Universidad
Carlos III de Madrid

Proyecto Final
Análisis de Proyectos de Investigación

Tratamiento de Datos

Elena Almagro Azor - Mario Golbano Corzo - Juan Muñoz Villalón

Universidad Carlos III de Madrid. Campues Puerta de Toledo.

October 31, 2024

Contents

1 Paso 1. Implementación de un pipeline para el preprocesado de los textos

El primer paso llevado a cabo en el proyecto es el preprocesado de textos, lo que es crucial para el posterior proceso de clasificación o regresión que se realizará. En el código entregado se observa cómo se adquieren los datos, seleccionando sólo la información que nos interesa. También se importan los códigos raíz, donde sólo se toma el primer valor para cada proyecto.

Una vez importados los datos, se procedió con el procesamiento de los textos, para lo que se descargaron las librerías necesarias. Gracias a ellas se definieron las funciones `wrangle_text` y `prepare_data`.

Con la primera función se consiguen dos tareas fundamentales. En primer lugar, se eliminan HTML tags y URLs. Esta acción es crucial para desprenderse del texto que no aporta información relevante y hace más difícil el proceso de comprensión y análisis del texto, por lo que no nos interesa mantenerlos para el proceso de vectorización. Por otra parte, también realiza la expansión de contracciones, ayudando con la estandarización del texto. En definitiva, `wrangle_text` es una función cuyo objetivo principal es limpiar el texto y librarse de las contracciones léxicas.

`Prepare_data` implementa la función `wrangle_text` y además se encarga de tokenizar, lematizar y eliminar palabras irrelevantes. El proceso de tokenización es primordial para transformar el texto en secuencias de palabras que puedan ser procesadas individualmente. Asimismo, reducir las palabras a su forma más básica facilita la comprensión. Esto se lleva a cabo con la lematización. Por último, se encarga de eliminar palabras irrelevantes que no aportan significado específico y son generalmente comunes, contribuyendo a reducir el tamaño del conjunto.

Con todo esto, se consigue un texto limpio y estandarizado que ayudará a obtener una clasificación o regresión más precisa y eficaz.

2 Paso 2. Representación vectorial de los documentos mediante tres procedimientos

Una vez se ha hecho el preprocesado de los datos podemos pasar a la vecotirzación de los mismos. En la presente práctica estudiaremos 3 tipos de representaciones vectoriales.

2.1 TFIDF

En primer lugar, trabajaremos con TFIDF. Como hemos visto en las clases de teoría, para trabajar con TFIDF tenemos que convertir nuestro dataframe de datos a un *corpus de Gensim*, que es una colección de todos los documentos (en este caso cada uno de los textos tokenizados y limpios). Además, se ha aplicado el método de N-gram para eliminar los tokens que no sean muy relevantes y juntar aquellos que aparezcan juntos con frecuencia.

Así, en la figura 1 se puede apreciar el primer texto del corpus tokenizado tras aplicar el *N-gram*, donde se comprueba, por ejemplo, que *ebola-virus, sierra-leone, densely-populated* han pasado a ser un solo token, ya que, efectivamente, se pueden entender como una sola idea (un solo token).

```
============= First text after N-gram replacement =============
['ebola_virus', 'modern', 'approach', 'developing', 'bedside', 'rapid', 'diagnostics', 'sofia_ref', '115843', 'current', 'ebola
_virus_disease', 'evd', 'outbreak', 'ha', 'caused', '5000', 'death', 'within', 'month', 'west_africa', 'guinea', 'sierra_leon
e', 'liberia', 'severely_affected', 'country', 'including', 'numerous', 'healthcare', 'worker', 'serious', 'public_health', 'cr
isis', 'international', 'concern', 'number', 'case', 'still', 'increasing', '11', 'month', 'first', 'case', 'wa', 'described',
'december', '2013', 'mid', 'november', '2014', 'approximately', '15000', 'individual', 'infected', 'epidemic', 'still', 'contro
l', 'direct', 'effect', 'outbreak', 'include', 'disruption', 'standard', 'medical', 'care', 'insecurity', 'social', 'disruptio
n', 'country', 'already', 'struggling', 'recover', 'decade', 'war', 'one', 'important', 'key', 'action', 'limit', 'stop', 'spre
ad', 'deadly', 'disease', 'identify', 'isolate', 'ebov', 'infected', 'patient', 'diagnosis', 'ebola_virus', 'infection', 'ha',
'past', 'performed', 'overwhelmingly', 'specialist', 'reference', 'laboratory', 'high', 'performance', 'molecular', 'serologica
l', 'culture', 'method', 'recent_year', 'many', 'function', 'mobilised', 'rapid', 'response', 'mobile', 'laboratory', 'indeed',
'several', 'unit', 'supported', 'european', 'cdc', 'usa', 'canadian', 'chinese', 'african', 'state', 'set', 'help', 'current',
'outbreak', 'performing', 'laboratory', 'diagnosis', 'collaboration', 'national', 'centre', 'west_africa', 'type', 'response',
'ha', 'effective', 'helping', 'control', 'past', 'evd', 'outbreak', 'rural', 'part', 'africa', 'ha', 'effective', 'controllin
g', 'current', 'outbreak', 'spreading', 'densely_populated', 'city', 'slum', 'area', 'unfortunately', 'diagnostic', 'procedur
e', 'currently', 'used', 'mobile', 'laboratory', 'associated', 'several', 'problem', 'limited', 'number', 'diagnostic', 'hub',
'ii', 'performing', 'diagnostic', 'test', 'correctly', 'requires', 'specialist', 'training', 'skill', 'experience', 'consequent
ly', 'limited', 'number', 'trained_staff']
```

Figure 1: Primer texto tokenizado tras aplicar N-gram

Y ahora, ya podemos pasar a la vectorización creando un diccionario de *Gensim* que contenga todos los tokens de nuestro corpus y aignarle un identificador (entero). Así, en el diccionario reusltan 120392 elementos (tokens). Pero antes de continuar, se quitan los elementos que aparezcan en muy pocos documentos y por tanto no den demasiada información acerca de ese texto, así como los que aparezcan en demasiados textos, y por consiguiente, no den suficiente información sobre su texto concreto. Haciendo esto se ha pasado a tener 35755 elementos en el diccionario.

A continuación, crearemos una versión numérica del corpus mediante el método **Bag-of-Words**, pasando de una lista de tokens a una lista de tuplas donde se indique el identificador de token y el número de ocurrencias. Y finalmente, podemos aplicar el método de vectorización TFIDF. Este método evalúa la importancia relativa de una palabra (un token en este caso) en relación con el conjunto de los textos, por lo que no depende únicamente del número de apariciones de una palabra en un texto, como en *BoW*, sino que depende de las apariciones en todo el corpus. Esto se hace

simplemente siguiendo una fórmula y como se ha visto en teoría, una vez tenemos el diccionario de *gensim* y los textos tokenizados es muy sencillo de implementar con los métodos de la propia librería de *gensim*. De esta forma, la representación del primer texto utilizando *TFIDF* es la que se muestra en la fig 2.

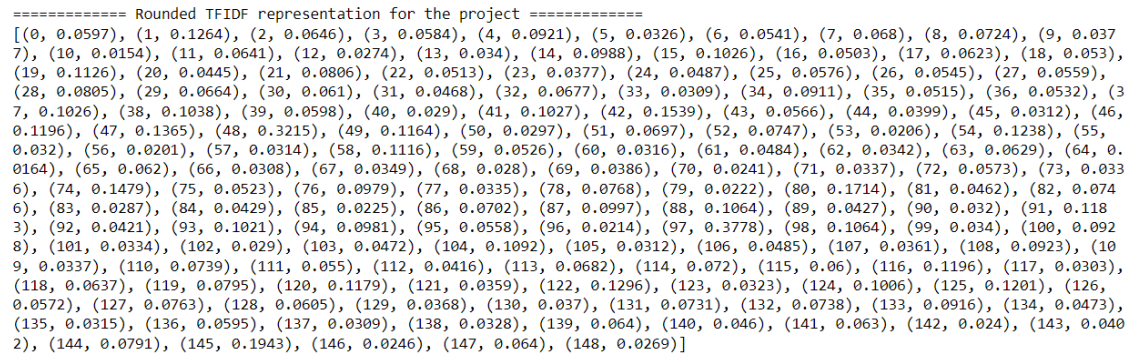


Figure 2: Representación del primero texto con TFIDF

2.2 Word2Vec

De manera similar a como hemos hecho en *TFIDF*, entrenaremos el modelo *Word2Vec* siguiendo lo visto en clase. Así, comenzaremos entrenando el modelo con arquitectura de skip-gram (predecir el contexto a partir de una palabra dada). Otros parámetros que usaremos son: *vectorsize = 200, mincount = 10*. Para el *vectorsize* se ha elegido 200 ya que es un valor relativamente alto, permitiendo una buena precisión para el gran corpus con el que estamos trabajando, sin llegar a ser muy demandante computacionalmente. Y en cuanto, al *mincount*, se ha elegido 10, ya que consideramos que si alguna palabra aparece menos de 10 veces en todos los documentos con los que se está trabajando, esta no debería aportar mucha información.

El modelo, entonces, lo que hace es, a partir del contexto, asigna un vector denso a cada palabra (token) de 200 elementos (en nuestro caso), de manera que cuanto más similar sean los embeddings (vectores) de 2 palabras, más parecidos serán. En la fig 3 podemos ver el embedding de la primera palabra. Así, al crear el modelo, hemos generado un diccionario con todo el vocabulario en embeddings.

Podemos comprobar que el número de elementos en el diccionario de *Word2Vec* es de 29620 palabras. Algo menos que el diccionario de *TFIDF*, pero relativamente similar. Además, esta diferencia es entendible debido a las diferentes técnicas de reducción de palabras empleadas.

```
============= Embedding of the first word =============
[-3.12416013e-02  -1.81236230e-02   6.12020530e-02   3.52362692e-02   7.14826956e-02   -7.72186518e-02   1.20362408e-01
6.87143877e-02    4.17842232e-02   -1.76577270e-01   -1.58324942e-01   1.22726187e-01   7.76977241e-02   3.75248909e-01
1.08797714e-01    6.74977675e-02   -2.86129359e-02   -1.30010709e-01   -6.87555224e-02   -1.78275824e-01   1.11746401e-01
2.56959468e-01    -3.49184185e-01   1.10961333e-01    -9.71902348e-03   8.55682939e-02    5.04117273e-03    3.73056501e-01
-5.47970682e-02   5.61433099e-02    -1.86915442e-01   -4.83765565e-02   -5.99576421e-02   -5.30197285e-02   -1.51275992e-01
-1.16336182e-01   1.61461905e-01    -1.87443823e-01   -2.04946324e-02   1.00903176e-01    -1.01187035e-01   2.30363652e-01
1.28145948e-01    -3.03511560e-01   -5.36974557e-02   -5.22858510e-03   6.04345137e-03    1.62239254e-01    1.14838012e-01
9.57783833e-02    3.10560539e-02    1.30435852e-02    -3.21621969e-02   1.26292139e-01    4.63788770e-02    4.16114405e-02
-1.04725525e-01   -3.07655364e-01   -2.70861872e-02   6.92118406e-02    1.36535197e-01    1.28112420e-01    -1.76210538e-01
1.09396420e-01    1.86012164e-01    2.31208399e-01    4.25165556e-02    1.69503823e-01    -7.34838396e-02   4.95147395e-01
-7.07807094e-02   6.16348498e-02    1.15920275e-01    -2.00370207e-01   2.08662629e-01    -3.74861434e-02   1.06573068e-02
1.40316203e-01    -5.15743867e-02   1.35348544e-01    1.37132704e-01    -1.14411734e-01   -1.54189005e-01   5.00385128e-02
-2.13191062e-01   5.55826947e-02    -1.67944729e-01   -3.64959985e-02   -1.03185410e-02   1.49337590e-01    1.89148739e-01
2.29615748e-01    2.48150930e-01    6.23486154e-02    2.80150265e-01    -5.88666089e-02   -5.73473759e-02   2.07837537e-01
-4.37807254e-02   -2.12307587e-01   -2.00152639e-02   -1.82528526e-01   -3.35788466e-02   -8.89260769e-02   -7.70941302e-02
1.77994445e-01    3.65869068e-02    -1.68208815e-02   1.75150424e-01    2.10492954e-01    -8.23503435e-02   1.56663135e-01
-1.69952646e-01   9.16616619e-02    4.65022437e-02    1.55199543e-01    -8.34933072e-02   -2.32943147e-01   -1.40412614e-01
-1.06905540e-02   -5.06857336e-02   1.04285451e-02    -2.11570948e-01   1.52047500e-01    2.18808293e-01    -3.08767915e-01
2.23709061e-03    -1.25612020e-01   -1.33778572e-01   1.14972964e-01    -9.68592614e-03   1.30106285e-01    2.17617936e-02
9.29547325e-02    1.05827861e-01    -1.94516718e-01   9.21968520e-02    -1.93188503e-01   2.75420112e-04    -8.96836594e-02
1.84009776e-01    -5.45821041e-02   -1.36394709e-01   -7.84362331e-02   9.45905596e-02    -3.19217588e-03   -1.03283621e-01
-5.67154773e-02   1.42483816e-01    -6.82601854e-02   -7.56135508e-02   -1.80825308e-01   1.86376125e-01    -1.39455562e-02
1.86144039e-01    7.71733299e-02    -6.17466867e-02   -2.09012121e-01   7.79133663e-02    2.39019200e-01    -1.17367692e-01
5.31129912e-02    -1.47187665e-01   1.31509885e-01    6.10008352e-02    1.70776054e-01    8.91038030e-02    -1.72378168e-01
-2.41432890e-01   -8.26283097e-02   6.57731518e-02    1.12143621e-01    1.00324705e-01    2.22632915e-01    -1.32135093e-01
6.78303614e-02    2.02543750e-01    -2.97677405e-02   1.19700417e-01    -1.20899327e-01   1.82092249e-01    -1.42548501e-01
-2.19887365e-02   -1.59011304e-01   4.78850640e-02    -1.97031721e-01   -1.18392564e-01   -2.01073572e-01   9.21344608e-02
3.59262414e-02    -2.36423939e-01   1.64380774e-01    8.51041228e-02    -1.15004778e-01   -9.69764888e-02   3.44719477e-02
-3.71778496e-02   1.45377859e-01   -2.01952562e-01   4.13826033e-02]
```

Figure 3: Embedding de la primera palabra (modelo Word2Vec)

Para terminar, podemos hacer uso de *t-SNE* para pasar las vectorizaciones conseguidas de una alta dimensión, a 2 dimensiones para poder representarlo. Al representarlo obtenemos la fig 4. Se muestran solo 500 embeddings para poder apreciarlos correctamente. En la figura se pueden apreciar grupos de palabras cercanos (vectores, embeddings similares). Por ejemplo: "disease, cancer, treatment, terapeuthic, drug, patient" o "academic, science, expert, university, knowledge", que obviamente, son palabras con significados, al menos contextual o semánticamente, similares.

Otro punto interesante que podemos estudiar son las *Out-of-the-vocabulary words*. Estas son palabras que aparecen en el diccionario inicial de *gensim* que creamos, pero no en el vocabulario del modelo entrenado en *Word2Vec*, debido al filtrado de palabras que hayamos hecho. El porcentaje de palabras que sí que aparece en el vocabulario es del 58.42%, por lo que se han filtrado un poco menos de la mitad de las palabras iniciales.

Para finalizar, ahora mismo tenemos los embeddings de todas las palabras, pero nosotros queremos los embeddings de cada uno de los textos. Esto se hace a partir de la lista de tokens de cada texto y sus respectivos embeddings, calculando una media de estos. Así, la vectorización resulta en una matriz con tantas filas como textos tenemos y un embedding de 200 elementos (en nuestro caso) para cada texto (columnas).

Figure 4: Representación de 500 embeddings del diccionario de Word2Vec

2.3 BERT (Embeddings contextuales basados en transformers)

Como último método de vectorización de los textos del *dataset* de documentos, se va a utilizar BERT (Bidirectional Encoder Representations from Transformers). Para poder trabajar correctamente con BERT y con nuestro *DataFrame*, tendremos que

convertir este a un objeto *DataSetDict*, donde lo dividiremos en 3 *subdatasets* (train, test y validación) que nos servirán en el futuro para el entrenamiento y validación del modelo BERT utilizado. Además, tuvimos que buscar un modelo de BERT y escogimos el modelo `bert-base-uncased`.

Tras definir las librerías necesarias, se cargó un tokenizador preentrenado BERT, cuya principal función es transformar textos en secuencias de tokens para que el modelo BERT funcione correctamente. Una vez cargado el tokenizador a utilizar y definida la función que realiza la acción de tokenizar, aplicamos dicha función a nuestro conjunto de datos, de donde se obtiene el nuevo conjunto de datos con el que trabajaremos `tokenized_preproc_dic`, donde se modificarion posteriormente algunas columnas. Es importante mencionar que el modelo BERT trabaja directamente con el texto sin procesar, utilizando el texto completo, por eso es necesario este paso de tokenización.

A continuación, se combinan los datos en lotes de igual longitud con ayuda de un padding. Con esto, se han preparado los datos para la implementación de BERT.

Llegados a este punto se procede con el entrenamiento del modelo, explicado en la siguiente sección.

3 Entrenamiento y evaluación de modelos de clasificación

3.1 BERT (Hugging Face)

3.1.1 Hugging Face Trainer

El primer paso antes de definir nuestro `Trainer` es definir una clase `TrainerArguments`, donde podemos especificar diferentes parámetros de entrenamiento como la frecuencia con la que evaluar y guardar los *checkpoints* del modelo, dónde guardarlos, etc. Hay muchos aspectos que se pueden personalizar, siendo los más importantes:

1. `num_train_epochs`: Número total de *epochs* de entrenamiento (un número alto puede causar *overfitting*). En nuestro caso se han utilizado 5.

2. `per_device_train_batch_size`: Tamaño del lote por dispositivo durante el entrenamiento. Hemos escogido 16.

3. `per_device_eval_batch_size`: Tamaño del lote por dispositivo durante la evaluación. Hemos escogido también 16.

4. `output_dir`: El directorio donde se va a guardar el modelo de salida y los archivos de configuración. Esto es bastante importante porque así podremos después cargar el modelo directamente sin necesidad de volverlo a entrenar.

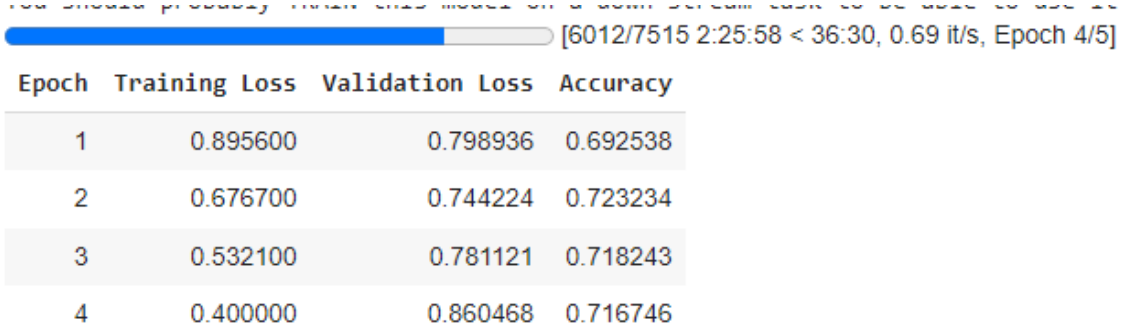
5. `learning_rate`: La tasa de aprendizaje inicial del optimizador AdamW. Hemos escogido un valor de $2e-5$.

6. `evaluation_strategy`: Estrategia de evaluación durante el entrenamiento: en este caso, la evaluación se realiza al final de cada época.

El segundo paso es definir nuestro modelo. Utilizaremos la clase *AutoModelForSequenceClassification* especificando 6 clases diferentes a la salida.

Por último, ya solo nos queda crear un `Trainer` donde le pasamos todos los objetos construidos hasta ahora (los argumentos, el modelo, el *dataset* de Train y el *dataset* de evaluación, el tokenizador y el *dataCollator*) y entrenar el modelo usando dicho `Trainer`.

En la figura 5 podemos ver las *epochs* realizadas durante el entrenamiento y debajo las métricas obtenidas en general para el entrenamiento.


[6012/7515 2:25:58 < 36:30, 0.69 it/s, Epoch 4/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.895600	0.798936	0.692538
2	0.676700	0.744224	0.723234
3	0.532100	0.781121	0.718243
4	0.400000	0.860468	0.716746

Figure 5: Métricas obtenidas al entrenar el modelo de BERT

```
TrainOutput(global_step=6012, training_loss=0.626111678417254,
metrics={'train_runtime': 8762.5283, 'train_samples_per_second': 13.717,
'train_steps_per_second': 0.858, 'total_flos': 2.2128811150375188e+16,
'train_loss': 0.626111678417254, 'epoch': 4.0})
```

A continuación podemos observar las métricas obtenidas para la evaluación utilizando el set de *eval* proporcionado al *trainer*.

```
{'eval_loss': 0.744223952293396, 'eval_accuracy': 0.723234339905166,
'eval_runtime': 123.1002, 'eval_samples_per_second': 32.551,
'eval_steps_per_second': 2.039, 'epoch': 4.0}
```

3.1.2 Obteniendo predicciones con Hugging Face

En primer lugar, se carga el modelo que ya ha sido entrenado y se define el entrenador de prueba para realizar predicciones en nuestro conjunto de datos usando el set de *test*, con lo que obtendremos las predicciones del modelo en forma de *logits*. Las predicciones finales serán aquellas con el índice más alto en cada grupo de *logits*.

Conociendo las predicciones finales se calculará el valor de precisión en la predicción, resultando ser de **0.7059**.

3.2 Scikit-learn: Random Forest

Para los modelos de vectorización de ***TFIDF, word2vec*** se empleará un método de clasificación de Random Forest. En ambos casos se hará uso de un *GridSearch* (de la librería *ScikitLearn* para encontrar los parámetros que hagan mejor la clasificación. De esta forma, el *GridSearch*, realiza validación cruzada, permitiendo evaluar el rendimiento de cada uno de los métodos mediante la precisión. Así, aplicando un mismo modelo de clasificación podremos comparar los rendimientos obtenidos para cada uno de los modelos de vectorización y determinar cuál de ellos es mejor (al menos para nuestro caso).

3.2.1 Clasificación para vectorización TFIDF

Para poder llevar a cabo la clasificación necesitamos poder convertir el diccionario que teníamos a unos vectores de Numpy. Esto se puede hacer mediante la función *corpus2dense* o *corpus2csc*. En nuestro caso, usaremos la primera ya que no tenemos una matriz de datos extremadamente grande.

Finalmente, como ya se ha indicado, se ha hecho un *gridsearch* con los siguientes parámetros:

```
'n_estimators': [50, 100, 200],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10]
```

Y se ha utilizado una validación cruzada con 5 divisiones (cv = 5).

Se ha determinado que los mejores parámetros son:

```
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
```

Devolviendo una precisión de **0.6151**.

3.2.2 Clasificación para vectorización Word2Vec

Al contrario que en el caso anterior, ahora podemos entrenar el modelo de clasificación directamente a partir de la matriz resultante de la vectorización. Siguiendo con el modelo llevado a cabo en el apartado anterior, realizaremos de nuevo un *gridsearch* con los mismos parámetros para un Random Forest y el mismo número de divisiones en la validación cruzada.

De nuevo, resulta que los mejores parámetros para el clasificador son

```
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
```

por lo que la comparación entre ambos modelos será más adeacuada. En el caso de la vectorización *Word2Vec*, hemos obtenido una precisión de **0.6408**

3.2.3 Comparación Modelos Word2Vec - TFIDF - BERT

En la tabla 1 podemos ver los resutlados obtenidos tanto en la vectorización como en el entrenamiento y evaluación del modelo de clasificación para *TFIDF*, *Word2Vec* y *BERT*.

	TFIDF	**Word2Vec**	**BERT**
Nº palabras en diccionario	35755	29620	-
Precisión	0.6151	0.6408	0.7059

Table 1: Tabla Comparativa TFIDF - Word2Vec

Centrándonos en los modelos *Word2Vec* y *TFIDF*, podemos observar que para *Word2Vec* se ha conseguido una mejor precisión con menos palabras en el vocabulario. Podríamos pensar que este resultado es consistente, dado que *Word2Vec* se trata de un modelo de vectorización contextual, tomando relaciones semánticas y similitudes entre palabras según el contexto de cada una, mientras que *TFIDF* simplemente se basa en la frecuencia de la aparición de las palabras en los tetxos y en todo el corpus.

Por otra parte, con el modelo BERT se observa una mayor precisión. Esto se puede deber a la capacidad del modelo para entender las relaciones semánticas y el significado de palabras dependiendo del contexto, por lo que para una comprensión más profunda, este modelo puede ser una mejor opción. No obstante, computacionalmente consume más recursos que los modelos anteriores.

4 Extensión

4.1 Estimación del texto más similar a una palabra

Como se ha estudiado en clase, *Word2Vec* implementa un método *most similar* para, dada una palabra, decir las palabras que más se acerquen (su embedding, vector) a esta, es decir, las más similares. A partir de esta idea, hemos pensado en desarrollar diferentes herramientas (funciones), para los modelos de vectorización que nos estime el texto que más se parezca, o mejor dicho, que más relación tenga con una palabra dada. Como ocurre, con la función de *most similar*, no se le podrá pasar una palabra que no esté en los diccionarios.

4.1.1 TFIDF

Para *TFIDF* se ha implementado una función que transforma la palabra que le pasemos a representación *TFIDF* al igual que hicimos al principio a partir del modelo entrenado y el diccionario creado. Esta función daría error si la palabra por la que se pregunta no está en el diccioanrio. Después, se crea una matriz de similitud utilizando *MatrixSimilarity* de *gensim* y se calcula la similitud coseno entre la palabra y todos los documentos del corpus, devolviendo un vector con la similitud coseno con cada uno de los textos. Así, podemos ver qué textos son los más cercanos, o los que más se pueden asociar a la palabra dada. Veamos algunos ejemplos. Veremos los 5 textos con mayor similitud coseno a las palabras:
Covid:

```
Texto #1 más similar: ID 101016065: COVID eXponential Programme
Tema asociado: 23 : Natural Sciences

Texto #2 más similar: ID 101016131: AI-based chest CT analysis enabling
rapid COVID diagnosis and prognosis
Tema asociado: 23 : Natural Sciences

Texto #3 más similar: ID 101034321: Entrepreneurial Multidisciplinary
scientists forging Pathways Onto clean Water, sustainable Energy
and Resources
Tema asociado: 25 : Engineering and Technology

Texto #4 más similar: ID 101036871: Holistic & Green Airports
Tema asociado: 29 : Social Sciences

Texto #5 más similar: ID 101029512: Participatory Pathways for Reshoring
European Manufacturing
Tema asociado: 29 : Social Sciences
```

Mediterranean:

```
Texto #1 más similar: ID 101002330: Discovering the Deep Mediterranean
Environment: A History of Science and Strategy (1860-2020)
Tema asociado: 31 : Humanities

Texto #2 más similar: ID 699818: Analysis of the Artistic Exchanges in
the Medieval Mediterranean between 12th and 15th Centuries through the
Geographical Information Systems (GIS): A Critical Review of Centre
and Peripheries
Tema asociado: 31 : Humanities

Texto #3 más similar: ID 693055: MEDRESET.A comprehensive, integrated,
and bottom-up approach to reset our understanding of the Mediterranean
space, remap the region,and reconstruct inclusive, responsive, and
flexible EU policies in it
Tema asociado: 29 : Social Sciences

Texto #4 más similar: ID 795465: Reassessing Late Ottoman Literatures
within a Mediterranean Framework
Tema asociado: 31 : Humanities

Texto #5 más similar: ID 101003394: Integrated analysis of coralline
algae  facies of the central Mediterranean since the Oligocene
Tema asociado: 23 : Natural Sciences
```

4.1.2 Word2Vec

Vamos a realizar un procedimiento similar al de antes pero para *Word2Vec*. Al igual que antes, se le pasa la palabra, el diccionario creado anteriormente y el dataframe con todos los textos y tampoco se podrá evaluar si la palabar objeto de estudio no está en el diccionario. Y el resto es el procedimiento equivalente al anterior también; se obtiene la representación vectorial de la palabra, se calcula la similitud coseno de la palabra con todos los textos y estos valores se almacenan en un vector, del que tomaremos los de mayor valor. Para comparar con el caso anterior, evaluaremos para las mismas palabras:
Covid:

```
Texto #1 más similar: ID 25886: COVID eXponential Programme
Tema asociado: 23 : Natural Sciences

Texto #2 más similar: ID 15172: PROPOSAL FOR FUNDING RESEARCH DEVELOPMENT
AND MANUFACTURING OF VACCINE AGAINST COVID-19
Tema asociado: 21 : Medical and Health Sciences

Texto #3 más similar: ID 2385: COVID-19 Telemedicine { an infectious disease
management tool for governments all over Europe that will reduce casualties
from COVID-19 as well as prepare governments for the next wave of epidemics.
```

```
Tema asociado: 21 : Medical and Health Sciences

Texto #4 más similar: ID 20190: Developing a diverse portfolio of vaccine
candidates for Rift Valley Fever, Chikungunya and Ebola
Tema asociado: 21 : Medical and Health Sciences

Texto #5 más similar: ID 4775: Corona Accelerated R&D in Europe
Tema asociado: 21 : Medical and Health Sciences
```

Mediterranean:

```
Texto #1 más similar: ID 7950: The structures of the Early Modern
Mediterranean shipbuilding
Tema asociado: 31 : Humanities

Texto #2 más similar: ID 21870: The construction of early modern
global Cities and oceanic networks in the Atlantic: An approach via OceaN's
Cultural HeritAge
Tema asociado: 31 : Humanities

Texto #3 más similar: ID 17728: Crop Production in the Levant and
International Trade Exchange: investigating coprolites and crop plant remains
from the 1st millennium CE Negev Highlands and Aravah Valley
Tema asociado: 31 : Humanities

Texto #4 más similar: ID 8812: Discovering the Deep Mediterranean Environment: A History of Science
Tema asociado: 31 : Humanities

Texto #5 más similar: ID 27177: Transported cultural landscapes:
the role of colonization processes in cultural landscape shaping
Tema asociado: 27 : Agricultural Science
```

Podemos comprobar que en el caso de la palabra **mediterranean** se obtienen 5 textos diferentes para los dos métodos, aunque todos tienen una relación muy estrecha con la palabra que le hemos pasado. No obstate, en cuanto a la palabra **covid** vemos que el primer texto es el mismo, aunque el resto no. No obstante, comparando los demás textos podemos ver que los que se obtienen con TFIDF no están estrictamente relacionados con el covid, sino que hablan de comparaciones de tiempos pre-covid o post-covid, mientras que en *Word2Vec* los textos que se han obtenido sí que son textos relacionados directamente con el covid.

En conclusión, usando estas simples herramientas, podemos decir que para buscar un texto relacionado con una temática concreta, es más eficiente el uso de *Word2Vec*, debido a ser un método contextual.