Categorización de comentarios Tóxicos utilizando LSTMs Bidireccionales y Attention

José Javier Jo Escobar
Universidad del Valle de Guatemala
jo14343@uvg.edu.gt

Resumen:

En la actualidad existen aún tabúes respecto a la salud mental y muchas personas tienden a expresar sus problemas en los sitios web o redes sociales. Los comentarios tóxicos son un tema psicológicamente difíciles de escalar. Para ello se presenta un marco para utilizar deep learning para identificar y clasificar automáticamente los comentarios tóxicos en 6 subcategorías identificadas. Se implementara una arquitectura LSTM bidireccional y se tratará de implementar un modelo de attention para ayudar a nuestra LSTM bidireccional. Se pretende asignar pesos a las categorías logrando así visualizar la activación de la red neuronal en respuesta a cada palabra dentro de una oración sin perder la lógica cualitativa de la intuición humana.

Introducción:

Muchos sitios web que muestran contenido enviado por el usuario deben tratar con comentarios tóxicos o abusivos. La moderación humana puede conllevar un riesgo psicológico para los moderadores, y es difícil de implementarlo a escala. Las posibles soluciones automatizadas requerirán no sólo una clasificación binaria (aceptable vs bloqueada) sino también clasificaciones detalladas para mantener la cortesía sin interferir con el discurso normal y proporcionar explicaciones a los usuarios cuando sus publicaciones están censuradas. Diferentes sitios web pueden desear tener diferentes políticas para tratar diferentes tipos de contenido tóxico. Por ejemplo, un sitio web puede permitir comentarios insultantes mientras bloquea las amenazas y el discurso de odio basado en la identidad.

En este proyecto se pretende mejorar la identificación y clasificación de los comentarios tóxicos en un dataset recopilado en este artículo, el dataset consiste en 159,571 comentarios de pláticas en Wikipedia los cuales fueron etiquetados por evaluadores humanos para la determinación de comportamiento tóxico. Los 6 tipos de toxicidad son: toxic, severe_toxic, obscene, threat, insult y identify_hate. Una muestra de las entradas del dataset se puede observar en la Figura 1.

En este reporte se describe un enfoque de referencia a este clasificación múltiple utilizando una red neural conformado por un feedforward de tres capas con promedios de embeddings en oraciones y word2vec como entradas. Se experimentará con LSTMs uni y bidireccionales con Gated recurrent unit

(GRU). Se pretende utilizar un modelo de LSTM bidireccional + attention para mejorar la identificación de pesos a las palabras de entrada.

Estado del arte:

comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0
Would you both shut up, you don't run wikipedi...	1	0	0	0	1	0
Hi! I am back again!\nLast warning!\nStop undo...	1	0	0	1	0	0
Thanks much - however, if it's been resolved, ...	0	0	0	0	0	0
Its common sense dumbass, it was a fight in th...	0	0	1	0	1	0
Exactly my thoughts, it doesn't belong in the ...	0	0	0	0	0	0

Figura 1: Una porción de comentarios del dataset de entrenamiento, se muestran la gran diversidad del corpus y capacidad de un comentario al tener múltiples etiquetas. También se puede ver que cuando un comentario es *toxic* es un subconjunto de *severe_toxic* pero no de las otras etiquetas.

Para este proyecto, se pretende investigar cómo las LSTMs mejorarían el rendimiento y también si se utiliza attention.

Objetivo

Análisis Exploratorio

Para convertir los datos de texto sin formato en forma utilizable, se tokenizo la entrada utilizando el paquete nltk [1]. Como cada uno de los comentarios tiene una longitud diferente, se rellenará o recortará las listas de tokens de cada comentario a una longitud uniforme y enmascarar los tokens hpadi en la capa de salida RNN (o capa de attention, una vez se comience a usar).

Para representar cada token de palabra en los comentarios de entrada, se realizó la incrustación de palabras usando 300- vectores globales pre entrenados dimensionales para la representación de palabras (GloVe) [2], y se entrenó utilizando estadísticas globales de coincidencia global word-word en 6 mil millones de tokens de corpus de Wikipedia y Gigaword [3] con un tamaño de vocabulario de 400k.

También se crearon vectores de entrada de índices de palabras en int32 utilizando el mapeo de word2id pre entrenado que incluye GloVe. Se retuvo un 30% aleatorio de los datos de entrenamiento para usar como dev set configurado para probar el rendimiento de los modelos. Algunas palabras ofensivas publicadas en línea no son parte de los 400k palabras que forman el vocabulario de GloVe pero están definidos como tokens *unk*. Luego de identificar las palabras ofensivas el rendimiento

debería mejorar, si un token contiene cómo substring cualquier miembro de la lista de vulgaridades comunes que no se encuentra fácilmente en substrings de palabras inofensivas se mapean a la raíz de la vulgaridad.

Otro dato importante sobre el dataset es el gran desequilibrio en cada ejemplo de las etiquetas (Ver Figura 1). Este problema se trato ponderando las contribuciones de las etiquetas inversamente a su ocurrencia en los datos de entrenamiento, normalizados a 1 comentario tóxico (la etiqueta más común). Estos pesos hacen que los gradientes más grandes se propaguen por errores en las etiquetas más raras. De esta manera, aprendemos más de cada ejemplo con las etiquetas poco comunes.

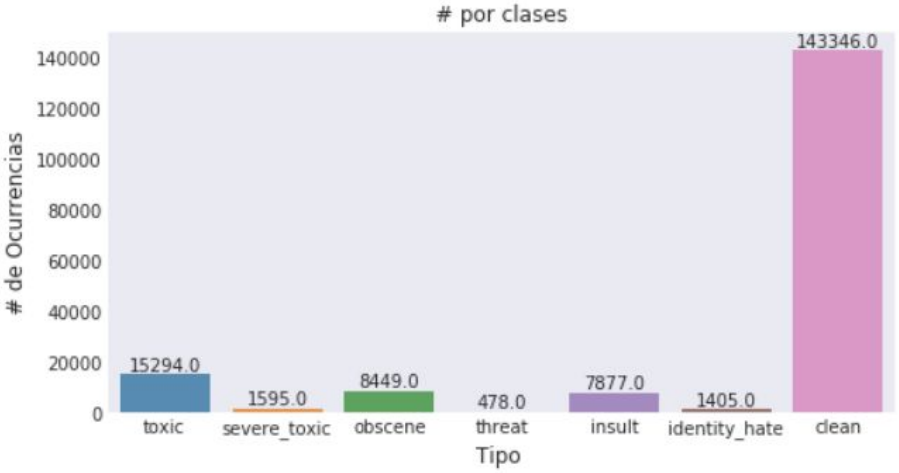


Gráfico 1: Conteo de ocurrencias de cada tipo de categoría.

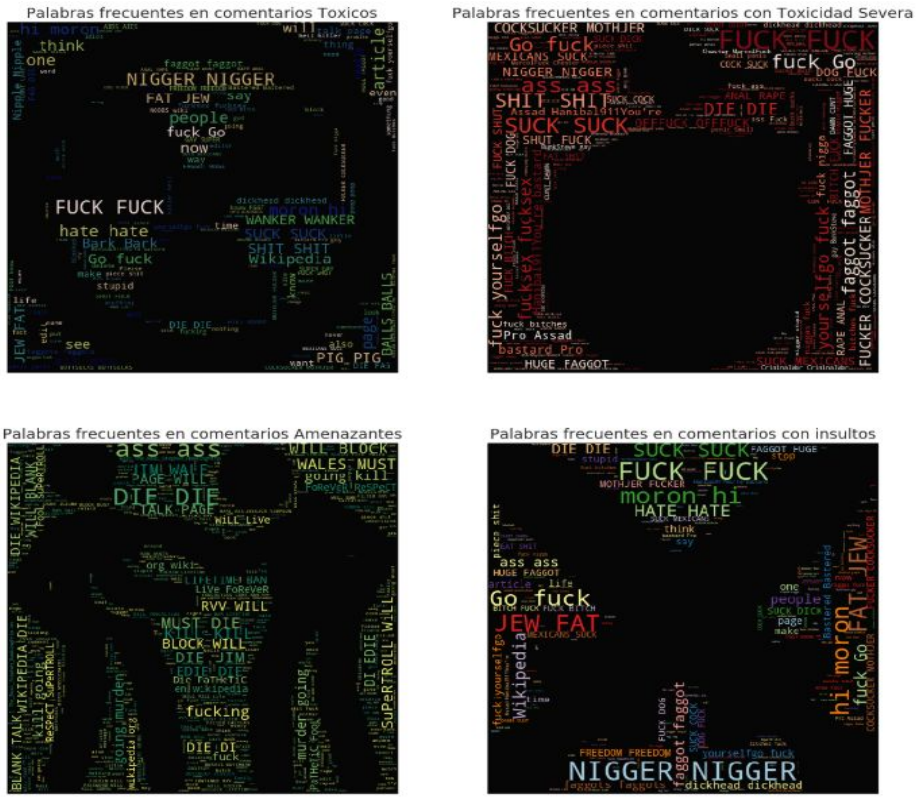


Figura 1: Se muestran nubes de palabras que representan las palabras más frecuentes en grande

Modelo de referencia

Se implementó una red neuronal feedforward totalmente conectada como nuestro modelo de referencia. Las entradas del modelo es el promedio de los embeddings de 300d GloVe para los tokens en cada comentario. Se debe tomar en cuenta que no es necesario rellenar comentarios en este modelo, ya que el procedimiento promedio reduce los comentarios de cualquier longitud a un tamaño uniforme.

La red neuronal consta de 3 capas ocultas con función de activación ReLU [4], con 30, 20, 10 unidades ocultas en cada capa oculta, respectivamente. Dado que el objetivo es realizar una clasificación de etiquetas múltiples (es decir, las etiquetas de toxicidad no son mutuamente excluyentes), las salidas de la última capa oculta se introducen en una capa de salida sigmoide (en lugar de una capa softmax, lo que obligaría a las probabilidades de 6 clases a agregar hasta 1) con 6 unidades, que corresponden a las probabilidades predichas de cada una de las 6 etiquetas. La función de cross entropy se usa como pérdida. La arquitectura neural se puede definir de la siguiente manera:

$$
\begin{aligned}
\boldsymbol{h}_1 &= \text{ReLu}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1) \\
\boldsymbol{h}_2 &= \text{ReLu}(\boldsymbol{h}_1\boldsymbol{W}_2 + \boldsymbol{b}_2) \\
\boldsymbol{h}_3 &= \text{ReLu}(\boldsymbol{h}_2\boldsymbol{W}_3 + \boldsymbol{b}_3) \\
\hat{\boldsymbol{y}} &= \sigma(\boldsymbol{h}_3\boldsymbol{W}_4 + \boldsymbol{b}_4) \\
J &= CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{6} y_i \log(\hat{y}_i)
\end{aligned}
$$

donde

$$\boldsymbol{x} \in \mathbb{R}^{B\times 300}, \boldsymbol{h}_1 \in \mathbb{R}^{B\times 30}, \boldsymbol{h}_2 \in \mathbb{R}^{B\times 20}, \boldsymbol{h}_3 \in \mathbb{R}^{B\times 10}, \hat{\boldsymbol{y}} \in \mathbb{R}^{B\times 6}, \boldsymbol{y} \in \mathbb{R}^{B\times 6}$$

B es el tamaño de nuestros batch.

En este modelo de referencia y a lo largo de este artículo, todos los bias son inicializados en cero, mientras que todas las matrices de pesos son inicializadas utilizando la inicialización de Xavier [5]. Con este modelo de referencia, encontramos un rendimiento del dev set relativamente bueno medido por las curvas Receiver Operating Characteristics (ROC) para cada tipo de toxicidad (ver Figura 2). La media de ROC y Area Under The Curve (AUC) en las 6 etiquetas en el conjunto de prueba ciega fue de 0.9490, que es un buen rendimiento para una referencia. La similitud entre el train y las curvas de desarrollo muestra que no teníamos un sobreajuste severo.

Dado que el número de negativos verdaderos (comentarios no tóxicos) en nuestro conjunto de datos es grande (ver Tabla 1), y las curvas ROC trazan la tasa positiva verdadera (T P R = T P / (T P + F N)) vs. la tasa de falsos positivos (F P R = F P / (F P + T N)), su apariencia está sesgada por el desequilibrio en nuestro conjunto de datos.

Para una medida más directa del rendimiento de nuestro clasificador para cada etiqueta, también se trazó la precisión frente al rendimiento de recuperación de nuestro clasificador de referencia (que se muestra a la derecha en la Figura 2). Aquí vemos que el rendimiento es mejor para las etiquetas que tienen más ejemplos de entrenamiento.

Arquitecturas Recurrentes

Nuestro modelo de referencia tuvo un buen rendimiento relativamente, pero podemos mejorarlo construyendo un modelo de red neuronal recurrente (RNN) que es capaz de posicionar las información de cada una de las palabras a través de un promedio sobre las oraciones.

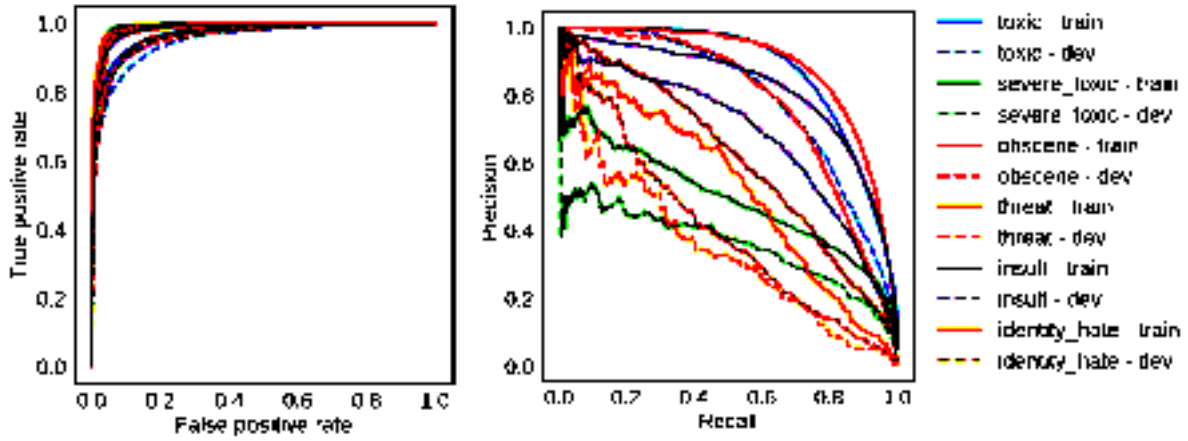


Figura 2: Las curvas ROC de recuperación de precisión para el rendimiento de nuestro modelo de referencia en cada tipo de toxicidad según la clasificación de nuestro modelo de referencia.

Se identificó que un modelo RNN con GRU y LSTM es mejor que un Vanilla RNN, pero las diferencias entre LSTM y GRU son pequeñas, entonces mejor nos enfocamos en las redes con LSTM.

Se implementaron los modelos con Python y tensorflow [6] para las celdas que conforman nuestra arquitectura LSTM, se utilizó la implementación original de LSTM que provee tensorflow. También se utilizan las ecuaciones de [7] que se muestran a continuación:

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(i)} + \boldsymbol{h}_{t-1} \boldsymbol{U}^{(i)}\right) \\
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(f)} + \boldsymbol{h}_{t-1} \boldsymbol{U}^{(f)}\right) \\
\boldsymbol{o}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1} \boldsymbol{U}^{(o)}\right) \\
\tilde{\boldsymbol{c}}_t &= \tanh\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1} \boldsymbol{U}^{(o)}\right) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned}
$$

La arquitectura LSTM permite que la información fluya largas distancias a través de la red desenrollada, pero también evita que los gradientes exploten. También se encontró que el uso de celdas GRU (que tienen propiedades similares) para esta tarea y se encontró un rendimiento comparable a las celdas LSTM, por lo que omitimos sus ecuaciones aquí [8].

Para una arquitectura LSTM unidireccional, es posible utilizar los estados ocultos del tiempo final como el estado de salida general h que se encuentra en la última capa de salida. Se encontró un mejor rendimiento pero, cuando usamos el promedio de elementos sobre los estados ocultos de todos los

time-steps (enmascarando cualquier estado oculto en time-steps donde el <pad> o token es entrada) es *h* , la entrada para la capa final de la arquitectura.

Con una arquitectura LSTM bidireccional, tenemos un conjunto de parámetros para un LSTM extendida hacia adelante y un conjunto separado de parámetros para un LSTM extendida hacia atrás que lee las listas de tokens de entrada que comienzan en el final en lugar del principio. Por lo tanto, para cada posición de token, tenemos dos estados de salida en cada time step, que tomamos el promedio antes de promediar en todos los time steps, o concatenamos para formar un solo estado de salida conjunta que luego se alimenta en la capa de attention que se describe a continuación. .

Para evitar el sobreajuste, agregamos regularización al estado de salida general de los LSTM o capa de attention aplicando un dropout [9] de 0.5. Finalmente, se utiliza una capa completamente conectada con sigmoide para convertir la salida del dropout de cada capa en las probabilidades de cada etiqueta.

Implementación de Attention

Para esta implementación de Attention, los vectores ocultos para cada palabra *ht* de los time steps recurrentes *t* se alimenta de una tangente hiperbólica no lineal para producir una nueva representación de palabra *vt*. El puntaje del attention *st* es calculado por el producto punto de similitud entre *vt* y el vector contexto de la palabra *ua*. Los pesos de attention αt se calculan al normalizar la puntuación del attention a través de todos los time steps vía la función de softmax. Finalmente, un vector de salida attention *h* es calculado como la suma ponderada de los estados ocultos originales *ht* de cada palabra, ponderado por los pesos de la capa de attention normalizados.

Experimentos:

Configuraciones

Se probaron modelos con una variedad de configuraciones de hiper parámetros en un nodo de AWS. Se enfocó en las expansiones del modelo en lugar de una búsqueda sistemática de configuraciones óptimas de hiper parámetros, manteniendo la tasa de aprendizaje para el optimizador Adam [11] fija en 0.0005. La disminución constante de la pérdida de entrenamiento (como se muestra a la izquierda en la Figura 3) muestra que esta elección de tasa de aprendizaje fue razonable.

La Tabla 2 muestra los resultados de un conjunto de experimentos que se hicieron implementando a medias la capa de attention, para evaluar la mejora del uso de estados ocultos más grandes, más capas y arquitecturas unidireccionales y bidireccionales. Descubrimos que las arquitecturas bidireccionales eran en todos los casos mejores que unidireccionales, más de 1 capa en cada dirección (si es bidireccional) condujo a un sobreajuste, y las celdas LSTM y GRU dieron un rendimiento casi equivalente. Por lo tanto, se utilizó un LSTM bidireccional de una sola capa como nuestro modelo comparativo para luego evaluarlo con attention.

Cada experimento se llevó alrededor de 1-3 horas, dependiendo del tamaño del modelo y del numero de épocas. Primero se corrieron todos los modelos para 50 épocas, pero se observó que el modelo iniciaba a tener un sobreajuste después de ~10 épocas (Ver figura 3). Por lo tanto, se implementó la detención temprana para que el modelo se guarde progresivamente después de cada época y logrará tener un mejor rendimiento para el dev set en cualquier época anterior.

También se implementó recortes al gradiente, logrando así garantizar que nuestros gradiente no explotarán durante el back propagation, pero debido a la arquitectura LSTM (ver discusión en Arquitecturas Recurrentes), la norma permaneció lo suficientemente baja durante el entrenamiento que el recorte era innecesario (Ver panel derecho de la Figura 3)

Cell type	Dirección	#Capas	Ocultas	Dev {ROC AUC}	Dev {AP}
LSTM	Unidireccional	1	50	.9610	.5009
LSTM	Unidireccional	1	100	.9683	.5473
LSTM	Unidireccional	2	50	.9546	.4470
LSTM	Bidireccional	1	50	.9747	.5788
LSTM	Bidireccional	2	50	.9747	.5882
GRU	Unidireccional	1	50	.9659	.5180
GRU	Bidireccional	1	50	.9755	.5837
GRU	Bidireccional	2	50	.9761	.5619

Tabla 2: Esta tabla muestra los resultados de la variación de hyper parámetros de cada modelo, estados ocultos, numero de capas y dirección para el modelo RNN sin la capa de attention.

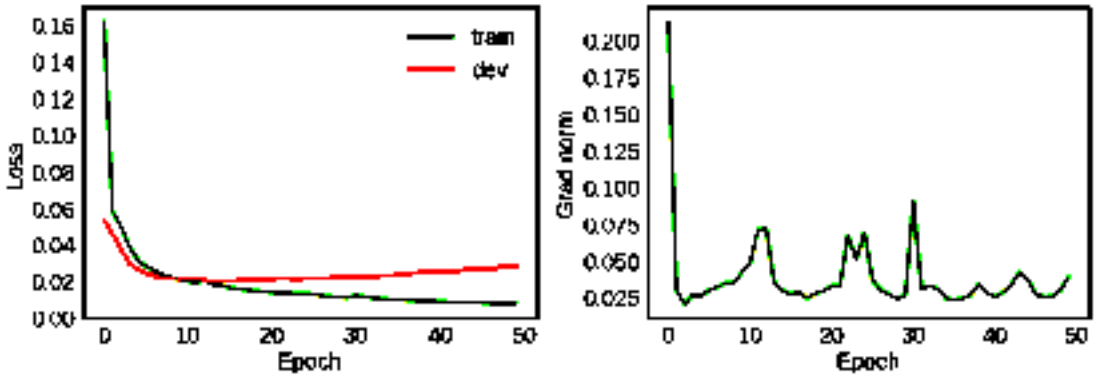


Figura 3: En la gráfica izquierda podemos observar una función de pérdida generada por las pruebas de entrenamiento. Podemos ver que incrementa la perdida luego de ~10 épocas indicando un sobreajuste, como lo mencionamos anteriormente al detenerlo antes guardamos el mejor modelo antes

de que el sobreajuste empiece. En la gráfica derecha podemos ver la normalización del gradiente durante el entrenamiento mostrándonos que no tenemos problemas con el gradiente.

Evaluación

Se configuró una evaluación del pipeline para trazar automáticamente las curvas de ROC y la curva de recuperación de la precisión para la clasificación de cada tipo de toxicidad. El pipeline también calcula el AUC ROC y la precisión promedio (AP) para la clasificación de cada tipo de toxicidad, y el AUC ROC medio en columna, el promedio de precisión y el promedio en columna en todos los tipos de toxicidad. Se realizaron las predicciones del modelo en un conjunto de prueba de 153,164 comentarios cuyas etiquetas fueron dadas por el artículo antes mencionado.

Modelo	Dev <AP>	Dev <ROC AUC>	Test <ROC AUC>
Referencia	0.5610	0.9669	0.9490
LSTM	0.5473	0.9683	0.9435
LSTM Bidireccional	0.5882	0.9747	0.9611
LSTM Bidireccional + Attention	**0.5989**	**0.9784**	**0.9689**

Tabla 3: Podemos observar un resumen de los resultados generados por los modelos que se investigaron y su accuracy promedio, también podemos ver la media de ROC AUC del dev set, y la media de ROC AUC del train set.

Los resultados para los tres modelos principales se muestran en la Tabla 3 y se sobresaltan los resultados en la clasificación de las múltiples etiquetas resultado de la implementación de un modelo LSTM Bidireccional con la capa de attention.

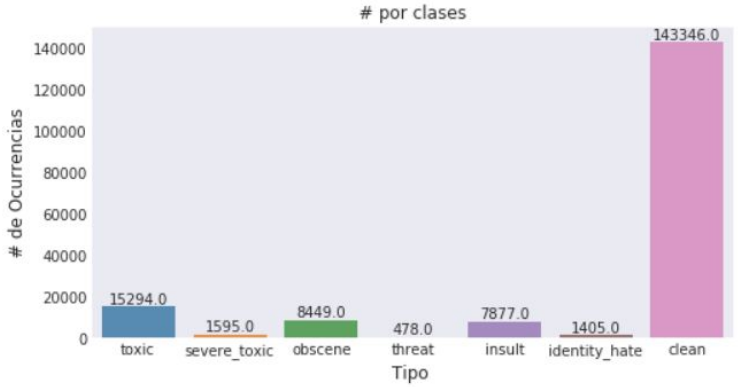


Figura 4: Podemos observar el desequilibrio que se tiene en los comentarios etiquetados en el dataset

En la Figura 5 vemos vemos que el rendimiento general de nuestro modelo final (LSTM bidireccional + attention) es muy bueno, con el ROC muy cerca de la esquina superior izquierda que representa una clasificación perfecta. En el panel de la derecha, vemos que las clases en las que el rendimiento fue

más bajo son aquellas que tienen la menor cantidad de ejemplos de entrenamiento (threat, severe_toxic y identify_hate).

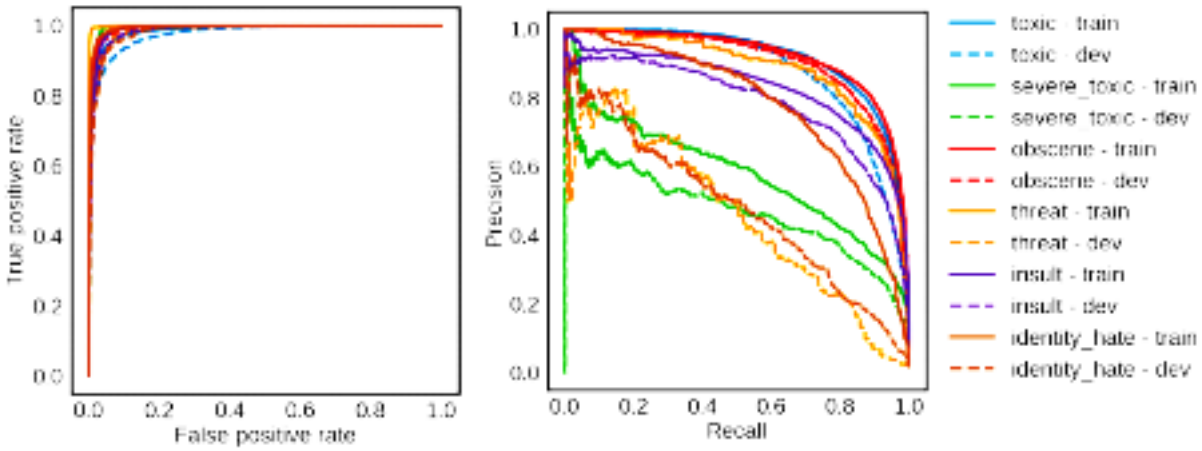
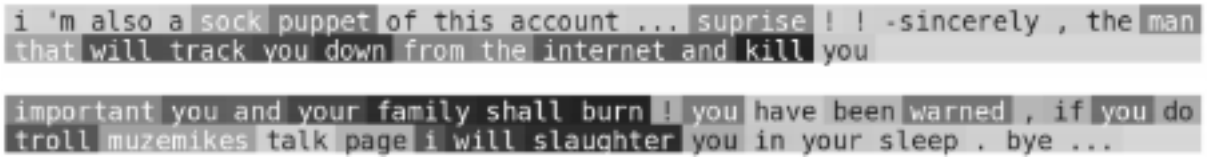


Figura 5: Las curvas ROC y la precisión en la recuperación para nuestro modelo final.

Hay cierto sobreajuste, cómo lo podemos ver en las líneas punteadas que se desplazan de las líneas continuas representan el rendimiento en los datos de entrenamiento.

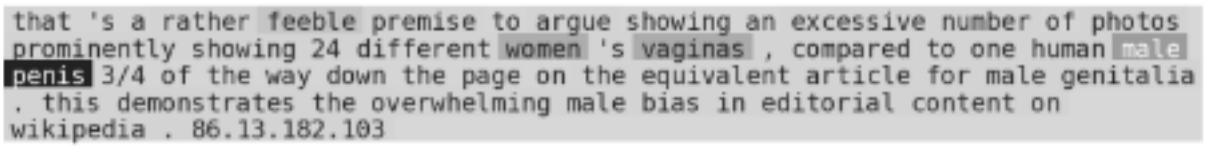
Evaluación Cualitativa por Attention

Con el mecanismo de la capa attention, el modelo es capaz de computar y normalizar los pesos para cada palabra de la oración. Para propósitos de diagnóstico, podemos visualizar los pesos de la capa attention en los comentarios seleccionados para entender cualitativamente cómo el modelo clasificó los comentarios de la manera en que lo hizo Por ejemplo, podemos ver los pesos de la capa attention de algunos ejemplos de verdaderos positivos: en este caso, los comentarios se clasificaron con éxito cómo *toxic* y *threat*.

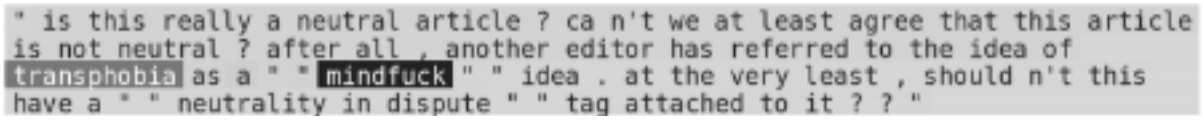


Los pesos de attention se destacaron con éxito mostrando las palabras y frases que suenan como una amenaza para los usuarios.

Este es un ejemplo de un falso positivo:



Aunque los pesos del modelo attention resalta algunas palabras parcialmente vulgares, el clasificador lo clasificó con éxito como negativo. Por otro lado, aquí hay un ejemplo de un falso positivo



El modelo identifico algunas palabras que podrían ser potencialmente tóxicas y se resaltaron los pesos del modelo attention, pero no reconoce que el comentarista está citando a otra persona para que haga un argumento razonable y, por lo tanto, lo clasifica erróneamente como tóxico y obsceno.

Conclusiones

Se implementó un LSTM bidireccional del framework de Tensorflow con una salida de la capa de attention logrando exitosamente clasificar varios subtipos de toxicidad en los comentarios. El rendimiento promedio final del ROC AUC en el conjunto de pruebas es de 0.9689 lo cual es bastante bueno. La precisión promedio fue de 0.5989.

El rendimiento es principalmente limitado por obscenidades nuevas o raramente utilizadas por el token hunki de GloVe. Se intentó mejorar esto mediante el mapeo de obscenidades compuestas que son desconocidas para la identificación de la obscenidad principal a la hora de realizar el preprocesamiento, pero el procesamiento elimina varios “Unk” o tokens desconocidos. El uso de de incrustaciones a nivel de caracteres o métodos convolucionales podrían mejorar aún más el rendimiento, ya que sería capaz de reconocer mejor los patrones de caracteres tóxicos dentro de una palabra desconocida. Además, una mayor regularización más allá del simple dropout podría ayudar a mejorar el rendimiento del modelo sin tener un sobreajuste.

Referencias

1. Bird, S., Loper, E., & Klein, E. (2009) Natural Language Processing with Python. OReilly Media Inc.
2. Pennington, J., Socher, R., & Manning, C. (2014) GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing.
3. Parker, R., et al. (2011) English Gigaword Fifth Edition LDC2011T07. DVD. Philadelphia: Linguistic Data Consortium.
4. Nair, V. & Hinton, G. E. (2010) Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning, Omnipress, USA, 807-814.\
5. Glorot, X., Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256.
6. Abadi, M., et al. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
7. Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. Neural Computation, 9(8):1735-1780.
8. Cho, K., et al. (2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, arXiv:1406.1078.
9. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014) Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 1 1929-1958.
10. Ivanov, I. (2018) Tensorflow implementation of attention mechanism for text classification tasks. https://github.com/ilivans/tf-rnn-attention/
11. Kingma, D. P. & Ba, J. (2014) Adam: A Method for Stochastic Optimization, arXiv:1412.6980.