

## TEMA 3 – HOJA DE EJERCICIOS II

Se plantean diferentes ejercicios basados en modelos preentrenados de embeddings estáticos y contextuales de palabras.

### Ejercicio 1. Embeddings Estáticos – Relaciones semánticas

Los modelos basados en embeddings estáticos proporcionan una representación vectorial de palabras en un espacio continuo de dimensiones fijas, donde cada palabra tiene un único vector asociado independientemente del contexto en el que aparezca. Estos modelos son capaces de capturar la semántica de las palabras a partir de su entrenamiento no supervisado sobre grandes corpus de texto.

Word2Vec es uno de los modelos que aprende estas representaciones vectoriales mediante el entrenamiento sobre ventanas de contexto. En concreto, emplea la técnica Skip-gram, que predice palabras de contexto a partir de una palabra objetivo, o CBOW, que predice la palabra objetivo a partir de su contexto.

Por otro lado, GloVe es otro de los modelos para la generación de embeddings estáticos, basándose para ello en el análisis de coocurrencias de palabras. Específicamente, GloVe basa su entrenamiento en la construcción de una matriz de coocurrencia de palabras, factorizándola para la obtención de representaciones vectoriales.

Asimismo, otro modelo relevante para la generación de estos embeddings es FastText. A diferencia de Word2Vec y Glove, FastText no trabaja con palabras individuales, sino que lo hace a nivel de subpalabra o n-gramas de carácteres. Esta representación permite una mayor robustez ante palabras infrecuentes o con errores tipográficos, solucionando el famoso problema de palabras fuera del vocabulario (OOV por sus siglas en inglés).

En este ejercicio, se desea explorar el uso de los modelos de embeddings estáticos de palabras Word2Vec, GloVe y FastText para el análisis de relaciones semánticas. Una propiedad interesante de estos modelos es su capacidad para capturar relaciones semánticamente analógicas, como las presentes en la famosa ecuación:

$$\text{king} - \text{man} + \text{woman} = \text{queen}$$

- Usa la biblioteca gensim para cargar los modelos preentrenados "word2vec-google-news-300", "glove-wiki-gigaword-300".

- Descarga el modelo de FastText a través del siguiente enlace <https://dl.fbaipublicfiles.com/fasttext/vectors-wiki/wiki.simple.zip>
- Una vez extraído el modelo de FastText "wiki.simple.bin", cargarlo mediante el siguiente código de ejemplo. Nota que <TU\_PATH> debe reemplazarse por la ubicación donde se encuentra el modelo en tu sistema de archivos:

```
from gensim.models.fasttext import load_facebook_model  
  
ft = load_facebook_model('<TU_PATH>wiki.simple.bin').wv
```

- Con Word2Vec, GloVe y FastText, emplea la función most\_similar() de dicha biblioteca para encontrar la palabra que mejor completa la anterior ecuación. ¿Qué palabras devuelve cada modelo? Reflexiona sobre los resultados.

Nota:

- <https://radimrehurek.com/gensim/downloader.html>
- [https://tedboy.github.io/nlp/generated/generated/gensim.models.Word2Vec.most\\_similar.html](https://tedboy.github.io/nlp/generated/generated/gensim.models.Word2Vec.most_similar.html)

## Ejercicio 2. Embeddings Estáticos – Palabras fuera del vocabulario

Tanto Word2Vec como GloVe y FastText proporcionan embedding estáticos a nivel de palabra. Dada una lista de palabras, podemos obtener una matriz de embeddings, donde las filas son las palabras que componen la lista y las columnas son el número de dimensiones que tienen los embeddings.

- Con Word2Vec, GloVe y FastText, obtén la matriz de embeddings del conjunto de palabras ["apple", "orange", "banana"]. Es decir, unir los embeddings de cada una de estas palabras individualmente y forman una matriz. ¿Qué dimensiones tienen? ¿Qué características tienen dichas matrices (dispersa vs densa, números enteros vs flotantes, ...)?
- Con Word2Vec, GloVe y FastText, intenta obtener el embedding de la palabra "orangge". ¿Qué modelos permiten obtenerlo? ¿Cuáles no? ¿A qué se debe?

Nota:

- <https://radimrehurek.com/gensim/models/word2vec.html>
- <https://radimrehurek.com/gensim/models/fasttext.html>

### Ejercicio 3. Embeddings Estáticos – Visualización de Embeddings

A través del algoritmo de reducción de dimensionalidad PCA se puede visualizar gráficamente el espacio de embeddings representado por estos modelos. A continuación, se muestra un código de ejemplo que permite visualizar un conjunto de palabras *words* dados sus correspondientes embeddings.

```
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

word_vectors_2d = pca.fit_transform(embeddings)

plt.figure(figsize=(8, 6))

for word, coord in zip(words, word_vectors_2d):

    plt.scatter(coord[0], coord[1], marker='o', label=word)
    plt.text(coord[0], coord[1], word, fontsize=12)

plt.grid()
plt.show()
```

- Emplea Word2Vec, GloVe y FastText para obtener los embeddings de la lista de palabras ["woman", "king", "man", "queen"]. Aplica PCA sobre estos embeddings y represéntalos gráficamente con Matplotlib. ¿Qué palabras se muestran cerca?
- Emplea Word2Vec, GloVe y FastText para obtener los embeddings de la nueva lista de palabras ["woman", "king", "man", "queen", "potato", "carrot", "tomato"]. Aplica PCA sobre estos embeddings y represéntalos gráficamente con Matplotlib. ¿Qué palabras se muestran cerca en esta ocasión?

#### Ejercicio 4. Embeddings Estáticos – Representación de frases

Aunque Word2Vec, GloVe y FastText proporcionan embeddings estáticos a nivel de palabra, existen diferentes técnicas para obtener embeddings a nivel de frase. Una de las formas más populares consiste en calcular la media de embeddings de los diferentes tokens que componen una frase. Otra estrategia también frecuente consiste en quedarse con el máximo de cada dimensión, resaltando las características más sobresalientes de cada token analizado.

Dada la frase de consulta "Dogs are domestic animals." y el conjunto de frases ["Dogs are pets.", "This is a dog.", "They are free today."]

- Tokeniza las anteriores frases y elimina stopwords y puntuación.
- Con Word2Vec, GloVe y FastText, obtén el vector promedio y el vector máximo de la frase de consulta y el conjunto de frases.
- Con Word2Vec, GloVe y FastText, obtén el vector de media ponderada por IDF. Para ello, construye un diccionario de (palabras, su valor de idf).
- Calcula la similitud coseno entre la frase de consulta y cada frase del conjunto de frases usando los anteriores vectores. Reflexiona sobre qué frases son consideradas más similares por estos modelos y qué vectores funcionan mejor (media, máximo, o media ponderada por IDF).

Nota:

- [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine\\_similarity.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html)
- <https://numpy.org/doc/2.2/reference/generated/numpy.mean.html>
- <https://numpy.org/doc/2.2/reference/generated/numpy.max.html>
- <https://numpy.org/doc/2.2/reference/generated/numpy.average.html>
- [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html#sklearn.feature\\_extraction.text.TfidfVectorizer.get\\_feature\\_names\\_out](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer.get_feature_names_out)

### Ejercicio 5. Embeddings Contextualizados – BERT

Uno de los modelos más representativos dentro de esta categoría de embeddings es BERT. A diferencia de ELMo, BERT utiliza una arquitectura basada en mecanismos de atención y transformers, lo que le permite considerar el contexto completo en el que se encuentra una determinada palabra al generar su representación vectorial.

De igual forma, BERT proporciona embeddings contextualizados a nivel de palabra en la salida de su última capa. Una vez obtenidos los outputs generados por BERT, es posible acceder a las representaciones vectoriales de las palabras procesadas mediante el atributo `last_hidden_state`:

```
tokens = tokenizer(text, padding=True, truncation=True,  
                   return_tensors="pt")  
  
with torch.no_grad():  
  
    outputs = model(**tokens)  
  
outputs.last_hidden_state.numpy()[:, 1:-1, :]
```

Dada nuevamente la frase de consulta "Dogs are domestic animals." y el conjunto de frases ["Dogs are pets.", "This is a dog.", "They are free today."] se pide lo siguiente:

- Con el modelo y tokenizador de BERT 'bert-base-uncased', obtén el vector promedio de la frase de consulta y el de cada frase del conjunto de frases.
- Con el modelo y tokenizador de BERT 'bert-base-uncased', obtén el vector máximo de la frase de consulta y el de cada frase del conjunto de frases.
- Para las anteriores vectorizaciones, calcula la similitud coseno entre la frase de consulta y cada una del conjunto de frases. Reflexiona sobre qué frases son consideradas más similares por estos modelos y qué vectores funcionan mejor (media o máximo).

Nota: <https://huggingface.co/google-bert/bert-base-uncased>