Tratamiento de datos

2024-25

Proyecto final

Oriol Espejo Bas - 100510042

Índice

Introducción y objetivos

El proyecto final de la asignatura tiene como propósito aplicar las técnicas y conocimientos adquiridos en el curso de Tratamiento de Datos para abordar un problema práctico de aprendizaje automático basado en datos textuales. Se trabajará con un conjunto de datos que contiene recetas de cocina, las cuales se abarcar tanto variables textuales como numéricas. Los objetivos para este proyecto son:

- Análisis de las variables de entrada y visualización de la relación entre variables.
- Preprocesamiento y análisis de datos a través de la implementación de un pipeline eficiente para la limpieza y normalización de textos.
- Vectorización textual aplicando distintos métodos.
- Entrenamiento y evaluación de modelos de regresión utilizando redes neuronales y la regresión mediante KNN.
- Implementación *fine-tunning* de un modelo preentrenado con *Hugging Face.*

A lo largo de este documento se detallará el proceso llevado a cabo para alcanzar los objetivos planteados, abarcando desde el análisis y preprocesamiento de los datos hasta la implementación y evaluación de modelos de aprendizaje automático. Se describirán las técnicas utilizadas para la vectorización textual y los enfoques empleados para resolver la tarea de regresión, justificando las decisiones tomadas en cada etapa. Además, se analizarán los resultados obtenidos mediante métricas adecuadas, comparando el rendimiento de las diferentes metodologías aplicadas.

Metodología

Para la realización de este trabajo, se ha seguido un enfoque práctico fundamentado en los conocimientos adquiridos durante el curso, adaptando y aplicando las técnicas vistas en clase para abordar las distintas etapas del proyecto. Se ha trabajado principalmente mediante los notebooks vistos en clase, entendiéndolos, completándolos y ajustando el código e implementaciones a las necesidades específicas de este proyecto.

Requisitos

El proyecto básico consiste en resolver una tarea de regresión, comparando el rendimiento obtenido al usar diferentes técnicas de vectorización de los documentos y al menos dos enfoques distintos de aprendizaje automático. Se detallan los requisitos específicos, a continuación:

- **Análisis de variables de entrada**. Visualización de la relación entre la variable de salida y algunas de las categorías de las recetas.
- **Implementación de un pipeline para el preprocesado de los textos**.
- **Representación vectorial** de los documentos mediante tres procedimientos diferentes:
  - TF-IDF
  - Word2Vec
  - *Embeddings* Contextuales
- **Entrenamiento y evaluación** de modelos de regresión utilizando al menos las dos estrategias siguientes de aprendizaje automático:
  - Redes Neuronales
  - KNN
- Comparación de lo obtenido en el paso 3 con el ***fine-tunning*** **de un modelo preentrenado con *Hugging Face*.**

Análisis de variables de entrada

Para cumplir con este requisito, se ha establecido una relación entre las categorías y las valoraciones asignadas a las recetas. En primer lugar, se han extraído y procesado los datos correspondientes a las categorías y sus respectivas valoraciones. Este análisis se ha abordado desde dos perspectivas fundamentales:

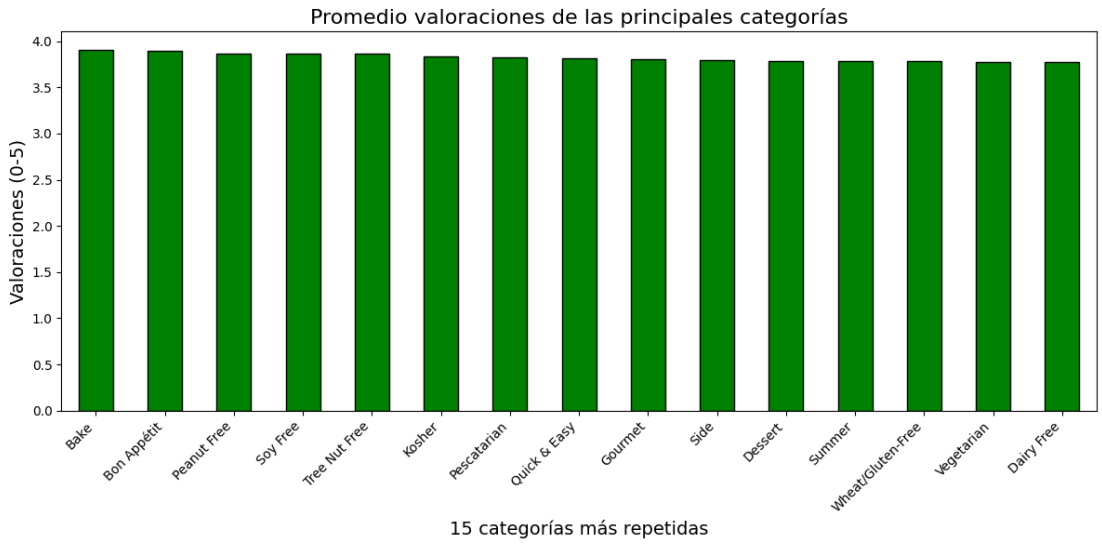
1. Identificación de las categorías más frecuentes y cálculo del promedio de sus valoraciones: Esta perspectiva es importante porque permite identificar las categorías más representativas del conjunto de datos. Al conocer cuáles son las categorías más recurrentes y cómo se valoran en promedio, es posible determinar tendencias generales y obtener una visión global de las preferencias del usuario o del mercado.
2. Selección de las categorías con las mejores valoraciones promedio: Este enfoque se centra en destacar las categorías mejor valoradas, independientemente de su frecuencia. Es relevante porque permite identificar aquellas áreas con mayor calidad percibida o satisfacción por parte de los usuarios. Este análisis es clave para resaltar fortalezas o aspectos diferenciadores, incluso en categorías menos frecuentes, pero altamente valoradas.

Identificación de las categorías más frecuentes y cálculo del promedio de sus valoraciones

Tras la extracción de los datos, se han identificado las 15 categorías más frecuentes y se han agrupado en una lista. Para cada una de estas categorías, se han recopilado las valoraciones de todas las recetas asociadas, calculando el promedio de dichas valoraciones. Finalmente, se ha generado una nueva lista que relaciona cada categoría con su promedio de valoración, la cual ha sido representada visualmente mediante un histograma.

```
has_category
Bake                 3.908764
Bon Appétit          3.893038
Peanut Free          3.866359
Soy Free             3.866067
Tree Nut Free        3.862685
Kosher               3.833198
Pescatarian          3.824789
Quick & Easy         3.814687
Gourmet              3.804622
Side                 3.794430
Dessert              3.789882
Summer               3.787642
Wheat/Gluten-Free    3.785798
Vegetarian           3.776110
Dairy Free           3.773004
Name: rating, dtype: float64
```

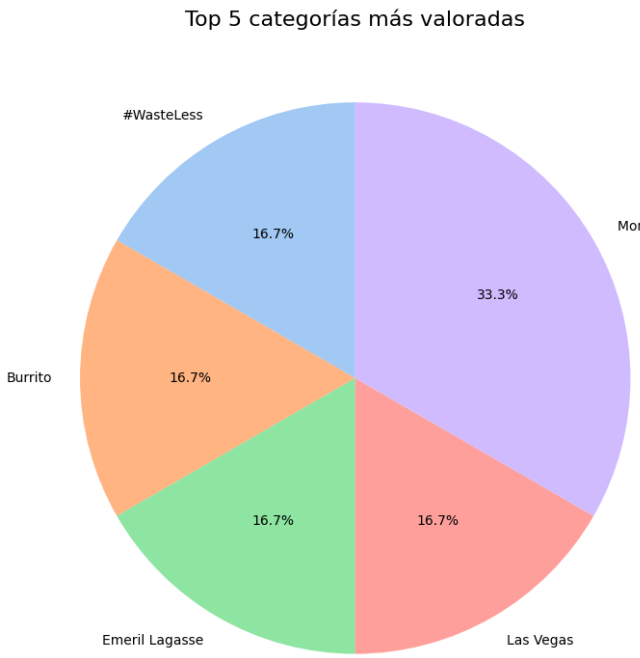
Lista de las 15 categorías más repetidas con sus respectivos promedios de valoración.



Como se observa en la tabla anterior y en la representación gráfica, las 15 categorías principales tienen un promedio de valoraciones muy similar, que varía entre 3.77 y 3.91. Esta similitud en los promedios podría estar relacionada con la alta frecuencia de aparición de estas categorías en el conjunto de datos, lo que puede influir en los valores promedio. Además, dado que las valoraciones oscilan en un rango de 0 a 5, este rango limitado podría dificultar la diferenciación clara entre las categorías más populares. Si el rango de valoraciones fuera más amplio, sería posible distinguir de manera más precisa las diferencias en la calidad percibida de las categorías más comunes.

Selección de las categorías con las mejores valoraciones promedio

Para abordar esta perspectiva, se ha comenzado con la extracción de los datos mencionados anteriormente y, a partir de ellos, se han ordenado las categorías de forma descendente según su promedio de valoraciones. Además, se ha calculado la frecuencia de aparición de cada una de estas categorías en el conjunto de recetas, con el fin de evaluar el impacto que tienen en el conjunto de datos. Este análisis permite observar no solo qué categorías tienen las valoraciones más altas, sino también cuán representativas son dentro de las recetas en general, lo que ayuda a entender mejor la relación entre la calidad percibida y la popularidad de las categorías.




```
Frecuencia de las 5 categorías más valoradas:
categories
#WasteLess          1
Burrito             1
Emeril Lagasse      1
Las Vegas           1
Mortar and Pestle   2
```

Una vez representados los datos, se observa que las 5 categorías mejor valoradas tienen una repercusión bastante limitada, ya que solo se repiten una o dos veces a lo largo de todo el conjunto de recetas. Esto sugiere que, aunque estas categorías reciben valoraciones altas, su presencia en el conjunto de datos es escasa. Es importante destacar que una receta puede estar asociada a una o más categorías, lo que puede llevar a que ciertas categorías con una puntuación elevada no tengan una representación significativa debido a su baja frecuencia en comparación con otras categorías más populares. Esto también indica que las categorías que concentran más recetas, aunque puedan tener valoraciones ligeramente más bajas, son probablemente las que reflejan mejor las preferencias generales de los usuarios.

Implementación de un pipeline para el preprocesado de los textos

Para poder realizar las vectorizaciones y sus representaciones, el entrenamiento y la evaluación del modelo que se ha construido, previamente se ha realizado un preprocesamiento de texto. Esta fase de preprocesamiento aportará en los datos:

- Mejora en la calidad de los datos gracias a la limpieza y a la estructuración que se hace de estos. Se eliminan caracteres innecesarios, palabras irrelevantes (como las *stop words*), y otros elementos que no aportan información significativa para el modelo.
- Normalización y estandarización del texto para reducir la variabilidad de los datos.

El procedimiento que se ha seguido para el preprocesamiento de los datos es el siguiente:

1. Eliminación de las filas que en el campo 'desc' tengan textos vacíos. Los textos vacíos no aportan información relevante al modelo, lo que podría afectar negativamente el rendimiento del análisis. Además, mantener estas filas puede introducir ruido en el modelo, distorsionando los resultados. Por lo tanto, eliminar las filas vacías mejora la precisión y eficiencia del preprocesamiento.
2. Eliminación de caracteres especiales y números de los textos de cada descripción. Se mantiene las palabras. De esta manera simplificamos el texto al máximo posible.
3. Una vez eliminados los caracteres especiales y los números, se han *tokenizado* los textos. Este paso es crucial porque convierte el texto en una estructura que los algoritmos pueden procesar fácilmente. Los principales beneficios de la *tokenización* son la simplificación del texto y la reducción de complejidad de este.
4. Después de realizar una nueva estructura en el texto mediante el proceso de *tokenizar*, se ha realizado la homogenización para estandarizar y limpiar los datos. Este proceso se ha dividido en tres fases:
    - Pasar a minúsculas para eliminar la variabilidad innecesaria del texto, mejorando la consistencia y reduciendo la complejidad, asegurando que las palabras se reconozcan sin importar cómo se escriban.
    - *Lemmataizer*: Con esta técnica se reduce las palabras a su raíz para así poder agrupar variantes de una palabra bajo un mismo término.
    - Eliminar las *stopwords* que no aportan valor semántico y que pueden distorsionar el análisis.

A continuación, se muestra con el primer documento el procedimiento de aplicar cada uno de estos pasos.

```
============= Original Description =============
This uses the same ingredients found in boudin blanc, the classic French white sausage. Start two days before serving.

============= After Wrangling Result =============
This uses the same ingredients found in boudin blanc the classic French white sausage Start two days before serving
```

```
============= After Wrangling Result =============
This uses the same ingredients found in boudin blanc the classic French white sausage Start two days before serving

============= After Tokenizing by words) =============
['This', 'uses', 'the', 'same', 'ingredients', 'found', 'in', 'boudin', 'blanc', 'the', 'classic', 'French', 'white', 'sausage', 'Start', 'two', 'days', 'before', 'serving']

============= Sentences tokeneized =============
[['This', 'uses', 'the', 'same', 'ingredients', 'found', 'in', 'boudin', 'blanc', 'the', 'classic', 'French', 'white', 'sausage', 'Start', 'two', 'days', 'before', 'serving']]
```

```
['this', 'uses', 'the', 'same', 'ingredients', 'found', 'in', 'boudin', 'blanc', 'the', 'classic', 'french', 'white', 'sausage', 'start', 'two', 'days', 'before', 'serving']
```

```
============= Lemmatized review  =============
['this', 'us', 'the', 'same', 'ingredient', 'found', 'in', 'boudin', 'blanc', 'the', 'classic', 'french', 'white', 'sausage', 'start', 'two', 'day', 'before', 'serving']
```

```
============= Lemmatized review  =============
['this', 'us', 'the', 'same', 'ingredient', 'found', 'in', 'boudin', 'blanc', 'the', 'classic', 'french', 'white', 'sausage', 'start', 'two', 'day', 'before', 'serving']

============= Clean lemmatized review  =============
[['us', 'ingredient', 'found', 'boudin', 'blanc', 'classic', 'french', 'white', 'sausage', 'start', 'two', 'day', 'serving'], ['siciliianstyle', 'tomato', 'sauce', 'ha', 'ton', 'mediterranean',
```

Vectorización

La vectorización es un procedimiento clave a la hora de tratar datos ya que convierte el texto, en este caso el preprocesado, en una forma que puede ser entendida y procesada por los modelos de aprendizaje automático. En este apartado, se han explorado tres enfoques diferentes para la representación vectorial de los documentos:

- TF-IDF (*Term Frequency-Inverse Document Frequency*): Este enfoque mide la importancia de las palabras en un documento en relación con su frecuencia en el corpus completo.
- Word2Vec: En este enfoque, los documentos se representan como el promedio de los *embeddings* de las palabras que los componen, lo que captura la similitud semántica entre palabras en función de su contexto en el corpus.
- *Embeddings* contextuales calculados a partir de modelos basados en *transformers*: Modelos como BERT y RoBERTa utilizan redes neuronales de *transformers* para generar representaciones contextuales de las palabras, es decir, palabras que pueden tener diferentes significados según el contexto.

Antes de realizar un análisis detallado de cada enfoque, es importante destacar la creación de un diccionario o corpus a partir del texto preprocesado utilizando la librería Gensim. Este diccionario contiene todas las palabras presentes en el texto preprocesadas, las cuales están ordenadas alfabéticamente. A continuación, se presenta el resultado del diccionario generado.

```
The dictionary contains 17842 terms
First terms in the dictionary:
0 : blanc
1 : boudin
2 : classic
3 : day
4 : found
5 : french
6 : ingredient
7 : sausage
8 : serving
9 : start
```

Cabe remarcar que este diccionario como indica la imagen anterior contiene 17842 términos por lo que ha sido sometido a un filtrado donde se han conservado las palabras que aparezcan en al menos 5 documentos del corpus y se han eliminado las palabras que aparezcan en más del 75% de los documentos del corpus.

```
The dictionary contains 4449 terms
First terms in the dictionary:
0 : blanc
1 : classic
2 : day
3 : found
4 : french
5 : ingredient
6 : sausage
7 : serving
8 : start
9 : two
```

Una vez creado el diccionario, se ha reestructurado el texto de los documentos utilizando la técnica BoW (*Bag of Words).* Esta técnica asocia cada palabra con un identificador único y cuenta cuántas veces se repite cada palabra en el documento. El modelo BoW convierte los documentos de texto en una representación numérica que los modelos de aprendizaje automático pueden procesar, permitiendo así que se puedan aplicar algoritmos de manera eficiente.

```
Representación BoW del primer documento:
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1)]
```

TF-IDF

Para la realización del TF-IDF se ha importado TF-IDF de la librería GenSim y se ha creado un objeto TF-IDF a partir del corpus reestructurado mediante la técnica BoW.

Se ha realizado dos iteraciones, en la primera iteración, se recorre el corpus BoW, y para cada documento se calcula su vector TF-IDF utilizando el modelo TFIDF. Este modelo devuelve un conjunto de tuplas, donde cada tupla contiene el ID de la palabra y su valor TF-IDF correspondiente. En la segunda iteración, se extraen los valores TF-IDF de cada tupla del vector y se almacenan en una nueva lista, obteniendo así la representación final del documento en términos de sus valores TF-IDF.

```
Representación del primer documento preprocesado:
['us', 'ingredient', 'found', 'boudin', 'blanc', 'classic', 'french', 'white', 'sau
Representación de TFIDF del primer documento:
[0.40898218334897296, 0.22135126622397988, 0.21783043492959261, 0.3099728126644617,
```

Word2Vec

Para la implementación de Word2Vec, se ha utilizado la librería GenSim, importando la clase Word2Vec para entrenar un modelo sobre el corpus obtenido mediante la técnica BoW. Se ha realizado una iteración sobre cada documento en el corpus, obteniendo el vector Word2Vec de cada palabra. Luego, se ha calculado el promedio de los vectores de las palabras presentes en el documento, lo que da una representación vectorial única para cada documento. En el caso de que un documento no contenga ninguna palabra que esté en el vocabulario del modelo, se asigna un vector de ceros como representación.

```
Representación de Word2Vec del primer documento:
[-0.19516437  0.62854946  0.07248502 -0.31608388  0.19986582 -1.079417
  0.22346649  1.3750455  -0.20973803 -0.2815936  -0.28142613 -1.3737518
 -0.3423893   0.22808045  0.20341626 -0.3747022   0.13624671 -0.9299024
  0.19847639 -1.0731872   0.30548543  0.39976025  0.07252727 -0.27807835
 -0.25840154  0.04985748 -0.55229723 -0.3477356  -0.41100246 -0.22389364
  0.6342769   0.27108738  0.01586879 -0.372543   -0.3091148   0.6813045
  0.28882748 -0.22818868 -0.10063823 -1.1401963  -0.05930333 -0.8044618
 -0.18822224  0.09597892  0.26081464 -0.12458082 -0.58998704  0.16479982
  0.311446    0.4125821   0.18831857 -0.40289384  0.01016849 -0.11616118
 -0.46105295  0.46176195  0.5585684   0.30508006 -0.611179    0.17207366
  0.05866716  0.17853774 -0.44043812 -0.0698002  -0.65596604  0.31145096
  0.13029411  0.2941228  -0.5073604   0.6914711  -0.32475388  0.3482808
  0.8455247  -0.4035571   0.57603246  0.40683973  0.00541775 -0.22651598
 -0.39754865  0.46031693 -0.23460935 -0.02498004 -0.4253681   1.0688001
 -0.0049177   0.24745443 -0.24366027  0.701291    0.9513112   0.26599276
  0.9080179   0.32722265  0.20165032 -0.059654    0.90574044  0.7190644
  0.3190643  -0.59981143  0.13984749  0.00291469]
```

BERT

BERT (Bidirectional Encoder Representations from Transformers) es un modelo de lenguaje avanzado basado en Transformers, desarrollado por Google, que ofrece representaciones contextuales de palabras y frases. A diferencia de enfoques como TF-IDF o Word2Vec, BERT considera el contexto bidireccional, es decir, el significado de una palabra en relación con todas las palabras de la frase. Esto permite capturar matices más complejos y precisos del lenguaje, especialmente en textos ambiguos o donde el contexto cambia el significado de las palabras. Para entender mejor BERT, se listan tres características claves:

1. BERT solo utiliza la parte de "*encoder*" del transformador. Esta decisión está alineada con el objetivo principal de BERT: crear representaciones contextuales profundas de texto para diversas tareas de procesamiento de lenguaje natural.
2. Como se ha mencionado anteriormente, es bidireccional, está diseñado para analizar el contexto completo de un texto (tanto antes como después de cada palabra). El problema de la bidireccionalidad es que no se puede realizar el preentrenamiento de modelos de lenguaje (LM) observando el futuro. Una posible solución es el lenguaje enmascarado LM.
3. Más adecuado para tareas que requieren comprensión completa de oraciones.

Para la representación vectorial de los documentos se han seguido los siguientes pasos:

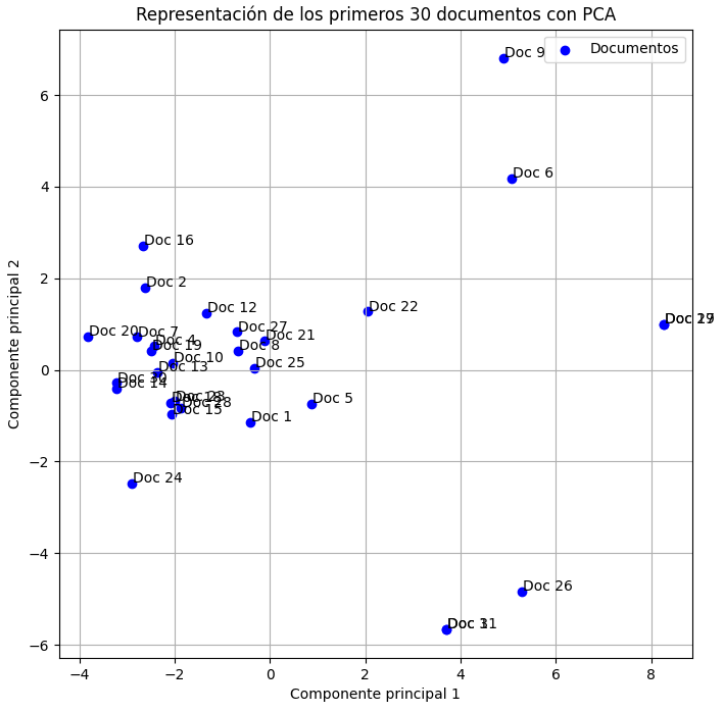
- Carga del modelo y tokenizador de BERT: Este *transformer* usa su propia clase BertTokenizer para convertir texto en tokens que pueda procesar.
- Preparación del texto: Se agrega [CLS] al inicio y [SEP] al final del documento, que son tokens especiales usados por BERT para indicar el comienzo y el final de una secuencia. Se tokeniza y se convierten los tokens en IDs que BERT utiliza internamente.
- Conversión a tensores: Se convierte el texto a representaciones numéricas.
- Generación de los embeddings con BERT.
- Todo este procedimiento se realiza bajo una iteración, ya que este procedimiento se aplica a cada documento del corpus. Es importante destacar que se usa el corpus sin aplicar BoW, ya que con BERT se usa el texto original.

A continuación, se muestra el resultado de BERT. Hay que destacar que para compilar este algoritmo se ha realizado en un tiempo de 39 minutos.

```
 3.8941e-02, -5.8174e-01,  5.6510e-01,  2.3084e-01,  4.6012e-02,
 7.6000e-02, -6.8837e-02,  3.7427e-01, -8.6130e-02, -6.1049e-01,
 6.5217e-03, -8.7563e-02,  3.1435e-01, -2.9354e-03, -1.6276e-01,
-7.5966e-01, -3.3922e-02, -2.0329e-01,  4.9519e-01,  3.1719e-01,
 3.3314e-01, -2.9109e-01, -1.3887e-01,  4.1150e-01,  2.1462e-01,
 1.4794e-01, -2.5320e-01, -3.4732e-01, -1.1140e-01, -1.1132e-01,
 1.8284e-01,  3.0624e-01, -1.4226e-02,  3.9004e-02, -2.2406e-01,
-5.8861e-01,  2.4969e-01,  2.2703e-01,  3.4188e-01,  1.0209e-01,
-1.6261e-01, -4.1228e-01, -3.6786e-01,  5.0388e-01,  4.6060e-01,
-4.7222e-01,  5.1876e-02, -5.5241e-01,  1.0946e-01, -5.7111e-01,
 6.1775e-01, -3.0122e-01,  2.2230e-02,  1.1675e-01, -2.0081e-02,
 3.1247e-01,  1.3532e-01,  1.2976e-01,  1.4317e-01,  3.0102e-01,
-2.8486e-01, -2.8731e-01,  3.2311e-01, -9.4902e-02,  1.7007e-01,
-9.7906e-02,  2.0585e-01,  4.2103e-01, -3.0205e-01, -3.8290e-02,
-2.5575e-01, -5.3869e-01, -3.0844e-01, -7.2784e-02,  6.1002e-01,
-4.1692e-01,  5.8037e-01,  8.6450e-02, -4.6866e-01,  3.5511e-01,
-8.4919e-02,  4.0123e-01,  5.3357e-03,  2.3319e-01,  2.6002e-01,
-1.4314e-01, -3.8710e-02, -2.2138e-01, -3.5490e-01,  2.2152e-01,
-4.6635e-01, -8.1929e-02,  2.1478e-01,  3.3011e-01,  2.1475e-01,
 2.8139e-01,  7.6740e-02,  2.6892e-01, -2.8342e-01, -3.4275e-01,
 2.2597e-01, -1.1835e-01, -7.7276e-02,  3.7495e-01,  7.3779e-02,
```

Trozo vector BERT primer documento

También se ha realizado un gráfico de dispersión de los 30 primeros documentos para ver que conclusiones se pueden extraer con los resultados de BERT. Se ha usado PCA para reducir cada embedding a 2 dimensiones para poder graficarlo.



Analizando la dispersión de estos 30 documentos se puede deducir:

- Los documentos que están más cerca los unos a los otros son semánticamente similares. En cambio, los que están más alejados son semánticamente distintos.
- Si se representaran todos los documentos y se vieran agrupaciones de documentos como la del gráfico podrían tener descripciones parecidas y las recetas serían similares.

Entrenamiento y evaluación

En este apartado se han entrenado y evaluado los modelos anteriores utilizando dos estrategias diferentes de aprendizaje automático, permitiendo comparar la capacidad predictiva y el rendimiento de distintas aproximaciones. El objetivo es ajustar modelos que sean capaces de predecir valores continuos (‘*ratings’*) en base a los datos preprocesados y representados previamente.

- Redes neuronales: Las redes neuronales son modelos de aprendizaje automático inspirados en el funcionamiento del cerebro humano. Están compuestas por una serie de capas de nodos (o “neuronas") conectados entre sí, que procesan la información de manera jerárquica. Se ha usado la librería PyTorch que será utilizada para diseñar, entrenar y evaluar las redes neuronales.
- KNN: Este método se basa en encontrar las observaciones más cercanas en el espacio de características y calcular un valor promedio para realizar predicciones.

Se ha usado la variable ‘*ratings’* como salida, la misma que la del primer requisito. Para entrenar y evaluar los tres métodos anteriores, se han almacenado los tres vectores de salida en distintas variables X, para así poder probar una tras otra. A continuación, se han usado la librería Scickit-lear para dividir el conjunto de datos en dos subconjuntos, uno para entrenar el modelo y otro para evaluarlo.

- X_train: El subconjunto de datos de entrada utilizado para entrenar el modelo.
- X_test: El subconjunto de datos de entrada utilizado para evaluar el rendimiento del modelo (conjunto de prueba).
- y_train: Las etiquetas correspondientes a X_train.
- y_test: Las etiquetas correspondientes a X_test.

Como se ha visto en la BERT, se aplica el mismo procedimiento de convertir a tensores estas variables, convirtiéndolas así en representaciones numéricas para que puedan ser entrenadas y evaluadas.

Para la evaluación de los entrenamientos se ha usado el siguiente parámetro:

- MSE (Error Cuadrático Medio): Es la media de los cuadrados de las diferencias entre las predicciones y los valores reales. Una menor cantidad de MSE indica que el modelo tiene un buen ajuste a los datos de prueba.

Red neuronal

Se ha definido una red neuronal de la clase *Regression* mediante la librería PyTorch. A continuación, se ha creado una instancia de la red neuronal con la dimensión del número de características del tensor del subconjunto de datos de entrada de entrenamiento. Finalmente se ha definido un número de épocas (iteraciones) para entrenar el modelo y se ha realizado el respectivo bucle.

Word2Vec

Resultado del entrenamiento:

- Epoch [10/50], Loss: 10.1225: Al principio del entrenamiento (después de la décima época), la pérdida es alta. Esto es normal, ya que el modelo todavía no ha aprendido patrones significativos.
- Epoch [20/50], Loss: 2.8031: Después de más épocas de entrenamiento, el valor de la pérdida disminuye significativamente. Esto indica que el modelo ha comenzado a aprender y a ajustar sus parámetros para mejorar el rendimiento.
- Epoch [30/50], Loss: 2.9703: Aquí vemos que la pérdida aumenta ligeramente en comparación con la época anterior. Esto puede ser indicativo de que el modelo está empezando a sobreajustarse o que el optimizador necesita un ajuste de parámetros.
- Epoch [40/50], Loss: 2.6409: La pérdida sigue disminuyendo, lo cual es una señal de que el modelo está mejorando y aprendiendo a hacer predicciones más precisas.
- Epoch [50/50], Loss: 2.4480: Al final del entrenamiento (después de 50 épocas), la pérdida sigue siendo relativamente baja, lo que indica que el modelo ha alcanzado un buen nivel de aprendizaje.

```
Epoch [10/50], Loss: 10.1225
Epoch [20/50], Loss: 2.8031
Epoch [30/50], Loss: 2.9703
Epoch [40/50], Loss: 2.6409
Epoch [50/50], Loss: 2.4480
```

TF-IDF

Para TF-IDF, se ha tenido que repetir el procedimiento del apartado 3, esta vez usando *padding* para preparar los datos en una matriz densa. Con el TF-IDF inicial, al convertir el vector TF-IDF antiguo a tensores no se podía realizar la acción.

```
Epoch [10/50], Loss: 13.7253
Epoch [20/50], Loss: 11.4813
Epoch [30/50], Loss: 7.7433
Epoch [40/50], Loss: 3.5311
Epoch [50/50], Loss: 2.3069
```

Como se puede apreciar, TF-IDF alcanza un buen nivel de aprendizaje en última instancia. Aun así, este modelo puede mejorar más si este se entrenará más. En este caso, la pérdida comienza en un valor más alto (13.7253) y disminuye de forma más gradual.

BERT

```
Epoch [10/50], Loss: 2.1986
Epoch [20/50], Loss: 1.9614
Epoch [30/50], Loss: 1.9477
Epoch [40/50], Loss: 1.8806
Epoch [50/50], Loss: 1.8099
```

La pérdida en este caso también es gradual, lo que sugiere que le modelo esta aprendiendo. En este caso, se puede ver como en las últimas 10 épocas varia muy poco lo que implica que puede estar llegando al mínimo local.

MSE

Modelo	MSE
W2V	2.5429
TF-IDF	2.4966
BERT	1.7850

El modelo TF-IDF tiene un MSE de 2.4966, lo que indica que realiza predicciones ligeramente mejores en promedio en comparación con W2V, cuyo MSE es de 2.5429. Esto sugiere que el modelo TF-IDF está mejor ajustado en términos de las predicciones realizadas.

En cuanto a BERT, que tiene un MSE de 1.7850, se observa una mejora significativa respecto a TF-IDF y W2V. El modelo BERT, al ser un modelo de aprendizaje profundo contextualizado, ofrece una representación mucho más precisa de los textos, capturando matices semánticos más complejos que los métodos tradicionales como W2V y TF-IDF. Este rendimiento superior refleja su capacidad para comprender el contexto de las palabras dentro de frases o textos, lo que le permite hacer predicciones más precisas en comparación con los otros dos modelos.

En conclusión, BERT muestra un mejor desempeño con el menor MSE.

KNN

Como se ha visto anteriormente el KNN encuentra las observaciones más cercanas en el espacio de características y calcular un valor promedio para realizar predicciones. Para este caso se ha ido variando el número de vecinos.

Modelo	N-neighbours	MSE
BERT	10	1.7765
BERT	20	1.7154
BERT	30	1.6975
BERT	50	1.6950
TF-IDF	10	1.7887
TF-IDF	20	1.7361
TF-IDF	30	1.7304
TF-IDF	50	1.7268
TF-IDF	100	1.7216
TF-IDF	200	1.7194
Word2Vec	10	1.8329
Word2Vec	20	1.7649
Word2Vec	30	1.7407
Word2Vec	100	1.7257
Word2Vec	200	1.7103

- Con BERT, al aplicar KNN se continúa obteniendo el mejor MSE. Sin embargo, a diferencia de Word2Vec o TF-IDF, al incrementar el número de vecinos hasta 100, se observa un aumento en el MSE.
- Para TF-IDF y Word2Vec ocurre el caso contrario ya que al aumentar hasta 200 sigue disminuyendo el MSE. Esto puede indicar que estos modelos logran capturar mejor las representaciones de los datos a medida que aumenta la cantidad de características consideradas, lo que mejora su capacidad para realizar predicciones más precisas.

Transformers

En este último apartado, volvemos a utilizar el modelo *transformer* BERT, pero esta vez aplicamos la técnica de *fine-tuning*, que consiste en ajustar un modelo preentrenado para que se especialice en una tarea específica. En el contexto de este proyecto, como hemos hecho anteriormente se hace un ajuste para predecir la valoración (rating) de una receta en función de su descripción. Se puede decir que se ha enseñado a BERT a aprender la relación entre el contenido textual de las recetas y las valoraciones.

Para la realización de este código se han seguido los siguientes pasos:

- Extracción de datos e importación de librerías necesarias del *transformer.*
- *Tokenización del texto mediante BERT, como se ha visto anteriormente.*
- División del conjunto de datos entre validación y entrenamiento como en el caso anterior.
- Creación del *dataset* compatible con *Hugging Face*.
- Configuración del entrenamiento. Se han elegido 10 épocas debido a la gran cantidad de tiempo que tarda en ejecutar BERT.
- Se ha entrenado y evaluado el modelo obteniendo los siguientes resultados.

Epoch	Training Loss	Validation Loss
1	No log	16.077709
2	14.893800	15.882895
3	14.893800	15.535700
4	14.395400	14.997594
5	14.395400	14.175160
6	13.201800	12.792418
7	13.201800	11.534040
8	11.068500	11.055355
9	11.068500	10.611235
10	9.868800	9.747421
11	9.868800	8.653713
12	8.289800	8.006978
13	8.289800	6.968947
14	6.411000	5.889246
15	6.411000	4.890109
16	4.696900	4.015092
17	4.696900	3.110121
18	3.085800	2.524754
19	3.085800	2.196552
20	2.033900	2.288405

Los resultados de entrenamiento y validación son buenos, en el caso que se haya aumentado el número de épocas se vería reflejada mejor la progresión positiva de los datos que se reflejan en esta tabla. La pérdida de entrenamiento disminuye progresivamente desde 14,89 en la segunda época hasta 2,03 en la última época. Esto indica que el modelo está aprendiendo y ajustando sus pesos, ya que la pérdida de entrenamiento se va reduciendo. También se puede concluir que el modelo generaliza bien a datos nuevos ya que en este caso se pasa de tener 16 a 2,28.

Se debe tener en cuenta también, la disminución constante tanto en la pérdida de entrenamiento como en la validación ya que sugiere que el modelo está aprendiendo de manera efectiva, sin sobreajustarse a los datos de entrenamiento.

MSE

En este caso, se ha logrado un MSE de 2,28, lo que indica que el modelo más efectivo ha sido el vector BERT entrenado y posteriormente evaluado utilizando el método KNN.

Extensión

La alimentación es un pilar fundamental para la supervivencia y el bienestar del ser humano. Mantener una dieta equilibrada es esencial para garantizar un cuerpo sano y prevenir enfermedades. Por ello, es crucial utilizar adecuadamente los alimentos, identificando los elementos que determinan su impacto en la salud.

En el caso del fichero analizado, las recetas tienen una serie de características que se han encontrado e investigado en la *web* "*Manual MSD*" para entender sus efectos en el cuerpo humano y la cantidad que se debe abarcar en el día a día:

- Cantidad de grasas: Hay grasas saludables y grasas saturadas que son malas para el organismo. Se debe limitar las grasas.
- Proteínas: Priorizar platos con un nivel de proteínas moderado.
- Calorías: Priorizar platos con densidad calórica moderada.
- Sodio: Limitar el sodio.

Se ha desarrollado un algoritmo diseñado para identificar las recetas más saludables, teniendo en cuenta una serie de características nutricionales clave. El objetivo principal de este algoritmo es ayudar a las personas que practican deporte a seleccionar recetas que mejor se adapten a sus necesidades dietéticas y de rendimiento.

1. El primer paso ha sido filtrar las recetas que contienen más del 60% del valor máximo de proteínas.
2. Después de haber seleccionado las recetas con un nivel adecuado de proteínas, se han ordenado las recetas de forma descendente en función de las calorías.
3. Finalmente, se ha aplicado los criterios contrarios a las grasas y al sodio. Para estas características, se han priorizado las recetas con menores cantidades de grasas y sodio.

A continuación se muestran las 5 recetas que se adaptan más a la dieta de un deportista.

Título	Calorías	Proteínas	Sodio	Grasas
Rice Pilaf with Lamb, Carrots, and Raisins	4157357	236489	3134853	221495
Rice Pilaf with Lamb, Carrots, and Raisins	4157357	236489	3134853	221495
Lamb Köfte with Tarator Sauce	4518216	166471	7540990	44198
Deep-Dish Wild Blueberry Pie	29997918	200210	27570999	1716279
Pear-Cranberry Mincemeat Lattice Pie	30111218	200968	27675110	1722763

Conclusión

Este trabajo ha permitido explorar diversas técnicas de tratamiento de datos, destacando la importancia creciente de este proceso en un mundo cada vez más impulsado por la orientación del dato. Actualmente, el tratamiento de datos se encuentra en auge, especialmente en el ámbito empresarial, donde las organizaciones reconocen su valor estratégico para la toma de decisiones y la optimización de procesos. La existencia de datos coherentes e íntegros es una condición necesaria para que las empresas se puedan beneficiar de la inteligencia artificial, entre otras cosas.

Este trabajo demuestra cómo un enfoque adecuado en el preprocesamiento de datos puede mejorar significativamente el rendimiento de modelos avanzados como Word2Vec, TF-IDF o BERT.