TEMA 3 – HOJA DE EJERCICIOS IV

Se plantean diferentes ejercicios basados en modelos preentrenados de embeddings estáticos 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:

$$king - man + woman = queen$$

- 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/nlps/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://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

Ejercicio 5. Embeddings Estáticos – Entrenamiento de los modelos.

Se plantea un ejercicio para entrenar un modelo de embeddings desde cero. En este caso, se van a seleccionar los tres libros de Jane Austen que están disponibles en el proyecto Gutenberg de NLTK.

Los pasos que hay que seguir son los siguientes:

1) Coger el texto de los tres libros.
2) Hacer un preprocesamiento de los textos. Utilizar esta vez spacy:
    a. Convertir todo a minúsculas.
    b. Eliminar todos los tokens que no sean palabras (signos de puntuación…)
3) Cargar el modelo de Word2Vec (`from gensim.models import Word2Vec`)
4) Entrenar el modelo (se puede utilizar el siguiente código):

```
w2v_model = Word2Vec(sentences=corpus_sentences,
                     vector_size=300,      # Dimensión del vector (300)
                     window=8,             # Ventana más amplia para capturar más contexto (8)
                     min_count=2,          # Filtrar palabras poco frecuentes (5)
                     workers=4,
                     sg=1,                 # Usar Skip-gram
                     epochs= 30,           # Más ciclos de entrenamiento
                     negative= 15,         # Tamaño de Muestra Negativa (Negative sampling)
                     alpha= 0.025,         # Learning rate inicial
                     min_alpha= 0.0001     # Learning rate final
                    )
```

5) Guardar el modelo (`w2v_model.save("austen_w2v.model")`).

Una vez hecho todo lo anterior, se pide lo siguiente:

1. Sacar estadísticas del modelo (dimensión de los vectores y número de palabras).
2. Mostrar las palabras más similares a ‘love’.
3. Mostrar las palabras más similares a ‘hate’.
4. Mostrar la similitud entre las palabras ‘love’ y ‘art’.
5. ¿Y si miramos una palabra que no esté seguro en el vocabulario? Por ejemplo, buscar las palabras más similares a ‘pianista’.
6. Buscar analogías entre

    love- hate + heart = ¿?

    Entonces, a la hora de buscar las palabras similares hay que indicar las palabras positivas (‘love’ y ‘heart’ en este caso) y negativas (‘hate’ en este caso).

Hacer lo mismo, pero entrenando un modelo de FastText.

```
ft_model = FastText(sentences=corpus_sentences,
                    vector_size=300,    # Aumentar dimensionalidad (300)
                    window=8,           # Ventana más amplia para capturar más contexto (8)
                    min_count=2,        # Mantener min_count bajo para capturar más variantes
                    workers=4,
                    sg=1,               # Skip-gram para mejor calidad
                    min_n=2,            # Tamaño mínimo de n-gramas
                    max_n=6,            # Tamaño máximo de n-gramas (aumentado para capturar más patrones)
                    epochs=30,          # Más ciclos de entrenamiento (30)
                    word_ngrams=1,      # Habilitar n-gramas de palabras
                    negative=15,        # Más muestras negativas
                    alpha=0.025,        # Learning rate inicial
                    min_alpha=0.0001    # Learning rate final
                )
```