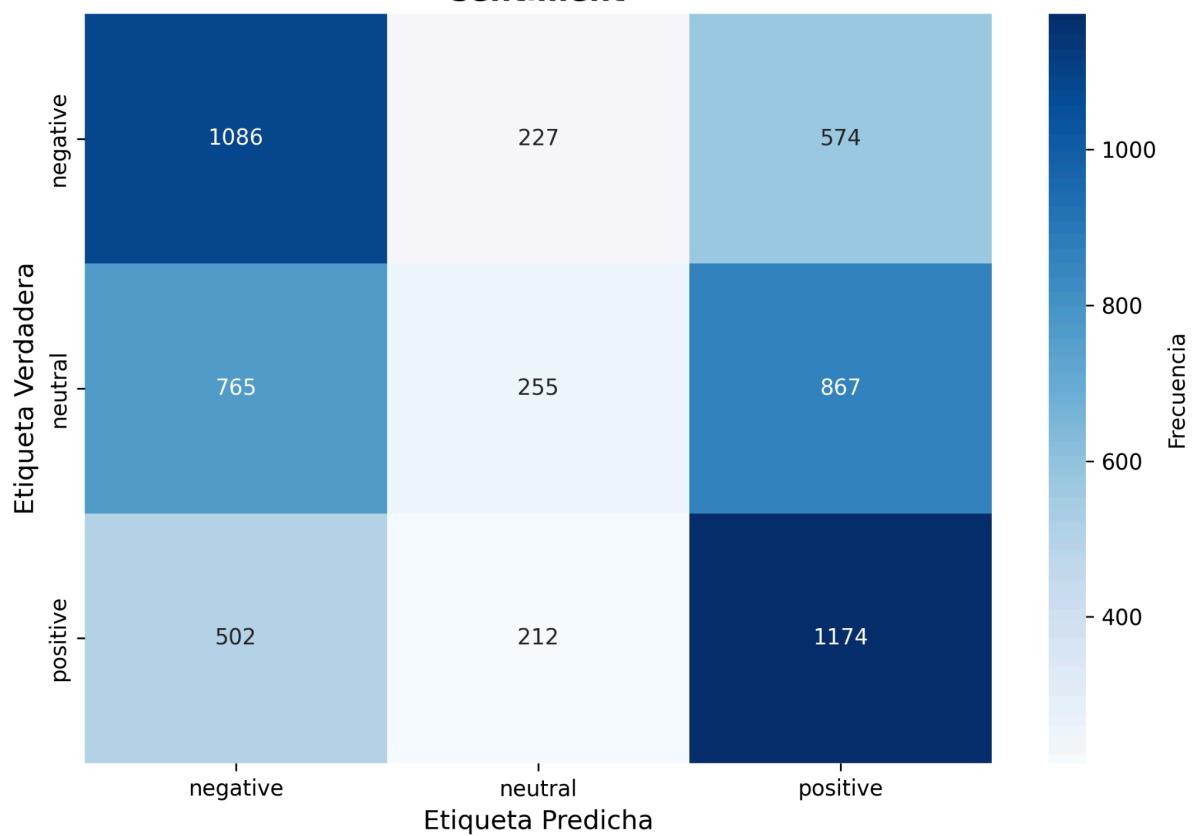
SVM (LinearSVC)
ngram_unigram_bigram_trigram

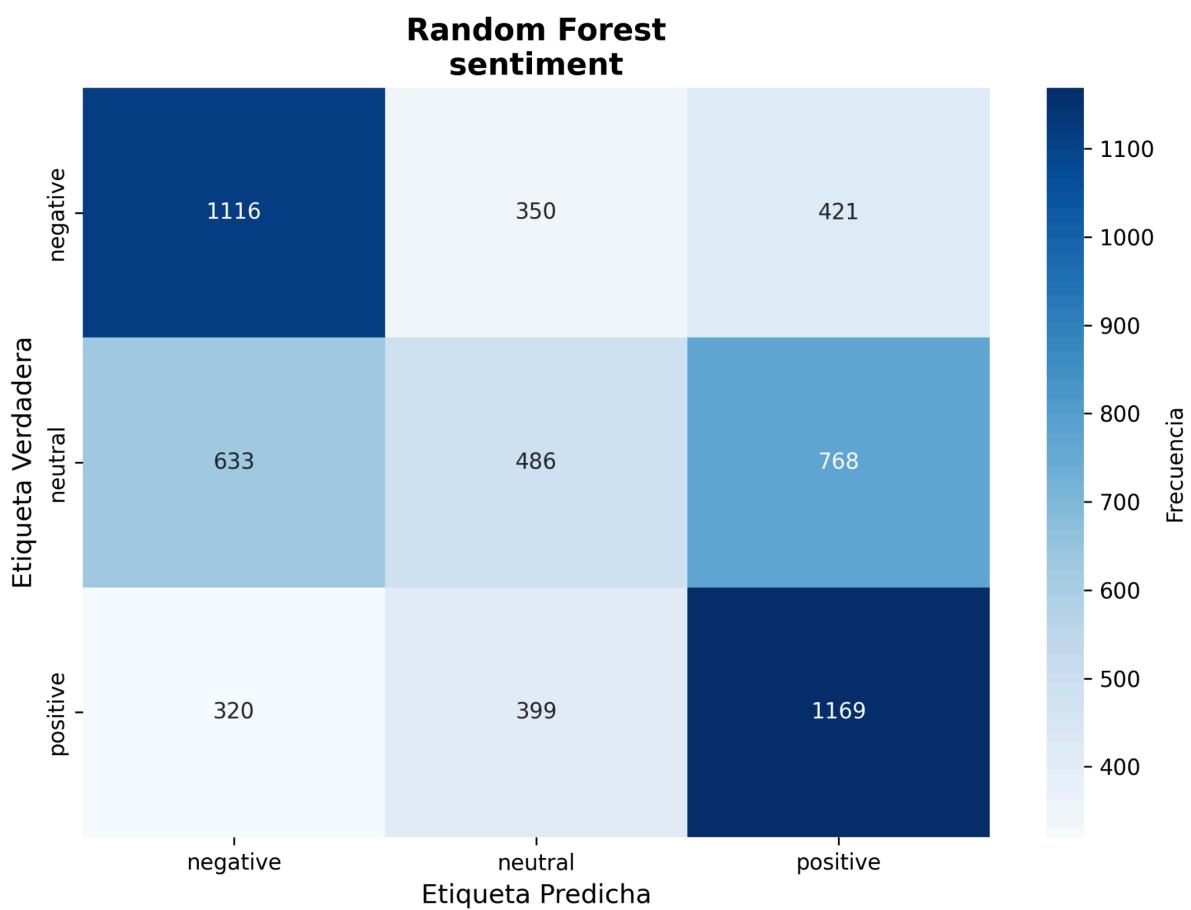


SVM (RBF)
ngram_unigram_bigram_trigram



Multinomial Naive Bayes sentiment





SVM (LinearSVC) sentiment

