Práctica 3.4

Procesamiento del lenguaje natural

Topic modeling vs K-Means

Introducción

Una de las aplicaciones más útiles en PLN es la clasificación de temas a partir de un *corpus* dado. Existen diferentes opciones para realizar esta clasificación de temas.

En esta práctica realizaremos un breve ejemplo de una de uso común en el campo del PLN, como es el ***Topic Modeling***. Una vez aplicada esta técnica, contrastaremos los resultados que arroje posteriormente con un algoritmo de clusterización de propósito general como es **K-Means**, a fin de observar cuál de los dos ofrece mejores resultados en la tarea de clasificación temática.

Esta práctica se divide en dos partes:

1) implementación de **LDA** mediante Scikit-Learn
2) comparación de resultados entre **LDA** y **K-Means**

1) Implementación de Topic Modeling

¿Qué es el Topic Modeling?

El **topic modeling** es una técnica no supervisada de PLN, capaz de detectar y extraer de manera automática relaciones semánticas latentes de grandes volúmenes de información. Estas relaciones, que se denominan *tópicos* o temas, son un conjunto de palabras que suelen aparecer juntas en los mismos contextos y nos permiten observar relaciones que seríamos incapaces de observar a simple vista.

Existen diversas técnicas que pueden ser usadas para obtener estos temas. Una de ellas es el algoritmo conocido como ***Latent Dirichlet Allocation*** (**LDA**). Este algoritmo implementa un modelo que devuelve, por un lado, los diferentes temas que componen la colección de documentos y, por otro lado, evalúa la presencia cada tema en cada documento del *corpus*. Los temas consisten en una distribución de probabilidades de aparición de las distintas palabras del vocabulario.

¿Cómo se implementa?

En este ejercicio trabajaremos con un corpus muy reducido de datos que nos permita comprender cómo funciona **LDA** de cara a su aplicación en una práctica posterior a un volumen de datos mayor. LDA está implementado en diferentes librerías. En este ejemplo recurriremos a **Scikit-Learn** por estar familiarizados con ella.

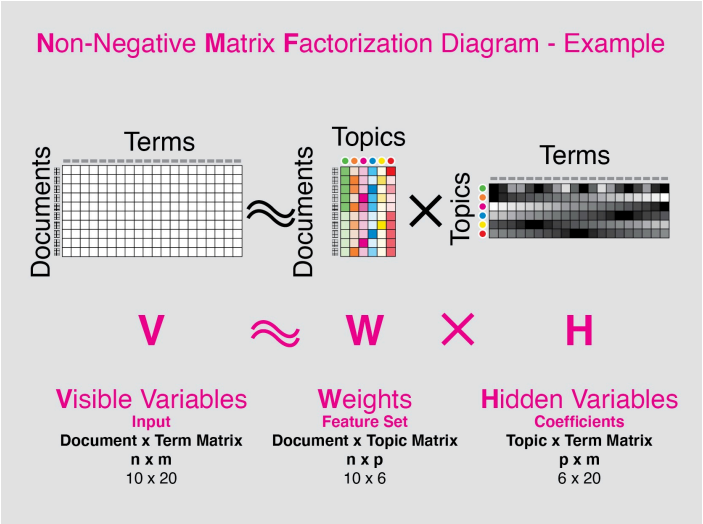
El conjunto de datos con el trabajaremos será un conjunto de **5 frases:**

```
# Corpus de documentos de ejemplo

corpus = [ "el cielo es azul", "las nubes son blancas", "el sol brilla",
"el cielo está despejado", "las estrellas brillan en la noche" ]
```

Pregunta: Antes de pasar a la explicación y codificación del modelo ... ¿serías capaz de diferenciar a simple vista alguno de los temas existentes en el corpus anterior?

Antes de usar **LDA** en **Scikit-Learn**, debemos entender cómo organiza los datos.



Scikit-Learn implementa **LDA** partiendo de lo que llama una **matriz de términos del documento** (**V** en la imagen) para descomponerla en dos matrices de menor dimensión, una **matriz de documento-temas** y una **matriz de temas-términos**:

- La **matriz de documento-temas** (**W** en la imagen) describe cómo se distribuyen los temas en los documentos. Cada fila representa un documento y cada columna representa un tema. Cada entrada de la matriz representa la frecuencia de aparición del tema en el documento correspondiente.
- La **matriz de tema-términos** (**H** en la imagen) contiene las palabras asociadas con cada tema. Es decir, tiene una fila por cada tema y una columna por cada término, y cada entrada representa la probabilidad de que el término pertenezca al tema.

Ahora incorporaremos el código necesario para usar LDA:

1) importa las siguientes librerías relacionadas con la vectorización de texto (*CountVectorizer* y *TfidfVectorizer*) y la modelo LDA (*LatentDirichletAllocation*):

```
from sklearn.feature_extraction.text import CountVectorizer,
from sklearn.decomposition import LatentDirichletAllocation
```

2) crea una variable que guarde el resultado de vectorizar (aplicando *fit* y *transform*) el *corpus* con las frases mediante `CountVectorizer`. Además del vectorizador, guarda en otra variable el diccionario de términos generado.

3) ahora generaremos el modelo. El número de temas que resultan de una agrupación más ajustada o coherente de términos no es algo que prescribe el modelo, sino algo que hay que decidir de antemano. O dicho de otro modo, debemos por prueba y error indicar al modelo qué número de temas queremos que el modelo genere, haciendo ajustes sucesivos.

   Comenzaremos pues probando, por ejemplo, con **2** temas:

```
# Probamos con 2 temas
no_topics = 2

# Vectorizador de términos
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

# Ajustamos el modelo LDA
lda_model = LatentDirichletAllocation(n_components=no_topics,
max_iter=10, random_state=0)
lda_model.fit(X)

# Obtenemos la distribución de temas para cada documento
doc_topic_distribution = lda_model.transform(X)

# Obtenemos la distribución de palabras para cada tema
topic_word_distribution = lda_model.components_
```

   En el código anterior, generamos el modelo LDA, indicando entre otros valores, el número de iteraciones que queremos que el algoritmo realice agrupando términos. Se entiende que a mayor número de iteraciones más precisa resultará la agrupación de términos por temas.

   Este ejemplo parte de un *corpus* demasiado reducido, con lo que 10 iteraciones son suficientes, pero si manejáramos un corpus con muchos documentos y miles de términos, el número de iteraciones debería ser superior.

4) finalmente, una vez preparado y ejecutado el modelo, necesitamos generar una salida por pantalla que nos permita mostrar los resultados que éste arroja.

   Por ejemplo, nos puede interesar mostrar los diferentes **temas** identificados por el modelo (que deberemos "etiquetar" en un segundo momento, es decir, darles un nombre porque el modelo no genera por sí mismo los nombre de los temas) junto a los **términos** y **documentos** asociados al mismo. Escribe con tus palabras en el mismo código cuál es el significado de cada término recibido.

   Lleva a tu cuaderno la siguiente función para generar dicha salida:

```
print("\nDistribución de palabras para cada tema:")
for i, topic_words in enumerate(topic_word_distribution):
    top_words_indices = topic_words.argsort()[-5:][::-1]
    top_words = [vectorizer.get_feature_names_out()[index] for index in
    top_words_indices] print(f"Tema {i+1}: {top_words}")
```

   Explica en el cuaderno qué contenido o función tienen las siguientes variables/métodos:

   1. topic_words
   2. top_words_indices
   3. top_words

   ¿Qué utilidad tiene en particular el método "argsort" en "topic_words"?

5) El código anterior debería arrojar un resultado similar al siguiente:

   Distribución de temas para cada documento:

   Documento 1: [0.00994676 0.99005324]

   Documento 2: [0.0052245 0.9947755]

   Documento 3: [0.00992818 0.99007182]

   Documento 4: [0.99001996 0.00998004]

   Documento 5: [0.00496406 0.99503594]

Distribución de palabras para cada tema:

Tópico 1: ['cielo', 'el', 'azul', 'sol', 'está']

Tópico 2: ['las', 'brillan', 'nubes', 'blancas', 'noche']

6) Finalmente, puede ser aconsejable usar algún tipo de métrica que nos ayude a evaluar la coherencia de los resultados obtenidos. Una opción para modelos tipo LDA es recurrir a métricas como la **puntuación de silueta** (*silhouette_score*).

Modifica el código del ejemplo previo con las siguientes líneas:

```
...
from sklearn.metrics import silhouette_score
...

#
# Añade aquí el código previo donde se creó el modelo
#

# Calcular la coherencia de los tópicos
coherence_scores = []
for i, topic_words in enumerate(topic_word_distribution):
    top_words_indices = topic_words.argsort()[-5:][::-1]
    top_words = [vectorizer.get_feature_names_out()[index] for index in
    top_words_indices]
    coherence_score = silhouette_score(X[:, top_words_indices],
    labels=doc_topic_distribution.argmax(axis=1))
    coherence_scores.append(coherence_score)

# Mostrar la coherencia de los tópicos
print(f"\nCoherencia de los tópicos para {num_topics} temas:")
for i, coherence_score in enumerate(coherence_scores):
    print(f"Tópico {i+1}: {coherence_score}")
```

7) El código anterior deberá generar una salida de este tipo:

Coherencia de los temas para 2 temas:

Tópico 1: 0.5355193967397891

Tópico 2: 0.6372118053629785

Observaciones:

- La coherencia se expresa como un valor numérico. En el ejemplo proporcionado, los valores de coherencia varían entre **0** y **1,** donde un valor más cercano a 1 indica una mayor coherencia.

- Al observar los valores de coherencia para diferentes números de temas, podemos analizar cómo la elección del número de tópicos afecta la calidad de los tópicos generados.

Ejercicios

1) Exploración de temas:

- **Modifica el código anterior encapsulando la creación y salida de resultados del modelo LDA** utilizando creando para ello una función llamada "explora_lda" que reciba como argumento el número de temas que LDA debe generar.

- Observa cómo cambian las distribuciones de temas y palabras con diferente número de temas (es decir, escogiendo 2, 3 y 4 temas, por ejemplo).

- **Pregunta**: ¿qué influencia tiene la elección del número de temas en la interpretación y utilidad del modelo?

2) Evaluación del modelo:

- Observa qué resultados arroja el modelo utilizando la métrica de puntuación de silueta para cada tema probando diferente número de tema.

- **Pregunta**: ¿qué conclusiones se puede extraer?

2) Implementación de K-Means

Como hemos visto, la técnica del *Topic Modeling* persigue hacer una agrupación de términos a partir del contenido de un *corpus* basada en cálculos de probabilidades. Otra opción para realizar esta clusterización se podría acometer mediante otro algoritmo de propósito general como **K-Means**.

¿Qué es K-Means?

K-Means es un algoritmo de agrupamiento sencillo de implementar, que agrupa datos en **k** grupos diferentes, donde "k" es un parámetro que se debe definir. La elección de k depende de la cantidad de grupos que se quiera obtener y de la naturaleza del conjunto de datos. Por tanto, se trata de un algoritmo de aprendizaje no supervisado, al no saber de antemano a qué clase (o grupo) pertenecen los datos a agrupar.

¿Cómo funciona?

Pasos:

1) El algoritmo comienza seleccionando 'k' puntos iniciales al azar a los que denomina "centroides" (puntos centrales) de los clusters.
2) Luego, asigna cada punto de datos al cluster más cercano en función de la distancia euclidiana entre el punto y los centroides.
3) Después de que cada punto de datos se asigna a un cluster, el algoritmo actualiza los centroides recalculando su posición como la media de todos los puntos en ese cluster. Este proceso se repite hasta que no se produzcan cambios en la asignación de clusters.

Si quieres una explicación más detallada y visual de esto último, te recomiendo revisar este breve vídeo.

¿Cómo se implementa?

En este caso, a diferencia de la implementación previa de LDA, usaremos **TfidfVectorizer** en lugar de **CountVectorizer** con **K-Means**, ya que, al reducir la influencia de las palabras comunes y resaltar las palabras más distintivas de cada documento, puede ayudar a mejorar la calidad de los grupos generados y hacer que sean más representativos y significativos.

Pasemos a implementarlo …

1) importamos las librerías necesarias:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
```

2) crea un objeto `TfidfVectorizer` y úsalo para vectorizar (aplicando *fit* y *transform*) el *corpus* con las frases. Además del vectorizador, guarda en una variable `terms` el diccionario de términos generado.

3) generamos el modelo buscando **3 temas** (prueba también con otros valores):

```
# Probamos con 3 agrupaciones
true_k = 3

model = KMeans(n_clusters=true_k, init="k-means++", max_iter=50,
n_init=1)

model.fit(tfidf)

order_centroids = model.cluster_centers_.argsort()[:, ::-1]
```

```
for i in range(true_k):
 print(f"Cluster {i}: ")
 for ind in order_centroids[i,:5]:
   print(f'{terms[ind]}', end=" ")
 print("\n")
```

4) muestra en formato Pandas la asociación entre cada frase del *corpus* y el número de clúster asignado cuando `true_k = 3`.

	documents	cluster
0	el cielo es azul	1
1	las nubes son blancas	0
2	el sol brilla	2
3	el cielo está despejado	1
4	las estrellas brillan en la noche	0

Pregunta: ¿se comporta K-Means mejor, igual o peor que LDA?