Toxic Detection
Prueba Técnica

Mercado Libre

César Daniel Garrido Urbano

29 de marzo de 2023

En este documento se presenta todo el proceso que se siguió para resolver el problema de *Toxic Detection*, sus resultados y el análisis de los mismos. De igual forma, se presentan las suposiciones, decisiones tomadas, modelos explorados, posibles mejoras al modelo y otras alternativas a la solución del problema. El código de respaldo que soporta este trabajo lo puede encontrar en `https://github.com/Cesard97/ToxicDetection.git` o adjunto a este documento en forma de `jupyter notebook`.

1. Exploración inicial de los datos

Para empezar, se realiza una exploración inicial de los datos, con la cual se busca cargar, visualizar y entender los datos con los que se va a trabajar. Asimismo, en esta primera etapa se busca poder limpiar los datos y evaluar la información relevante al problema. Así las cosas, se tiene un set de datos con 514.555 mensajes y su respectiva etiqueta de lenguaje ofensivo, 1 si lo es, 0 si no lo es (véase fig 1).

	message	label
0	Hola, dale actualizar a la pagina o la tecla F5	0
1	Ningún mujer te va a dejar por un error. porque todos somos humanos y el algún momento la cagamos pero siempre hay la oportunidad de mejorar en la cagas que uno hace y más si te hace feliz	1
2	Entonces, si está haciendo una correspondencia en R ^ 2, ¿cómo dibujaría G (x) = {y es un elemento de R2 tal que el producto escalar de x e y es 0} si G (3,4)	0
3	victoria con el pelito corto me voy a matar	1
4	@LoloPeniche PeriodistaIIIII\nEs una puta verdulera la HDSPTM... https://t.co/6wyiTra4ko	1
...	...	...
514550	Argentina: *apoya a la Alemania Nazi en la guerra*\nAlemania: *pierde la guerra*\nArgentina teniendo que explicarle la situación al resto del mundo: https://t.co/eQo4Pqru2r	1
514551	La rata esta de @robertomadrazo invitando a votar, ya pasó no??, para darle aire a su partido corrupto y hablando de tiranías. No mames ratero que te compre quien no te conoce. https://t.co/T7Xm9ZLJxE	1
514552	No dale no hay problema muchísimas gracias no me sabía ese dato no encuentro problema alguno solo me surgió esa duda espero no haber incómodado muchas gracias	0
514553	Mi nombre es "onassis amaya cc 84083683	0
514554	Hola buenas tardes me podrian indicar donde puedo llevar el reloj por garantia se los compre en el mes de febrero. Gracias	0

Figura 1: Pequeña muestra del *dataset* que se tiene para la prueba

Se tiene en total un número considerablemente alto de datos, lo que permitirá hacer una estimación más precisa del error al separar los datos en validación y prueba. No obstante, el procesamiento de todos estos datos tendrá un alto costo computacional y requerimientos de memoria dependiendo del enfoque que se decida seguir para la solución.

Por otro lado, dada la longitud y escritura particular de los mensajes, pareciera que estos provienen de una red social como *twitter*, por lo que será necesario tener en cuenta cosas como usuarios e hipervínculos para no afectar el análisis, ni agregar *tokens* innecesarios en la vectorización de los mensajes.

Adicionalmente, este resulta ser un problema de clasificación binaria balanceado, al tener una relación de datos de clase 0 (57,5 %) y clase 1 (42,5 %) bastante equitativa. Por esta razón no será necesario enfocarse mucho en balancear clases o en en verificar el tipo de error que se está teniendo, en principio el *accuracy* será una buena medida para evaluar el desempeño de los modelos.

Finalmente, en esta primera exploración de los datos se procede a realizar la separación del dataset en un subset de entrenamiento (80 %) y un subset de prueba (20 %). *Split* típico para así poder evaluar el desempeño del algoritmo con datos que los modelos nunca hayan visto. De igual forma, dadas las restricciones de tiempo, poder computacional y memoria, se decidió probar todo el desarrollo primero sobre un subset de 100.000 datos (una quinta parte del total de datos).

1.1. Tokenización y limpieza de datos

En esta etapa se estandarizaron (o normalizaron) los datos realizando las siguientes acciones de preprocesamiento sobre los mensajes:

- **Estandarizar la capitalización.** Todos los caracteres se trabajaron en minúscula.
- **Remover tildes.** Si bien en el español, en algunos casos, las tildes tienen la función de cambiar el significado de ciertas palabras, en las redes sociales es común que los usuarios las omitan (o incluso las usen de forma incorrecta). El estandarizar los mensajes de esta manera permitirá reducir el vocabulario (número de palabras únicas) y en la mayoría de los casos no debería afectar el significado de la oración.
- **Tokenizar** (separar las palabras de una oración en *tokens* que posteriormente podrán ser convertidos en vectores). Dada la naturaleza de los datos, se utilizó un tokenizador especifico para la twitter, el `TweetTokenizer` de la librería `nltk`. Este permite remover usuarios o lidiar con *hashtags*, comunes en este tipo de mensajes.
- **Remover *stopwords*, hipervinculos y signos de puntuación.** Esto permite quitar del mensaje información muchas veces irrelevante para la clasificación del texto.
- ***Setmming.*** Reducción de las palabras (*tokens*) a su raíz por medio de reglas definidas para el idioma.

Este preprocesamiento de los datos se implementó a través de una función en `python`, con el objetivo de que sea modular y fácilmente alterable. Esto pues, a pesar de mencionar las razones por las que pueden ser útiles estos pasos de normalización, no siempre son beneficiosos. De hecho, en la solución final para este problema, se decidió no implementar el paso de *Stemming*, pues la reducción de palabras a su raíz resultaba confusa en varios casos, sobretodo dada la naturaleza del lenguaje coloquial predominante en los mensajes. Una vez finalizada la estandarización se obtiene la siguiente información sobre la longitud de los mensajes (véase fig. 2). Donde se observa que la mayoría de los mensajes, son cortos y casi ninguno supera las 40 palabras.

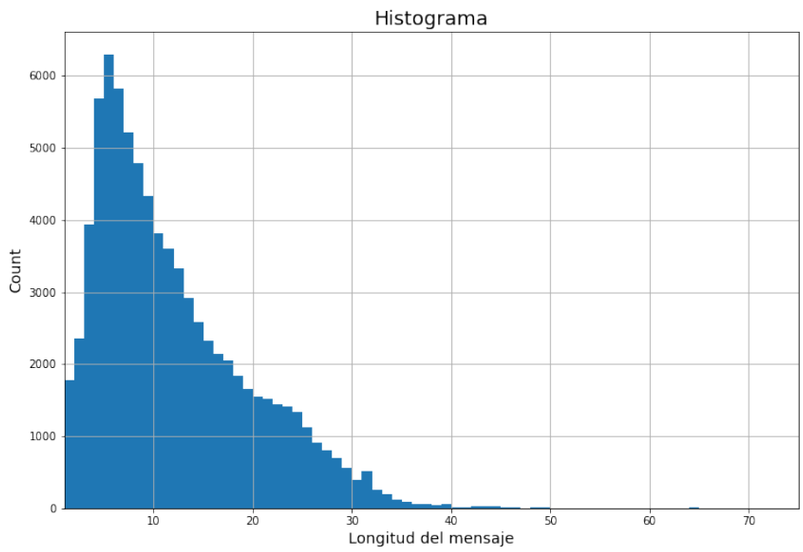

Figura 2: Histograma con la cantidad de *tokens* por mensaje en el dataset de entrenamiento

2. Embeddings

Una vez *toklenizados* y estandarizados los mensajes, se procede a representar dicho texto como un vector que pueda ser utilizado por un modelo de clasificación. Para esto existen varias posibilidades, para este reto se decidió explorar tres de las más conocidas y ampliamente usadas:

2.1. Bag of Words (BOW)

Con esta sencilla forma de vectorización, se construye un vector de 1s y 0s del tamaño del vocabulario para cada mensaje. Donde los 1s estarán ubicados en la posición de la palabra (o *token*) que contenga dicho mensaje. Dada la longitud de los mensajes (en promedio 10 *tokens*) este será un vector muy disperso (contendrá en su mayoría 0s).

A pesar de su simpleza, esta forma de vectorización tiene un gran reto y es el de manejar y guardar en memoria esta gran cantidad de datos. Para el entrenamiento es necesario cargar con una matriz de tamaño: $datos \times vocabulario$. Lo que quiere decir, para este caso en especifico, una matriz de $400{,}000 \times 10{,}000$ datos aproximadamente (el tamaño del vocabulario puede variar dependiendo del preprocesamiento pero usualmente se encontraba entre 8.000 y 15.000 *tokens*).

Con esto en mente, un paso sensato a realizar es el de reducción de dimensionalidad. Con un algoritmo como el de Análisis de Componentes Principales (o PCA por sus siglas en inglés), se puede reducir la dimensionalidad de la representación de cada uno de los mensajes de entre 10.000 y 15.000 al valor de dimensiones que uno desee. En este caso para tratar de preservar la mayor cantidad de información sin comprometer el costo computacional y los requerimientos de memoria se redujo a 1.000 dimensiones. No obstante, este es un valor sujeto a cambio, dada la disponibilidad de recursos computacionales.

Aun así, dado el costo computacional y los requerimientos de memoria, esta forma de vectorización se implementó solo con 100.000 datos.

2.2. Word2Vec

Una forma más elaborada de vectorización como esta permite construir *embeddings* a partir del contexto. Es decir, se utilizan las palabras cercanas (contexto), para entrenar un modelo que trate de predecir una palabra (este es el principio detrás del modelo *CBOW*). De esta manera, se toman los pesos de dichos modelos que tienen decodificados el contexto de cada una de las palabras y se obtiene un vector (del tamaño que uno desee) como representación de la palabra.

Ahora bien, dado que se esta tratando de clasificar oraciones y no palabras, existen dos formas de fijar el tamaño con representación: La primera, es promediando los vectores de cada palabra en la oración y tomando ese promedio como el vector que representa la oración. La segunda, es concatenando cada una de las palabras. Ahora bien, esta última aproximación requiere definir un número de palabras fijo, en donde si la oración tiene un número menor se completará el vector con 0s y si se tiene un número mayor se cortará la oración en ese punto. Ambos enfoques fueron implementados y probados en esta solución.

2.3. Deep Pre-trained Transformer

Finalmente, una de las formas de vectorización más recientes y populares, es la que ofrecen los grandes modelos de lenguaje pre-entrenados, algunos incluso *fine-tuned* para ciertas tareas. Esta implementación resulta ser la más sencilla por lo que ni siquiera es necesario tokenizar los datos, sino solo ejecutar el modelo de lenguaje con la oración tal cual como se tiene. En este caso, se realizó solo un pequeño pre-procesamiento para remover hipervinculos y usuarios de los mensajes. Se utilizó un modelo pre-entrenado y *fine-tuned* para la similitud de frases en español basado en `distiled-roberta`.

Vale la pena aclarar que en todos los casos se encontró un número de *outliers* con mensajes, caracteres y palabras (*tokens* en general) muy raros que no lograron codificarse de manera correcta. No obstante, el número de estos mensajes estuvo siempre al rededor del 0,2 % del total, por lo que se codificaron con un vector de ceros.

3. Selección de Modelo

Ya vectorizados los mensajes que se desean clasificar, solo resta entrenar distintos clasificadores basados en árbol de decisión, para obtener el modelo deseado. Para esto se trabaja con 3 alternativas: Un simple árbol de decisión ***(Decission Tree)***, un modelo de *bagging* que agrega varios árboles de decisión ***(Random Forest)***, un modelo de gradient boosting que agrega también aprendices débiles ***(XGBoost)***.

3.1. Resultados

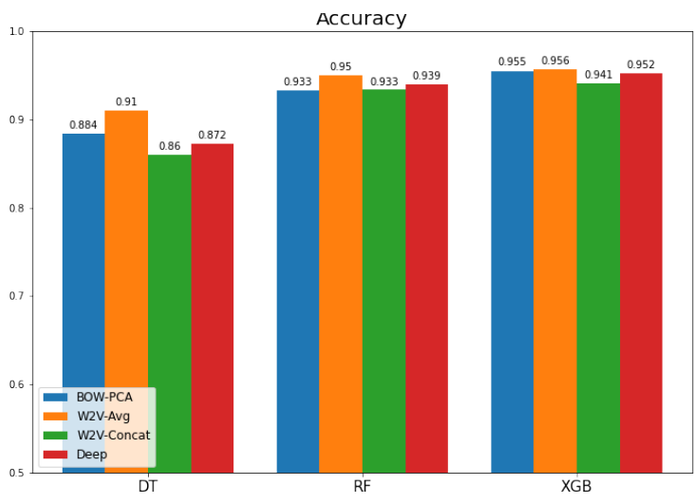

Figura 3: *Accuracy* de los modelos y *embeddings* utilizados con el set de pruebas

Cuadro 1: Métricas de desempeño para los modelos entrenados con 80.000 datos

Model	Test	Accuracy	Precision	Recall	F1-Score
XGB	**W2V-Avg**	**0.9561**	0.9688	0.9272	**0.9475**
XGB	BOW-PCA	0.9548	**0.9768**	0.9160	0.9454
XGB	Deep	0.9517	0.9564	**0.9296**	0.9428
RF	W2V-Avg	0.9500	0.9597	0.9217	0.9404
XGB	W2V-Concat	0.9408	0.9481	0.9115	0.9294
RF	Deep	0.9390	0.9459	0.9095	0.9273
RF	W2V-Concat	0.9333	0.9365	0.9056	0.9208
RF	BOW-PCA	0.9326	0.9569	0.8821	0.9180
DT	W2V-Avg	0.9101	0.8979	0.8914	0.8946
DT	BOW-PCA	0.8837	0.8689	0.8576	0.8632
DT	Deep	0.8722	0.8491	0.8531	0.8511
DT	W2V-Concat	0.8596	0.8330	0.8403	0.8367

Cuadro 2: Métricas de desempeño para los modelos entrenados con 400.000 datos

Model	Test	Accuracy	Precision	Recall	F1-Score
XGB	**W2V-Avg**	**0.959635**	**0.972400**	**0.931514**	**0.951518**
RF	W2V-Avg	0.952979	0.961708	0.926304	0.943674
XGB	W2V-Concat	0.938947	0.944010	0.910423	0.926912
RF	W2V-Concat	0.937519	0.943835	0.907041	0.925072
DT	W2V-Avg	0.917628	0.902491	0.903956	0.903223
DT	W2V-Concat	0.874902	0.849677	0.857522	0.853581

En ambos entrenamientos (con 80.000 y con 400.000 datos) el modelo con mejor *accuracy* resulta ser*XGBoost*, seguido muy de cerca por *RandomForest* y por último *DecisionTree*. Asimismo, la forma de vectorización que más se destacó en los tres modelos fue la de Word2Vec con la agregación que promedia las palabras de la oración (véase figura 3). Note que a pesar de que no se logró realizar todas las pruebas estipuladas con el *dataset* completo, los resultados son consistentes y el desempeño es casi el mismo entrenando con 80.000 o con 400.000 datos. Es de hecho el algoritmo más débil (DT) el único que muestra una pequeña mejora, con la mayor cantidad de datos.

Ahora bien, al ser este un problema de clasificación binaria balanceada, la métrica de mayor importancia viene a ser el *accuracy*, porque nos dice que tantos aciertos tuvo sobre ambas clases. No obstante, se puede observar que otras métricas que miden el desempeño en cuanto a falsos positivos y falsos negativos presentan altos valores con el clasificador *XGBoost*. Incluso el valor más alto del *F1-Score*, el cual pondera *Precision* y *Recalll* es el del clasificador *XGBoost* con un *embedding* de W2V en ambos casos (véase cuadros 1 y 2).

4. Conclusiones

4.1. Análisis de Resultados

En términos generales se puede observar que los resultados apuntan a que todos los modelos logran resolver el problema de clasificación. Esto pues se tienen *accuracies* por encima del 90 % con los tres modelos. Teniendo en cuenta que este es un problema de clasificación binario balanceado (donde un modelo aleatorio tendría un *accuracy* de alrededor del 50 %), estos resultados son bastante prometedores y que denotan un generalización adecuada de la tarea.

Sin embargo, los métodos de ensamble como *XGBoost* y *RandomForest* presentan siempre una mejora de algunos puntos sobre el simple árbol de decisión y estos a su vez tienen un desempeño muy parecido. Se destaca también la similitudes que presentan los distintos métodos de *embedding*, incluso el simple método de *BOW* con *PCA*, logra resultados comparables con los métodos complejos, inclusive superando el *embedding* del modelo pre-entrenado en el árbol de decisión.

4.2. Posibles mejoras y trabajo futuro

En esta primera aproximación y solución del problema se realiza una exploración sobre distintos métodos de vectorización y varios modelos de clasificación basados en árboles de decisión. No obstante, todas estas implementaciones están sujetas a mejoras pues cada una de ellas contiene un gran número de hiperparámetros que en no fueron optimizados y prácticamente se usaron sus valores por defecto. Vale la pena considerar que la optimización de estos valores puede incluso cambiar cual resulta ser la mejor aproximación y el mejor modelo.

De igual forma, quedan por explorar algunas de las variantes que no se pudieron implementar por restricciones de capacidad computacional, como la vectorización *BOW* (incluso sin *PCA*), que mostró un desempeño comparable a los otros tipos de vectorización. Asimismo, probar otros tipos de modelos profundos pre-entrenados, más allá de la versión de `distilroberta` usada, incluso sería interesante *fine-tune* alguna versión de estos modelos para la tarea especifica en cuestión.

Adicionalmente, en esta primera aproximación el enfoque se limita a explorar la solución del problema con modelos basados en árboles de decisón, pero existen otras alternativas que valdría la pena tener en cuenta. Ya sea la de utilizar modelos más complejos, como redes neuronales con capas de recurrencia, lstm o de atención que sean capaces de capturar mejor el sentido de la frase o incluso directamente modelos ya pre-entrenados para tareas similares a los que se les puede hacer un *fine-tuning* para mejorar el desempeño. O aproximaciones más sencillas como las propuestas por los *lexicones*, en donde se detecta la clase a partir de la probabilidad de que ciertos *tokens* aparezcan en un mensaje ofensivo o no.

Ahora bien, aunque el modelo todavía presenta espacios para mejora, su desempeño es suficiente para poder ser implementado en producción. Para esto se podría utilizar como base el *notebook* desarrollado y con algún servicio en la nube como el de *SageMaker Studio*, se pueden re-entrenar estos modelos, optimizar sus hiperparámetros y desplegar en un *endpoint* que le permita a los usuarios hacer llamados al modelo. Este modelo en particular podría ser útil para detectar tweets que están siendo ofensivos contra la empresa o incluso ayudar a enrutar usuarios que estén utilizando lenguaje ofensivo con asistentes que estén mejor preparados para esto. En primer lugar se piensa en tweets por lo que el dataset con el que fue entrenado el modelo contiene principalmente estos, pero podría extrapolarse a otros casos de estudio, en cuyo caso un re-entrenamiento con datos más similares a estos vendría bien.