Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INTELIGENCIA ARTIFICIAL

Procesamiento de Lenguaje Natural I

PRÁCTICA OBLIGATORIA EVALUABLE

CLASIFICACIÓN DE IDEOLOGÍAS EN TUITS

Daniel del Nuevo Montero
Carmen Maeso

1. Introducción

El objetivo de esta práctica es realizar una tarea de clasificación de tuits en español en función de su ideología política implícita: *moderate_left*, *left, moderate_right* y *right.*

Para este problema, se aplicarán múltiples técnicas vistas en clase, tales como distintos tipos de preprocesamiento textual, representaciones vectoriales (BoW, TF-IDF), modelos de aprendizaje automático clásicos (Naive Bayes, SVM), así como enfoques de aprendizaje profundo (embedding contextual y redes neuronales).

El conjunto de datos proporcionado está compuesto por dos ficheros:

- ***train.csv***: conjunto de entrenamiento con 28.065 tuits.
- ***development.csv***: conjunto de validación con 4.678 tuits.

Cada fila representa un tuit acompañado de metainformación sobre el usuario que lo publicó. Para esta práctica, nos centramos en la columna ***tweet*** como entrada y en ***ideology_multiclass*** como salida a predecir.

	id	label	gender	profession	ideology_binary	ideology_multiclass	tweet
0	37732	@user239	female	politician	right	moderate_right	@user, propuestas de futuro 💪#no tocar las ref...
1	38447	@user13	female	journalist	left	left	@user @user Podemos entrar en bucle hasta el i...
2	10758	@user8	female	politician	left	moderate_left	👉Como gallega me produce profunda decepción v...
3	33860	@user262	male	politician	right	right	Y ahora, tras este raro de amable discusión so...
4	11677	@user341	female	politician	right	right	1-No se trata de defender o no la labor de @us...

Figura 1: Muestra de contenido ***train.csv*** cargado como *pandas*.

Además, visualizamos el histograma de clases donde aplicaremos la siguiente **codificación**: *moderate_left* : 0; *left* : 1; *moderate_right* : 2; *right* : 3.

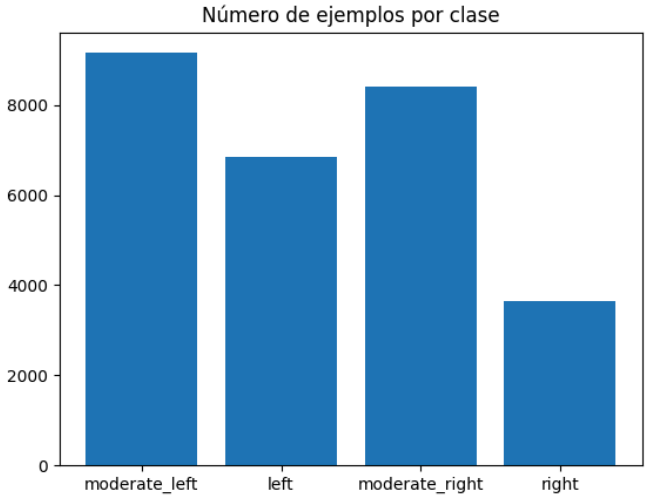

Figura 2: Histograma de clases: *moderate_left*, *left, moderate_right* y *right.*

2. Propuestas

2.1. Preprocesamiento

Para el preprocesamiento utilizaremos la librería SpaCy con el modelo “es_core_news_sm”, ya que los textos de la entrada se encuentran en castellano.

Para procesar los datos textuales antes de generar representaciones vectoriales, se ha utilizado la función **preprocess_sent**, que combina el uso de **expresiones regulares** (**re**) y la biblioteca **SpaCy**.

Esta función permite aplicar múltiples transformaciones sobre el texto original mediante parámetros configurables, detallados a continuación:

- **lowercase**: Si es True, convierte todo el texto a minúsculas.
- **remove_urls**: Si es True, elimina cualquier dirección URL del texto.
- **remove_punct**: Si es True, elimina todos los signos de puntuación.
- **remove_numbers**: Si es True, elimina todos los números del texto.
- **remove_stopwords**: Si es True, elimina las palabras vacías (stopwords) utilizando la lista de SpaCy.
- **lemmatize**: Si es True, devuelve las formas lematizadas de los tokens.
- **remove_hashtags**: Si es True, elimina los hashtags del texto.
- **remove_mentions**: Si es True, elimina las menciones a otros usuarios (comunes en X/Twitter).
- **replace_emojis**: Si es True, reemplaza emojis por su descripción textual (por ejemplo, "😊" → "smiling face").
- **reduce_repeated_chars**: Si es True, reduce repeticiones consecutivas de caracteres a una sola ocurrencia (por ejemplo, "soooo" → "so").
- **min_token_length**: Define la longitud mínima de un token para ser conservado en el texto resultante

Aunque vemos que esta función posibilita muchas combinaciones de preprocesado, nos hemos centrado en dos: procesado mínimo y procesado estándar.

2.1.1. Procesamiento mínimo

En el **preprocesamiento mínimo**, se aplican muy pocas modificaciones al texto:

- Se conservan las menciones, URLs, signos de puntuación, números, emojis, hashtags y stopwords.
- No se realiza lematización, ni se reduce la repetición de caracteres.
- El texto se transforma únicamente a minúsculas (por defecto), y se conserva cualquier token con una longitud mínima de 1 carácter.

Este enfoque pretende preservar al máximo la estructura original del texto para modelos menos sensibles al ruido.

2.1.2. Procesamiento estándar

En el **preprocesamiento estándar**, se lleva a cabo una limpieza algo más profunda, aunque sigue siendo parcial:

- Se eliminan los números y se realiza lematización de los tokens.
- No se eliminan URLs, menciones, signos de puntuación ni stopwords, y tampoco se reemplazan emojis ni se reducen caracteres repetidos.
- Al igual que en el caso mínimo, se utilizan minúsculas por defecto y se mantiene una longitud mínima de token igual a 1.

Este nivel busca un equilibrio entre limpieza básica y retención semántica.

Además, para agilizar el desarrollo del proyecto y evitar repetir el preprocesamiento en cada ejecución, los datos procesados de los conjuntos de entrenamiento y validación se guardaron en archivos .pkl (Pickle). De este modo, pudieron cargarse directamente en futuras ejecuciones, reduciendo considerablemente el tiempo de carga y procesamiento.

2.2. Representación de la entrada de texto

Para entrenar los modelos de clasificación, fue necesario transformar los textos en una representación numérica que pudiera ser procesada por algoritmos de aprendizaje automático. En este trabajo se han considerado dos enfoques principales:

- **Representaciones clásicas mediante el modelo Bag of Words (BoW),** utilizando distintos esquemas de pesado (binario, TF, TF-IDF).

- **Embeddings estáticos**, incluyendo tanto inicialización aleatoria como embeddings preentrenados (FastText).

Inicialmente también se exploró el uso de **embeddings contextuales** - como se puede ver en el *notebook*- mediante modelos basados en Transformers (como BERT). Sin embargo, debido a las

limitaciones de hardware —específicamente falta de GPU y memoria RAM insuficiente para procesar lotes grandes de texto—, esta opción fue descartada en la versión final del sistema.

2.2.1. Representación BoW (Bag of Words)

El modelo clásico de representación de Bag of Words (BoW) transforma cada documento en un vector de tamaño fijo basado en la presencia o frecuencia de las palabras del vocabulario. En este trabajo se exploraron tres variantes. Cada una de las representaciones BoW se usa para los dos preprocesados distintos:

2.2.1.1. Bow con pesado binario

La primera variante fue BoW con pesado binario, donde cada palabra del vocabulario se representa como un valor 1 si aparece en el documento, o 0 si no aparece. Este enfoque no tiene en cuenta la frecuencia de aparición, sino únicamente la presencia o ausencia de las palabras, lo cual puede ser útil para modelos simples.

2.2.1.2. BoW con pesado TF (Term Frequency)

En segundo lugar, se aplicó BoW con pesado TF (Term Frequency). En este caso, el vector contiene la frecuencia absoluta de cada palabra en el documento, es decir, cuántas veces aparece. Aunque esta representación incorpora más información que la binaria, puede dar demasiado peso a palabras frecuentes que no aportan valor discriminativo.

2.2.1.3. BoW con pesado TF-IDF (Term Frequency – Inverse Document Frequency).

Por último, se utilizó BoW con pesado TF-IDF (Term Frequency – Inverse Document Frequency). Esta variante pondera las palabras no sólo en función de su frecuencia en el documento, sino también considerando cuán comunes son en el conjunto total de documentos. De esta forma, se penalizan las palabras muy frecuentes (como conectores o términos genéricos) y se da mayor importancia a aquellas que son más específicas o informativas.

2.2.2. Representación con embeddings

Además de las representaciones clásicas basadas en frecuencias, también se exploraron enfoques más modernos mediante **embeddings estáticos**. Este tipo de representación convierte cada palabra en un vector denso de valores reales, diseñado para capturar relaciones semánticas y sintácticas entre palabras. En este proyecto se utilizaron dos enfoques distintos de embeddings estáticos:

2.2.2.1. Embeddings estáticos con inicialización aleatoria

En primer lugar, se usaron **embeddings con inicialización aleatoria**. En este caso, se empleó una capa Embedding de Keras que se inicializa con valores aleatorios y se entrena conjuntamente con el

modelo de clasificación. Este enfoque permite aprender representaciones personalizadas adaptadas al conjunto de datos, pero parte de cero, por lo que necesita más datos y entrenamiento para ser eficaz.

2.2.2.2. Embeddings preentrenados con FastText

Por otro lado, se utilizaron **embeddings preentrenados con FastText**, un modelo desarrollado por Facebook AI. Se cargaron los vectores distribuidos entrenados sobre grandes corpus en español (cc.es.300.vec), y se asignó a cada palabra de nuestro vocabulario su vector correspondiente. Estas representaciones fueron utilizadas como pesos fijos o ajustables en la capa de embedding. FastText ofrece la ventaja de tener información semántica aprendida previamente, lo que mejora el rendimiento especialmente cuando los datos de entrenamiento son limitados.

2.2.2.3. Embeddings contextuales

También se intentó el uso de embeddings contextuales mediante modelos basados en la arquitectura Transformer, como BERT. A diferencia de los embeddings estáticos, los embeddings contextuales generan una representación distinta para una misma palabra dependiendo del contexto en el que aparece, capturando así de forma más precisa significados y relaciones semánticas complejas.

Sin embargo, durante el desarrollo del proyecto, se encontraron limitaciones importantes de recursos: el procesamiento de grandes volúmenes de datos con BERT requería una GPU con mayor capacidad y una cantidad de memoria RAM superior a la disponible en el entorno de trabajo (Google Colab).

2.3. Modelos de Aprendizaje Automático

De acuerdo con las representaciones generadas, se diseñaron y entrenaron distintos modelos de clasificación. Para ello, se distinguieron dos enfoques principales: por un lado, se utilizaron **algoritmos clásicos** de aprendizaje automático aplicados sobre representaciones tipo BoW, y por otro, se exploraron **modelos de aprendizaje profundo**, aprovechando especialmente las capacidades de las representaciones distribuidas mediante embeddings.

2.3.1. Modelos clásicos

En este apartado se emplean diversos modelos clásicos de aprendizaje automático para llevar a cabo la tarea de clasificación, utilizando principalmente representaciones Bag of Words (BoW) del texto.

2.3.1.1. Naive Bayes

Naive Bayes es un clasificador probabilístico que se basa en el teorema de Bayes. Dado un documento d y un conjunto de clases C, el modelo selecciona la clase c que maximiza la probabilidad a posteriori P(c|d).

2.3.1.1.1. BernoulliNB

Para la representación BoW con pesado binario, se utilizó la clase BernoulliNB de sklearn, diseñada para trabajar con características booleanas (presencia o ausencia de palabras

El clasificador **BernoulliNB** obtiene unos resultados modestos, con un accuracy cercano al 55% en ambos tipos de preprocesamiento. El preprocesamiento **estándar** aporta una leve mejora en términos de macro F1 (0.479 frente a 0.469), indicando que la lematización, limpieza de puntuación y eliminación de stopwords ayudan a reducir ligeramente el ruido en el texto.

Modelo	Preprocesamiento	Accuracy	Macro F1
BernoulliNB	Mínimo	0.558	0.469
BernoulliNB	Estándar	0.558	0.479

Tabla 1: Métricas modelos BernoulliNB con procesamiento mínimo y estándar.

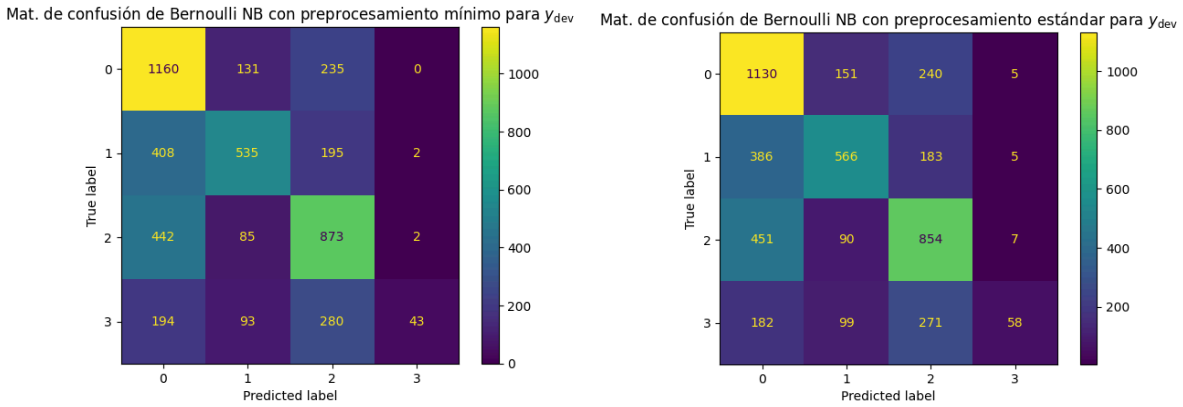

Figura 3: Matrices de confusión modelos BernoulliNB con procesamiento mínimo y estándar.

2.3.1.1.2 ComplementNB

Para las representaciones BoW con pesado TF y TF-IDF, se empleó ComplementNB, una variante más robusta ante clases desbalanceadas y adecuada para frecuencias continuas.

ComplementNB, diseñado para pesos continuos y clases desbalanceadas, mejora un poco los resultados respecto a Bernoulli. En general, el mejor macro F1 (0.553) se obtiene con **pesado TF** y **preprocesamiento mínimo**, lo que sugiere que eliminar información excesiva puede penalizar ligeramente el rendimiento.

Modelo	Peso	Preprocesamiento	Accuracy	Macro F1
ComplementNB	TF	Mínimo	0.578	0.554
ComplementNB	TF	Estándar	0.572	0.548
ComplementNB	TF-IDF	Mínimo	0.577	0.542
ComplementNB	TF-IDF	Estándar	0.574	0.540

Tabla 2: Métricas de los distintos modelos ComplementNB.

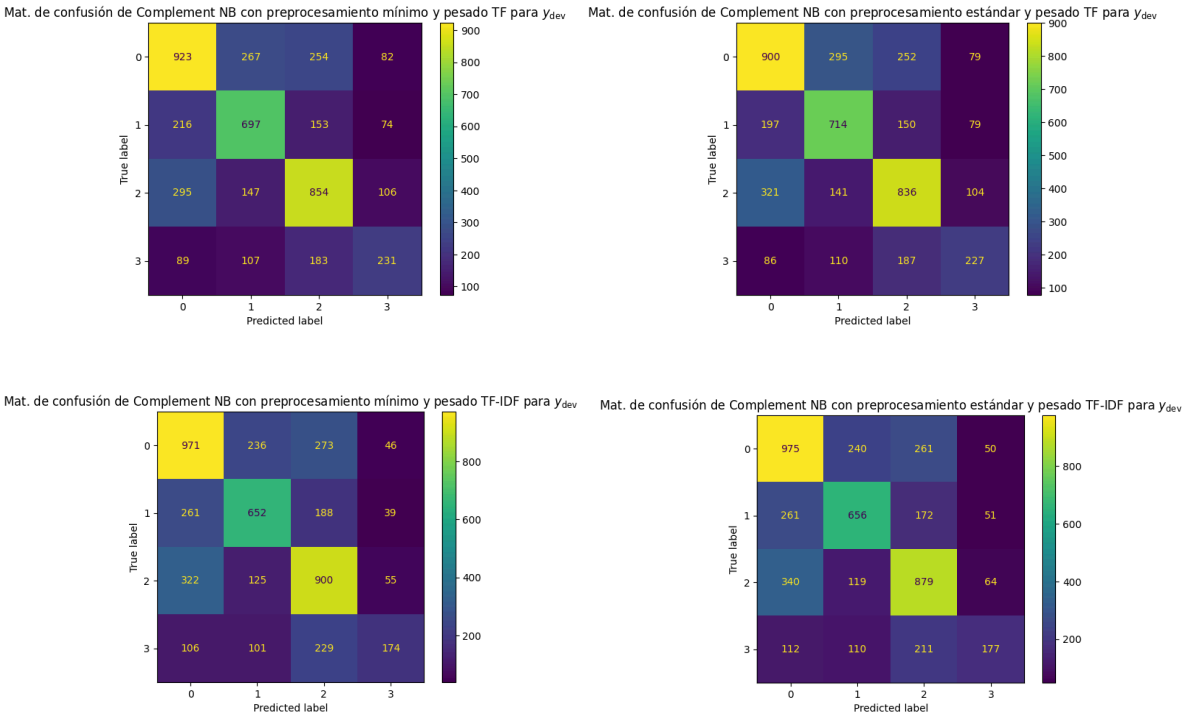

Figura 4 : Matrices de confusión de los distintos modelos ComplementNB.

2.3.1.2. Support Vector Machine

Las máquinas de vectores de soporte son algoritmos supervisados utilizados tanto para clasificación como para regresión. En clasificación, SVM busca construir hiperplanos óptimos que separen las distintas clases maximizando el margen entre ellas. Este enfoque es especialmente eficaz en espacios de alta dimensión, como los generados por BoW.

En cuanto a los clasificadores **SVM**, los resultados son más discretos de lo esperado, con macro F1 por debajo de 0.54 en todos los casos. La mejor configuración se alcanza con **TF-IDF y preprocesamiento mínimo**, alcanzando un macro F1 de 0.539. Es destacable que la diferencia entre representaciones es pequeña, y que las versiones con preprocesamiento estándar no ofrecen mejoras sustanciales.

Modelo	Peso	Preprocesamiento	Accuracy	Macro F1
SVM	Binario	Mínimo	0.535	0.511
SVM	Binario	Estándar	0.532	0.510
SVM	TF	Mínimo	0.531	0.508
SVM	TF	Estándar	0.525	0.504
SVM	TF-IDF	Mínimo	0.566	0.539
SVM	TF-IDF	Estándar	0.563	0.538

Tabla 3: Métricas de los distintos modelos SVM.

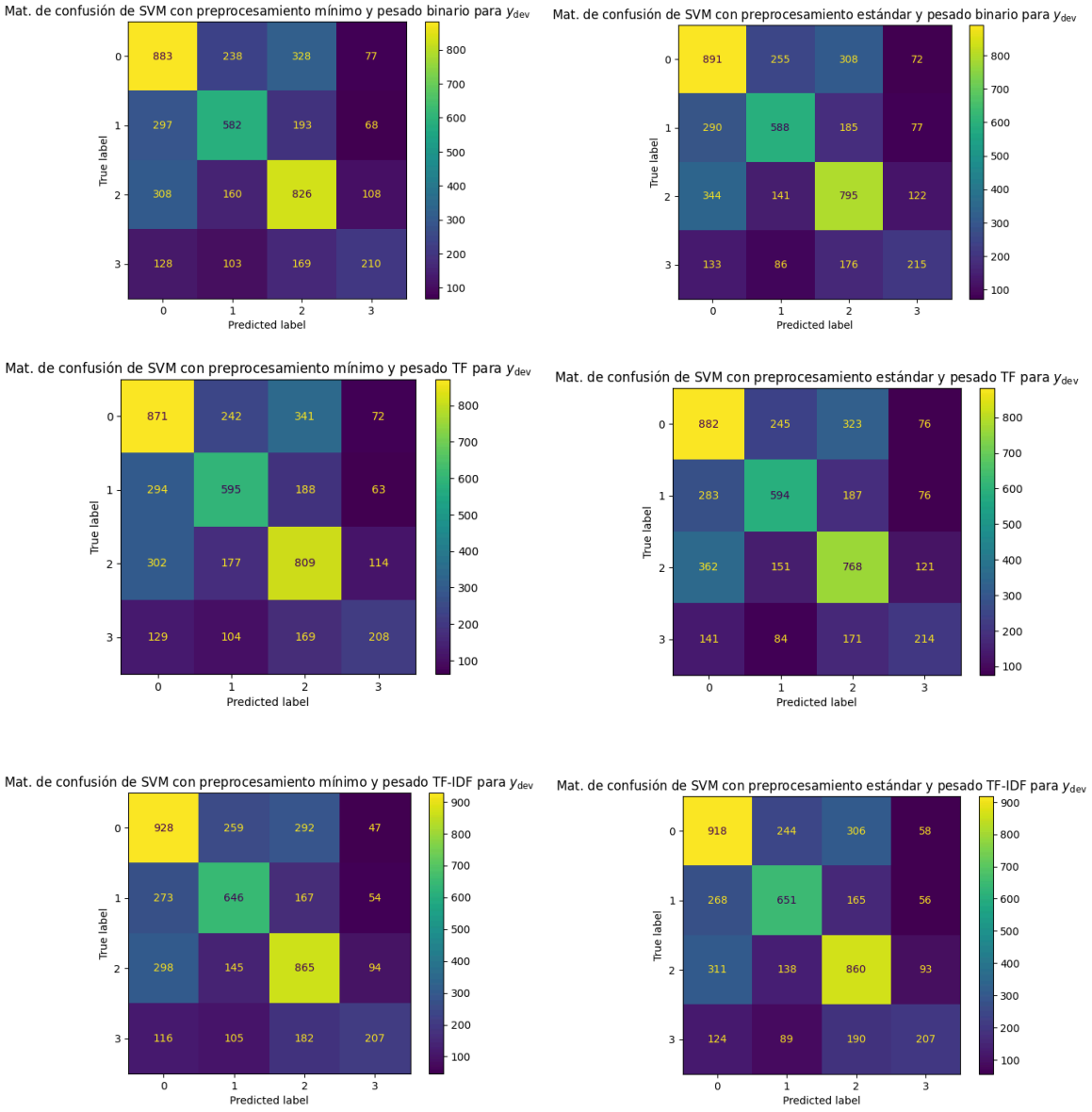

Figura 5: Métricas de los distintos modelos SVM.

2.3.1.1. Evaluación de modelos clásicos

En conjunto, el mejor rendimiento se alcanza con **ComplementNB** sobre representaciones **TF**, ligeramente por delante de SVM con **TF-IDF**. La eliminación "agresiva" de información mediante preprocesamiento estándar no siempre es beneficiosa, especialmente en los modelos que se basan en conteo de frecuencias como SVM y Naive Bayes.

2.3.2. Modelos de Aprendizaje Profundo

Además de los modelos clásicos, se entrenaron diferentes arquitecturas de redes neuronales con el objetivo de aprovechar representaciones más ricas del lenguaje y mejorar la capacidad de generalización del sistema. No obstante, los resultados con embeddings estáticos no fueron tan buenos como se esperaba. Y lamentablemente, como ya se ha comentado, por motivos de hardware no se pudo implementar el modelo basado en transformers, que era el enfoque más adecuado para esta tarea.

El análisis se centra, por tanto, en dos tipos de representaciones basadas en embeddings:

- **Embeddings estáticos inicializados aleatoriamente**
- **Embeddings estáticos preentrenados con FastText**

Para facilitar la comparación, en esta sección se omiten los modelos entrenados con preprocesamiento mínimo, ya que no se observaron diferencias significativas con respecto al preprocesamiento estándar.

2.3.2.1. Con embeddings estáticos inicializados aleatoriamente

2.3.2.1.1. DNN (Red Neuronal Densa)

Para entrenar las DNN se ha probado un paso de PCA (que finalmente hemos descartado) y se ha probado con dos arquitecturas: una con dos capas ocultas densas y activación ReLU y otra con 3 capas densas con Batch Normalization y Dropout. No obstante, el modelo muestra un claro estancamiento en el aprendizaje y un resultado muy bajo. La mejor combinación apenas logra un 0.48 de Macro F1-score.

Modelo	Accuracy	Precision	Recall	Macro F1-score
DNN + Embedding estático (aleatorio)	0.5006	0.4865	0.4817	0.4801

Tabla 4: Métricas modelo DNN + Embedding estático (aleatorio).

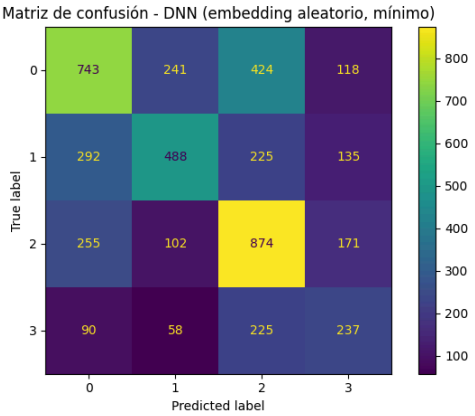

Figura 6: Matriz de confusión modelo DNN + Embedding estático (aleatorio).

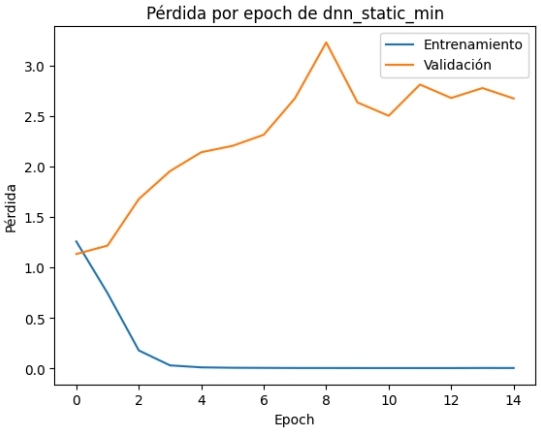

Figura 7: Pérdida por época DNN + Embedding estático (aleatorio).

2.3.2.1.2. CNN (Red Convolucional 1D)

La red incluye varias capas convolucionales y de pooling, diseñadas para capturar patrones locales en la secuencia de palabras. Sin embargo, los resultados son similares al entrenamiento con DNN.

Modelo	Accuracy	Precision	Recall	Macro F1-score
CNN+ Embedding estático (aleatorio)	0.4974	0.4812	0.4653	0.4684

Tabla 5: Métricas modelo CNN + Embedding estático (aleatorio).

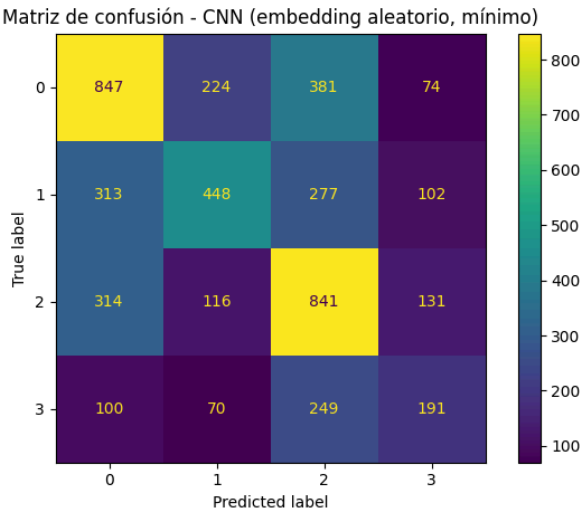

Figura 8: Matriz de confusión modelo CNN + Embedding estático (aleatorio).

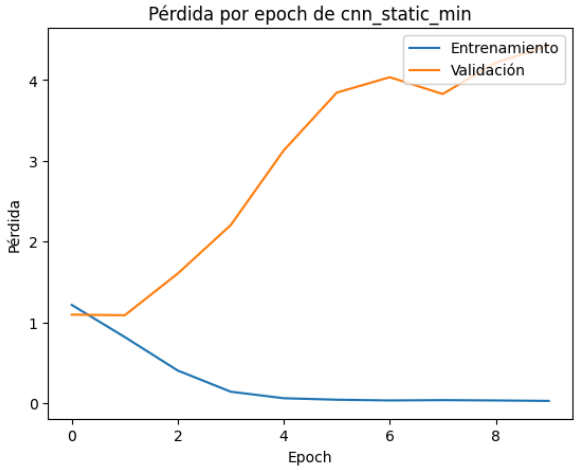

Figura 9: Pérdida por época CNN + Embedding estático (aleatorio).

Como vemos, los embeddings estáticos no tienen suficiente capacidad de aprendizaje, llevando al modelo a un overfitting.

2.3.2.2. Con embeddings preentrenados con FastText

En las primeras pruebas de embeddings preentrenados con FastText se volvía a ver la tendencia al sobreajuste, por lo tanto, se decidió incluir EarlyStopping

Se mantuvo la misma arquitectura de red que en el caso anterior, pero utilizando embeddings de FastText cargados previamente y ajustables (trainable=True).

2.3.2.2.1. DNN

Modelo	Accuracy	Precision	Recall	Macro F1-score
DNN+ Embedding estático preentrenado FastText	0.5517	0.5369	0.5202	0.5222

Tabla 6: Métricas modelo DNN + Embedding preentrenado en FastText.

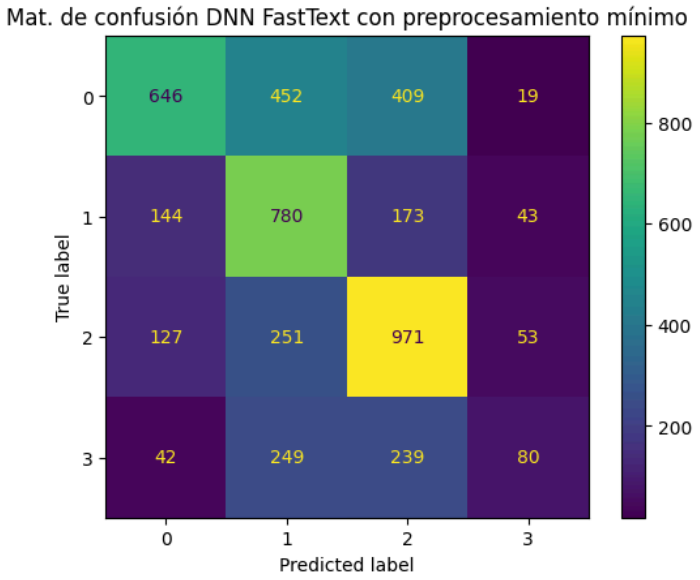

Figura 10: Matriz de confusión modelo DNN + Embedding preentrenado en FastText.

2.3.2.2.2. CNN

Modelo	Accuracy	Precision	Recall	Macro F1-score
CNN+ Embedding estático preentrenado FastText	0.5594	0.5601	0.5060	0.5125

Tabla 7: Métricas modelo CNN + Embedding preentrenado en FastText.

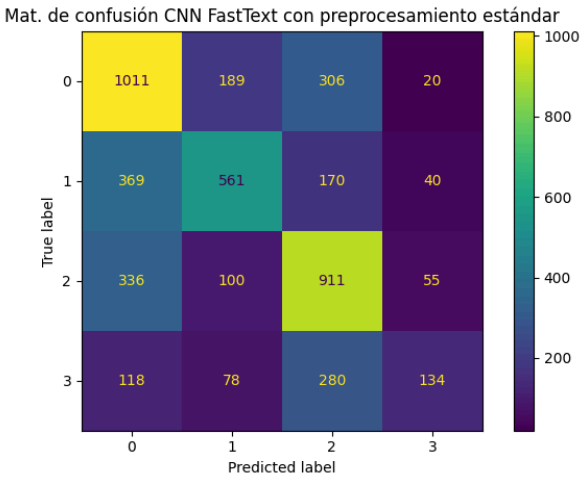

Figura 11: Matriz de confusión modelo CNN + Embedding preentrenado en FastText.

2.3.2.2.3. LSTM

Se probó únicamente con embeddings de FastText por su capacidad para capturar dependencias contextuales a largo plazo. No obstante, no se incluyó finalmente porque no aportaba mejoras significativas.

3. Conclusiones

Este trabajo ha abordado la tarea de clasificación de tuits en español según su ideología política, utilizando una variedad de técnicas de PLN y algoritmos de aprendizaje automático, tanto clásicos como basados en aprendizaje profundo.

En primer lugar, se definieron dos estrategias de preprocesamiento textual: una mínima, que conserva casi toda la información original del tuit, y otra estándar, que introduce técnicas de limpieza como la lematización y eliminación de números. Sin embargo, a lo largo del estudio se observó que las diferencias de rendimiento entre ambas estrategias eran poco significativas en la mayoría de modelos.

En cuanto a la representación del texto, se exploraron enfoques clásicos basados en el modelo Bag of Words (binario, TF y TF-IDF), así como representaciones mediante embeddings estáticos. Dentro de estos últimos, se probaron tanto embeddings con inicialización aleatoria como embeddings preentrenados con FastText. A pesar de los intentos iniciales, no fue posible emplear embeddings contextuales como BERT debido a las limitaciones de hardware (principalmente memoria RAM y GPU insuficiente), lo que limitó el uso de modelos con mayor capacidad expresiva.

Respecto a los modelos clásicos, los resultados más prometedores se obtuvieron con el clasificador ComplementNB usando pesado TF sobre el texto con preprocesamiento mínimo, alcanzando un Macro F1 cercano al 0.55. Por su parte, los modelos SVM ofrecieron un rendimiento más estable pero algo inferior, destacando el uso de representaciones TF-IDF.

En cuanto a los modelos de aprendizaje profundo, los resultados no mejoraron demasiado. Se experimentó con arquitecturas DNN, CNN y LSTM. Los modelos que usaban embeddings estáticos inicializados aleatoriamente no lograron superar a los modelos clásicos, con una clara tendencia al sobreajuste y una capacidad limitada de generalización.

Los mejores resultados dentro del aprendizaje profundo se obtuvieron al emplear embeddings preentrenados con FastText, particularmente con modelos DNN y CNN, alcanzando macro F1-score en torno a 0.52–0.54. No obstante, las mejoras respecto a modelos clásicos fueron modestas, y los intentos con LSTM no aportaron beneficios significativos, posiblemente debido a la complejidad del modelo frente a la cantidad y calidad de los datos disponibles.

Finalmente, cabe destacar que, aunque se probaron múltiples arquitecturas y combinaciones, los resultados globales se mantuvieron dentro de márgenes similares. Esto sugiere que la tarea de clasificación ideológica en tuits es compleja y está sujeta a numerosos factores como ambigüedad del lenguaje, polarización difusa y ruido inherente al texto en redes sociales.

Como líneas futuras, el uso de embeddings contextuales mediante modelos como BERT sería muy prometedor para capturar matices semánticos y contextuales que los modelos actuales no logran representar adecuadamente.

4. Referencias

- Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed., Chapter 4). Draft. https://web.stanford.edu/~jurafsky/slp3/

- Scikit-learn. (n.d.-a). *sklearn.naive_bayes.BernoulliNB*. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

- Scikit-learn. (n.d.-b). *sklearn.naive_bayes.ComplementNB*. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.htm

- Wang, K. (2019). *Fake news detection using traditional machine learning and deep learning* [Project report, Stanford University]. https://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26644050.pdf