Procesamiento del Lenguaje Natural: Proyecto final

Ángel Corrales Cuevas

José Lorente López

Alejandro Torres Muñoz

Máster en Inteligencia Artificial Aplicada

Procesamiento del Lenguaje Natural

1. Preprocesado de textos

En primer lugar, para realizar el primer paso de obtención de los **tokens**, se ha debido realizar la carga de los distintos documentos que conforman el **corpus** proporcionado. Así, al observar los datos proporcionados en el archivo *projects.xlsx,* se debe destacar que para este primer paso se ha realizado lo siguiente:

1) Unión de atributo **title** con el atributo **summary**. De esta manera, se obtiene un único texto que representa a cada documento, y sobre el que se va a realizar el preprocesado. La idea de añadir el título a la representación es porque puede contener más información útil de manera condensada, que represente claramente la categoría a la que está asociada el documento. Dicha unión resulta en el atributo **text.**
2) Preprocesado del atributo **text,** a través de los siguientes pasos de manera secuencial:
    - **a) Wrangling:** Consiste en realizar una primera limpieza de posibles tags derivados del HTML, así como expandir las contracciones del habla inglesa, ya que se observa que los textos de los proyectos están escritos en inglés.
    - **b) Tokenization:** Consiste en segmentar el texto de cada proyecto en una lista de tokens. Es decir, se separa el texto en una lista con cada una de las palabras que lo componen.
    - **c) Homogenization:** Consiste en reducir la lista obtenida en el paso anterior, de manera que los tokens que tengan un mismo significado semántico sean agrupados en un sólo token. En concreto, para el caso de este proyecto, y ya que se ha demostrado que su funcionamiento suele ser mejor, se optará por reducir los tokens a través de sus lemas.
    - **d) Cleaning:** El último paso del preprocesado se corresponde a la realización de una limpieza final de los lemas obtenidos. Para ello, se eliminarán aquellos lemas que:
        - i) Sean extraños, es decir, apenas aparezcan en el conjunto global de los textos.
        - ii) Aquellos lemas que sean muy comunes en todos los componentes del corpus y que por lo tanto no aporten ninguna información diferencial en los documentos.

El resultado de todos los pasos anteriores ha sido guardado en el archivo *projects_preprocessed.xlsx,* sobre el atributo **clean_text.**

2. Representación vectorial

Con respecto a la representación vectorial, han sido obtenidos los embeddings de distintas representaciones a través de distintas técnicas que dan un embedding a cada token. Debido a que cada documento está representado como la unión de varios embeddings, se ha creado la función ***get_review_vector*** para representar el embedding final de un documento. No es más que la media de los embeddings de los tokens contenidos en dicho documento. Así, se han calculado los siguientes embeddings :

- **TF-IDF** : La representación TF-IDF de un token es el resultado de la multiplicación de :
  - TF(w,d) : Es la frecuencia de un término, y representa su importancia dentro de un documento. Se calcula como el número de veces que aparece el token dentro de un documento dividido entre el número total de tokens que aparece en dicho documento(la suma de sus frecuencias).
  - IDF(w) : Es la frecuencia inversa de aparición de dicho token dentro del conjunto del corpus. Se calcula como el logaritmo de la división del número total de documentos del corpus entre el número de documentos en los que aparece dicho token. Por tanto, un token que esté muy centrado en pocos documentos, obtendrá un valor alto; mientras que para tokens que aparecen en todos los documentos se obtendría un valor muy bajo.

  Los valores obtenidos en dicha representación se han guardado en la sparse matrix ***sparse_matrix_tfidf_2.npz.***
- **Word2Vec :** Word2Vec es una técnica de extracción basada en tratar de obtener los embeddings de un token en base al contexto que le rodea, es decir, en base a los tokens de su entorno. Para ello, esta técnica hace uso de una ventana que va desplazando y se computan las probabilidades de las tokens vecinos contenidos en dicha ventana basada en la palabra central (skip-gram). De esta manera, se va entrenando por descenso de gradiente para obtener la máxima likelihood para los tokens contenidos en dicha ventana**.** Los valores obtenidos en dicha técnica se ah guardado en la sparse matrix ***sparse_matrix_w2v_2.npz,*** sobre la cual se ha elegido realizar la representación con vectores de 200 elementos.
- **Glove :** Glove se basa en tratar de ampliar la lógica empleada en Word2Vec, haciendo uso de las estadísticas globales de representación de cada token, a través de la matriz X. A su vez cada token es representado por un vector w_i. Así. la función de pérdida que se computa durante el entrenamiento

para ir actualizando los valores de los vectores que representan los tokens son : $J(\Theta) = \sum f(X_{ij}) * (w_i * w_j + b_i + b_j - log(X_{ij}))$

De esta manera se, se introduce la función $f()$ que sirve para no dar demasiado valor a las ocurrencias entre tokens demasiado comunes, así como dar bajo valor a las uniones poco comunes. Se debe destacar que el token i representa el token que se está evaluando, y el token j representa cualquier otro token del diccionario. Por otro lado, los valores de la matriz X_{ij} son estáticos una vez se recorre todo el corpus, y representa la ocurrencia del token j en el entorno del token i. Los valores obtenidos en dicha técnica se han guardado en la sparse matrix ***sparse_matrix_glove_2.npz,*** con la cual se ha elegido realizar la representación con vectores de 50 elementos.

- **FastText:** FastText representa un modelo que hace uso de subwords para poder procesar palabras que se encuentran fuera del vocabulario con el que se ha entrenado, las OOV words. Así, cada palabra es representada por la suma de los vectores de las subpalabras contenidas en dicha palabra.

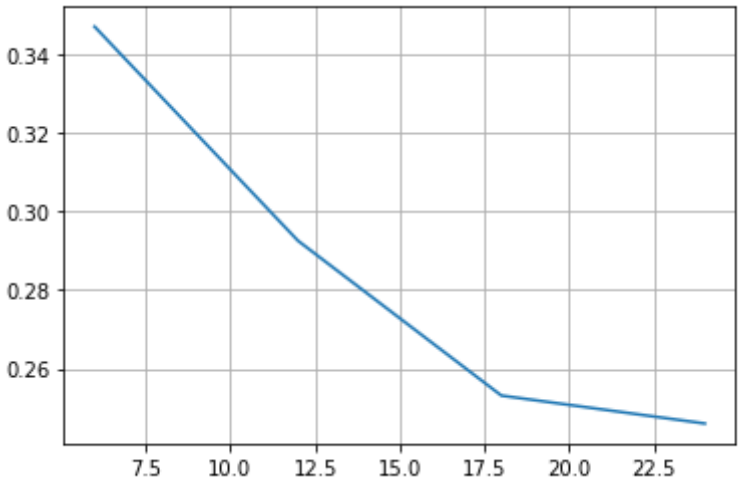
  Tras ello, se computa el likelihood entre dos tokens de la misma manera que se realiza en Word2Vec. Así, los valores obtenidos en dicha técnica se han guardado en la sparse matrix ***sparse_matrix_ft_2.npz***, con la cual se ha elegido realizar la representación con vectores de 300 elementos.

- **LDA:** LDA es una extensión de LSI, en la que los tópicos se caracterizan por una distribución probabilística a través de los tokens contenidos en el diccionario; y los documentos con una distribución probabilística entre los topics disponibles.

  Así, cada topic se asigna con una distribución Dirichlet caracterizada por el parámetro $\boldsymbol{\mu}$ sobre los tokens del diccionario; y cada documento con una misma distribución caracterizada por el parámetro α . En el caso de que los parámetros tomen valores bajos, las distribuciones tendrán una baja varianza y estarán concentradas en unos pocos tokens/topics; mientras que para valores altos las distribuciones estarán muy distribuidas.

  El entrenamiento de dicho modelo se basa en, para cada token contenido en un documento, samplear en su propia distribución de topics y, para cada topic seleccionado, samplear en su distribución de tokens. Finalmente, el objetivo es maximizar la probabilidad de seleccionar el token correcto. Por lo tanto, de esta manera y tras realizar el entrenamiento, al final se tendrá una distribución de topics para cada documento, y una distribución de tokens para cada topic. Así, tal y como se puede observar, el número de topics es la variable que puede ser seleccionada, y en función de su valor obtendremos más o menos valor en las predicciones.

Así, con respecto a LDA, para elegir el número de topics se ha evaluado la coherencia del modelo con 6,12,18 y 24 topics. Así, la gráfica obtenida ha sido la siguiente:



Tal y como se puede observar, la asignación elegida ha sido de 6 tópicos.

3. Clasificación de primer nivel

Finalmente, una vez realizado el pre-procesamiento de los textos, así como la obtención de los distintos embeddings, se ha procedido a realizar la clasificación en base a la categoría de primer nivel asignada a cada uno de los documentos.

Sin embargo, de manera previa a mostrar los resultados obtenidos, se ha de destacar que el DataFrame con las categorías asignadas se ha guardado en el archivo *df_projects_preprocessed_embeddings_2.xlsx,* donde la categoría asignada a cada documento se encuentra en la columna *first_category.* Para saber mejor cómo ha sido realizada la asignación de categorías a cada documento, y debido a la falta de espacio en la memoria, se ruega consultar el apartado "Filtrado de los SciVocCodes" en el archivo Python.

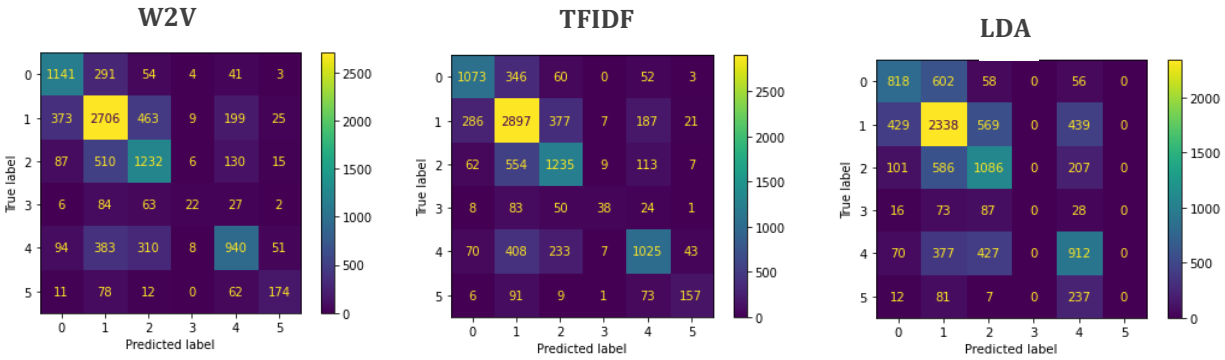
1. Clasificación de primer nivel haciendo uso de embeddings

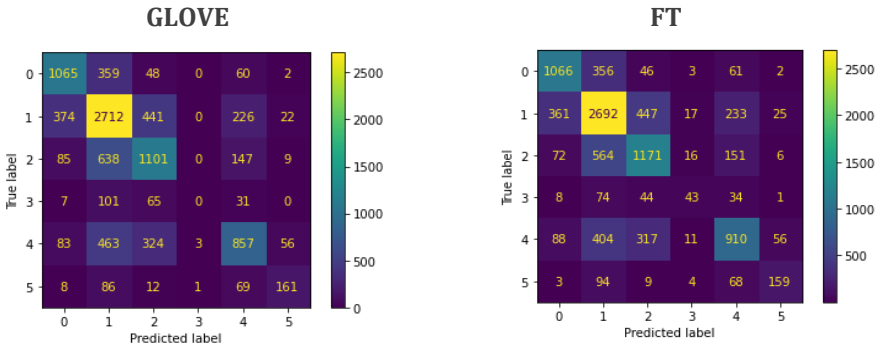
Con respecto a los resultados obtenidos en la clasificación haciendo uso exclusivo de los embeddings obtenidos, se ha hecho uso de **SVC** para la clasificación. Tras haber realizado un grid search para obtener los mejores hiperparámetros para cada modelo, se han obtenido los siguientes valores :

Modelo	W2V	GloVe	FastText	TFIDF	LDA
Accuracy	64,63%	61,31%	62,82%	66,82%	53,59%
Tam.Embeddings	200	50	300	35268	6

Tal y como se puede observar, el modelo que aporta una mejor clasificación es TFIDF, pero a cambio de un tamaño de la matriz de entrenamiento empleada mucho mayor. Por otro lado, W2V se aproxima bastante a cambio de una carga computacional bastante inferior, por lo que puede ser considerado como la opción idónea. Sin embargo, se ha de destacar que el tiempo de cómputo para obtener los embeddings de W2V es mucho mayor que el tiempo necesario para obtener los embeddings de TFIDF, por lo que en la práctica dicha carga computacional se compensa.

Finalmente, las matrices de confusión obtenidas han sido las siguientes:





2. Clasificación de primer nivel con variables extras

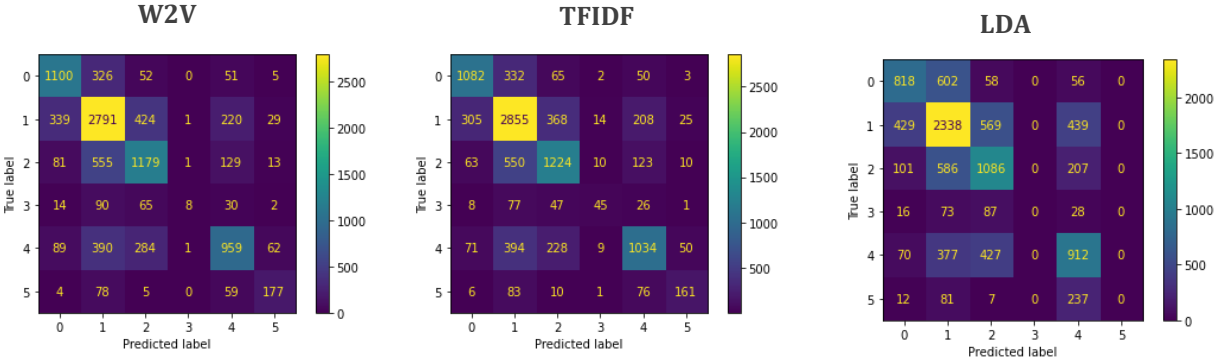
Se ha explorado el hacer uso de otras variables presentes en los metadatos del data frame complementariamente junto a los embeddings para realizar la clasificación de textos. Para ello, las variables utilizadas han sido:

- Coste total del proyecto
- Duración total del proyecto
- País principal del proyecto
- Tópico del proyecto

Estas variables han sido añadidas como una característica/columna más al conjunto de datos tras el último embedding. Los resultados obtenidos han sido los siguientes:

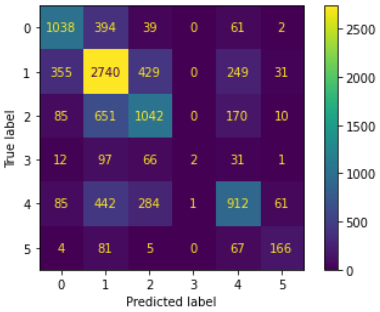
Modelo	W2V	GloVe	FastText	TFIDF	LDA
Accuracy	64,64%	61,38%	62,21%	66,57%	54,68%
Embeddings + Var	204	54	304	35272	10
Diff. Sin Metadatos	+0,01%	+0,07%	-0,61%	-0,24%	+1,09%

Se puede concluir que, en general, el hecho de añadir variables adicionales para nuestro clasificador no consigue aumentar las prestaciones significativamente. Se puede ver como en los embeddings de menos características como LDA o GloVe, aumenta algo más la precisión, pero en los que hay más como W2V apenas hay cambios e incluso con

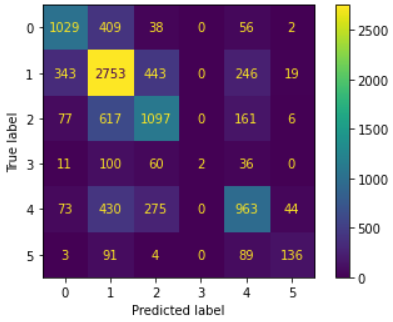


empeoramiento en algunos casos como FastText.

GLOVE



FT



Tal y como se puede observar, la distribución de precisión (en tanto por ciento) a lo largo de las distintas clases ha sido tal que:

	SIN METADATOS					**CON METADATOS**				
CLASE	W2V	TFIDF	LDA	GLOVE	FT	W2V	TFIDF	LDA	GLOVE	FT
0	**74,38**	69,94	53,32	69,42	69,49	**71,71**	70,73	53,33	67,67	67,08
1	71,68	**76,74**	61,99	71,84	71,31	73,37	**75,63**	61,93	72,03	72,37
2	62,22	**62,37**	54,84	55,60	59,14	60,21	**61,82**	54,84	53,22	56,03
3	10,78	11,76	0	0	**21,07**	3,83	**22,06**	0	0,96	0,96
4	52,63	**57,39**	51	47,98	50,95	53,73	**57,89**	64,73	51,09	53,95
5	**51,6**	46,58	0	47,77	47,18	**54,81**	47,78	0	51,39	42,11

4. Distribución de grafos

Como ya sabemos, un grafo en corpus LDA es un gráfico que representa a los documentos en el corpus como nodos y las relaciones entre ellos como líneas o aristas. Cada nodo representa un documento y cada arista representa una relación entre dos documentos. La importancia de crear grafos en este tipo de problemas radica en que estos permiten visualizar de manera gráfica y sencilla la relación entre diferentes documentos en un corpus de texto. Esto puede ser muy útil para analizar la distribución de tópicos en el corpus, encontrar patrones o tendencias en los textos y para facilitar la navegación y búsqueda de información. Además, los grafos, también pueden ser útiles para identificar documentos similares o relacionados, lo que puede ser de gran ayuda en aplicaciones como la recomendación de contenidos o la búsqueda de información en grandes corporaciones de texto.

Para comprender los resultados que vamos a mostrar, es necesario entender el concepto de “layout”. Un layout es una forma de representar visualmente los nodos y las aristas de un grafo. Los layouts son utilizados para darle forma y estructura al grafo y para facilitar su comprensión y análisis. Hay muchas formas diferentes de representar los documentos en un grafo, y la elección de un layout en particular dependerá de las necesidades y objetivos del análisis como es normal. Existen diferentes tipos de layouts, veamos en los que nos vamos a enfocar:

1. Random Layout

El "random layout" es un tipo de layout que se utiliza para representar los nodos y las aristas de un grafo de manera aleatoria. Es decir, en este layout no se tiene en cuenta la relación entre los nodos ni la estructura del grafo, sino que se ubican los nodos de manera aleatoria en el espacio disponible. Este layout es útil en algunos casos para tener una primera impresión de la estructura del grafo y para detectar patrones o tendencias de manera rápida. Sin embargo, es importante tener en cuenta que el "random layout" no es adecuado para analizar detalladamente la relación entre los nodos y la estructura del grafo, ya que no tiene en cuenta estos aspectos.

Dado este análisis, veamos el resultado de aplicar este layout a nuestro corpus LDA. Tal y como vemos en la figura 1, el resultado proporciona información nula al colocar, cada nodo, en una posición completamente aleatoria.

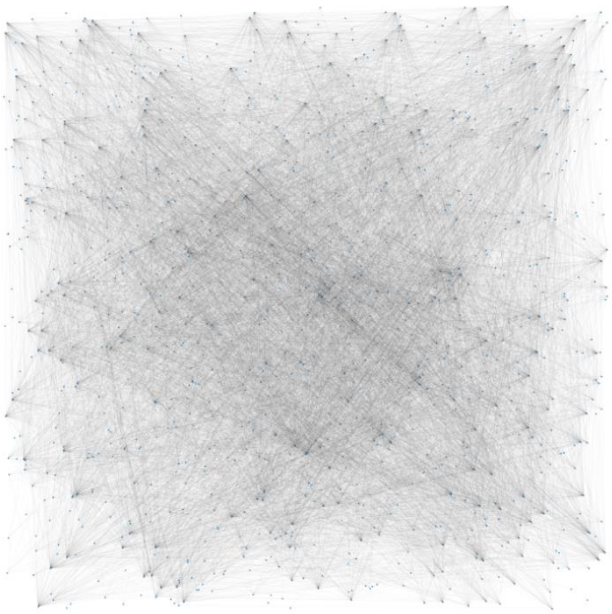


Figura 4.1: Representación del Random Layout en nuestro corpus

2. Spring Layout

El "Spring Layout usando Fruchterman-Reingold" es un tipo de layout que se utiliza para representar los nodos y las aristas de un grafo de manera dinámica y visualmente atractiva. Este layout se basa en el modelo de "resortes" de Fruchterman y Reingold, que trata a los nodos del grafo como partículas que se atraen o repelen entre sí según la fuerza de las relaciones entre ellos. En este layout, los nodos del grafo se representan como partículas que se mueven en el espacio y que se atraen o repelen entre sí según la fuerza de las relaciones entre ellos. Esto permite visualizar de manera dinámica y atractiva la estructura del grafo y la relación entre los nodos. Si bien nos permite analizar el corpus de forma sencilla e intuitiva, debemos tener en cuenta que este tipo de layout usando “Fruchterman-Reingold” puede ser computacionalmente costoso.

Veamos el resultado obtenido al aplicar este layout a nuestro corpus:

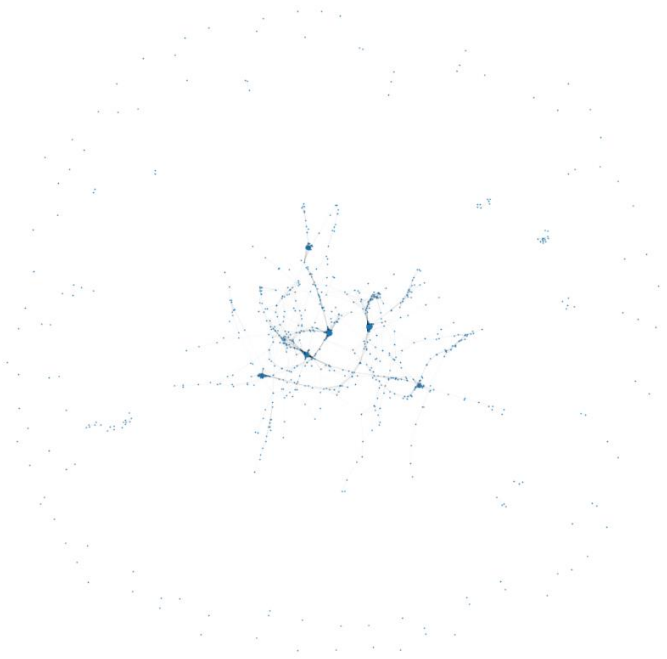


Figura 4.2: Representación del Spring Layout en nuestro corpus

Como podemos ver, algunos nodos o grupos de nodos están aislados del resto en el grafo (coinciden con aquellos con un solo enlace, lo cual tiene sentido al guardar relac ión con pocos documentos del corpus y ser "outliers"). Podemos centrar la atención en e l grupo con más conexiones entre sí y obviar esos documentos menos relevantes en el c orpus al completo:

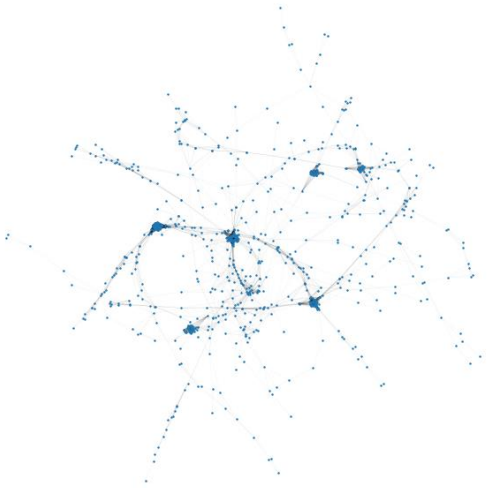


Figura 4.3: Enfoque de los nodos más relevantes en el Spring Layout

Vemos claramente diferenciados 6 conjuntos principales de nodos (asociados a los 6 topics definidos en el modelo LDA). Estos nodos se asocian a los más representativos de cada topic y el resto en el grafo están ligados a documentos que tocan varios topics simultáneamente.

3. Force Layout

Este layout se basa en el concepto de "fuerzas" que actúan sobre los nodos del grafo y que determinan su posición en el espacio. En el "force layout", se aplican diferentes tipos de fuerzas a los nodos del grafo para determinar su posición en el espacio. En el resultado vemos (al igual que en el Spring) 6 conjuntos de nodos conectados entre sí (haciendo referencia a los 6 topics LDA establecidos). Sin embargo, este layout facilita el entendimiento a nivel visual de las relaciones entre documentos del corpus

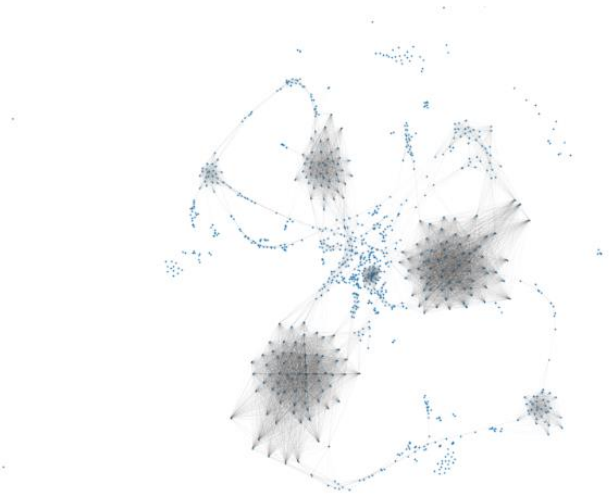


Figura 4.4: Representación del Force Layout en nuestro corpus