

Tarea #2 de PLN-DL

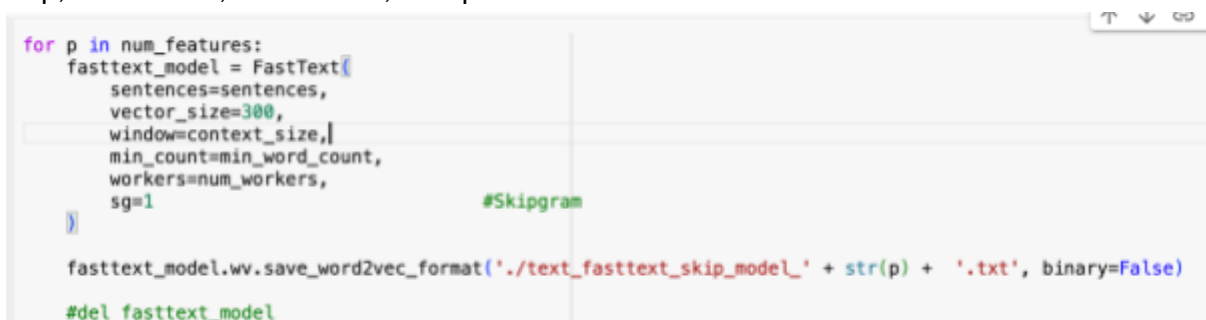
Representación vectorial con Fasttext, embeddings y NER

Procesamiento de lenguaje natural con Deep Learning

PUNTO 1. OBTENCIÓN DEL VECTOR DE EMBEDDINGS DE DATASETS

1.1 Fasttext. A principios de 2017, Facebook AI Research publicó un artículo que presentaba a [FastText](#) como 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.

Objetivo. Generar el archivo **text_fasttext_skip_modelo_300.txt** con las palabras vectorizadas por el modelo de Fasttext de los dos datasets [Europarl y JRC](#). La idea es obtener un solo archivo. En la siguiente figura se muestra el código para producir el archivo de palabras únicas vectorizadas con un embeddings de 300. No deben haber palabras stop, ni números, ni símbolos, ni espacios en blanco



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

Para ello se debe limpiar el texto raw total y solamente obtener palabras únicas y para el entrenamiento y generación del archivo se debe generar una lista de listas de sentencias de variable **sentences**. El modelo se debe entrenar con todo el conjunto de sentencias de los dos datasets.

PUNTO 2. RECONOCIMIENTO DE ENTIDADES NOMBRADAS USANDO RNNs

El reconocimiento de las entidades nombradas es una tarea transversal en las aplicaciones de PLN tales como:

- Clasificación de texto
- Extracción de relaciones
- Chatbots
- Traducción de texto
- Análisis de sentimientos, etc.

Hay tres enfoques para el reconocimiento de entidades nombradas (su sigla en inglés: Named Entity Recognition (NER)); un primer enfoque que usa CRFs con el entrenamiento de vectores de características de forma tales como las propias palabras, postag, ventanas de las palabras a la izquierda y derecha, primeros caracteres en mayúsculas, etc. Un segundo enfoque con redes neuronales profundas (deep learning) usando Bi-LSTMs y el tercer enfoque que utiliza los embeddings dinámicos y transformers tales como BERT, ELMO y FLAIR que contextualizan las sentencias de entradas usando preentrenados de más de un lenguaje y transfieren al modelo toda la caracterización de las asociaciones y contextualización de las palabras de la sentencia de entrada. En este punto se proponen el enfoque de deep learning para dos modelos usando Bi-LSTMs+ CRFs bajo el dataset [Biobert](#).

Objetivo. Entrenar dos RNNs; un BiLSTM con enterización y otro BiLSTM + embeddings con el archivo vectorizado del primer punto. La idea es entrenar los dos modelos sobre el dataset [Biobert](#) y llenar la siguiente tabla.

| Resultados de entrenamiento del BiLSTM+CRF | | | | Épocas | Batch_size |
|---|-----------|--------|----------|--------|------------|
| Modelo | Precision | Recall | F1-score | | |
| BiLSTM+CRF+conll2002-enterización | 46.7% | 44.6% | 45.6% | 30 | 512 |
| BiLSTM+CRF+conll2002+embeddings | 49.4% | 50.1% | 49.8% | 30 | 512 |
| BiLSTM+CRF+ Biobert -enterización | | | | | 32,64,128 |
| BiLSTM+CRF+ Biobert +embeddings | | | | | |

**embeddings=archivo vectorizado con fasttext de [Europarl y JRC](#)*

PUNTO 3. FINE TUNING PARA NER

Esta tarea tiene como **objetivo** la implementación de un finetuning de NER sobre los tres modelos de preentrenado de [bert-base-uncased](#), [BETO](#) y [XML-Roberta](#) sobre el dataset de [Biobert](#) y usar los pipelines de los modelos para construir una pequeña caja de texto para probar los modelos. Como ayuda, en [Ner](#) se encuentran los modelos de [distilbert](#) y [BETO](#) entrenados sobre [conll2002](#). Para el desarrollo del ajustes de los dos modelos se deben realizar las siguientes rutinas:

2.1 Primero se debe almacenar su token de autenticación del sitio web de Hugging Face (¡regístrese aquí si aún no lo ha hecho!), luego ejecute la siguiente celda e ingrese su nombre de usuario y contraseña:

2.2 Cargar el dataset [Biobert](#) desde HF, realizar las pruebas del dataset, conjunto train, test y val, además de verificar el nombre de la etiquetas de anotado.

2.3 Implementar la fase de preprocesamiento definiendo el tokenizador del modelo de [bert-base-cased](#), alinear los tokens y etiquetas con los tokenizadores de [bert-base-cased](#) y [BETO](#). En esta fase también se debe realizar el padding o relleno de manera dinámica usando la librería **DataCollatorWithPadding**.

2.4 Implementar el ajuste del modelo usando la tarea de clasificación del modelo preentrenado para la clasificación de entidades, para ello se usa la librería **AutoModelForTokenClassification**.

2.5 Definir la estrategia de entrenamiento usando **TrainingArguments**, en la que se definen el nombre del modelo con el que va quedar registrado en HF, el número de épocas, la estrategia de evaluación, la tolerancia o rata de aprendizaje.

2.6 Implementar el **Trainer** o fit del modelo en la que se deben definir el conjunto de entrenamiento, el de validación y poner en marcha el entrenamiento de los dos modelos.

2.7 En la fase de entrenamiento se debe llamar un procedimiento que obtenga las métricas de desempeño de precisión, recall y f1-score y accuracy de cada una de las épocas para cada uno de los dos modelos. Esto se debe ver reflejado en la API de Hugging Face en donde deberán subir los modelos.

La idea es llenar todos los datos de la tabla entrenando los modelos.

| Resultados de entrenamiento de Finetuning-transformers | | | | Épocas | Batch_size |
|---|-----------|--------|----------|--------|------------|
| Modelo | Precision | Recall | F1-score | | |
| Distilbert + conll2002 | 64.3% | 63.1% | 63.7% | 2 | 16 |
| BETO + conll2002 | 83.1% | 85.2% | 84.1% | 2 | 16 |
| bert-base-uncased + Biobert | | | | 10 | 16 |
| BETO + Biobert | | | | 10 | 16 |

| | | | | | |
|--|--|--|--|---|---|
| XML-Roberta + Biobert | | | | 5 | 16 (Tratar de usar 16 bits, sino probar con 1,2,4,8) |
|--|--|--|--|---|---|

Nota: Los ambientes de trabajo son dos [colab](#) para aplicaciones que no necesiten más de 15 Gz y [Kaggle](#) para aplicaciones mayores a 15 Gz