Universidad Carlos III de Madrid

Curso Inteligencia Artificial en las Organizaciones

Práctica Final

FECHA: 15/12/2024
ENTREGA: ÚNICA
GRUPO: 7

Nombre: Amina Errami Maslaoui
NIA: 100472114

Nombre: Aitana Antonia Ortiz Guiño
NIA: 100472097

Nombre: Martín Portugal Gónzalez
NIA: 100472279

Nombre: Alba Vidales Casado
NIA: 100472236

ÍNDICE

Introducción

La predicción de los sentimientos en textos representa un desafío clave en el ámbito del procesamiento de lenguaje natural (NLP) y la minería de datos. Con la creciente cantidad de datos generados diariamente en plataformas como redes sociales, la capacidad de analizar y clasificar automáticamente las emociones expresadas en los textos se convierte en una herramienta fundamental para diversas aplicaciones, desde la mejora de la experiencia del cliente hasta la toma de decisiones en campañas publicitarias.

Este trabajo se centra en el uso de técnicas de minería de texto y algoritmos de aprendizaje automático para predecir los sentimientos expresados en tweets categorizados por emociones. A partir de un dataset extraído de Kaggle, que contiene miles de tweets clasificados en seis emociones principales (0: sad, 1: joy, 2: love, 3: anger, 4: fear, 5: surprise), se aplicarán procesos avanzados de análisis textual. La primera etapa del estudio empleará la herramienta RapidMiner, enfocándose en el preprocesamiento de datos mediante técnicas como la tokenización, la eliminación de stopwords y la conversión del texto en vectores utilizando métodos como TF-IDF. Posteriormente, se implementarán modelos clásicos de clasificación como métodos de árboles como Random Forest junto con modelos de Deep Learning, para explorar su efectividad en la predicción de emociones. Además, se implementará un análisis de sentimientos y se generarán n-gramas para intentar mejorar la capacidad predictiva del modelo.

En una segunda etapa, se utilizarán representaciones avanzadas del lenguaje, como Word Embeddings, que permitirán capturar relaciones semánticas y contextuales en los textos. Estas técnicas, combinadas con modelos clásicos como los anteriormente mencionados, buscarán mejorar la capacidad predictiva y ofrecer una visión más precisa de los sentimientos asociados a los tweets.

El objetivo principal de este trabajo es comparar el rendimiento de diferentes enfoques y modelos en la predicción de emociones, evaluando métricas como precisión, recall y accuracy. Este documento detalla el proceso completo, desde el preprocesamiento y la extracción de características hasta la implementación de los modelos, la validación cruzada y el análisis de resultados, proporcionando así una visión integral de las metodologías utilizadas y su impacto en la clasificación de sentimientos.

Entendimiento de negocio

1. Predicción de sentimientos en textos de redes sociales

La detección y clasificación de sentimientos en textos, especialmente en plataformas como redes sociales, es una tarea fundamental en el ámbito del análisis de datos. Las emociones expresadas en tweets, publicaciones y comentarios tienen un impacto significativo en áreas como la mercadotecnia, la atención al cliente y el análisis de la opinión pública. Entender cómo las personas se sienten y comunican en línea permite a las organizaciones tomar decisiones más informadas y adaptar sus estrategias de forma proactiva.

En el caso de los tweets, la naturaleza breve y a menudo informal del texto plantea retos importantes, como la ambigüedad semántica, el uso de jerga y abreviaturas, y la presencia de ironía o sarcasmo. Por ello, abordar este problema requiere no solo herramientas avanzadas, sino también enfoques especializados para estructurar, interpretar y analizar estos datos de manera eficiente.

El análisis de sentimientos a partir de tweets categorizados en emociones, es una tarea que puede ofrecer insights valiosos en diferentes contextos, desde campañas publicitarias hasta monitoreo de crisis. A partir de un dataset predefinido, este trabajo busca demostrar cómo la minería de texto y las técnicas de aprendizaje automático pueden revelar patrones ocultos en la comunicación humana digital.

2. Objetivo del uso de modelos para la predicción de emociones

El objetivo principal de este estudio es implementar y comparar diferentes modelos de aprendizaje automático para predecir emociones en textos cortos, utilizando como base un dataset de tweets categorizados por emociones. Inicialmente, se empleará RapidMiner para realizar el preprocesamiento textual y aplicar modelos clásicos como Random Forest. Estas técnicas permitirán establecer un punto de referencia en la clasificación de sentimientos.

Posteriormente, se integrarán enfoques más avanzados basados en Word Embeddings, como ConceptNet Numberbatch, GloVe o BERT, que capturan relaciones semánticas y contextuales más profundas entre palabras. Estas representaciones vectoriales servirán como entrada para modelos de aprendizaje profundo, como redes neuronales, que son capaces de identificar patrones complejos y relaciones no lineales en los datos.

Cada modelo será evaluado en términos de precisión, recall y F1-score, permitiendo identificar cuál es más adecuado para esta tarea. Además, se analizará la influencia de los preprocesamientos aplicados, como la tokenización, la eliminación de stopwords y la representación del texto mediante TF-IDF. Estos procesos son clave para transformar datos no estructurados en una forma adecuada para el análisis computacional.

En resumen, este trabajo no solo busca predecir emociones en textos cortos con alta precisión, sino también explorar cómo diferentes enfoques y modelos pueden complementarse para abordar los desafíos únicos de este tipo de análisis. Esto permitirá desarrollar herramientas útiles para el análisis de sentimientos en entornos reales, optimizando estrategias basadas en la interpretación de datos emocionales.

Compresión de los datos

El archivo CSV contiene datos estructurados sobre tweets en inglés, con información relevante para la predicción de sentimientos, en formato delimitado por punto y coma (;). Cada fila representa un tweet específico, organizado en los siguientes campos:

- **Text**: Texto completo del tweet. Contiene la opinión o emoción expresada por el usuario, que servirá como entrada para los modelos de análisis de sentimientos. Ejemplo: *"I can't believe how amazing this day has been!"*.
- **Label**: Número que representa la emoción asociada al tweet, categorizado de la siguiente manera:
    - 0: Tristeza (*sad*).
    - 1: Alegría (*joy*).
    - 2: Amor (*love*).
    - 3: Enojo (*anger*).
    - 4: Miedo (*fear*).
    - 5: Sorpresa (*surprise*).

    Este campo será utilizado como la variable objetivo para los modelos de aprendizaje automático.

RapidMiner

Preparación de datos

Para iniciar el proceso de preparación de datos para minería de texto, seleccionaremos los atributos relevantes del conjunto de datos: Text y Label. La columna Text contiene el contenido completo del tweet, mientras que la columna Label asigna un número que representa la emoción asociada al texto. Este formato permitirá estructurar los datos de forma óptima para el análisis y la predicción de sentimientos.

Paso 1: Creación de la Nueva Variable Label_Text

Para facilitar la comprensión de los resultados y la interpretación en herramientas como RapidMiner, se añadirá una nueva variable llamada Label_text, que convierte los valores numéricos de la columna Label en descripciones textuales de las emociones. Esta variable incluye las siguientes categorías:

- Sad
- Joy
- Love
- Anger
- Fear
- Surprise

Este paso no altera la naturaleza de los datos originales, sino que proporciona una representación más intuitiva para los análisis, especialmente durante las visualizaciones y la validación del rendimiento del modelo.

Paso 2: Selección de Atributos y Preparación de Datos

Una vez creada la columna Label_text, eliminaremos el atributo Label, dado que ya hemos sintetizado su información en Label_Text, por lo que conservaremos únicamente los siguientes atributos:

- **Text:** Campo de texto que contiene el contenido completo del tweet.
- **Label_text:** Variable categórica textual de la emoción para facilitar la interpretación del modelo.

Al tener un número reducido de atributos, nos aseguramos de que los datos sean manejables y específicos para las tareas de clasificación y minería de texto.

Paso 3: Definición de las Variables de Predicción y Texto

Configuraremos **Label_text** como la variable objetivo o *label* que deseamos predecir. La columna **Text** se utilizará como una variable regular, proporcionando el contenido textual que será analizado y procesado para extraer características clave mediante técnicas de procesamiento de lenguaje natural (NLP).

Paso 4: Conversión a Texto para Minería de Documentos

Para aplicar técnicas de minería de texto, es fundamental que el campo **Text** esté en formato de texto puro. Esto permitirá utilizar el campo en herramientas específicas de procesamiento de texto. Durante esta etapa, se llevarán a cabo las siguientes operaciones:

- **Tokenización**: División del texto en palabras o términos.
- **Eliminación de palabras irrelevantes** (*stopwords*): Para reducir el ruido en los datos.
- **Lematización o stemming**: Simplificación de palabras a su forma base.
- **Generación de vectores**: Representación numérica de palabras o frases mediante técnicas como TF-IDF o Word Embeddings.

Este proceso estructurará los datos de manera eficiente, permitiendo la implementación de modelos predictivos para la clasificación de sentimientos según las emociones definidas en las etiquetas.

Process Documents

1. Tokenización (Tokenize)

El primer paso en el proceso de minería es la **tokenización**, que consiste en dividir el texto en unidades mínimas conocidas como **tokens**. Un **token** es cualquier elemento, como una palabra o símbolo, que se considera una unidad independiente en el análisis de texto. En este caso, utilizamos un enfoque de "non-letters" que separa los tokens basándose en caracteres que no son letras, como espacios, puntuación y símbolos. Esto nos permite segmentar el texto de manera efectiva, eliminando elementos irrelevantes que puedan interferir en el análisis posterior.

2. Eliminación de Stopwords (Stopwords Removal)

Una vez tokenizado, el siguiente paso es la eliminación de **stopwords**. Las **stopwords** son palabras comunes del lenguaje, como "the," "is," "and" o "of," que suelen ser de alta frecuencia pero con poco significado semántico en la minería de texto. Aunque el texto principal está en inglés, algunas reseñas incluyen español y chino, por lo que el enfoque será eliminar solo las stopwords en inglés. Esto reducirá el "ruido" en los datos, dejando palabras más relevantes para el análisis.

3. Transformación a Minúsculas (Lowercase Transformation)

Para asegurar consistencia en el análisis, transformamos todas las palabras a **minúsculas**. Esto evita la duplicidad de tokens, donde palabras como "Love" y "love" se interpretan como distintas solo por la diferencia de mayúsculas. Con este paso, se mejora la homogeneidad y se reduce la cantidad de variaciones en los datos.

4. Filtrado de Tokens por Longitud

A continuación, aplicamos un **filtro de longitud de tokens** para descartar aquellos demasiado cortos o demasiado largos, lo cual ayuda a eliminar ruido adicional en los datos. En este caso, establecemos un **mínimo de 3 caracteres** y un **máximo de 25 caracteres**. Esto excluye tokens irrelevantes (por ejemplo, letras individuales o palabras extremadamente largas que pueden ser errores de escritura) y mejora la calidad de los datos textuales para el análisis.

5. Stemming (Stem - Snowball)

Finalmente, utilizamos un proceso llamado **stemming** para reducir las palabras a sus raíces o formas básicas. El **algoritmo Snowball** se aplica en inglés y permite reducir palabras derivadas a una forma común, como convertir "taking" en "take" o "slightly" en "slight". Esto ayuda a agrupar términos similares y facilita la detección de patrones, ya que palabras con la misma raíz se tratarán como equivalentes. El stemming en inglés se adapta mejor a la mayoría del texto de las reseñas, aunque en algunos casos puede no tener el mismo efecto en otras lenguas.

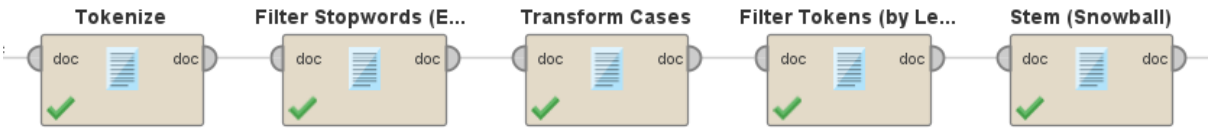


Imagen. RapidMiner Process

Prune Method

Además de los diversos procedimientos aplicados en el paso de **Process Documents**, hemos implementado un método de poda (**prune method**) con el objetivo de eliminar términos que no aporten valor significativo al análisis y, al mismo tiempo, optimizar el rendimiento y los tiempos de ejecución del proyecto. Este enfoque resulta esencial, sobre todo cuando trabajamos con grandes volúmenes de texto, ya que reduce la cantidad de datos procesados sin sacrificar información relevante.

El **prune method** es una técnica utilizada para reducir la dimensión de la matriz de términos, eliminando términos que no cumplen con ciertos criterios de frecuencia. Al reducir el número de términos, mejoramos la eficiencia de los procesos posteriores (por ejemplo, modelos de machine learning) y evitamos problemas de almacenamiento o uso excesivo de memoria. Esta técnica es particularmente útil en análisis de texto, donde muchas palabras pueden ser poco frecuentes o demasiado comunes, contribuyendo poco a la diferenciación entre documentos.

Para implementar el prune method, hemos configurado un rango de poda porcentual basado en el umbral de frecuencia de los términos en los documentos, con los siguientes límites:

- **Límite inferior: 0,3%**. Este umbral elimina términos que aparecen en menos del 0,3% de los tweets. La presencia de estos términos es tan baja que se considera que no aportan valor significativo al análisis general, ya que su contribución al significado global es mínima y no justifican el espacio de procesamiento.
- **Límite superior: 90%**. Este umbral elimina términos que aparecen en más del 90% de los tweets, ya que suelen ser palabras muy comunes o generales que no diferencian una tweet de otro. Estos términos son menos útiles para la segmentación o clasificación, y mantenerlos aumentaría innecesariamente el tamaño de la matriz de términos.

En este proyecto, hemos optado por configurar el prune method para priorizar la optimización por **memoria**. Esta configuración permite:

- **Reducir el uso de memoria** al almacenar sólo los términos relevantes en el conjunto de datos, evitando una sobrecarga de términos innecesarios.

- **Mejorar la eficiencia de procesamiento**, ya que un conjunto de datos más pequeño requiere menos tiempo para ejecutarse en etapas posteriores, como en la generación de la matriz TF-IDF o en el entrenamiento de modelos predictivos.
- **Mantener la precisión** del análisis al eliminar datos redundantes o poco informativos que aportarían poco a la extracción de insights o patrones.

Lista de palabras

Una primera etapa del análisis consiste en generar una lista de palabras ordenadas según la cantidad de veces que aparecen en el conjunto de tweets. Esta frecuencia permite identificar términos recurrentes que pueden revelar temas y patrones comunes en las emociones expresadas por los usuarios.

Dado que los tweets están categorizados en diferentes emociones mediante el campo **Label_Text**, también es posible separar la frecuencia de palabras por emoción. Esta clasificación muestra qué palabras o frases aparecen más comúnmente en cada categoría, ayudando a detectar tendencias específicas.

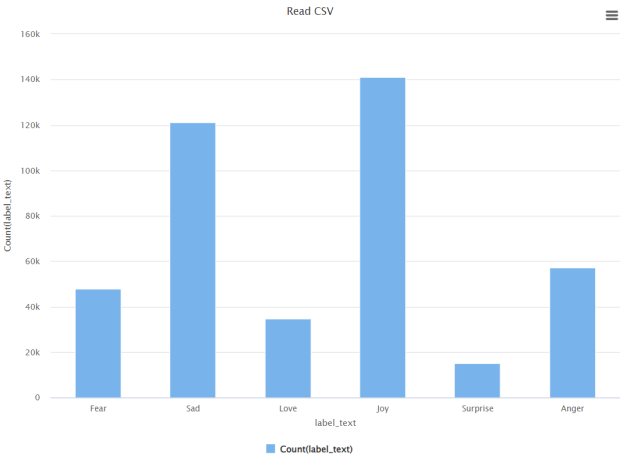
Por ejemplo, en tweets etiquetados con *Joy*, podrían encontrarse palabras como "happy," "celebrate," o "thankful," mientras que en los asociados con *Anger* podrían predominar términos como "hate," "bad," o "disaster." Esta información es clave para comprender los factores que contribuyen a cada emoción, proporcionando una base para mejorar los modelos de predicción y extracción de sentimientos.

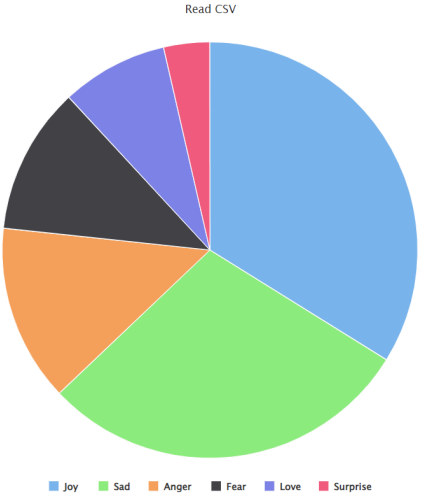
Matriz de términos (TF-IDF)

Al aplicar **TF-IDF** (Term Frequency-Inverse Document Frequency) a la matriz de términos, transformamos los datos iniciales para resaltar los términos más relevantes en cada documento (en este caso, cada tweet). Este enfoque permite no solo contabilizar la frecuencia de palabras, sino también evaluar su importancia relativa en función del contexto general, lo que ayuda a reducir el impacto de palabras muy comunes que no aportan información significativa (como "the" o "a").

Uso del Operador Sample

Una parte fundamental en esta etapa de preparación de los datos ha sido el balanceo de las clases. Esto se debe a que, tras realizar el proceso descrito anteriormente, las categorías representadas en el campo **Label_Text** están desbalanceadas. Se obtuvo la siguiente distribución inicial de datos:





Este desequilibrio hace necesario aplicar técnicas de balanceo de datos, como sobremuestreo, submuestreo o algoritmos que manejan desbalanceos, para asegurar que los modelos no favorezcan una clase en detrimento de las demás. Para abordar el problema del desbalance, se utilizó exclusivamente el operador **Sample** en RapidMiner. Este operador permite especificar el número de instancias por clase en el conjunto de datos, lo cual es fundamental para equilibrar las categorías y optimizar el rendimiento de los modelos predictivos.

El enfoque consistió en establecer un límite de **6000** instancias por clase. Este número fue elegido cuidadosamente para mantener un balance entre la diversidad de datos y la capacidad computacional, ya que los modelos se benefician de conjuntos representativos y manejables.

Con esta configuración, se redujo el tamaño total del conjunto de datos a aproximadamente **36000 instancias** (6 clases x 36000 instancias por clase). Esta reducción no solo ayudó a equilibrar las clases, sino que también facilitó el procesamiento y optimización de los algoritmos utilizados en etapas posteriores, permitiendo una mejora en la eficiencia computacional sin comprometer la calidad del análisis.

Modelos

A la hora de plantear los modelos decidimos crear 4 modelos de Deep Learning y 1 modelo de Random Forest, ya que estas técnicas fueron las que mejores resultados obtuvimos en la práctica 2, por lo tanto lo usaremos como base para desarrollar nuevos modelos.

Como el objetivo de la práctica es explorar cómo pueden afectar distintos preprocesamientos de los datos al modelo para ello utilizaremos un modelo de deep learning con los mismos parámetros para los 4 modelos pero cambiando el preprocesamiento.

Por otro lado contaremos con un único modelo de Random Forest con los hiperparametros obtenidos en la práctica 2 para darle más variedad a los modelos.

MODELO	Nº FOLDS	ACTIVATION	SAMPLING TYPE	Nº DE CAPAS	Nº DE NEURONAS	EPOCHS
Modelo Deep Learning	20	Rectifier	Stratified Sampling	2	25/15	20

MODELO	Nº FOLDS	Nº TREES	SAMPLING TYPE	CRITERION	MAXIMAL DEPTH	VOTING STRATEGY
Modelo 5	20	100	Stratified Sampling	Information Gain	50	Confidence Vote

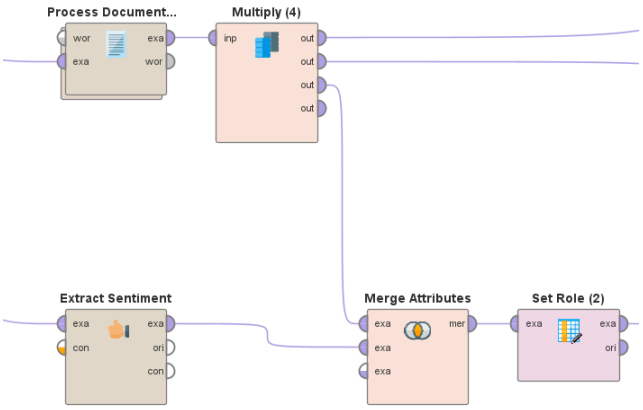
Modelo 1 y Modelo 5

El preprocesamiento de los datos para estos dos modelos contarán con la configuración más básica con tokenize, filtrado de stopwords, normalización de las palabras a lower case, filtro de longitud de los tokens entre 2 y 25 y stemming para reducir las palabras a sus raíces o formas básicas.

Modelo 2

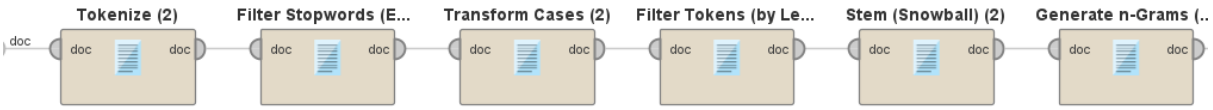
Este modelo suma al preprocesamiento anterior el análisis de sentimiento, para que asigne una puntuación de sentimiento al texto indicando si el contenido es positivo, negativo o neutral mediante el modelo VADER en la cual se asigna una puntuación entre -1 y 1, donde las puntuaciones negativas indican un sentimiento negativa y viceversa para las positivas

Esta puntuación, o **sentiment score**, se obtendrá como un nuevo atributo y lo seleccionaremos como único atributo para esta rama del proceso utilizando un filtro llamado Select Attribute. Esto permitirá aislar la puntuación de sentimiento para su procesamiento separado.



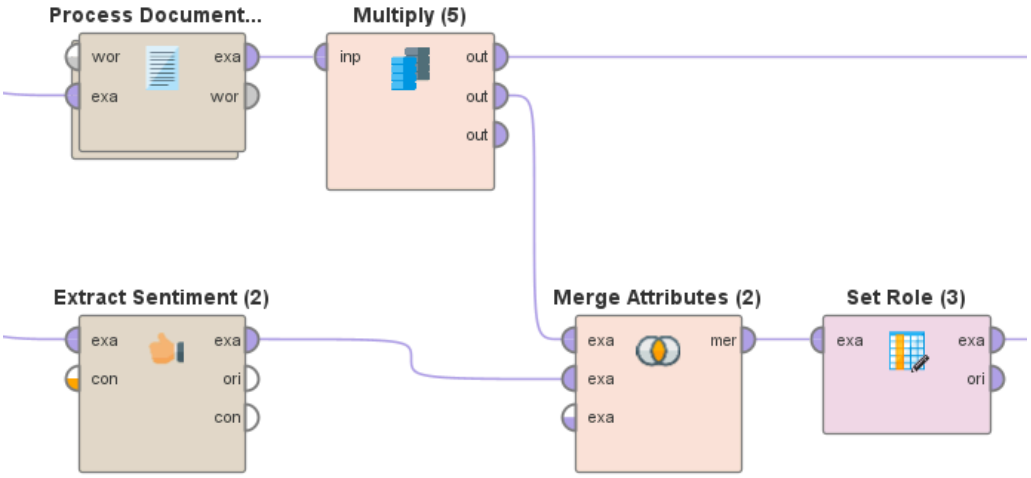
Modelo 3

Para este modelo añadimos la creación de n-gramas sobre el procesamiento original. Utilizaremos tanto **unigramas** (una sola palabra) como **bigramas** (combinaciones de dos palabras) para capturar términos individuales y frases clave de las reseñas. Este enfoque permite identificar patrones de expresión que podrían indicar aspectos positivos o negativos específicos en el servicio, y se integra bien con los datos de sentimiento, permitiendo un análisis más detallado de los términos que contribuyen a una percepción positiva o negativa.



Modelo 4

Y para este último modelo decidimos introducir el uso de n-gramas con el extract sentiment. Todo ello con el objetivo de valorar si realmente estos cambios en los preprocesos de los datos mejoraron el entrenamiento del modelo.



Resultados

El accuracy de los modelos obtenidos:

MODELOS	ACCURACY
Modelo 1	75.57±1.01%
Modelo 2	80.17±1.08%
Modelo 3	76.4±1.12%
Modelo 4	80.88±1.04%
Modelo 5	75.03±1.02%

Y aquí la comparación T-Test entre los distintos modelos.

A	B	C	D	E	F
	0.756 +/- 0.010	0.750 +/- 0.010	0.802 +/- 0.011	0.764 +/- 0.011	0.809 +/- 0.010
0.756 +/- 0.010		0.105	0.000	0.018	0.000
0.750 +/- 0.010			0.000	0.000	0.000
0.802 +/- 0.011				0.000	0.042
0.764 +/- 0.011					0.000
0.809 +/- 0.010					

Con estas dos tablas observamos que los mejores modelos son aquellos que emplean el extract sentiment (Modelo 2 y Modelo 4), y aunque los resultados que aplican los n-gramas son ligeramente superiores a aquellos que no lo hacen, no se trata de un factor diferenciador a la hora de entrenar el modelo.

Matrices de Confusión

Modelo 1

Deep Learning 4. Performance

	true Joy	true Sad	true Love	true Anger	true Surprise	true Fear	class precision
pred. Joy	3240	1262	319	347	49	281	58.93%
pred. Sad	1658	3842	223	651	64	514	55.26%
pred. Love	703	211	5305	86	37	54	82.94%
pred. Anger	120	379	53	4652	36	199	85.53%
pred. Surprise	169	76	45	42	5725	512	87.15%
pred. Fear	110	230	55	222	89	4440	86.28%
class recall	54.00%	64.03%	88.42%	77.53%	95.42%	74.00%	

Modelo 2

Deep Learning 6. Performance 3

	true Joy	true Sad	true Love	true Anger	true Surprise	true Fear	class precision
pred. Joy	4344	590	450	158	53	143	75.71%
pred. Sad	566	4500	141	879	87	672	65.74%
pred. Love	691	186	5246	68	34	49	83.61%
pred. Anger	117	417	61	4642	33	206	84.77%
pred. Surprise	173	76	53	40	5711	512	86.99%
pred. Fear	109	231	49	213	82	4418	86.59%
class recall	72.40%	75.00%	87.43%	77.37%	95.18%	73.63%	

Si empezamos revisando estos dos modelos, recordemos que son aquellos con el preproceso base sin uso de n-gramas. Observamos que el accuracy del modelo sin extract sentiment es notablemente menor que el modelo que lo usa, pero si nos enfocamos en las predicciones de cada clase, nos daremos cuenta de que las emociones de Amor, Enojo, Sorpresa y Miedo mantienen una precisión y un recall bastante parecido entre los dos modelos. Sin embargo las clases de felicidad y tristeza se ven mejoradas con un aumento del recall del 54 al 72% y del 64 al 75% respectivamente. Por lo que deducimos que el extract sentiment influye especialmente en estas dos emociones.

Modelo 3

Deep Learning 7. Performance 4

	true Joy	true Sad	true Love	true Anger	true Surprise	true Fear	class precision
pred. Joy	3313	1206	317	308	73	284	60.23%
pred. Sad	1650	3975	222	725	59	540	55.43%
pred. Love	659	180	5346	69	30	43	84.50%
pred. Anger	119	362	45	4690	24	193	86.32%
pred. Surprise	157	61	37	27	5752	511	87.88%
pred. Fear	102	216	33	181	62	4429	88.17%
class recall	55.22%	66.25%	89.10%	78.17%	95.87%	73.82%	

Modelo 4

Deep Learning 8. Performance 5

	true Joy	true Sad	true Love	true Anger	true Surprise	true Fear	class precision
pred. Joy	4405	559	487	157	67	152	75.60%
pred. Sad	632	4600	145	871	83	681	65.60%
pred. Love	608	151	5259	64	24	40	85.57%
pred. Anger	100	395	36	4672	19	160	86.81%
pred. Surprise	159	62	33	30	5721	509	87.83%
pred. Fear	96	233	40	206	86	4458	87.09%
class recall	73.42%	76.67%	87.65%	77.87%	95.35%	74.30%	

En los modelos 3 y 4 la principal novedad es la incorporación de bi-gramas en los datos, sin embargo al contrastarlos con los modelos sin n-gramas no se observa una mejoría significativa en las métricas de evaluación, de hecho son bastantes similares. Esto da a entender que para nuestro modelo la importancia recae más sobre el extract sentiment que sobre el uso de bigramas.

Modelo 5

Random Forest. Performance 11

	true Joy	true Sad	true Love	true Anger	true Surprise	true Fear	class precision
pred. Joy	2177	469	94	102	11	99	73.75%
pred. Sad	2617	4601	266	889	78	688	50.34%
pred. Love	824	276	5486	183	20	114	79.47%
pred. Anger	111	347	37	4557	6	209	86.52%
pred. Surprise	177	87	77	72	5862	561	85.75%
pred. Fear	94	220	40	197	23	4329	88.29%
class recall	36.28%	76.68%	91.43%	75.95%	97.70%	72.15%	

Y en este último modelo donde el accuracy era similar al primer modelo vemos que las clase Alegría está mucho más desbalanceada en el recall, pero por otra parte consigue que emociones como Amor o Sopresa las clasifique bastante bien, mejor que ninguno de los otros modelos. Seguramente si hubieses añadido el extract sentiment hubiese clasificado mejor Alegría y Tristeza.

Implementación en Python

Tras desarrollar la limpieza de los datos y la creación de modelos para la clasificación de emociones, el enfoque se dividió en varias etapas principales:

Preparación de los datos

1. Lectura y filtrado inicial del conjunto de datos

Se utilizó un archivo CSV que contenía textos etiquetados con emociones (campo label_text) para su procesamiento, el mismo archivo que utilizado en el RapidMiner. Así mismo recibió la misma limitación de cantidad de registros por clase a un máximo de 5000. Todo ello para conseguir una cantidad de datos suficientemente grande como para que el modelo identifique las características de los patrones pero con un tamaño manejable para entrenar los modelos y buscar los hiperparametros.

Se ha utilizado la misma cantidad de datos por emoción para tener un dataset correctamente balanceado.

2. División de datos

Dividir los datos originales en dos subconjuntos:

- **Train**: Utilizado para entrenar el modelo y ajustar los parámetros.
- **Test**: Reservado exclusivamente para evaluar el rendimiento del modelo en datos que no ha visto antes, simulando situaciones del mundo real.

⅔ de los datos irán a train (20.000) y un ⅓ (10.000) irán a test, repartido a partes iguales para cada una de las clases.

3. Limpieza y preprocesamiento del texto

El texto pasó por un proceso de **normalización**, que incluyó:

- Conversión a minúsculas, evitando así posibles errores en la escritura de palabras reduciendo así el tamaño del dataset y evitando la duplicación de palabras.
- Tokenización usando nltk.wordpunct_tokenize, en la cual se divide el texto en unidades más pequeñas llamadas **tokens** (normalmente palabras o signos de puntuación). Todo ello permite analizar el texto a nivel de palabras y subpalabras permitiendo otros cambios como eliminación de stopwords o lemanización.
- Eliminación de **stopwords** (palabras comunes que no aportan significado relevante al análisis, como "the" o "and"), al tratarse de un dataset en ingles utilizamos el corpus NLTK en ingles.
- Retención únicamente de caracteres alfanuméricos, para centrarse exclusivamente en letras, eliminando así signos de puntuación o caracteres especiales evitando ruido a la hora de centrarse en la clasificación de emociones.
- Lematización basada en la parte del discurso (POS) para reducir las palabras a su forma base, utilizando WordNetLemmatizer. Esto se hace para reducir la redundancia del vocabulario, ayudando a identificar el significado real de las palabras en vez de sus formas flexionadas.

- En el caso específico de la librería utilizada NLTK se crean etiquetas POS específicas para cada tipo de palabra. J para adjetivo, N para nombre, V para verbo y R para adverbio.

Word Embeddings

Un **Word Embedding** es una técnica utilizada en procesamiento de lenguaje natural (**NLP**) para representar palabras en un espacio vectorial de características de dimensiones fijas. En lugar de tratar las palabras como entidades independientes (como en las representaciones de bolsa de palabras o TF-IDF), los embeddings capturan relaciones semánticas y contextuales entre las palabras, codificándolas como vectores densos.

Los Word Embeddings se caracterizan por dar representaciones densas ya que la mayoría de los valores no son 0, frente a técnicas tradicionales como bolsas de palabras donde las representaciones pueden ser muy dispersas. Además capturan las relaciones entre palabras con significados similares tienen vectores cercanos en el espacio vectorial.

Las ventajas del uso de Word Embeddings es capturar las relaciones semánticas entre palabras de manera más profunda, permite reducir la dimensionalidad del texto ya que son vectores de dimensiones fijas, siendo esto muy útil en procesamiento de grandes vocabularios y permiten un entrenamiento más eficiente al ser más densos y compactos que otras técnicas.

Para ello hemos decidido utilizar 3 embeddings (ConceptNet Cumberbatch, GloVe y BERT) preentrenados con distintos enfoques, características y corpus para apreciar las diferencias en rendimiento.

1. ConceptNet Numberbatch

ConceptNet Numberbatch es un modelo de word embeddings que se basa en la base de conocimiento semántica ConceptNet, una red diseñada para capturar relaciones conceptuales entre palabras y frases, tales como "es un", "parte de", "causa", entre otras. Este enfoque va más allá de las simples asociaciones basadas en corpus textuales al integrar relaciones explícitas que se encuentran en ConceptNet, enriquecidas con información adicional obtenida de corpora textuales y otros modelos de embeddings.

Estos embeddings son de tipo estático, lo que significa que cada oracion o concepto tiene una representación fija, independiente del contexto en el que aparece y trabaja en un espacio vectorial de 300 dimensiones. Sin embargo, su enfoque distintivo radica en su capacidad para incorporar conocimiento explícito de relaciones semánticas, lo que lo convierte en una herramienta ideal para tareas que requieren razonamiento estructurado o comprensión conceptual más profunda.

Una de sus principales fortalezas es su habilidad para capturar relaciones conceptuales explícitas, lo que lo hace adecuado para aplicaciones como sistemas expertos, chatbots y tareas que requieren razonamiento lógico. Esta fortaleza de no solo depender de corpus textuales permite representar de manera efectiva conceptos menos comunes o aquellos que no aparecen con frecuencia en los datos de entrenamiento.

No obstante, ConceptNet Numberbatch tiene limitaciones inherentes a su naturaleza estática. Al no ser sensible al contexto dinámico en el que las palabras aparecen dentro de frases o párrafos, no puede diferenciar entre los diferentes significados de una palabra polisémica. Además, aunque su cobertura de vocabulario es amplia, puede ser menor en comparación con modelos basados puramente en corpus textuales masivos.

2. GloVe (Global Vectors for Word Representation)

GloVe (Global Vectors for Word Representation) es un modelo de embeddings estáticos diseñado para capturar relaciones semánticas y asociaciones contextuales entre palabras. Fue entrenado en corpus textuales extremadamente grandes, como Wikipedia y Common Crawl, lo que le permite ofrecer una amplia cobertura y precisión en sus representaciones. GloVe se basa en una técnica de factorización de matrices que analiza las coocurrencias de palabras en un corpus, utilizando esta información para generar vectores densos que codifican relaciones semánticas y sintácticas de manera compacta.

Cada palabra es representada por un único vector fijo, independientemente del contexto en el que aparezca. Esto lo convierte en un modelo eficiente y directo para diversas tareas de procesamiento de lenguaje natural, especialmente aquellas que requieren análisis semántico básico. Una característica distintiva de GloVe es su capacidad para capturar relaciones vectoriales de palabras que reflejan patrones semánticos y analógicos.

Sin embargo, como modelo estático, GloVe tiene limitaciones importantes. Al representar cada oración con un único vector fijo, no puede distinguir entre los diferentes significados que una palabra puede tener según su contexto, misma limitación que comparte con el modelo de ConcepNet. Esto lo hace menos adecuado para tareas que requieren desambiguación semántica o análisis detallado de palabras polisémicas. Además, GloVe no utiliza relaciones explícitas entre conceptos, ya que se basa exclusivamente en estadísticas de co-ocurrencias textuales.

3. BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers) es un modelo de lenguaje contextual desarrollado por Google que introduce embeddings dinámicos y altamente contextuales.

Construido sobre la arquitectura Transformer, BERT se entrena en dos tareas principales: **Masked Language Modeling (MLM)**, donde ciertas palabras en una oración se ocultan y el modelo debe predecirlas, y **Next Sentence Prediction (NSP)**, donde se aprende la relación entre pares de oraciones consecutivas. Este enfoque lo dota de una capacidad excepcional para comprender el significado en un contexto más amplio y complejo.

BERT está disponible en varias versiones, con diferencias en su tamaño y capacidad. La versión base utiliza una dimensionalidad de 768 y contiene 12 capas de transformadores. Este embedding genera representaciones dinámicas que adaptan el significado de una palabra según su contexto, gracias a su naturaleza bidireccional que analiza tanto las palabras previas como las siguientes en una oración

Sin embargo, BERT también presenta limitaciones, principalmente relacionadas con sus requisitos computacionales. Al ser un modelo grande y complejo, requiere una cantidad significativa de recursos tanto para su entrenamiento como para su implementación. Esto puede dificultar su uso en entornos con restricciones de hardware o donde la velocidad es crucial.

Comparación Resumida

Aspecto	ConceptNet Numberbatch	GloVe	BERT
Estático/Dinámico	Estático	Estático	Dinámico
Entrenamiento	Basado en relaciones semánticas	Co-ocurrencias textuales	Máscara y contexto (Transformer)
Dimensionalidad	300	300	768
Contexto	No contextual	No contextual	Contextual
Velocidad	Muy rápido	Muy rápido	Relativamente lento
Aplicación Ideal	Razonamiento semántico	Análisis semántico básico	NLP avanzado y contextual

Modelos

Para evaluar el desempeño de los embeddings generados, decidimos probar varios modelos de aprendizaje automático, partiendo de la premisa de que los mejores modelos utilizados previamente en RapidMiner eran **Deep Learning** y **Random Forest**. Con este punto de partida, exploramos otros algoritmos, incluyendo:

- **K-Nearest Neighbors (KNN):** Un modelo básico y fácil de implementar, aunque limitado en su capacidad para capturar relaciones complejas.
- **Gradient Boosting (Boost Gradient):** Reconocido por su alta precisión, pero computacionalmente costoso debido a la naturaleza secuencial de su entrenamiento.
- **Random Forest:** Un modelo robusto y eficiente para datos tabulares, que combina múltiples árboles de decisión para mejorar la precisión.
- **Redes Neuronales Artificiales (RNA):** Modelos complejos que pueden captar relaciones no lineales y patrones intrincados en los datos.

Random Forest

Random Forest es un algoritmo de aprendizaje automático basado en un conjunto de árboles de decisión. Su principal ventaja es reducir el sobreajuste y mejorar la precisión mediante el promedio de predicciones de múltiples árboles entrenados en diferentes subconjuntos del conjunto de datos. Utiliza un enfoque de "ensamble" donde cada árbol aporta una predicción y la decisión final se toma mediante votación (para clasificación) o promediado (para regresión). Para calibrar los hiperparametros decidimos utilizar el embedding de Bert como datos base, y posteriormente probamos los otros embeddings con los hiperparametros escogidos.

La búsqueda de hiperparámetros incluyó variables clave como:

- **Número de árboles (n_estimators):** Determina cuántos árboles forman parte del bosque.
- **Profundidad máxima (max_depth):** Controla la complejidad de cada árbol individual.
- **Mínimo de muestras para dividir un nodo (min_samples_split):** Número mínimo de muestras requeridas para dividir un nodo interno.
- **Mínimo de muestras en un nodo hoja (min_samples_leaf):** Número mínimo de muestras que debe tener una hoja (nodo final).

```python
param_grid = {
    'n_estimators': [50, 100, 200, 300, 400, 500],      # Número de árboles
    'max_depth': [None, 10, 20, 30],                    # Profundidad máxima del árbol
    'min_samples_split': [2, 5, 10, 20, 30, 40],        # Mínimo de muestras para dividir un nodo
    'min_samples_leaf': [1, 2, 4]                       # Mínimo de muestras en un nodo hoja
}
```

Para esta búsqueda de hiperparametros utilizamos GridSearchCV de la librería sklearn, en la que destacamos el uso de validación cruzada de 5 particiones y el uso de accuracy como métrica para sacar los mejores hiperparametros.

```python
# Configurar el GridSearchCV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,  # Validación cruzada con 5 particiones
    scoring='accuracy',
    verbose=2,
    n_jobs=-1  # Usar todos los núcleos disponibles
)
```

BERT

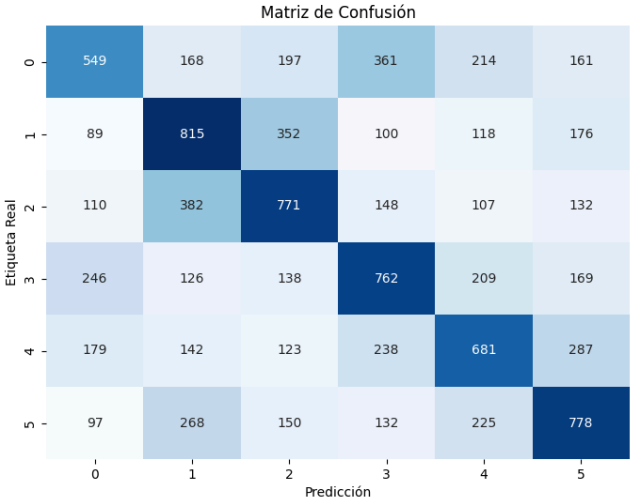
Tras realizar el gridsearch, encontramos que el accuracy medio es de 0.44 dejando bastante que desear a nivel de accuracy. Si analizamos más en profundidad por clases vemos que todas las emociones están en valores similares de predicción lo que lleva a pensar que el modelo desarrollado no explota de la mejor manera posible el word embedding de Bert.

Si analizamos la Matriz de Confusión observamos claramente que la clase peor precedida es Tristeza, mientras que el resto se encuentran a un nivel similar alrededor del 750 True Positives, cabe destacar que se encuentran puntos de fricción en relación a las clases 1 y 2, por lo que el modelo está teniendo problemas para diferenciar entre Alegría y Amor. Otro punto de fricción importante se encontraría entre las emociones Tristeza y Enojo.

```
Accuracy en el conjunto de prueba: 0.44
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.43      0.33      0.38      1650
           1       0.43      0.49      0.46      1650
           2       0.45      0.47      0.46      1650
           3       0.44      0.46      0.45      1650
           4       0.44      0.41      0.43      1650
           5       0.46      0.47      0.46      1650

    accuracy                           0.44      9900
   macro avg       0.44      0.44      0.44      9900
weighted avg       0.44      0.44      0.44      9900
```



GloVe

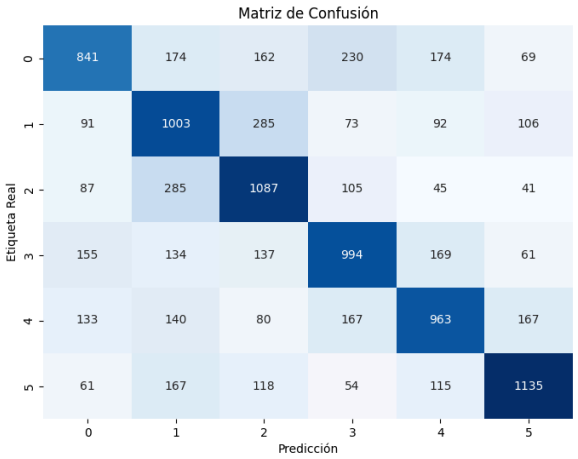
Como podemos observar los resultados de este embedding son superiores a los de Bert, sin llegar a tener resultados brillantes aumenta el accuracy hasta un 0.60. Si analizamos para cada clase, observamos que los true positivo siguen siendo los más bajos para la emoción Tristeza, aunque mejorando al anterior modelo hasta un 0.51. En el resto de clases, hay un aumento destacable de la precisión y de los true positive, aunque todos se encuentran por debajo del 0.69 de recall.

Si analizamos la Matriz de Confusión observamos que seguimos encontrando dificultades en las predicciones entre las emociones 1 y 2, pero en menor medida que con el embedding anterior. En el resto de resultados notamos como el modelo falla menos en la gran mayoría de casos.

```
Entrenando el modelo con los mejores hiperparámetros...
Accuracy en el conjunto de prueba: 0.6083838383838384
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.61      0.51      0.56      1650
           1       0.53      0.61      0.56      1650
           2       0.58      0.66      0.62      1650
           3       0.61      0.60      0.61      1650
           4       0.62      0.58      0.60      1650
           5       0.72      0.69      0.70      1650

    accuracy                           0.61      9900
   macro avg       0.61      0.61      0.61      9900
weighted avg       0.61      0.61      0.61      9900
```



ConcepNet

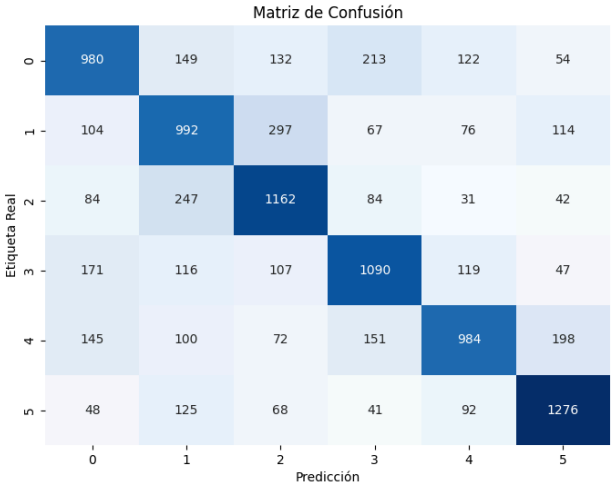
Como podemos observar el accuracy del modelo entrenado con estos embeddings, mejora en 0.05 la accuracy hasta el 0.654, lo cual puede no parecer mucho pero si analizamos los true positive observamos que aumentan en la mayoría de clases excepto en la clase 1 (Alegría) la cual es 0.01 menor, un error que consideramos despreciable.

Así que por el momento, el modelo de Random Forest entrenado con los embeddings de ConcepNet se postula el mejor, sin embargo ahora compararemos con el modelo de Red de Neuronas.

```
Accuracy en el conjunto de prueba: 0.6549494949495
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.64      0.59      0.62      1650
           1       0.57      0.60      0.59      1650
           2       0.63      0.70      0.67      1650
           3       0.66      0.66      0.66      1650
           4       0.69      0.60      0.64      1650
           5       0.74      0.77      0.75      1650

    accuracy                           0.65      9900
   macro avg       0.66      0.65      0.65      9900
weighted avg       0.66      0.65      0.65      9900
```



Red de Neuronas

Por otro lado desarrollamos una red de neuronas tras los resultados del Random Forest decidimos implementar una red de neuronas.

Las redes neuronales son una opción poderosa para problemas de clasificación de texto debido a su capacidad para aprender patrones complejos y no lineales. A diferencia de los modelos tradicionales, pueden manejar datos textuales representados como embeddings, capturando relaciones semánticas profundas entre palabras y frases. Esta flexibilidad es esencial en tareas como análisis de sentimientos, clasificación de noticias o identificación de temas.

Una de sus principales fortalezas es su capacidad para generalizar datos no vistos, siempre que se entrenen adecuadamente. Técnicas como Dropout y Batch Norm 1d ayudan a evitar el sobreajuste, mejorando la precisión en el conjunto de prueba. Además, su estructura modular permite agregar más capas y neuronas según las necesidades del proyecto, lo que hace a las redes neuronales escalables y adaptables a diferentes problemas.

Parámetros

La métrica indispensable para evaluar la precisión del modelo es la **loss** o función de pérdida ya que mide la discrepancia entre las predicciones del modelo y las etiquetas verdaderas del conjunto de datos, indicando cuán incorrectas fueron las predicciones del modelo para un conjunto de datos determinado.

En el caso de las redes neuronales, el objetivo es **minimizar la pérdida** durante el proceso de entrenamiento, es decir, hacer que las predicciones del modelo sean lo más precisas posibles. La función de

pérdida calcula un valor numérico que se utiliza para ajustar los pesos de las conexiones en la red neuronal a través del proceso de retropropagación (backpropagation).

En nuestro modelo utilizaremos *CrossEntropyLoss*, que es una función de pérdida diseñada específicamente para tareas de clasificación multiclase En resumen, la loss es una métrica crucial que guía el proceso de entrenamiento de redes neuronales, asegurando que el modelo evolucione para realizar predicciones más precisas con el tiempo.

La **learning rate** (tasa de aprendizaje) es un hiperparámetro fundamental en el entrenamiento de redes neuronales. Es un valor que controla el tamaño de los pasos que da el modelo al ajustar sus pesos en cada iteración del entrenamiento. La tasa de aprendizaje determina cuánto deben cambiar los pesos del modelo en respuesta al error cometido (calculado a través de la función de pérdida).

Si la tasa de aprendizaje es **muy baja**, el modelo aprenderá de manera muy lenta, y el proceso de entrenamiento podría tardar mucho tiempo en converger (alcanzar el mínimo de la función de pérdida). Sin embargo, si la tasa de aprendizaje es **demasiado alta**, el modelo podría saltar el mínimo de la función de pérdida y no converger adecuadamente, lo que podría llevar a una mayor oscilación o incluso a la divergencia del proceso de aprendizaje.

Por último, el **batch size** es otro hiperparámetro importante en el entrenamiento de redes neuronales. Se refiere al número de muestras (ejemplos) que se procesan antes de realizar una actualización de los pesos del modelo. En lugar de actualizar los pesos después de cada ejemplo (como en el caso del **Stochastic Gradient Descent** o **SGD**), los modelos con un **batch size** mayor procesan varios ejemplos de entrenamiento a la vez antes de realizar una actualización.

En cuanto al tamaño del lote, puede variar desde batchs **pequeños** (16, 32 o 64) pueden permitir que el modelo se entrene más rápido y usar menos memoria, pero podrían ser más ruidosos en el proceso de optimización. O batchs **grandes** (128, 256 o más) pueden proporcionar una estimación más precisa del gradiente y una actualización más estable, pero requieren más memoria y pueden ser más lentos en términos de tiempo de entrenamiento.

Nuestro Modelo

El modelo sigue una arquitectura de **red neuronal densa** (también llamada *fully connected* o *feedforward*), que pasa los datos a través de varias capas de neuronas conectadas, con activaciones y normalización para mejorar el entrenamiento.

Primera capa (entrada):

- nn.Linear(embedding_dim, 512): Esta es la capa de entrada, que toma un vector de embeddings de dimensión embedding_dim y lo mapea a un espacio de 512 dimensiones. Aquí, embedding_dim es la longitud del vector de características de cada palabra o frase en el conjunto de datos, y 512 es la cantidad de neuronas en esta capa.
- nn.BatchNorm1d(512): La normalización por lotes (Batch Normalization) ayuda a estabilizar el aprendizaje al normalizar la activación de cada capa. Esto permite que la red entrene más rápido y con una mayor precisión.
- nn.ReLU(): La función de activación ReLU (Rectified Linear Unit) introduce no linealidad al modelo. ReLU convierte todos los valores negativos en 0 y deja los valores positivos tal cual, lo que ayuda a que el modelo aprenda relaciones no lineales en los datos.

- nn.Dropout(0.5): La regularización Dropout ayuda a prevenir el sobreajuste al apagar aleatoriamente el 50% de las neuronas durante cada paso de entrenamiento. Esto hace que el modelo no dependa excesivamente de ciertas neuronas y se generalice mejor.

Capas ocultas intermedias:

Cada una de estas capas hace lo siguiente:

- nn.Linear(in_features, out_features): Es una capa densa que toma una entrada de características de in_features (el tamaño de la capa anterior) y la mapea a out_features (el número de neuronas de la capa actual). La cantidad de neuronas en cada capa se va reduciendo progresivamente, lo que ayuda a realizar una compresión de la información a medida que avanzamos en la red.
- nn.BatchNorm1d(out_features): Aplica la normalización por lotes a cada capa para estabilizar el proceso de entrenamiento y reducir la dependencia de la inicialización de los pesos.
- nn.ReLU(): Aplicamos la activación ReLU en cada capa intermedia para introducir no linealidades en el modelo.
- nn.Dropout(0.4): Se aplica Dropout con una tasa de 40% en cada capa intermedia, para reducir el riesgo de sobreajuste durante el entrenamiento.

Última capa (salida):

- nn.Linear(64, num_classes): Esta capa final toma las 64 neuronas de la capa anterior y las mapea a num_classes neuronas, donde cada neurona representa una clase en la que el modelo debe clasificar el texto de entrada, en este caso 6 clases.

```
class TextClassifier(nn.Module):
    def __init__(self, embedding_dim, num_classes):
        super(TextClassifier, self).__init__()
        self.fc = nn.Sequential(

            nn.Linear(embedding_dim, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.5),

            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.4),

            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(0.4),

            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, num_classes)
        )

    def forward(self, x):
        return self.fc(x)
```

Y tras valorar distintos parámetros nos quedamos con un batch size de 64, un learning rate de 1e-3 y 20 epochs, el cual nos dio los mejores resultados tanto en loss como en el recall medio de las clases. Para ello valoramos varios cambios como un batch size de 128, learning rates desde 1e-1 hasta 1e-5 y hasta 40 epochs.

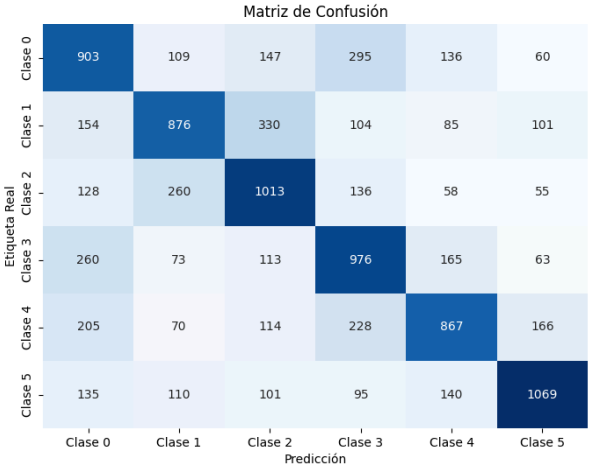
Bert

```
Epoch 20/20: Loss = 0.8712
Train Metrics:
Class 0: {'TP': 2580, 'FP': 953, 'FN': 770, 'TN': 15797, 'Precision': 0.7302575714690065, 'Recall': 0.7701492537313432, 'F1': 0.7496731076565452}
Class 1: {'TP': 2426, 'FP': 510, 'FN': 924, 'TN': 16240, 'Precision': 0.8262942779291553, 'Recall': 0.7241791044776119, 'F1': 0.7718740057270123}
Class 2: {'TP': 2858, 'FP': 916, 'FN': 492, 'TN': 15834, 'Precision': 0.7572866984631691, 'Recall': 0.8531343283582089, 'F1': 0.8023582257158899}
Class 3: {'TP': 2776, 'FP': 970, 'FN': 574, 'TN': 15780, 'Precision': 0.7410571276027763, 'Recall': 0.8286567164179105, 'F1': 0.7824126268320182}
Class 4: {'TP': 2304, 'FP': 493, 'FN': 1046, 'TN': 16257, 'Precision': 0.8237397211297819, 'Recall': 0.687761194029850?, 'F1': 0.7496339677891654}
Class 5: {'TP': 2839, 'FP': 475, 'FN': 511, 'TN': 16275, 'Precision': 0.8566686783343391, 'Recall': 0.8474626865671642, 'F1': 0.8520408163265306}
Test Metrics:
Class 0: {'TP': 903, 'FP': 882, 'FN': 747, 'TN': 7368, 'Precision': 0.5058823529411764, 'Recall': 0.5472727272727272, 'F1': 0.5257641921397379}
Class 1: {'TP': 876, 'FP': 622, 'FN': 774, 'TN': 7628, 'Precision': 0.5847797062750334, 'Recall': 0.5309090909090909, 'F1': 0.5565438373570522}
Class 2: {'TP': 1013, 'FP': 805, 'FN': 637, 'TN': 7445, 'Precision': 0.5572057205720572, 'Recall': 0.6139393939393939, 'F1': 0.5841983852364475}
Class 3: {'TP': 976, 'FP': 858, 'FN': 674, 'TN': 7392, 'Precision': 0.5321701199563795, 'Recall': 0.5915151515151515, 'F1': 0.5602755453501722}
Class 4: {'TP': 867, 'FP': 584, 'FN': 783, 'TN': 7666, 'Precision': 0.5975189524465886, 'Recall': 0.5254545454545455, 'F1': 0.5591744598516608}
Class 5: {'TP': 1069, 'FP': 445, 'FN': 581, 'TN': 7805, 'Precision': 0.7060766182298547, 'Recall': 0.6478787878787878, 'F1': 0.675726927939317?}
```

Precisión media: 0.574
Recall medio: 0.575
Loss: 0.8712

Al igual que con el modelo de Random Forest, el tratamiento del texto con Bert ha llevado a unos resultados de precisión y Recall bastante mediocres, aunque mejores.

Al igual que en el modelo anterior se repiten esos puntos de fricción entre Alegria - Amor y Tristeza - Enojo, errores los cuales se pueden explicar debido a la cercanía de estas emociones



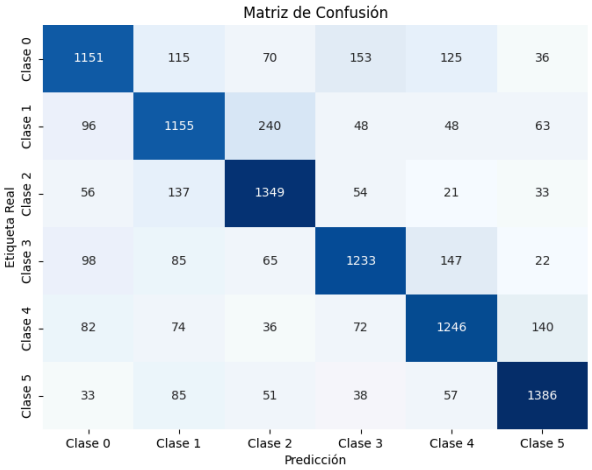
GloVe

```
Epoch 20/20: Loss = 0.5431
Train Metrics:
Class 0: {'TP': 2900, 'FP': 200, 'FN': 450, 'TN': 16550, 'Precision': 0.9354838709677419, 'Recall': 0.8656716417910447, 'F1': 0.8992248062015503}
Class 1: {'TP': 2803, 'FP': 332, 'FN': 547, 'TN': 16418, 'Precision': 0.8940988835725678, 'Recall': 0.8367164179104477, 'F1': 0.8644564379336931}
Class 2: {'TP': 3186, 'FP': 501, 'FN': 164, 'TN': 16249, 'Precision': 0.864117168496176, 'Recall': 0.951044776119403, 'F1': 0.9054995026289612}
Class 3: {'TP': 3062, 'FP': 284, 'FN': 288, 'TN': 16466, 'Precision': 0.9151225343693963, 'Recall': 0.9140298507462686, 'F1': 0.9145758661887694}
Class 4: {'TP': 2898, 'FP': 357, 'FN': 452, 'TN': 16393, 'Precision': 0.8903225806451613, 'Recall': 0.8650746268656716, 'F1': 0.8775170325510977}
Class 5: {'TP': 3215, 'FP': 362, 'FN': 135, 'TN': 16388, 'Precision': 0.8987978753145094, 'Recall': 0.9597014925373134, 'F1': 0.9282517684423272}
Test Metrics:
Class 0: {'TP': 1151, 'FP': 365, 'FN': 499, 'TN': 7885, 'Precision': 0.7592348284960422, 'Recall': 0.6975757575757576, 'F1': 0.7271004421983575}
Class 1: {'TP': 1155, 'FP': 496, 'FN': 495, 'TN': 7754, 'Precision': 0.6995760145366444, 'Recall': 0.7, 'F1': 0.6997879430475612}
Class 2: {'TP': 1349, 'FP': 462, 'FN': 301, 'TN': 7788, 'Precision': 0.7448923246824959, 'Recall': 0.8175757575757576, 'F1': 0.7795434845420399}
Class 3: {'TP': 1233, 'FP': 365, 'FN': 417, 'TN': 7885, 'Precision': 0.7715894868585732, 'Recall': 0.7472727272727273, 'F1': 0.7592364532019704}
Class 4: {'TP': 1246, 'FP': 398, 'FN': 404, 'TN': 7852, 'Precision': 0.7579075425790754, 'Recall': 0.7551515151515151, 'F1': 0.7565270188221008}
Class 5: {'TP': 1386, 'FP': 294, 'FN': 264, 'TN': 7956, 'Precision': 0.825, 'Recall': 0.84, 'F1': 0.8324324324324324}
```

Precisión media: 0.759
Recall medio 0.759
Loss: 0.5431

Vemos una clara mejoría respecto a su homólogo en Random Forest, la precisión sube del 0.61 hasta el 0.75 y sobre el modelo de RNA utilizado con Bert sube en 0.2.

Analizando los fallos del modelo observamos que en menor medida se mantiene los problemas de clasificación para las emociones de Alegría y Amor, pero sin embargo notamos una mejoría elevada en todos los campos de la matriz que están por debajo de la diagonal, en especial

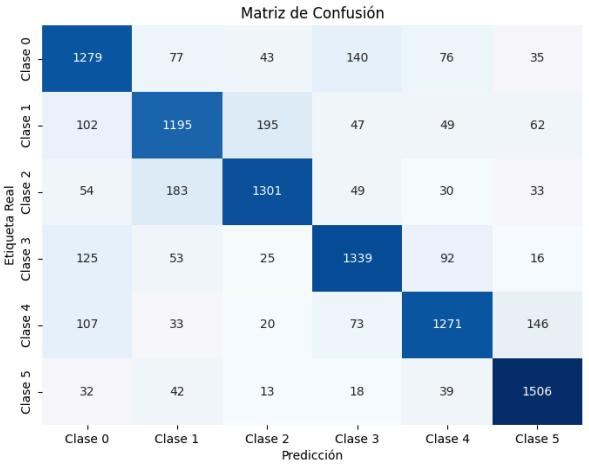


predicciones erróneas de la clase 0 con la clase 2 y 3 y predicciones de la clase 3 con la clase 4.
Siendo especialmente interesante este último punto, ya que si las predicciones que hace de la clase 3 sobre la clase 4 descienden especialmente (un 68%), aquellas en el contrario, es decir las que son predicciones de la clase 4 sobre la clase 3, disminuyen pero en mucha menor medida. (un 10%)

ConceptNet

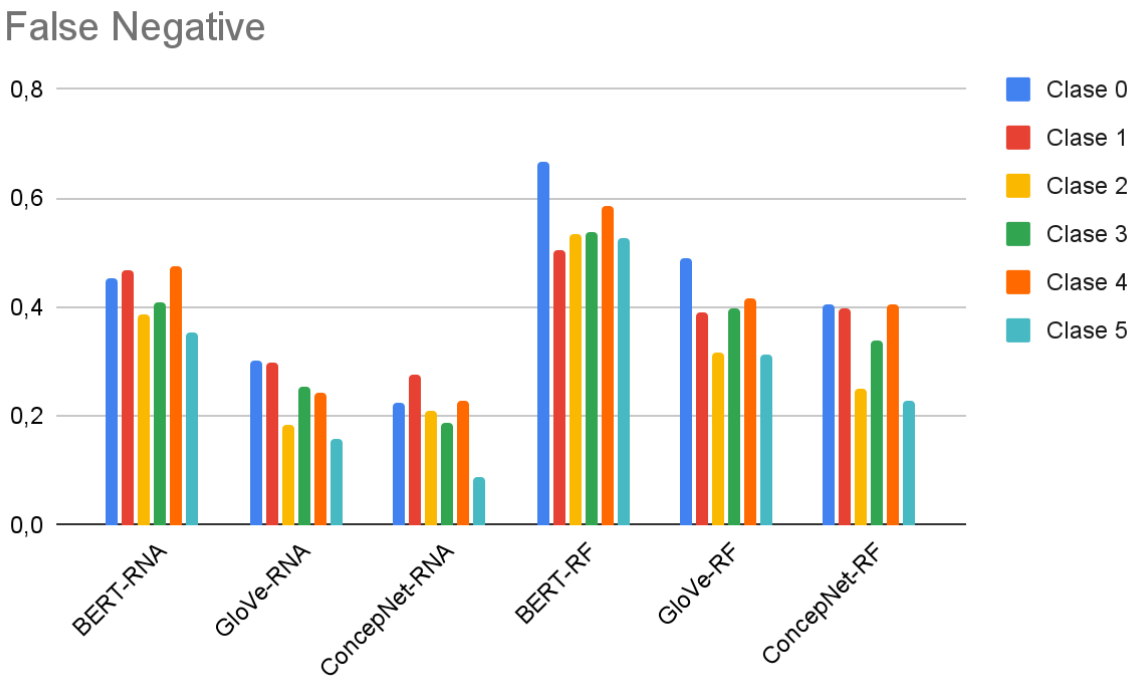
```
Epoch 20/20: Loss = 0.4197
Train Metrics:
Class 0: {'TP': 3178, 'FP': 253, 'FN': 172, 'TN': 16497, 'Precision': 0.9262605654328184, 'Recall': 0.9486567164179105, 'F1': 0.9373248783365286}
Class 1: {'TP': 2937, 'FP': 184, 'FN': 413, 'TN': 16566, 'Precision': 0.9410445370073695, 'Recall': 0.8767164179104477, 'F1': 0.90774223458507}
Class 2: {'TP': 3154, 'FP': 262, 'FN': 196, 'TN': 16488, 'Precision': 0.9233021077283372, 'Recall': 0.9414925373134329, 'F1': 0.9323086018326929}
Class 3: {'TP': 3159, 'FP': 158, 'FN': 191, 'TN': 16592, 'Precision': 0.9523665963219777, 'Recall': 0.9429850746268656, 'F1': 0.9476526173691315}
Class 4: {'TP': 2972, 'FP': 167, 'FN': 378, 'TN': 16583, 'Precision': 0.9467983434214718, 'Recall': 0.887164179104476, 'F1': 0.9160117121282171}
Class 5: {'TP': 3325, 'FP': 351, 'FN': 25, 'TN': 16399, 'Precision': 0.9045157780195865, 'Recall': 0.9925373134328358, 'F1': 0.946484486194136}
Test Metrics:
Class 0: {'TP': 1279, 'FP': 420, 'FN': 371, 'TN': 7830, 'Precision': 0.7527957622130665, 'Recall': 0.7751515151515151, 'F1': 0.7638100925649448}
Class 1: {'TP': 1195, 'FP': 388, 'FN': 455, 'TN': 7862, 'Precision': 0.7548957675300063, 'Recall': 0.7242424242424242, 'F1': 0.7392514692236312}
Class 2: {'TP': 1301, 'FP': 296, 'FN': 349, 'TN': 7954, 'Precision': 0.8146524733876017, 'Recall': 0.7884848484848485, 'F1': 0.801355097012627}
Class 3: {'TP': 1339, 'FP': 327, 'FN': 311, 'TN': 7923, 'Precision': 0.8037214885954381, 'Recall': 0.8115151515151515, 'F1': 0.8075995174909528}
Class 4: {'TP': 1271, 'FP': 286, 'FN': 379, 'TN': 7964, 'Precision': 0.8163134232498395, 'Recall': 0.7703030303030303, 'F1': 0.7926410975990023}
Class 5: {'TP': 1506, 'FP': 292, 'FN': 144, 'TN': 7958, 'Precision': 0.8375973303670745, 'Recall': 0.9127272727272727, 'F1': 0.8735498839907192}
```

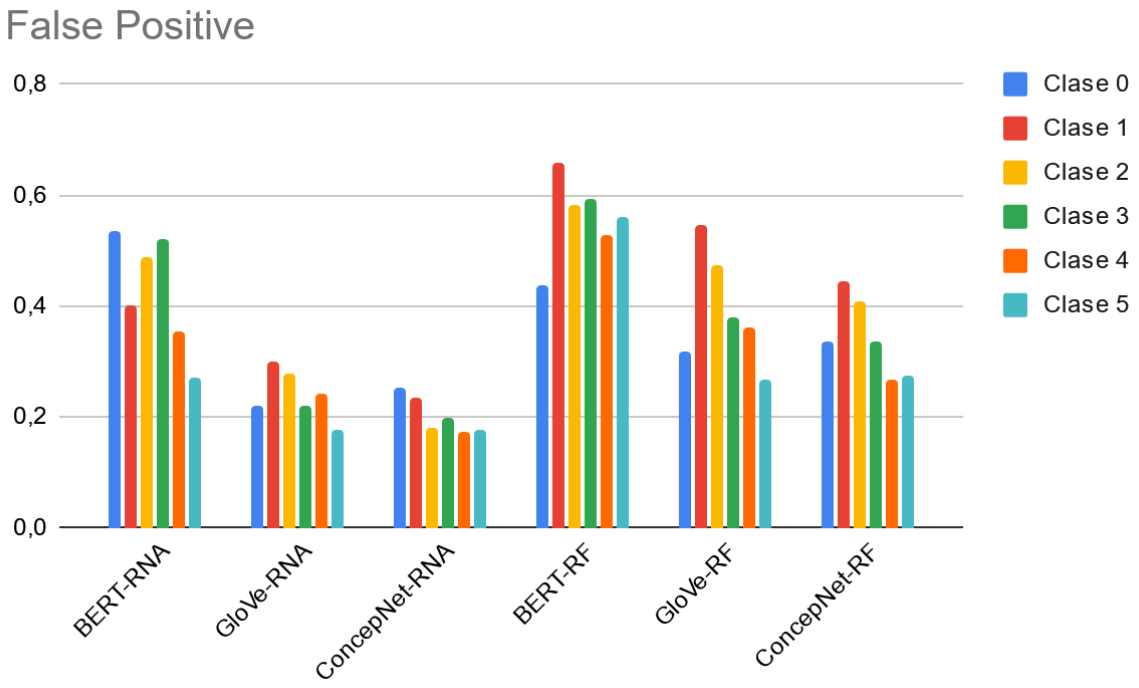
Precisión media: 0.796
Recall medio: 0.796
Loss: 0.4197

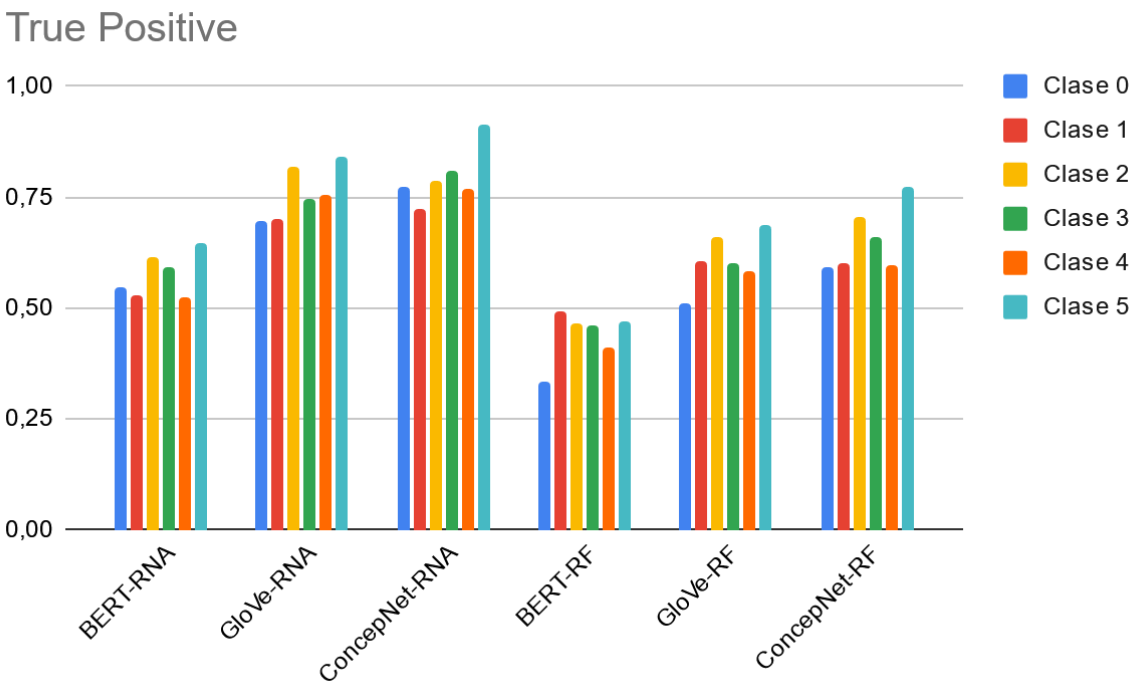


En este caso frente a su modelo de Random Forest vemos una clara mejoría, y frente a los otros dos modelos de RNA resulta ser el menor Loss y con las mejores precisiones y Recalls pero no tan lejos del modelo entrenado con Embeddings de GloVe. (<0.05)

Cuenta con los mismos puntos de fricción que los anteriores modelos pero en menor medida.




False Positive
0,8
0,6
0,4
0,2
0,0
BERT-RNA
GloVe-RNA
ConcepNet-RNA
BERT-RF
GloVe-RF
ConcepNet-RF
Clase 0
Clase 1
Clase 2
Clase 3
Clase 4
Clase 5


True Positive
1,00
0,75
0,50
0,25
0,00
BERT-RNA
GloVe-RNA
ConcepNet-RNA
BERT-RF
GloVe-RF
ConcepNet-RF
Clase 0
Clase 1
Clase 2
Clase 3
Clase 4
Clase 5

Resultados

Si observamos los resultados de la predicción de las clases, se evidencia que los modelos entrenados con la **red neuronal** superan significativamente a los entrenados con **Random Forest**. Esto sugiere que la arquitectura neuronal, con su capacidad para capturar patrones complejos y relaciones no lineales, es más adecuada para el problema en cuestión.

En cuanto al rendimiento de los **word embeddings**, destaca que el modelo **ConceptNet-RNA** obtuvo los mejores resultados, mientras que **BERT** mostró el peor desempeño.

Posibles razones del bajo rendimiento de BERT:

1. **Dimensionalidad y Complejidad:**
   BERT utiliza un embedding de 756 dimensiones, considerablemente más alto que las 300 dimensiones de **GloVe** y **ConceptNet**. Si el modelo no está bien ajustado o los datos no son lo suficientemente complejos, el alto número de dimensiones podría haber introducido ruido en lugar de mejorar la representación semántica.
2. **Escasez de Palabras Polisémicas:**
   Una de las fortalezas de BERT es su capacidad para manejar palabras polisémicas gracias a su naturaleza contextual y dinámica. Si el dataset no incluye suficientes palabras con múltiples significados, esta ventaja de BERT podría no haberse aprovechado plenamente, haciendo que embeddings estáticos como GloVe y ConceptNet sean más efectivos.
3. **Estructura de los Datos:**
   Si los datos contienen términos poco frecuentes o un vocabulario que no coincide bien con el corpus de preentrenamiento de BERT, el modelo podría haber tenido dificultades para generar representaciones precisas. En contraste, ConceptNet, diseñado para comprender relaciones conceptuales, podría haber aprovechado mejor estas conexiones semánticas.

Cabe destacar que Conceptnet utiliza un enfoque basado en relaciones semánticas ya definidas por agentes humanos donde dataset más pequeños y no tan especializados ni específicos pueden tener un alto rendimiento.

Sí, es muy probable que BERT hubiera arrojado mejores resultados con un dataset de mayor tamaño ya que es un modelo de lenguaje contextual preentrenado en grandes corpus, pero su rendimiento en tareas específicas mejora significativamente cuando se ajusta con datos adicionales mediante el proceso de fine-tuning.

En este caso, con solo 30.000 tweets, el modelo podría no haber tenido suficiente información contextual específica para la tarea, lo que limita su capacidad para generar representaciones precisas.

En cambio, ConceptNet, al basarse en relaciones semánticas explícitas, no requiere grandes cantidades de datos para funcionar bien. Sin embargo, BERT, al ser un modelo más complejo y adaptable, muestra su verdadero potencial cuando se entrena con datasets más extensos y variados.

Comparación

Crear modelos de aprendizaje automático en RapidMiner y Python implica enfoques muy diferentes en términos de implementación, control, flexibilidad y resultados. Ambos entornos ofrecen ventajas y desventajas según las necesidades del proyecto y el nivel de experiencia técnica del usuario.

En cuanto a la implementación y facilidad de uso, RapidMiner destaca por su interfaz gráfica intuitiva basada en arrastrar y soltar, lo que permite construir modelos sin necesidad de programar. Esto facilita una curva de aprendizaje rápida y hace que la herramienta sea ideal para principiantes. Sin embargo, esta misma simplicidad implica limitaciones en términos de personalización y control técnico. Además, algunas de sus funciones avanzadas están disponibles únicamente en versiones de pago. Por su parte, Python ofrece una flexibilidad total al permitir implementar desde algoritmos básicos hasta modelos complejos mediante bibliotecas especializadas como Scikit-learn, TensorFlow o PyTorch. Aunque su punto de entrada es más complicado debido a la necesidad de instalar entornos y dependencias, permite crear modelos completamente personalizados y escalables, siendo una opción poderosa para proyectos de cualquier tamaño.

El control y la personalización también son aspectos clave a considerar. En RapidMiner, muchos procesos están preconfigurados y se ejecutan de manera automática, lo que reduce la complejidad técnica pero también limita la capacidad de ajustar algoritmos a necesidades específicas. Python, en cambio, ofrece control absoluto en cada etapa del desarrollo del modelo, desde la preparación de datos hasta la implementación de algoritmos personalizados. Esto permite ajustar cualquier parámetro y probar técnicas experimentales, aunque implica mayor complejidad y posibles errores durante el desarrollo.

En términos de flexibilidad y creatividad, RapidMiner está diseñado para realizar tareas comunes de análisis de datos mediante módulos predefinidos, lo que restringe la capacidad de implementar métodos no convencionales. Python ofrece un entorno abierto donde el usuario puede experimentar libremente y aprovechar una vasta comunidad de desarrolladores que crea y comparte nuevas bibliotecas y recursos.

Cuando se trata de aplicaciones en proyectos reales, RapidMiner es adecuado para prototipos rápidos, proyectos educativos y análisis exploratorios. Sin embargo, su uso en proyectos de producción complejos es limitado debido a sus restricciones técnicas y de integración. Python, en cambio, es ideal para investigaciones avanzadas, sistemas de producción y procesamiento de datos a gran escala, siendo una herramienta esencial en proyectos de inteligencia artificial y aprendizaje profundo.

En conclusión, RapidMiner es una excelente opción para usuarios que buscan rapidez y facilidad de uso en proyectos sencillos y educativos, mientras que Python es la mejor alternativa para desarrolladores experimentados que necesitan libertad creativa y control total en proyectos complejos y escalables. La elección entre ambos depende del tipo de proyecto, el nivel técnico del equipo y los objetivos específicos del desarrollo.