Proyecto 2 – Tarea-examen 2

Lenin Pavón Alvarez

Mayo 2025

Resumen

Los grandes modelos de lenguaje (LLMs por sus siglas en inglés) son capaces de generar texto (y diverso contenido multimedia) a un ritmo sin precedentes. Es así que surge la necesidad de distinguir entre aquellos productos que fueron creados por humanos y aquellos que fueron creados usando LLMs. En este proyecto planeo explorar y explicar un técnica de watermarking reciente [1] de manera computacional.

Palabras clave. Teoría de Códigos Procesamiento de Lenguaje Natural Grandes Modelos de Lenguaje Códigos Correctores de errores *Robust Binary Code*

Índice

1. Antecedentes

1.1. Loros estocásticos (Grandes modelos de Lenguaje)

Un trabajo que en su momento fue muy controversial (y lo sigue siendo) fue un paper por el cual una de sus autoras (Timnit Gebru) perdió su empleo en Google. En *On the Dangers of Stochastic Parrots: Can language Models Be Too Big* [2] las autoras advierten con respecto a la trayectoria preocupante del desarrollo de Modelos de Lenguaje y los peligros asociados a su entrenamiento y uso por la población en general.

Las autoras definen a un modelo de lenguaje como

> we understand the term *language model* to refer to systems whuch are trained on string predicion tasks: that is, predicting the likelihood of a token (character, word or string) given either its preceding context or (in bidirectional and masked LMs) its surrounding contexts.

Es así que las autoras los suelen abreviar con LMs (*language models*) por sus siglas en inglés. Actualmente en la literatura se suele usar la abreviación LLMs (*large language models*) haciendo referencia al enorme tamaño de los datasets (datos de entrenamiento) a partir de los cuales se construye el modelo.

Las autoras presentan un conjunto de varias situaciones problemáticas, en resumen:

1. Costo ambiental y financiero (§3)
2. Datos de entrenamiento que superan la capacidad humana de entenderlos y documentarlos (§4).
    a) Aunque se obtiene una gran cantidad de datos, estos no necesariamente son representatitvos de puntos de vista no hegemónicos (e.g. sur global, hablantes de lenguas indígenas, etcétera).
    b) Aún cuando la realidad social cambia, es costoso reentrenar a los modelos con nuevos datos de entrenamiento. Así los LLMs se quedan con una visión estática de nuevos marcos conceptuales[1].
    c) Al no curar los datos, los LLMs pueden quedar entrenados para replicar los sesgos existentes en sus datos de entrenamiento.
    d) Al no invertir en la curación de estos datos de entrenamiento se genera una deuda documental con respecto a éstos.
3. Las salidas (*outputs*) de los LLMs se pueden malinterpretar como entendimiento del lenguaje natural (NLU).

[1] No encontré una referencia en línea disponible, pero el Grupo de Ingeniería Lingüística de la UNAM trabaja en detección de homofobia usando modelos de lenguaje.

Es particularmente este último problema el que origina la necesidad de distinguir entre la salida un LLM y texto[2] producido por un ser humano. Las autoras terminan §7 con la siguiente pregunta:

> Could LMs be built in such a way that synthetic text generated with them would be watermarked and thus detectable?

La respuesta de la industria ha sido ni siquiera entretener la pregunta. ChatGPT se liberó al mundo sin este tipo de *watermarking* que pudiera hacer su salida detectable. Sin embargo han surgido técnicas que permiten identificar la salida de estos LLMs sin necesidad de la cooperación (directa) de sus desarrolladores. Particularmente, ha habido un desarrollo reciente con respecto al uso de códigos correctores de errores para la detección de éstos.

1.2. Códigos Correctores de Errores

1.2.1. Tokenización

Antes de poder comenzar a hablar acerca del uso de códigos correctores de errores (ECC, por sus siglas en inglés) hay que recordar que nuestro espacio es de la forma $\{0,1\}^k$ para alguna $k \in \mathbb{Z}$, es así que necesitamos una manera mediante la cual podamos convertir lenguaje natural a nuestro vocabulario.

Usando Byte-Pair Encoding (BPE) [3] realizamos un análisis del corpus que nos brinda un vocabulario a nivel subpalabra originalmente pensado para traducción automática pero que se presta para el entrenamiento de diferentes arquitecturas. Se puede consultar un ejemplo del tokenizador que usa el LLMs de Meta (llama) en el siguiente notebook.

Por último, lo que resulta de este proceso de tokenización es un identificador único que es capaz de lidiar con palabras que nunca antes ha visto o que son compuestas. En el notebook anterior usé *psiconeuroinmunoendocrinología* y un saludo del náhuatl, y pese a ser cadenas de texto de baja (o nula) frecuencia en los datos de entrenamiento, el modelo es capaz de asociarlos a componentes a nivel subpalabra.

1.2.2. Definiciones

Chao et al. [1] toman a $k \in \mathbb{Z}^+$, la distancia de Hamming d_{H} en $\{0,1\}^k$ con $d_{\mathrm{H}}(u,v)$ la cantidad de coordenadas que difieren entre u y v [4] para así definir

Definición 1. Sean $k, n \in \mathbb{Z}^+$ con $k \leqslant n$, un *código corrector de errores (ECC)* es un mapeo inyectivo $\mathcal{C} : \{0,1\}^k \hookrightarrow \{0,1\}^n$.

De aquí se puede definir que la *codificación* de un mensaje m se obtiene aplicándole $\mathcal{C}$ y $\mathcal{C}(m)$ se define como la *codeword* o palabra clave. Por último la tasa de codificación es k/n.

De igual manera, podemos definir la distancia de corrección de error de $\mathcal{C}$. Ésta es la $t > 0$ más grande tal que para cualquier mensaje arbitrario $m \in \{0,1\}^n$ existe a lo más una palabra

[2] En el contexto de los LLMs multimodales debemos pensar en texto como cualquier representación digital, ya sea texto plano, audio, imágenes, video, código, etcétera.

clave $c \in \mathcal{C}$ en a bola de centro m y radio t con la distancia de Hamming. En otras palabras buscamos a la t más grande tal que existe una única c que cumple que $d_{\mathrm{H}}(m, c) \leqslant t$. Podemos caracterizar a nuestro código con esta t y podemos decir que $\mathcal{C}$ es un $[n, k, 2t+1]$-código.

Dado un $[n, k, 2t+1]$-código podemos pensar en un mapeo decodificador $\mathcal{C}^{-1}$. Veamos que está bien definido pues por construcción para cualquier $m \in \{0,1\}^n$ existe a lo más un $c = \mathcal{C}^{-1}(m)$ tal que $d_{\mathrm{H}}(w, c) \leqslant t$. En caso de no existir, la decodificación se puede escoger de manera arbitraria.

2. Watermarking

2.1. Técnicas de watermarking

2.1.1. Hard Red List

2.1.2. Soft Red List

2.1.3. Robust Binary Code (RBC)

Conectando el apartado 1.2.2 y el apartado 1.2.1 podemos pensar en el conjunto de datos de entrenamiento que tenemos para un LLM. Posteriormente, podemos descomponerlo a nivel subpalabra, permitiéndonos procesar vocabulario que no se encuentra en nuestro corpus. De cualquier forma, estas subpalabras se asocian a un número que tiene una representación binaria en forma de bits. Es así que para un vocabulario[3] $\mathcal{V}$ de tamaño $|\mathcal{V}|$ necesitamos $l = \lceil \log_2 |\mathcal{V}| \rceil$. De aquí podemos usar nuestro algoritmo que convierte a binario $\Gamma : \mathcal{V} \to \{0,1\}^l$.

Ahora podemos pensar en la distribución[4] inducida dado un token cualquiera, como los LLMs se entrenan para servir en tareas de predicción de texto tiene sentido preguntarse

> Dados los primeros n bits, ¿cuál es la probabilidad de que el siguiente bit sea un uno?

Denotemos a los primeros n bits como $b_{1:n}$ podemos denotar esta probabilidad con q_{n+1} tal que

$$q_{n+1} = \mathbb{P}(\mathrm{B}_{n+1} = 1 | b_{1:n})$$

Ahora tomemos a nuestro mensaje $m = \{\mathrm{M}_i\}_{i=1}^{|m|} \subset \{0,1\}^k$ con $|m|$ la cantidad de tokens en el mensaje. Podemos escoger a un ECC $\mathcal{C}$ al del cual podemos obtener a una palabra clave $\mathrm{Y}_i = \mathcal{C}(\mathrm{M}_i) \in \{0,1,\}^n$. Posteriormente podemos tomar una muestra de bits $(\mathrm{B}_i)_{i=1}^n$ correlacionada con Y_i.

[3] de subpalabras, aunque no es forzoso verlo de esta manera. Existen algoritmos diferentes al BPE que son capaces de procesar palabras íntegras como Bag of Words (BoW).

[4] Aunque es una interpretación común pensar en esta distribución, a nivel computacional es muy costoso obtener esta probabilidad. Los atajos computacionales usados como el *top-k sampling* devuelven un número muy cercano al que debería ser la probabilidad real pero formalmente esta distribución no está bien definida. Sin embargo, en su uso cotidiano, no se suele hacer esta distinción.

Para obtener esta muestra definimos una v.a. $\mathrm{B} \sim \mathrm{Ber}(q)$ con $q \in [0,1]$, por construcción podemos simular una Bernoulli a partir de una v.a. $\mathrm{U}' \sim \mathrm{Unif}[0,1]$ tal que

$$\mathrm{B} = \mathbb{1}_{\{\mathrm{U}' \leqslant q\}}$$

Una consecuencia inmediata es que U', B son v.a. correlacionadas. Ahora sea veamos que para una $\mathrm{Ber}(1,2)$

$$f(k; \frac{1}{2}) = \left(\frac{1}{2}\right)^k \left(\frac{1}{2}\right)^{1-k} = \frac{1}{2}$$

Es así que podemos observar que $\mathrm{B} = \mathbb{1}_{\left\{\left((1-\mathrm{Y})+\mathrm{U}\right)/2 \leqslant q\right\}} \sim \mathrm{Ber}(q)$ para $\mathrm{Y} \sim \mathrm{Ber}\left(\frac{1}{2}\right)$ y $\mathrm{U} \sim \mathrm{Unif}[0,1]$ vemos que por el parámetro de Y podemos pensar en que $(1-\mathrm{Y})$ nos permite obtener un bit $\mathrm{B} \sim \mathrm{Ber}(q)$ de una manera no uniforme, sesgada a Y. A este proceso se le llama *Correlated Binary Sampling Channel* (CBSC).

Una de las propiedades más importantes que tiene es que $\mathbb{P}[\mathrm{Y} = \mathrm{B}] = 1 - 2|1/2 - q| \geqslant \frac{1}{2}$. En particular la desigualdad se vuelve estricta cuando $q \notin \{0,1\}$ (que podemos pensar como una medida de la entropía de B y por tanto cuando $q = 0, 1$ siempre sabremos cuánto valdrá B), por lo que B está sesgada a Y.

Esta técnica aprovecha este sesgo pero que define a $\mathrm{Y} = (\mathrm{Y}_1, \cdots, \mathrm{Y}_n)$ a partir de una serie de bits conocida. En general por el desarrollo anterior, podemos ver que los bits generados serán sesgados a Y permitiéndonos identificar que provienen de un modelo de lenguaje con la marca de agua explicada anteriormente.

Además RBC utiliza ECCs para modificar en medida de lo posible a $d_{\mathrm{H}}(\mathrm{B}, \mathrm{Y})$ haciendo que en general $d_{\mathrm{H}}(\mathrm{B}, \mathrm{Y}) < \frac{n}{2}$. Esto mediante la selección de un mensaje $\mathrm{M} \in \{0,1\}^k$ que por la definición 1 hace que cuando tomemos un $[n, k, 2t+1]$-ECC $\mathcal{C}$ seamos capaces de tener la capacidad de corregir hasta t errores en $\mathrm{Y} = \mathcal{C}(\mathrm{M}) \in \{0,1\}^n$.

Robust Binary Code. Ahora, retomando todo lo detallado hasta este momento podemos detallar el algoritmo de Chao et Al[1].

Teniendo $\mathcal{V}$ el vocabulario de tamaño $|\mathcal{V}|$ obtenido de nuestra corpora, definimos a $l = \lceil \log_2 |\mathcal{V}| \rceil$ la cantidad de bits necesarios para almacenar cada token en binario, entonces tenemos a $\Gamma : \mathcal{V} \to \{0,1\}^l$ nuestro convertidor binario con su inversa Γ^{-1}[5]. Tomamos también a un $[n, k, 2t+1]$-ECC $\mathcal{C}$ y definimos a la ventana de entrada $w_{\text{in}} = \left\lceil \frac{k}{l} \right\rceil$, la ventana de salida $w_{\text{out}} = \left\lceil \frac{k}{l} \right\rceil$, y la variable aleatoria de cadenas de texto binario $\mathrm{R} \sim \mathrm{Unif}\{0,1\}^k$. De igual forma podemos tomar a nuestro modelo de lenguaje y tomar la distribución del $n+1$-ésimo token dada una ventana de n tokens en su expresión binaria como

$$p_{\text{bin}}(s_1, ..., s_n) = p_{\text{bin}}(\mathrm{S}_{n+1} | s_1, ..., s_n)$$

Por último, dados una cantidad de w_{in} tokens X_i generados por el modelo de lenguaje y N la

[5] Donde podemos solucionar el problema de la inexistencia de la imagen inversa tomando una l suficientemente grande de modo que podamos tomar arbitrariamente un elemento de $\{0,1\}^l$ para cada elemento de $\mathcal{V}$

cantidad de tokens a generar:

1. Iteramos desde $w_{\text{in}} + 1$ hasta N. En cada iteración i realizamos lo siguiente:
   a) Definimos el mensaje $\text{M} \in \{0,1\}^k$ como $\text{M} = \Gamma(\text{X}_{i-w_{\text{in}}:(i-1)})_{1:k} \oplus \text{R}$
   b) Definimos a $\text{Y} = \mathcal{C}(\text{M}) \in \{0,1\}^n$
   c) Creamos w_{out} tokens de la siguiente manera
      1) Tomamos una muestra desde $j = 1$ hasta n bits B_j usando el muestreo sesgado a Y descrito con anterioridad con $q = p_{\text{bin}}(\text{X}, \text{B}_{1:(j-1)})$
      2) Posteriormente, tomamos una muestra desde el bit $j = n + 1$ hasta el bit $w_{\text{out}} \cdot l$ con una Bernoulli de parámetro $q = p_{\text{bin}}(\text{X}, \text{B}_{1:(j-1)})$ con X el texto generado con anterioridad
      3) Por último regresamos el volcabulario correspondiente a B, es decir $\Gamma^{-1}(\text{B})$
   d) Aumentamos a i por w_{out} unidades
2. Al terminar las iteraciones, regresamos todos los tokens generados

Con $\oplus$ el operador bit a bit XOR.

2.2. Detección de watermarking

Para llevar a cabo la detección de nuestra marca de agua tomamos la distancia de Hamming del mensaje M_i, es decir el obtenido en cada iteración del RBC, y lo comparamos con el mensaje que se obtiene aplicando el decodificador $\mathcal{C}^{-1}$ del bloque que se generó usando M_i. Tras cada cálculo, se le resta a k y se realiza por cada una de las ventanas $(\text{N} - w_{\text{out}} - w_{\text{in}} + 1)$. Al final se realiza una prueba de comparación binomial, donde si se detecta que los tokens se generaron con un sesgo a nuestra distribución binomial, entonces se detectará la marca de agua.

3. Validación experimental

Se puede consultar el consultar el notebook dado con anterioridad para ver un ejemplo de cómo se implementan los algoritmos anteriores.

4. Conclusión

Aunque existían métodos de watermarking previos a la escritura de la referencia principal, los Códigos Correctores de Errores permiten sesgar de una manera eficiente y poco detectable a los textos generados por modelos de lenguaje. Esto a su vez nos permite lidiar con algunos de los problemas que uso descontrolado genera a la sociedad, potencialmente permitiendo identificar rápidamente desinformación y propaganda generada de manera automatizada.

Referencias

1. Chao, P., Sun, Y., Dobriban, E. y Hassani, H. *Watermarking Language Models with Error Correcting Codes* arXiv:2406.10281. Feb. de 2025. http://arxiv.org/abs/2406.10281 (2025).
2. Bender, E. M., Gebru, T., McMillan-Major, A. y Shmitchell, S. *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?* en. en *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (ACM, Virtual Event Canada, mar. de 2021), 610-623. ISBN: 9781450383097. https://dl.acm.org/doi/10.1145/3442188.3445922 (2025).
3. Sennrich, R., Haddow, B. y Birch, A. *Neural Machine Translation of Rare Words with Subword Units* arXiv:1508.07909. Jun. de 2016. http://arxiv.org/abs/1508.07909 (2025).
4. Kerl, J. *An introduction to coding theory for mathematics students* sep. de 2004. https://johnkerl.org/doc/kerl-ecc-intro.pdf.
5. Kirchenbauer, J. *et al. A Watermark for Large Language Models* arXiv:2301.10226. Mayo de 2024. http://arxiv.org/abs/2301.10226 (2025).
6. Beckmann, P. Natural languages as error-correcting codes. en. *Lingua* **28,** 251-264. ISSN: 00243841. https://linkinghub.elsevier.com/retrieve/pii/002438417190060X (2025) (ene. de 1971).
7. Kaplan, N. *Error-Correcting Codes: The Mathematics of Communication* en. Jul. de 2022. https://www.math.uci.edu/~nckaplan/research_files/momath_slides.pdf.
8. Calvino, A. *An Introduction to Coding Theory with Reed-Solomon Codes as a Real-World Example* Tesis doct. (Benedictine College, mar. de 2019). https://acalvino4.github.io/CodingTheory/CodingTheory.pdf.