Memoria - Proyecto Final

Transmisión de datos - Master Ingeniería de Telecomunicaciones

José Manuel García

Juan Manuel Espinosa

# Introducción

Esta memoria o documentación constituye parte del proyecto final del curso de Tratamiento de Datos, del Máster en Ingeniería de Telecomunicación. Este proyecto tiene como objetivo principal aplicar las técnicas y conocimientos adquiridos a lo largo del curso para abordar un problema de aprendizaje automático utilizando documentos textuales. En particular, se trabajará con un conjunto de datos que recopila recetas de cocina llamado **full\_format\_recipes.json** que contiene mas de 20.000 instancias de recetas de la web *Epicurious*. Este dataset incluye tanto información textual como variables numéricas, lo que ofrece la oportunidad de trabajar con diversas metodologías de tratamiento y modelado de datos.

El objetivo específico del proyecto es resolver un problema de regresión utilizando la puntuación de las recetas (*rating*) como variable de salida. Para ello, se realizará una comparación entre distintas estrategias de representación vectorial de los documentos y se implementarán al menos dos enfoques diferentes de aprendizaje automático el de redes neuronales RNN y ENFOQUE2. Además, el proyecto incluye un análisis crítico de los resultados obtenidos y una extensión basada en la expansión de embeddings contextuales basada en transformers que permita explorar nuevas ideas dentro del ámbito del tratamiento de datos textuales.

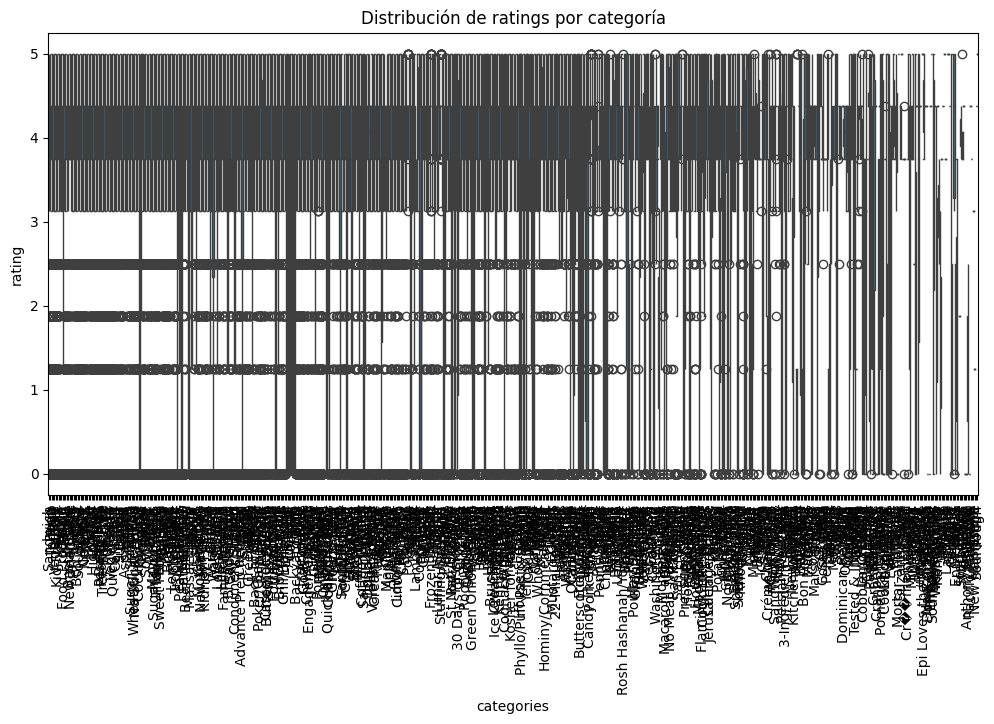
# Proyecto básico

## Análisis de variables de entrada

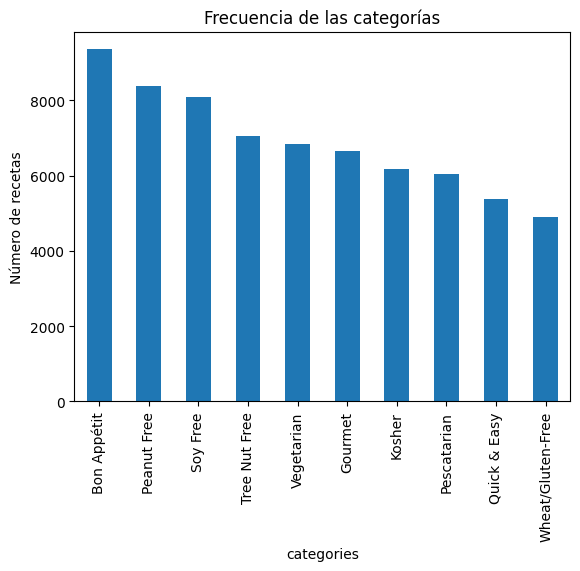
Los primeros compases de nuestro proyecto consisten en la descarga del archivo JSON que contiene todos los datos que se van a usar y la importación de las principales librerías con las herramientas básicas de trabajo. Más adelante se van implementando librerías más concretas, se podrían haber introducido todos los *imports* aquí, pero se ha decidido ir introduciendo las herramientas más concretas conforme vayan siendo necesarias, para tener más plasticidad a la hora de programar, la programación sea modular y que sea más intuitivo su uso.

Una vez con la información volcada en la variable *df* se procede a hacer el Análisis de las variables de entrada. Al no tener conocimiento del grueso de la información los primeros enfoques de análisis surgen de la intuición y de la lógica, teniendo en cuenta que son recetas de cocina.

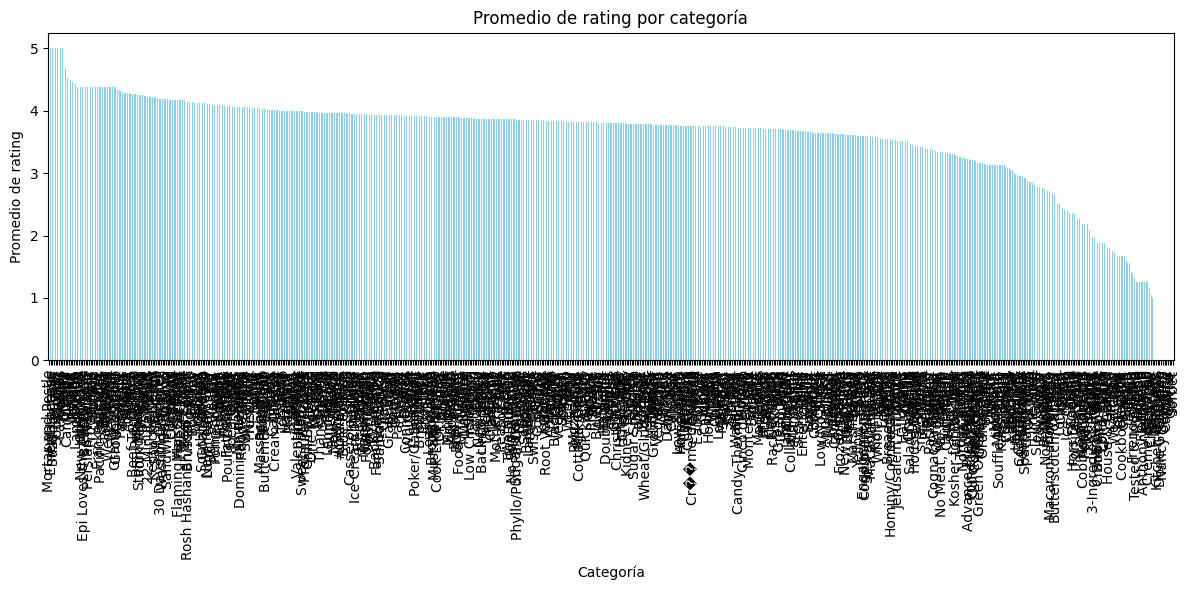
La primera idea era relacionar el apartado de categorías (variable textual) con el ranking (variable numérica de salida), pero no se tuvo en cuenta la magnitud de datos,



Al haber tantas categorías era imposible interpretar nada por lo que se tomó el top 10 categorías para ver las más relevantes.

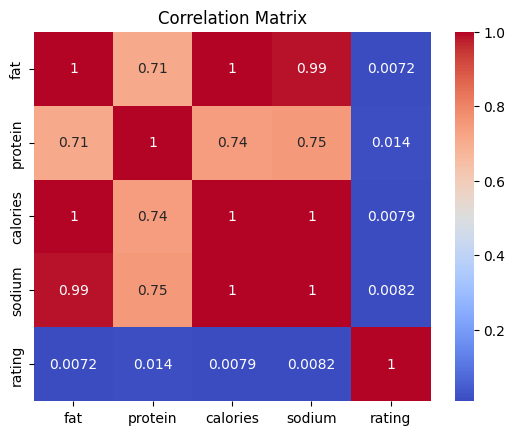


En este caso se puede observar que prácticamente todas las categorías se centran en preferencias o necesidades nutricionales.

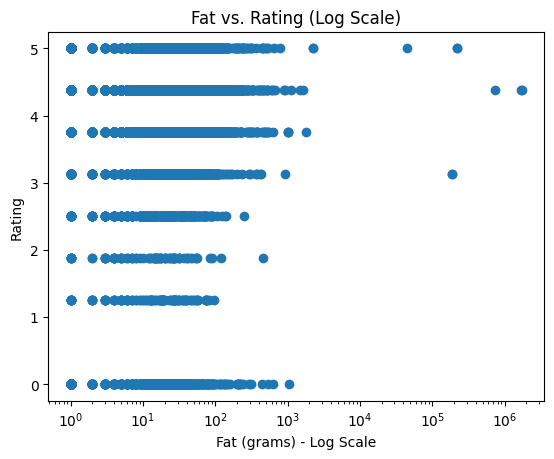


De forma global se puede ver la fluctuación del rating de cada categoría (habiendo obtenido la media)

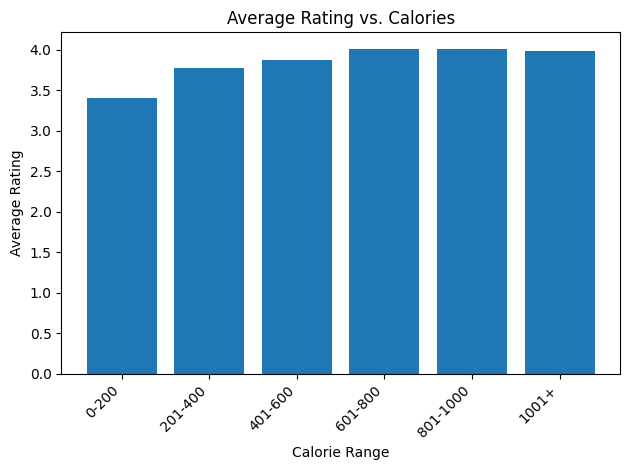
También se ha hecho una matriz de correlación entre el rating y las distintas clases numéricas relacionadas con las propiedades nutricionales del alimento (fat, calories, sodium, protein) mostrada como un heatmap.



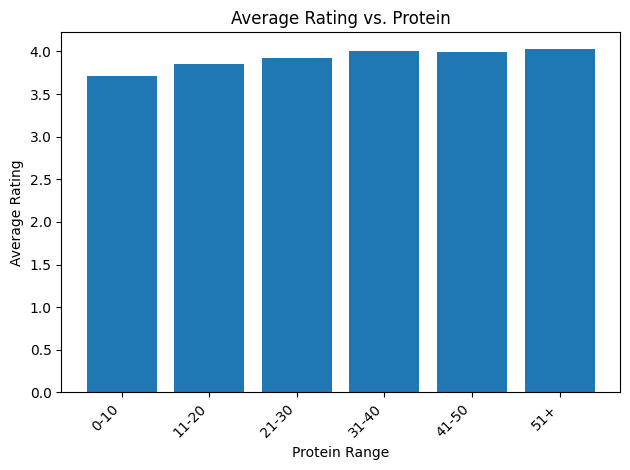
De esta graficación lo más relevante era la correlación entre fat - grasas, calories - calorías y sodium - sodio (y en menor medida protein - proteínas). Siendo variables muy similares como se probará más adelante.

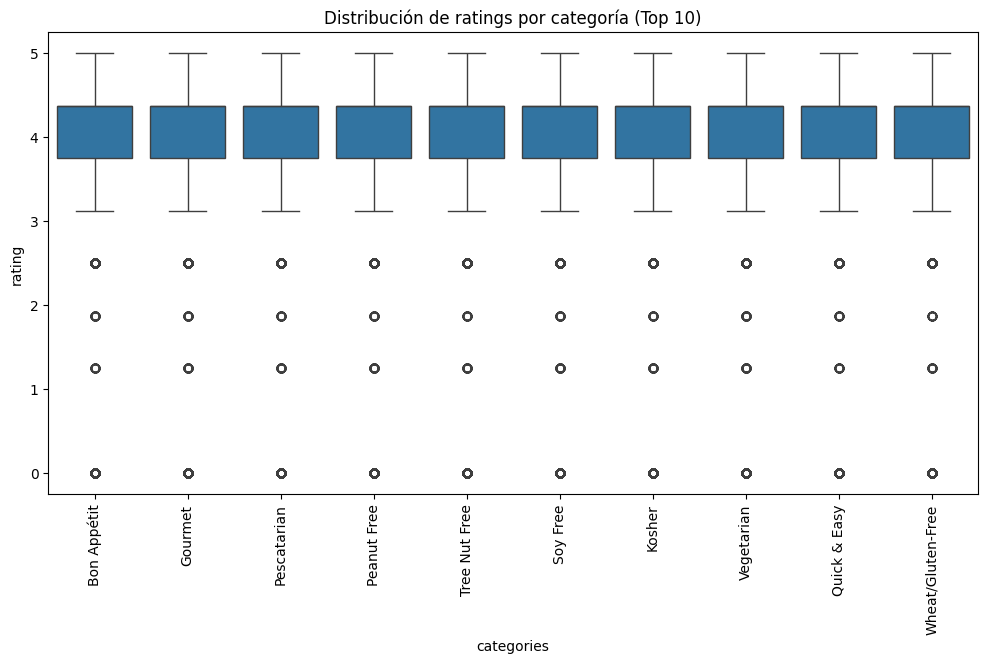


En este punto se empezó a hacer gráficas relacionando las distintas clases numéricas nutricionales con el rating que es nuestro valor de salida. En el caso de la cantidad de grasa se puede observar su relación en escala logarítmica y destaca que mejora el rating cuanto más contenido de grasa tiene. Lo más clave de este punto fue darse cuenta de que hay una serie de parámetros con valores a NaN o ‘0 no porque se les haya otorgado esa valoración, sino porque está vacía.



De igual manera que en el caso de las grasas. hay una relación directa entre el aumento de calorías y de rating (hasta llegar a un punto donde se estabiliza)

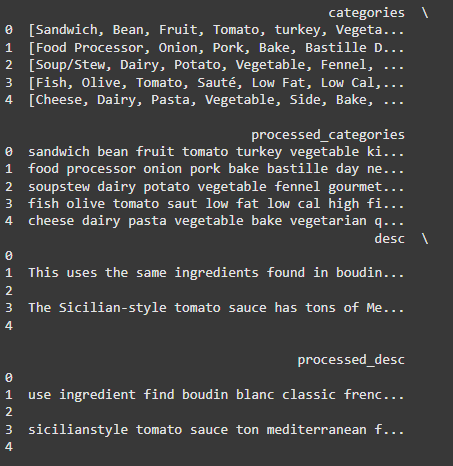
En el caso de la proteína, pese a existir la misma tendencia. es más leve.



Con todo este contexto se pasa al procesado del texto del pipeline las variables textuales más relevantes, categories y desc.

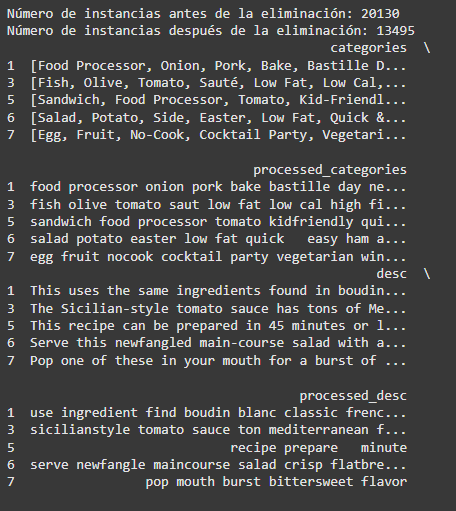
## Implementación de un *pipeline* para el preprocesado de los textos

En este punto se procesan las variables textuales escogidas para el pipeline, que en este caso son categories y desc.



Como se puede observar en el output del método, el procesado del pipeline elimina todos los símbolos que no son interesantes desde la perspectiva de las redes neuronales, como las mayúsculas, símbolos especiales, signos de puntuación, separación, guiones, barras, números

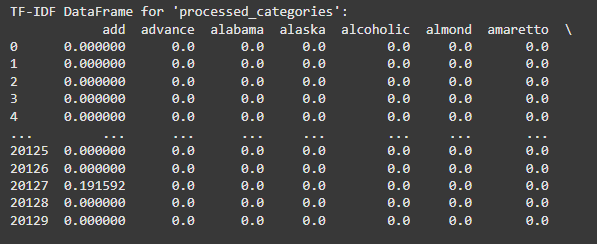
Si observamos el output de desc, continuamos con la idea que vimos antes de que hay ciertos campos que de forma recurrente se encuentran vacíos. Mientras que en variables numéricas esta cuestión era disculpable ya que no se podía diferenciar entre el 0 asignado “por defecto” al obtenido por rating, aquí no es lo mismo ya que se puede detectar cual desc está vacía y cual no, esta no es una problemática cualquiera ya que una vez procesado, estos campos se convierten en ceros, los cuales pueden pervertir y desviar el enfoque del entrenamiento, siendo además desc una de los campos más importantes era una situación que no se podía ignorar. Es así que se hizo un purgado de todas las instancias de todas las recetas con ese campo vacío.



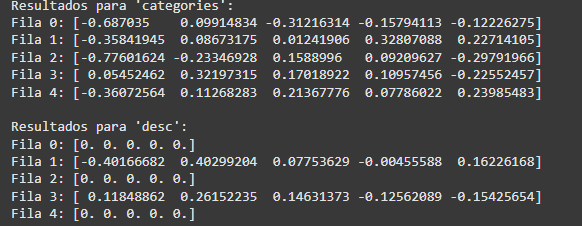
## Representación vectorial de los documentos

Para la vectorización se han usado 3 métodos:

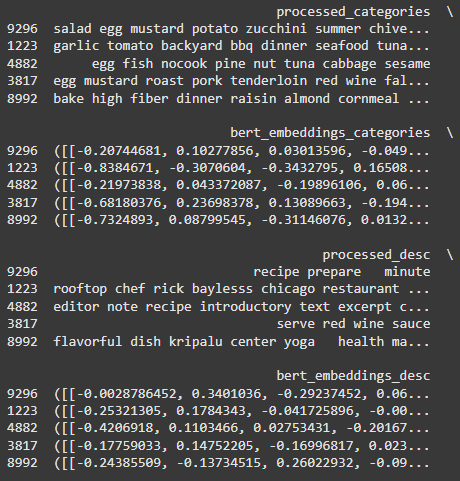
El método TF-IDF



El método Word2Sec



El método BERT por embeddings



En este 3er caso, el proceso al usar embeddings contextuales requiere de una cantidad muy alta de capacidad de procesado, que teniendo en cuenta además los diferentes ajustes y testing se vuelve inmanejable incluso pese a ya tener un dataset reducido. Por lo que se ha reducido la cantidad de recetas a una fracción del total antes de pasarlo al embedding, por extensión el resultado vectorizado de éste será igual de reducido.

## Entrenamiento y evaluación de modelos de regresión

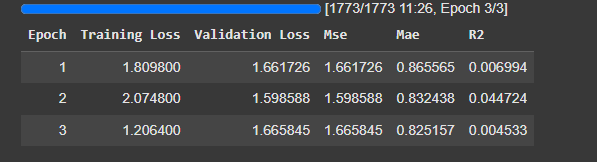
Para este propósito, se han planteado diversas arquitecturas de red neuronal a fin de probar la que diera el mejor resultado. En principio, se supone que las redes neuronales recurrentes (RNN) nos ofrecen las prestaciones que son necesarias para el análisis de textos.

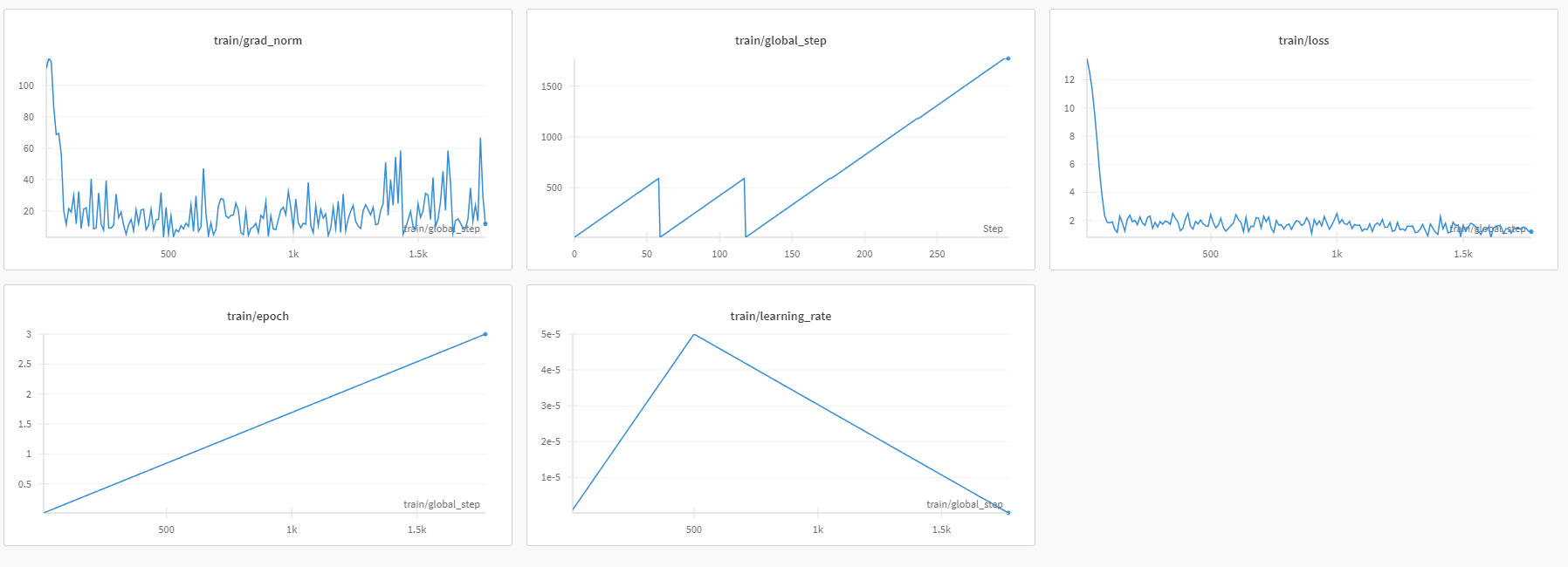
* *RegressionModel*: Modelo de capas *fully-connected* planteado para emular un problema de regresión.
* *ratingNet*: Modelo con una capa GRU central y sin función de activación (emula el problema de regresión).
* *ratingNetfc*: Versión de *ratingNet* con una capa fully-connected adicional.

Como se indicó, la que mejores prestaciones ofreción fue la RNN, es decir, *ratingNet*.

## *Fine-tuning* de un modelo preentrenado con *Hugging Face*

Se toma de nuevo la salida del pipeline y se usa el fine-tunning del modelo preentrenado de Huggign Face para comparar las características del entrenamiento del paso anterior con éste.





# Extensión

## Nuevos Embeddings contextuales

Una vez con el proyecto básico cumplido y funcional, pasamos a la parte de extensión del proyecto.

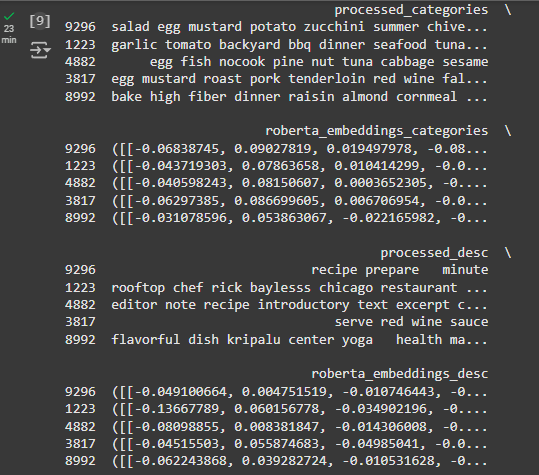
La decisión de expandir el proyecto que hemos tomado ha sido la de ampliar la fase de vectorización con 2 nuevos métodos de embeddings contextuales. Esta idea surge del interés de aprovechar los métodos ya creados para experimentar las diferencias y similitudes al trabajar con embeddings contextuales de distintas características.

Los nuevos embeddings contextuales basados en transformers escogidos han sido RoBERT y DistilBert. Hay 2 razones principales para esta elección: la primera es que al ser embeddings “hijos” o derivados de BERT se nos facilita bastante su implementación, la segunda es que al usar una misma base se pueden extraer conclusiones mucho más precisas a la hora de entender las diferencias entre uno y otro.

### RoBERTa

Es un embedding contextual basado en transformers derivada de BERT que cuenta con un tamaño de datos mucho mayor, que no cuenta con NSP y con dinamismo entre capas. Esto implica desde la teoría, una velocidad bastante más reducida, pero una precisión notablemente más alta.

A la hora de vectorizar se ha hecho más que notable la diferencia de tiempo de procesado, siendo varias magnitudes mayor.

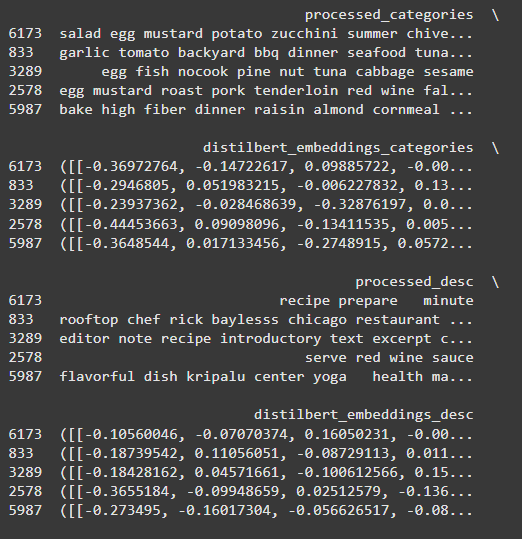


Tiempo 23 minutos

### DistilBERT

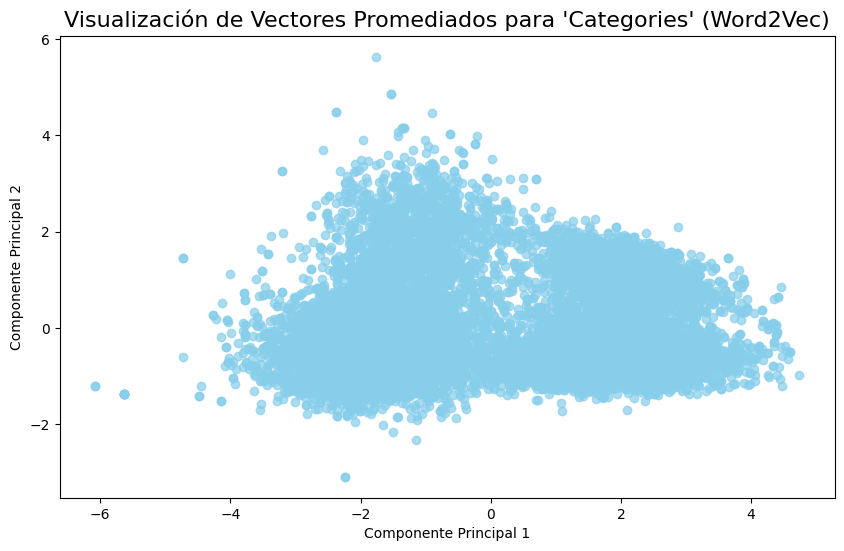
En cuanto a DIstilBERT se dice que es una versión “destilada” de BERT, por lo que tiene un tamaño bastante más reducido y comprimido que el original. Esto en la teoría implica la velocidad más alta y una precisión comparable.

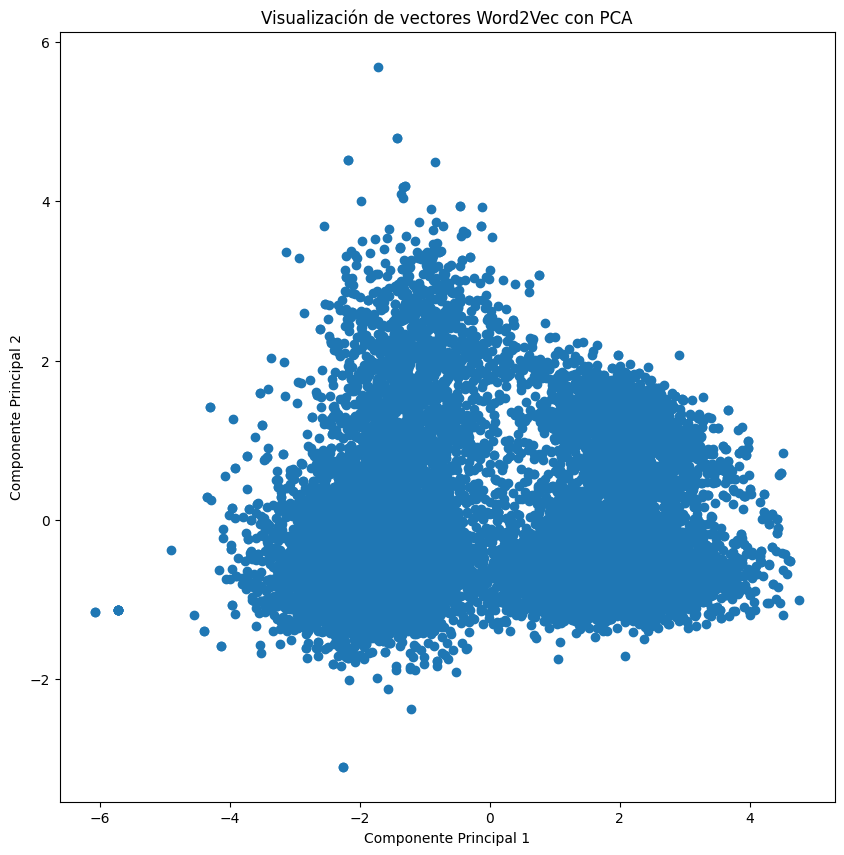
A la hora de vectorizar se ha hecho más que notable la diferencia de tiempo de procesado, siendo varias magnitudes mayor.

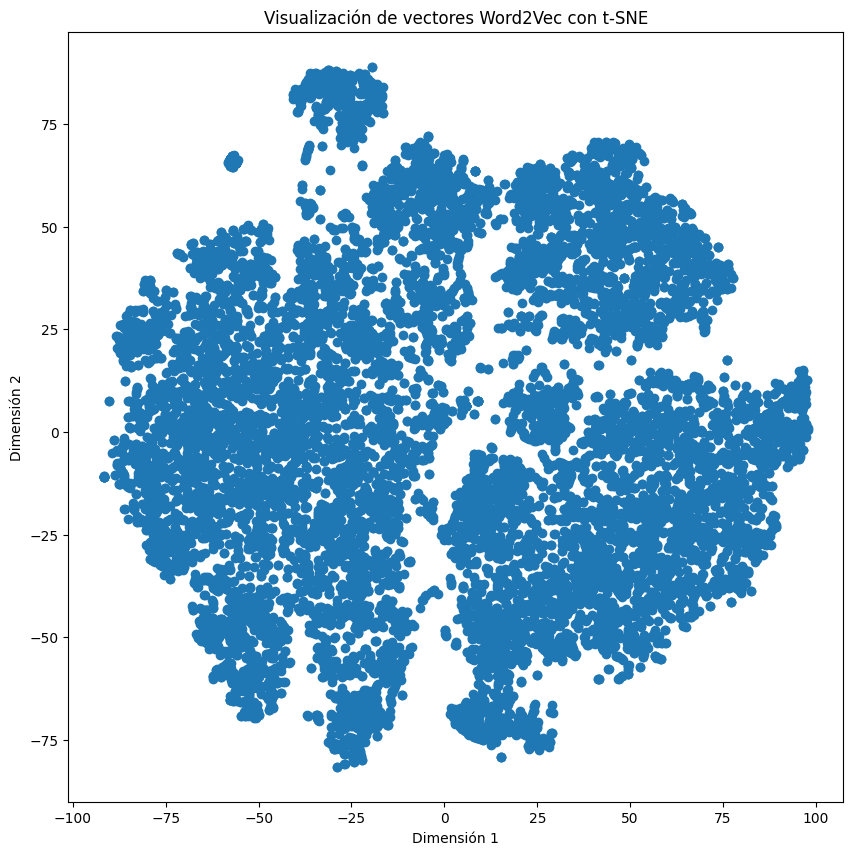


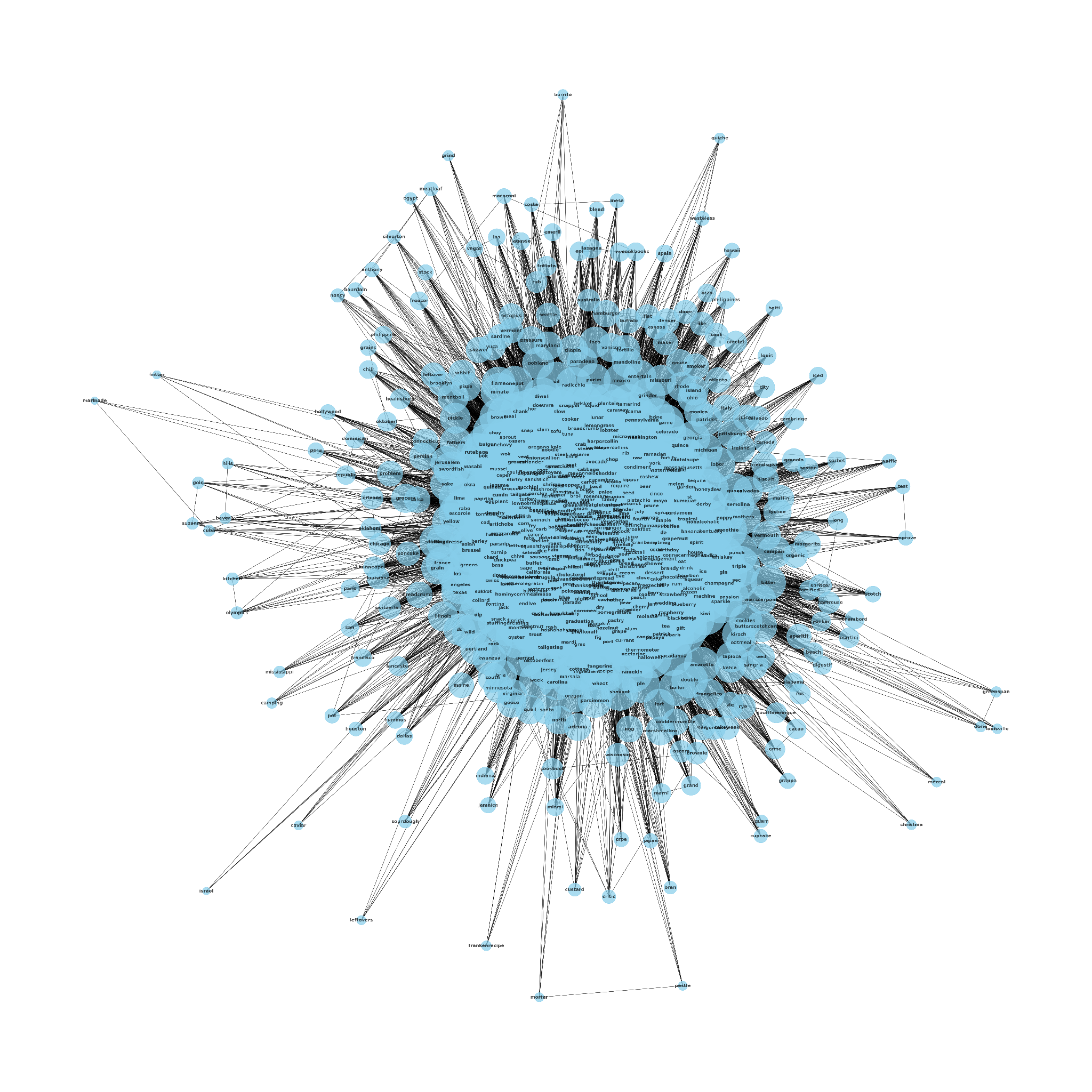
Tiempo:12 minutos

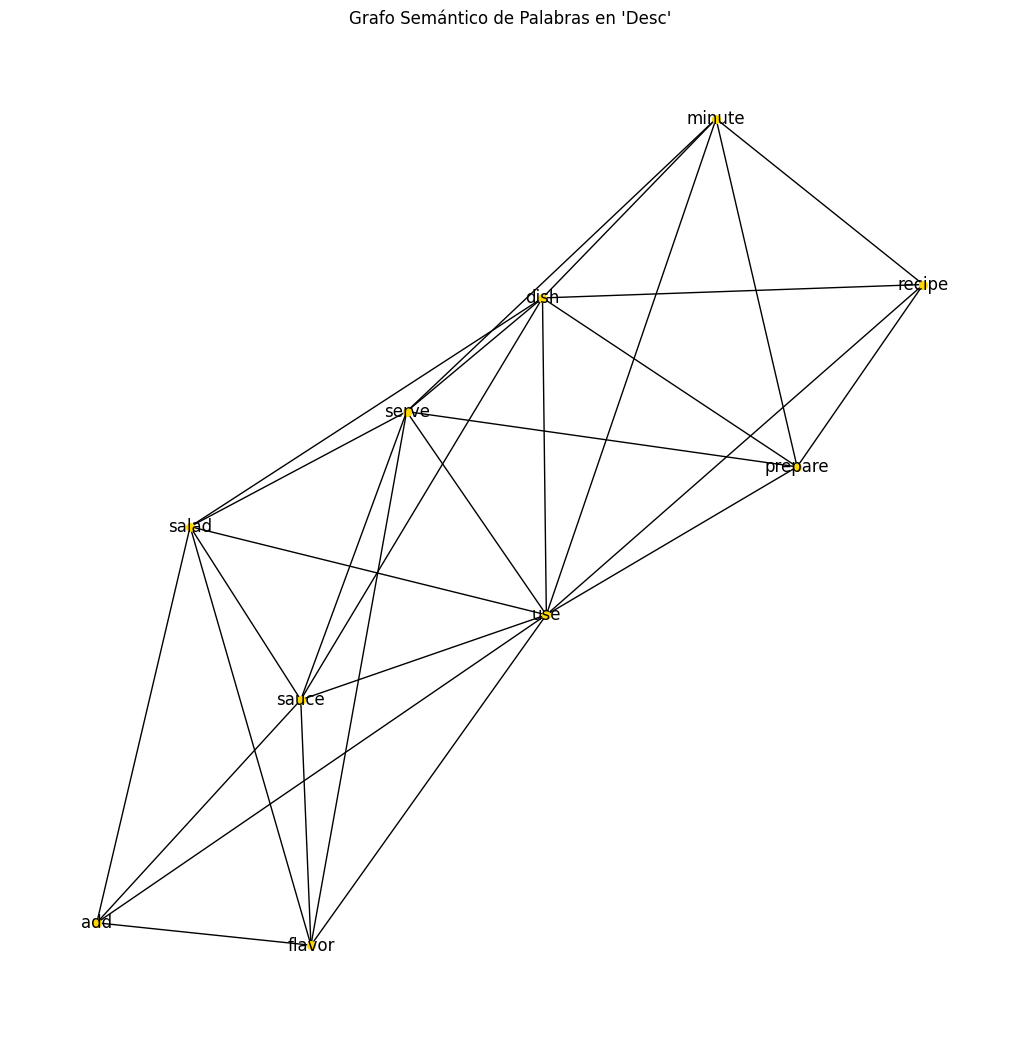
# Visualización de grafos











# Anexo

## Resultados de los modelos

| Resultado | [Modelo]  [Hiperparámetro]  [Vectorización] | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ratingNet  256  bert | ratingNet  128  bert | ratingNet  512  bert | ratingNet  128  word2vec | ratingNet  512  roberta | ratingNet  256  roberta | ratingNet  128  distilbert | ratingNet  512  distilbert | ratingNet  256  distilbert |
| Épocas óptimas | 4000 | 5500 | 3000 | >10000 |  |  |  |  |  |
| Train MSE | 0.0037 | 0.0000 | 0.0031 |  |  |  |  |  |  |
| Test MSE | 4.1361 | 4.1295 | 3.9014 |  |  |  |  |  |  |
| R^2 | -0.9212 | -0.9181 | -0.8121 |  |  |  |  |  |  |