Modelos de lenguaje distribucionales sobre el corpus de entrevistas*

Vladimir Vargas Calderón**

27 de marzo de 2021

Resumen

En este documento se reporta la creación de un recurso lingüístico asociado a las entrevistas de la Comisión para el Esclarecimiento de la Verdad, la Convivencia y la No Repetición (CEV) que permite representar vectorialmente el texto. Dicho recurso lingüístico fue creado mediante el modelo de *word embeddings* FastText. Se obtiene, por lo tanto, la representación vectorial de palabras encontradas en los relatos de las personas entrevistadas en las distintas regiones del país, constituyendo no solo una herramienta para la explotación de textos en la CEV, sino una herramienta lingüística de gran valor a nivel nacional, que incluye representación de palabras características de lugares y grupos poblacionales de Colombia y sus regiones. Este documento, por lo tanto tiene como objetivo mostrar aspectos teóricos del recurso lingüistico generado que será explotado en el futuro cercano dentro de la CEV.

1. Introducción

Dentro del equipo de texto de la Comisión para el Esclarecimiento de la Verdad, la Convivencia y la No Repetición (CEV), hemos decidido construir un recurso lingüístico que permite tener un mapa de "strings", en el sentido computacional de la palabra, a vectores en un espacio vectorial $\mathbb{R}^N$ con dimensión N. A este tipo de recursos lingüísticos se les conoce como *word embeddings* [17] y facilitan realizar diversas tareas como clasificación, clustering, regresión, entre otros. El primer modelo de lenguaje que produjo *word embeddings* de forma exitosa y popular en el mundo fue Word2Vec [9, 10, 11]. Este modelo concretó ideas de la teoría distribucional lingüística en la que las palabras tienen sentido gracias a su correlación y co-ocurrencia con otras palabras: el significado depende del contexto en el que se usan las palabras [4, 12].

La posibilidad de tener un mapa que nos lleva del espacio de los textos a un espacio vectorial de dimension fija N permite incluir el texto como características numéricas que pueden alimentar cualquier modelo estadístico (o de machine learning). Usualmente, distintas tareas de machine learning requieren de un input dado con un conjunto de características fijo para producir un output. En este caso, el conjunto de características de input son las mismas componentes de la representación vectorial de los textos. Dichas tareas incluyen análisis de sentimientos en donde se espera clasificar un texto en aspectos de opinión positivos o negativos [14]; modelamiento de temas, en donde se espera descubrir qué temas latentes existen dentro de un corpus de texto, i.e.una colección de representaciones vectoriales para cada texto [8, 16]; estimación de puntajes de algún producto según un texto dado [13]; entre otros.

Aún más, la construcción de este tipo de modelos de lenguaje, permite realizar operaciones entre palabras que dan cuenta de recursos lingüísticos muy interesantes. Uno de ellos es la analogía, en donde se intenta encontrar una palabra X que satisfaga A es a B como C es a X, con las palabras A, B, C dadas [5]. Cada palabra, según el modelo de lenguaje, tendrá una representación vectorial: $\vec{x}_A, \vec{x}_B, \vec{x}_C \in \mathbb{R}^N$, y la palabra X se encuentra mediante el cálculo de $\vec{x}_{\tilde{X}} := \vec{x}_A - \vec{x}_B + \vec{x}_C$, y la posterior búsqueda de una palabra X del conjunto de palabras V en el modelo de lenguaje, tal que $||\vec{x}_{\tilde{X}} - \vec{x}_X|| = \text{mín}_{Y \in V}(||\vec{x}_{\tilde{X}} - \vec{x}_Y||)$.

Este documento presenta la metodología seguida para construir el recurso lingüístico de *word embedding* aplicado al corpus de entrevistas de la CEV. Para ello se siguieron dos grandes pasos metodológicos: en la

*El presente informe se entrega como cumplimiento de la etapa 4 del **plan de trabajo** realizado por Vladimir Vargas Calderón y aprobado por la Comisión y el PNUD, por medio del contrato 0000046508.

**vladimir.vargas@comisiondelaverdad.co

sección 2 se explica detalladamente el proceso de preprocesamiento de texto. Después, en la sección 3 se explica conceptualmente el modelo FastText [6, 7, 1], que es una evolución computacional y lingüística de Word2Vec.

2. Preprocesamiento

En la CEV hemos construido una librería de Python para realizar preprocesamiento de texto llamada `preprocess`, actualmente en su versión `v0.0.2`. El preprocesamiento de texto tiene como finalidad disminuir la variabilidad lingüística de un texto, y la retención de aquellas palabras que aportan a la semántica del mismo. En particular, los pasos realizados en el preprocesamiento por la librería `preprocess` son:

- **Normalización de codificación del texto**. Debido a las distintas codificaciones de texto, estandarizar y normalizar esta codificación puede ser importante para pasos posteriores del preprocesamiento.
- **Caracteres alfabéticos a minúsculas**. Para disminuir la variabilidad del texto se pasa el texto a minúsculas. Esto se hace debido a que "Asesinato" y "asesinato" hacen referencia a lo mismo para el lector humano, pero son palabras diferentes para el computador.
- **Remoción de puntuación**. En principio no nos interesa la puntuación, sino únicamente una cadena de palabras que tienen significado por las palabras a su alrededor. La única "puntuación" que nos interesa es el cambio de párrafo por un punto y aparte, ya que usualmente los párrafos son grupos de ideas. En las entrevistas, los párrafos corresponden a intervenciones de los interlocutores. Por este motivo, nos interesa concatenar todas las oraciones de un mismo párrafo como si fuera una gran oración.
- **Remoción de *stopwords***. Algunas palabras no aportan al significado o las relaciones entre palabras importantes (e.g.sujetos, verbos, adjetivos). A las palabras que no aportan las llamamos *stopwords*. Entre ellas se encuentran palabras como la, el, como, donde, algún, entre otros.
- **Lematización**. Este es un proceso clave del preprocesamiento: se lleva cada palabra a su lema, que es la forma canónica, la forma de diccionario de una palabra. Por ejemplo, el lema "ir" representa todas las siguientes formas flexionadas: "voy", "yendo", "vamos", entre otros. Para realizar la lematización se utiliza el framework que provee la librería spaCy.
- **Tokenización**. Usualmente, para la realización de algunas de las tareas de preprocesamiento anteriormente descritas es necesario convertir una cadena de texto en una lista de textos pequeños, llamados *tokens*, que son la mínima partición de caracteres del texto que posee un significado. Para textos "bien escritos", estos *tokens* son palabras y signos de puntuación. En contextos de redes sociales, emoticones o risas ("jajaja") también son *tokens*. Otros elementos pueden ser definidos como tokens, tales como colocaciones comunes, e.g."gracias a Dios", puede ser tokenizado a "gracias_a_Dios".

La librería 'preprocess' permite realizar estos pasos de preprocesamiento de forma muy sencilla. Por ejemplo, la librería, así como un modelo de lenguaje natural (que incluye tokenización y lematización) de spaCy pueden ser cargados de la siguiente manera:

```
from preprocess import PreprocessPipeline
import spacy

nlp = spacy.load("es_core_news_sm")
```

Después, se puede definir un pipeline de preprocesamiento mediante `pp = PreprocessPipeline(nlp=nlp)`. La clase `PreprocessPipeline` permite inicializar objetos con una serie de opciones. La función de inicialización y su documentación se presenta a continuación:

```
def __init__(
    self,
    normalise: bool = True,
    lower: bool = True,
    remove_punctuation: bool = True,
    remove_stopwords: bool = True,
    lemmatise: bool = True,
    tokenise: bool = True,
```

```
    string_punctuation: bool = True,
    symbols: List[str] = None,
    language: str = "spanish",
    additional_stopwords: Union[str, List[str]] = None,
    mwes: Union[str, List[str]] = None,
    mwe_tokeniser: MWETokenizer = None,
    nlp: spacy.Language = None,
):
    """
    Initialises a preprocessing pipeline.

    Parameters
    ----------
    normalise: bool
        Whether to normalise text with a proper encoding.
    lower: bool
        Whether to lowercase text.
    remove_punctuation: bool
        Whether to remove punctuation.
    remove_stopwords: bool
        Whether to remove stopwords.
    lemmatise: bool
        Whether to perform lemmatisation.
    tokenise: bool
        Whether to tokenise text or not.
    Other args are defined in the functions from `preprocess.preprocess`
    """
```

Sin embargo, con la inicialización del objeto anteriormente presentada es posible no solamente preprocesar textos individuales, sino preprocesar textos aprovechando todos los núcleos de preprocesamiento de la máquina, realizando computos distribuidos sobre todos los hilos:

```
# Definición del pipeline
pp = PreprocessPipeline(nlp=nlp)
# Ejemplo de un texto:
print(pp("hola amiguitos! qué tal todo?"))
# Ejemplo de mil textos procesados en 3 hilos de forma paralela
print(pp(["hola amiguitos! qué tal todo?"]*1000, num_cpus=3))
```

en donde se usan 3 CPUs.

3. FastText[1]

FastText es una librería que crea *word embeddings*. Esto significa que un string s es mapeada a un vector en un espacio vectorial $\mathbb{R}^N$. El método de FastText comparte las ideas de embebimiento de otros modelos anteriores como Word2Vec [1, 7, 10]. A continuación, se mostrarán las ideas generales de cómo se construye un *word embedding*. El lector interesado en más detalles es referido a artículos como [2, 3, 15], en donde se muestra formalmente el método con detalle matemático.

Considere un conjunto de textos o documentos $\mathcal{D}$. Podemos crear un conjunto llamado vocabulario $\mathcal{V}$ como el conjunto de todas las palabras contenidas en los documentos presentes en $\mathcal{D}$. Podemos ordenar el conjunto $\mathcal{V}$

[1] Esta sección está basada en la última versión del trabajo divulgado en un trabajo de la autoría de Vladimir Vargas-Calderón, llamado *"Machine learning for assessing quality of service in the hospitality sector based on customer reviews"*, que está en rondas de revisión en la revista *Information Technology & Tourism*.

de manera arbitraria, pero por simplicidad, asumimos que tenemos un vocabulario ordenado alfabéticamente. Sea $V = |\mathcal{V}|$ el tamaño del vocabulario. Considere un mapa de codificación *one-hot*

$$\phi : \mathcal{V} \to \{0,1\}^V \subset \mathbb{R}^V \tag{1}$$

$$w_i \mapsto \phi(w_i) = \vec{\phi}_i \tag{2}$$

definido como una función que toma el i-ésimo elemento del vocabulario (en el orden alfabético preestablecido) y lo mapea a un vector $\vec{\phi}_i$ cuya única componente distinta de 0 es la i-ésima componente, tomando el valor 1. El embebimiento, es definido como una matriz W de tamaño $N \times V$ que toma un vector del vocabulario codificado por medio de la función *one-hot* ϕ, y lo lleva a un vector en el espacio embebido $\mathbb{R}^N$, en donde $N \ll V$. En general el vector en el espacio embebido es denso.

Por lo tanto, el embebimiento se lleva a cabo, para una palabra cualquiera del vocabulario $w_i \in \mathcal{V}$, por medio de la operación $W\phi(w_i) = W\vec{\phi}_i$. Note que, debido a la construcción de la función ϕ, esta operación indica la selección de la i-ésima columna de W. La característica principal de este modelo de embebimiento es que palabras que sean semánticamente similares, tendrán representaciones vectoriales similares en el espacio embebido. Formalmente esto significa que

$$\frac{\vec{w}_i \cdot \vec{w}_j}{(||\vec{w}_i||\,||\vec{w}_j||)} \approx 1 \tag{3}$$

para palabras similares $w_i, w_j \in \mathcal{V}$.

Naturalmente, una pregunta a responder es: ¿cómo puede uno medir similitud semántica? Mikolov etal. [10] definen la similitud semántica mediante un problema de predicción que tiene su orígen en la hipótesis distribucional de la lingüística [4], que postula que palabras semánticamente similares se utilizan en contextos similares. Por ejemplo, se espera que las palabras "cortesía" y "amabilidad" compartan representaciones vectoriales similares debido a que pueden ser encontradas en textos con contextos similares. El contexto se define formalmente como el conjunto de palabras que rodean una palabra de interés. La cantidad de palabras alrededor de la palabra de interés tenidas en cuenta dentro del contexto usualmente se conoce como el tamaño del contexto. La definición del contexto nos permite postular el problema de predicción que define la similitud semántica: dado un contexto alrededor de una palabra de interés w_i, ¿se puede predecir que la palabra de interés es w_i? O, similarmente, dada una palabra de interés w_i, ¿se puede predecir su contexto? Estas dos preguntas son dos modos o configuraciones de las arquitecturas tipo Word2Vec, y se conocen como la bolsa de palabras continua (CBOW) o el *skip-gram*, respectivamente.

Como ejemplo, consideremos la configuración CBOW. Consideremos una parte de una oración que consiste de una palabra de interés w (note que eliminamos el sub-índice) y un contexto de tamaño 4: $w_1\, w_2\, w\, w_3\, w_4$. Usaremos las palabras del contexto para predecir la palabra de interés. Esto se realiza primero promediando el vector de representación de las palabras de contexto:

$$\vec{w}_c := \frac{1}{4}\sum_{i=1}^{4} W\phi(w_i) = \frac{1}{4}\sum_{i=1}^{4} \vec{w}_i \tag{4}$$

La predicción de la palabra de interés se realiza[2] calculando $W^T\vec{w}_c$, que debería ser igual a la representación *one-hot* de la palabra w, es decir, a $\vec{\phi}_w$. Los elementos de matriz de W pueden ser aprendidos mediante cualquier algoritmo de minimización de una función de pérdida, como la entropía cruzada categórica, construida muestreando pares de tipo palabra de interés, palabras de contexto, y prediciendo palabras de interés dadas sus palabras de contexto.

FastText eleva esta idea recién presentada para aprender embebimientos de información encontrada en sub-palabras. En vez de lidiar con un vocabulario de palabras, FastText considera un vocabulario de cadenas de n caracteres. Para entender esto, consideremos una oración que contiene la palabra "cortesía". Definimos

[2]Aquí la predicción se realiza con la misma matriz W. Sin embargo, en la práctica, hay dos matrices W_1 y W_2, cada una correspondiente a un embebimiento. De esta manera, la predicción realmente se realiza mediante $W_2\vec{w}_c = \frac{1}{4}W_2\sum_{i=1}^{4} W_1\phi(w_i)$.

dos caracteres especiales ⟨ y ⟩ para marcar donde una palabra comienza y donde termina. De esta manera, "cortesía" se transforma en "⟨cortesía⟩". Si consideramos cadenas de 5 caracteres, obtenemos la siguiente descomposición de "cortesía":

- ⟨cort
- corte
- ortes
- rtesí
- tesía
- esía⟩

Ahora, se aprenden representaciones vectoriales para cada una de las cadenas de 5 caracteres encontradas en el vocabulario, de la misma manera en que se hacía con las palabras de contexto. Ahora, la representación de una palabra es el promedio de la representación de las cadenas que forman su descomposición. Esto puede ser extendido para representar oraciones también promediando las representaciones de sus palabras. Este modelo es muy potente debido a que permite construir representaciones vectoriales de palabras nunca antes vistas en el corpus.

En el repositorio de FastText de la CEV se encuentra, por el momento, disponibilizado un modelo de FastText para realizar tareas asociadas a machine learning utilizando representaciones vectoriales del texto.

Referencias

[1] Bojanowski, P., E.Grave, A.Joulin, and T.Mikolov (2016, July). Enriching Word Vectors with Subword Information. *arXiv e-prints*, arXiv:1607.04606.

[2] Dyer, C. (2014). Notes on noise contrastive estimation and negative sampling. *arXiv preprint arXiv:1410.8251*.

[3] Goldberg, Y. and O.Levy (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

[4] Harris, Z.S. (1954). Distributional structure. *Word10*(2-3), 146–162.

[5] Hartmann, N., E.Fonseca, C.Shulby, M.Treviso, J.Rodrigues, and S.Aluisio (2017). Portuguese word embeddings: Evaluating on word analogies and natural language tasks. *arXiv preprint arXiv:1708.06025*.

[6] Joulin, A., E.Grave, P.Bojanowski, M.Douze, H.Jégou, and T.Mikolov (2016, December). FastText.zip: Compressing text classification models. *arXiv e-prints*, arXiv:1612.03651.

[7] Joulin, A., E.Grave, P.Bojanowski, and T.Mikolov (2016, July). Bag of Tricks for Efficient Text Classification. *arXiv e-prints*, arXiv:1607.01759.

[8] Li, C., H.Wang, Z.Zhang, A.Sun, and Z.Ma (2016). Topic modeling for short texts with auxiliary word embeddings. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 165–174.

[9] Mikolov, T., K.Chen, G.Corrado, and J.Dean (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[10] Mikolov, T., I.Sutskever, K.Chen, G.Corrado, and J.Dean (2013). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

[11] Mikolov, T., W.-t. Yih, and G.Zweig (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 746–751.

[12] Sahlgren, M. (2008). The distributional hypothesis. *Italian Journal of Disability Studies20*, 33–53.

[13] Suryadi, D. and H.Kim (2018). A systematic methodology based on word embedding for identifying the relation between online customer reviews and sales rank. *Journal of Mechanical Design140*(12).

[14] Tang, D., F.Wei, N.Yang, M.Zhou, T.Liu, and B.Qin (2014). Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1555–1565.

[15] Vargas-Calderón, V. and J.E. Camargo (2019). Characterization of citizens using word2vec and latent topic analysis in a large set of tweets. *cities92*, 187–196.

[16] Yi, F., B.Jiang, and J.Wu (2020). Topic modeling for short texts via word embedding and document correlation. *IEEE Access8*, 30692–30705.

[17] Zamani, H. and W.B. Croft (2016). Embedding-based query language models. In *Proceedings of the 2016 ACM international conference on the theory of information retrieval*, pp. 147–156.