Tarea #1 de PLN
Representación vectorial con Fasttext, tokenización y embeddings
Procesamiento de lenguaje natural con Deep Learning

Introducción. El objetivo de este taller es familiarizarse con una serie de conceptos relacionados con datasets, limpieza de textos, tokenización, embeddings, visualización. En la carpeta compartida de drive van a encontrar tres aplicaciones que les ayudarán en el taller. La primera es **word2vec_español.ipynb** que tiene a su vez tres partes; 1) el uso de un archivo ya vectorizado y creación del modelo **model**, este es un preentrenado que genera Gensim `from gensim.models.word2vec import Word2Vec`.2) La consulta de palabras similares con **model.most_similar ()** con respecto a las posibles palabras del vocabulario. 3) La visualización de palabras en el plano cartesiano donde **tsne** reduce la dimensión de 300 a la dimensión 2 para poder visualizar las palabras. Para el problema 2, se recomienda que no se tome el archivo de **palabras.csv** sino que se adapte el código para procesar el conjunto de palabras del texto de Cien años de soledad. La otra aplicación es **word2_skip-gram.ipynb** que sirve para generar el archivo vectorizado de las palabras de un texto o dataset. En este caso el ejemplo se hizo con Word2vec para generar el vectorizado de la biblia y para la tarea, es similar pero con Fasttext, esto es para que con este archivo generado se puedan realizar consultas con **model.most_similar().** La tercera aplicación **Word2vec_scratch (1).ipynb** es la que se debe usar para el problema 2 y parte de la visualización de la aplicación de **word2vec_español.ipynb.**

1. Conceptos básicos de insumo para el taller

1.1 Fasttext. A principios de 2017, Facebook AI Research publicó un artículo que presentaba a FastText. FastText es una biblioteca liviana, gratuita y de código abierto que permite a los usuarios aprender representaciones de texto y clasificadores de texto. Funciona en hardware genérico estándar. FastText se basa en el método skip-gram pero **mitiga** la limitación de palabras que no están en el vocabulario. La idea importante de FastText en la generación de embeddings de las palabras desconocidas es que se dividen las palabras en secuencias más pequeñas de caracteres llamados n-gramas. Por ejemplo, para n = 2, un 2-grama para la palabra perro sería; “<pe”, “rro>” y una secuencia especial “<perro>” que denota la palabra completa. Este método es eficaz porque aprende representaciones de subpalabras que se comparten entre diferentes palabras y por lo tanto, una palabra no vista se disecciona en sus n-gramas de composición que muy probablemente se hayan visto durante el entrenamiento. El embedding de la palabra final se calcula como la suma de sus embeddings de los

n-gramas constituyentes. Como se puede observar en la siguiente figura FastText usa el concepto de subpalabras y n-gramas a diferencia del Word2vec que usa la asociación de la palabra centro con la palabra más cercana. La innovación más importante de FastText es que enriqueció el modelo de Word2vec con la información de subpalabras.

En la siguiente figura se muestra el código de un ejemplo de la definición de características de construcción para vectorizar un texto usando la librería de Gensim con **from gensim.models import FastText.** Hay que instalar Gensim**.**

```
[ ] num_features = [300]                          #Dimensionalidad del embeddings
    min_word_count = 1                            #Umbral mínimo de recuento de palabras
    num_workers = multiprocessing.cpu_count()     #Número de subprocesos que se ejecutaren en paralelo
    context_size = 5                              #Longitud de la ventana de contexto
    seed = 1
```

Luego se definen las características del modelo **fasttext_model** y en la siguiente figura se genera el archivo **text_fasttext_skip_modelo_300.txt** con las palabras vectorizadas por el modelo de Fasttext. Hay que tener en cuenta que el archivo vectorizado para Fasttext usa el formato de word2vec aunque el modelo se diferencia en la forma de vectorizar. En la variable **sentences** se definen las sentencias del datasets como una lista de listas de las sentencias del dataset.

```
for p in num_features:
    fasttext_model = FastText(
        sentences=sentences,
        vector_size=300,
        window=context_size,
        min_count=min_word_count,
        workers=num_workers,
        sg=1                                #Skipgram
    )

    fasttext_model.wv.save_word2vec_format('./text_fasttext_skip_model_' + str(p) +  '.txt', binary=False)

    #del fasttext_model
```

1.2 Limpieza de textos (text cleaned)

La limpieza del texto consiste en quitarle al texto las palabras que más se repiten como los determinantes, adjetivos, adverbios, conocidas como las palabras stop. Con nltk se pueden visualizar y con base en ella se pueden definir reglas para que sean eliminadas del texto.

```
import nltk
nltk.download('stopwords')
```

Para este taller la limpieza del texto consiste de:

1. Eliminación de los números del texto
2. Eliminación de las palabras stopwords
3. Eliminación de la puntuación
4. Eliminación de otros símbolos especiales

En la siguiente figura se definen las expresiones regulares en Python usando la librería re para limpiar una sentencia de texto.

```
newstring = [i for i in nom if not i.isdigit()]
newstring = [re.sub(r'[0-9]','', w) for w in newstring]
stop_words = stopwords.words('spanish')
newstring = [w for w in newstring if not w in stop_words]
re_punc = re.compile('[%s]' % re.escape(string.punctuation))
newstring = [re_punc.sub('', w) for w in newstring]
newstring = [re.sub("\!|\'|\?|\¿|\¡|\«","",w) for w in newstring]
sentencias1.append(newstring)
```

2. *Problemas a solucionar*

El taller consiste de dos partes:

2.1 Entrenar Fasttext y comparar con Word2vec.

Objetivo. La idea es tratar de preprocesar el dataset Europarl (Large Corpus Spanish), tokenizarlo, limpiarlo, obtener el modelo de embedding usando Fasttext y comparar la similaridad con el modelo Word2vec

2.1.1 Problema 1. (35 puntos)

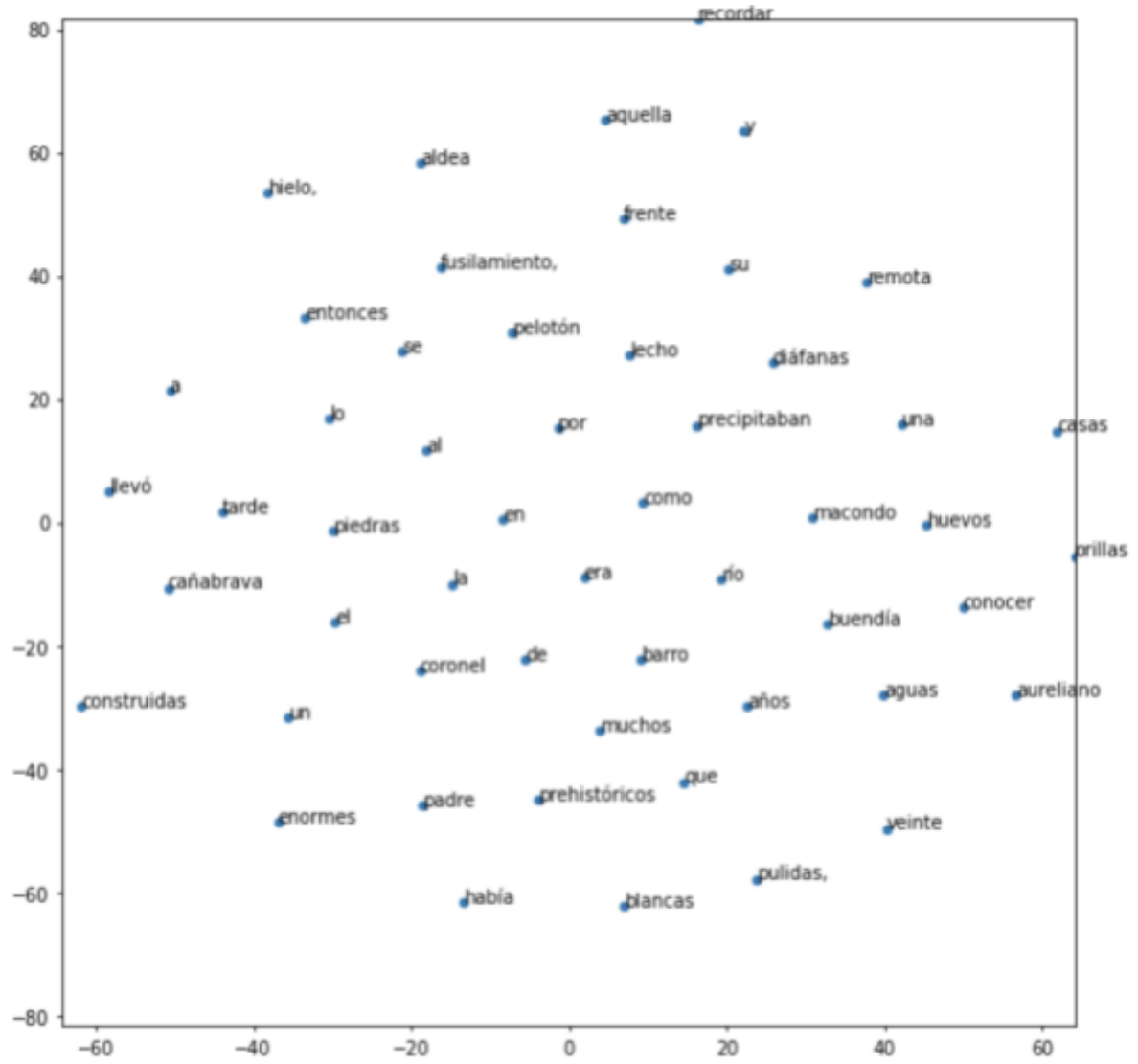
Dado el dataset Europarl (Large Corpus Spanish) del corpus Large corpus Spanish se debe realizar varios pasos hasta obtener el embedding de cada una de las palabras del corpus usando FastText. Pare ello se deben realizar las siguientes tareas de preprocesamiento:

1) Traer el dataset de Huggin Face
2) Segmentar el texto en sentencias, se debe crear una lista de listas con cada sentencia y luego tokenizadas por palabras.
3) Limpiar cada una de las sentencias tokenizadas usando el criterio definido en el punto 1.2.
4) Definir las características de entrenamiento y creación (num_features, sg, context_size, etc) para el uso de FastText (sugerencia; ver numeral 1.1)
5) Finalmente, debe generarse el archivo vectorizado de todas las palabras del texto del dataset Europarl (Large Corpus Spanish) con FastText . Para probar la similaridad de las palabras con respecto a los modelos de Word2vec y FastText se deden probar mínimo tres palabras del vocabulario del dataset y comparar la similaridad del archivo generado con respecto al modelo de Word2vec que puede ser generado con las mismas sentencias de Europarl (Large Corpus Spanish).

2.1 Word2vec scratch para un texto pequeño. (15 puntos)

Experimentar sobre el word2vec-scratch el siguiente texto ***"Muchos años después, frente al pelotón de fusilamiento, el coronel Aureliano Buendía había de recordar aquella tarde remota en que su padre lo llevó a conocer el hielo. Macondo era entonces una aldea de veinte casas de barro y cañabrava construidas a la orilla de un río de aguas diáfanas que se precipitaban por un lecho de piedras pulidas, blancas y enormes como huevos prehistóricos".*** Entrenar el conjunto de palabras del texto, proponga como parámetros, vectores de dimensión 30 y 50 con ajuste de ventana de 3. Crear un procedimiento que visualice todas las palabras del texto en el plano cartesiano usando tsne para las dos dimensiones (30 y 50) con el fin de comparar la similaridad de las palabras en ambas visualizaciones. (Para visualizar las palabras de ambas dimensiones se puede adaptar el código de la siguiente figura que se encuentra en el notebook de **word2vec_español-ipynb).** En términos generales se deben

visualizar los vectores de **W1 +b1** de las palabras del texto. Esta es una visualización posible.



Estas cercanías de palabras se puede comprobar con la distancia euclideana (`def euclidean_dist(vec1, vec2):`) que se encuentra implementada por palabra en **word2vec_scratch.ipynb.** Se deben probar tres palabras.

3. Forma de entrega

Entregar los notebooks:

1) El código de la solución del problema 1, además del archivo vectorizado del dataset Europarl.

2) El código de la solución del problema 2. Para los dos embeddings 30 y 50 se deben visualizar el conjunto de palabras del texto.

4. Criterios evaluativos

4.1 Problema 1

Actividad	Criterio	puntaje
1 Traer el dataset Europarl de HF	Imprimir el número de filas del texto del datasets	5 puntos
2. Segmentación de sentencias y tokenización de las sentencias	Imprimir la lista de listas de las sentencias ya tokenizadas	10 puntos
3. Limpiar cada una de las sentencias tokenizadas	Imprimir algunas las listas de sentencias solo con palabras (limpias)	10 puntos
4. Definir las características de Fasttext	Revisar que los valores estén correctos para generar el embedding de 300.	3 puntos
5. Obtención del archivo vectorizado de Fasttext y pruebas	Verificar que se generó el archivo vectorizado de Fasttext y las tres consultas de palabras del dataset para los dos modelos de Fasttext (con el archivo creado) y Word2vec con el modelo sobre Europarl.	7 puntos

4.2 Problema 2

Actividades	Criterio	Puntaje
Entrenar el texto de Cien años de soledad para generar embeddings de dimensión 30 y 50	Imprimir la salida de los vectores de **W1+b1** para los dos embeddings (30 y 50)	4 puntos

Visualizar las palabras del texto con los embeddings de 30 y 50	Visualización en tsne de los vectores del texto con los embeddings de dimensión de 30 y 50	8 puntos
Probar la cercanía con la distancia euclidiana	Verificar el uso de la distancia euclidiana para mirar la cercanía de las tres palabras con respecto a los vectores **W1+b1**	3 puntos