AMAZON - CLASIFICACIÓN DE PRODUCTOS DE FOTOGRAFÍA

TÓPICOS AVANZADOS DE ANALÍTICA

ANGUIE GARCIA
MILI GALINDO
SONIA RAMÍREZ
LOURDES RODIL

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN ANALÍTICA PARA LA INTELIGENCIA DE NEGOCIOS
24 DE MAYO DE 2024

Tabla de Contenido

AMAZON - CLASIFICACIÓN DE PRODUCTOS DE FOTOGRAFÍA

1 Adquisición de los datos

Para este proyecto se utilizó el dataset “Amazon fotos” incluido en la librería Pytorch[1] usado en el paper denominado ““Pitfalls of Graph Neural Network Evaluation”[2]. En este conjunto de datos los nodos representan los bienes y los enlaces representan que dos bienes frecuentemente se compran juntos. Las características de los nodos son reseñas de productos codificadas en *bag-of-words*, y las etiquetas de clase las proporciona la categoría de producto.

La tarea es asignar productos a su respectiva categoría de producto, dadas las características de los nodos.

1.1 Análisis Descriptivo de los datos

El conjunto de datos tiene las siguientes características:

Nombre	# Nodos	# Edges	# features	# Clases
Photo	7.650	238.162	745	8

Sobre este dataset se eliminaron los nodos aislados, quedando 7.535 nodos. Además, se visualizó el grafo completo (ver ilustración 1) y un subgrafo principal a partir de la selección de la componente más grande como se presenta en la ilustración 2.

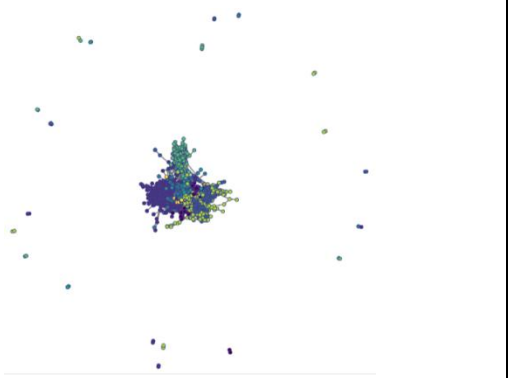


Ilustración 1 - Grafo Completo

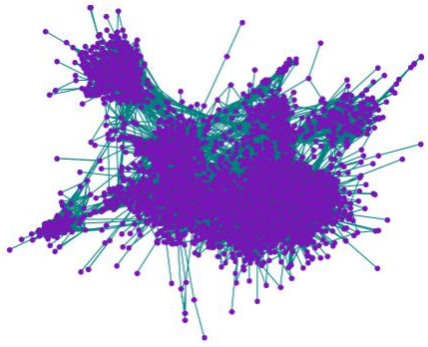


Ilustración 2 - Subgrafo principal

Se observa que dataset de Amazon, el 80% de los nodos cuenta presenta tiene un rango de grados entre 1 y 45 grados o conexiones:

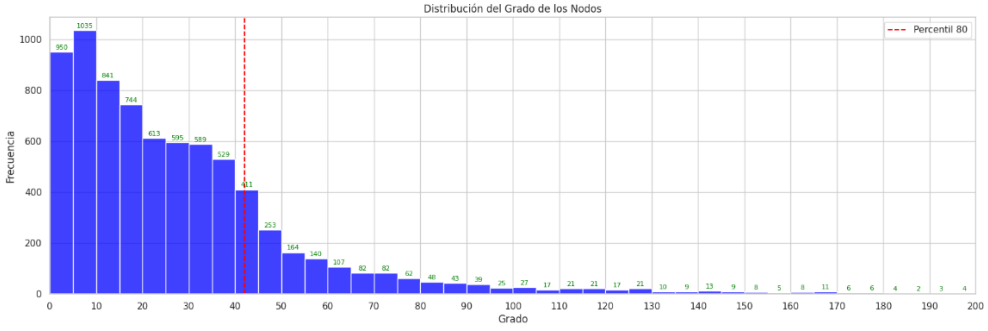


Ilustración 3 - Distribución del grado de los nodos en el grafo

[1] https://pytorch-geometric.readthedocs.io/en/2.5.0/generated/torch_geometric.datasets.Amazon.html
[2] Oleksandr Shchur, Pitfalls of Graph Neural Network Evaluation. Technical University of Munich, Germany.

Adicionalmente, como se observa en la ilustración 4, las 8 clases del dataset tienen una participación entre el 4.3% y el 25.4%, en donde existen dos clases predominantes (# 6 y #1) que representan el 47% de nodos del dataset:

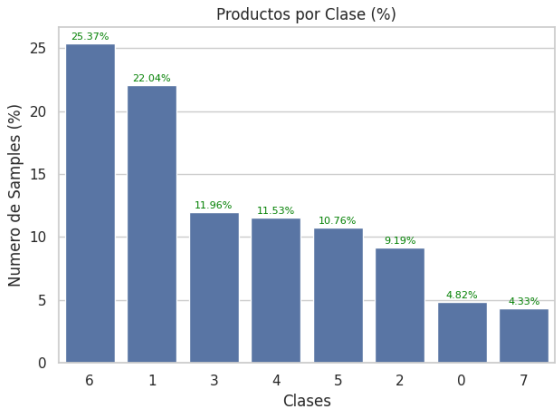


Ilustración 4 - Distribución de nodos por clase

1.2 Métricas del grafo

Se evaluaron las propiedades globales del grafo, observándose que el número de conexiones medio de los nodos del grafo es 31, una densidad (Número de conexiones / Número posible de conexiones) de 0.0040 y una transitividad o coeficiente de agrupamiento global de 0.1773, es decir, un nivel relativamente bajo de agrupamiento en el grafo.

2 Feature Engineering

En este proyecto, no se realizó feature engineering adicional en el dataset, ya que el conjunto de datos empleado ya estaba configurado específicamente para su uso en modelos de grafos, con características preprocesadas y optimizadas, tanto a nivel de nodos como de aristas, lo que facilitó su integración directa en arquitecturas de redes neuronales de grafos. Por lo tanto, la decisión de no realizar feature engineering adicional está justificada por la calidad y la preparación previa del dataset, lo cual optimizó el flujo de trabajo y garantizó la utilización eficiente de los recursos computacionales.

3 Modelación

Para el modelamiento de los grafos se realizó separación del grafo en *"train" con un 70%* y *test con 30%*. En el dataset de entrenamiento se generó una separación de validación de 10%.

Train	Validation	Test	Total
4.747	527	2261	7.535

Tabla 1 – Separación del dataset Amazon

Se realizaron 10 modelos que incluyen un proceso de tunning para varios de ellos así:

#	Nombre del modelo
1	Perceptrón Multicapa (MLP)
2	Perceptrón Multicapa (MLP) con tunning
3	GCN (Graph Convolutional Network)
4	GCN (Graph Convolutional Network) con tunning

#	Nombre del modelo
5	GATv2Conv
6	GATv2Conv con tunning
7	GATConv
8	GATConv con tunning
9	GraphSAGE
10	GraphSAGE con Early Stopping

Tabla 3 – Tabla de parámetros utilizados en cada modelo de predicción

3.1 Modelo Perceptrón Multicapa (MLP)

Se construyó una red neuronal de tipo Perceptrón Multicapa (MLP) utilizando PyTorch para clasificar clases del grafo.

- Este modelo consta de dos capas lineales, funciones de activación ReLU para introducir no linealidad, y mecanismos de regularización mediante dropout con una probabilidad del 50% durante el entrenamiento, tanto antes como después de la primera capa lineal.
- La salida del modelo se normalizó utilizando la función log_softmax para obtener probabilidades logarítmicas adecuadas para la clasificación.
- Finalmente, el modelo está optimizado con el algoritmo Adam, configurado con una tasa de aprendizaje de 0.01 y un decaimiento de peso de 5e-4 para evitar el sobreajuste.

```python
class MLP(torch.nn.Module):
    def __init__(self, dim_input, hidden_channels, dim_output):
        super(MLP, self).__init__()
        torch.manual_seed(42)
        self.lin1 = Linear(dim_input, hidden_channels)
        self.lin2 = Linear(hidden_channels, dim_output)
        self.dropout = 0.5
        self.optimizer = torch.optim.Adam(self.parameters(),
                                          lr=0.01,
                                          weight_decay=5e-4)

    def forward(self, x):
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin1(x)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin2(x)
        return F.log_softmax(x, dim=1)
```

3.2 Modelo Graph Convolucional Network – GCN

Este modelo consta de cinco capas de convolución de grafos (GCNConv), con las siguientes características:

- La primera capa transforma las características de entrada de tamaño dim_input a 32 dimensiones.
- Las siguientes dos capas intermedias mantienen un tamaño de 32 dimensiones, luego la cuarta capa reduce las dimensiones a 16 y la quinta capa a 8, finalizando con una capa de salida que ajusta las dimensiones al tamaño dim_output.
- Cada capa de convolución de grafos está seguida por una función de activación ReLU, que introduce no linealidad y ayuda al modelo a aprender representaciones más complejas de los datos de entrada.
- El modelo utiliza el optimizador Adam, configurado con una tasa de aprendizaje de 0.01 y un decaimiento de peso de 5e-4, para actualizar los parámetros de manera eficiente y prevenir el sobreajuste.
- La salida final del modelo incluye tanto las características procesadas como las probabilidades logarítmicas normalizadas usando log_softmax.

```python
class GCN(torch.nn.Module):
  def __init__(self, dim_input, dim_output):
      super().__init__()
      torch.manual_seed(42)
      self.conv1 = GCNConv(dim_input, 32)
      self.conv2 = GCNConv(32, 32)
      self.conv3 = GCNConv(32, 16)
      self.conv4 = GCNConv(16, 8)
      self.conv5 = GCNConv(8, dim_output)
      self.optimizer = torch.optim.Adam(self.parameters(),
                                        lr=0.01,
                                        weight_decay=5e-4)

  def forward(self, x, edge_index):
      x = self.conv1(x, edge_index)
      x = F.relu(x)
      x = self.conv2(x, edge_index)
      x = F.relu(x)
      x = self.conv3(x, edge_index)
      x = F.relu(x)
      x = self.conv4(x, edge_index)
      x = F.relu(x)
      x = self.conv5(x, edge_index)
      return x, F.log_softmax(x, dim=1)
```

3.3 Modelo GATv2Conv

Este modelo consta de dos capas de convolución de grafos basadas en atención (GATv2Conv), con las siguientes características:

- La primera capa transforma las características de entrada de tamaño dim_input a dim_hidden utilizando múltiples cabezas de atención (por defecto, 8 cabezas), lo que permite al modelo captar relaciones complejas entre los nodos al enfocar diferentes partes del grafo.
- La salida de esta capa es luego procesada por una función de activación ReLU y un dropout con una probabilidad del 50% para prevenir el sobreajuste.
- La segunda capa de atención toma la salida procesada, ajustada a *dim_hidden* * heads, y la transforma a *dim_output* con una única cabeza de atención, preparando las características finales para la clasificación.
- El modelo utiliza el optimizador Adam con una tasa de aprendizaje de 0.01 y un decaimiento de peso de 5e-4, lo que facilita la actualización eficiente de los parámetros y ayuda a evitar el sobreajuste.
- Finalmente, la salida del modelo incluye tanto las características procesadas de los nodos como las probabilidades logarítmicas normalizadas utilizando log_softmax.

```python
class GAT(torch.nn.Module):
  def __init__(self, dim_input, dim_hidden,
               dim_output, heads=8, lr=0.01, wd = 5e-4):
    super().__init__()
    torch.manual_seed(42)
    self.gat1 = GATv2Conv(dim_input, dim_hidden, heads=heads)
    self.gat2 = GATv2Conv(dim_hidden*heads, dim_output, heads=1)
    self.optimizer = torch.optim.Adam(self.parameters(),
                                      lr=lr,
                                      weight_decay=wd)

  def forward(self, x, edge_index):
    x = self.gat1(x, edge_index)
    x = F.relu(x)
    x = F.dropout(x, p=0.5, training=self.training)
    x = self.gat2(x, edge_index)
    return x, F.log_softmax(x, dim=1)
```

3.4 Modelo Graph Attention Network convolucional (GATConv)

Este modelo consta de cinco capas de convolución basadas en atención (GATConv), así:

- La primera capa toma las características de entrada *dim_input* y las transforma en 32 canales de salida con 32 cabezas de atención.
- Cada capa subsiguiente reduce progresivamente el tamaño de las características mientras mantiene múltiples cabezas de atención: la segunda capa transforma a 16 canales con 16 cabezas, la tercera a 8 canales con 8 cabezas, la cuarta a 4 canales con 4 cabezas, y la quinta ajusta a la dimensión de salida *dim_output* sin múltiples cabezas.
- Cada capa de convolución está seguida por una función de activación ReLU para introducir no linealidad y permitir que el modelo aprenda representaciones más complejas.
- El modelo utiliza el optimizador Adam con una tasa de aprendizaje de 0.01 y un decaimiento de peso de 5e-4 para actualizar eficientemente los parámetros y evitar el sobreajuste.
- La salida final del modelo incluye tanto las características procesadas de los nodos como las probabilidades logarítmicas normalizadas usando *log_softmax*.

```python
class GATc(torch.nn.Module):
  def __init__(self, dim_input, dim_output):
    super().__init__()
    torch.manual_seed(42)
    self.conv1 = GATConv(in_channels=dim_input, out_channels=32, heads=32)
    self.conv2 = GATConv(in_channels=32 * 32, out_channels=16, heads=16)
    self.conv3 = GATConv(in_channels=16 * 16, out_channels=8, heads=8)
    self.conv4 = GATConv(in_channels=8 * 8, out_channels=4, heads=4)
    self.conv5 = GATConv(in_channels=4 * 4, out_channels=dim_output)
    self.optimizer = torch.optim.Adam(self.parameters(),
                                      lr=0.01,
                                      weight_decay=5e-4)

  def forward(self, x, edge_index):
    x = self.conv1(x, edge_index)
    x = F.relu(x)
    x = self.conv2(x, edge_index)
    x = F.relu(x)
    x = self.conv3(x, edge_index)
    x = F.relu(x)
    x = self.conv4(x, edge_index)
    x = F.relu(x)
    x = self.conv5(x, edge_index)
    return x, F.log_softmax(x, dim=1)
```

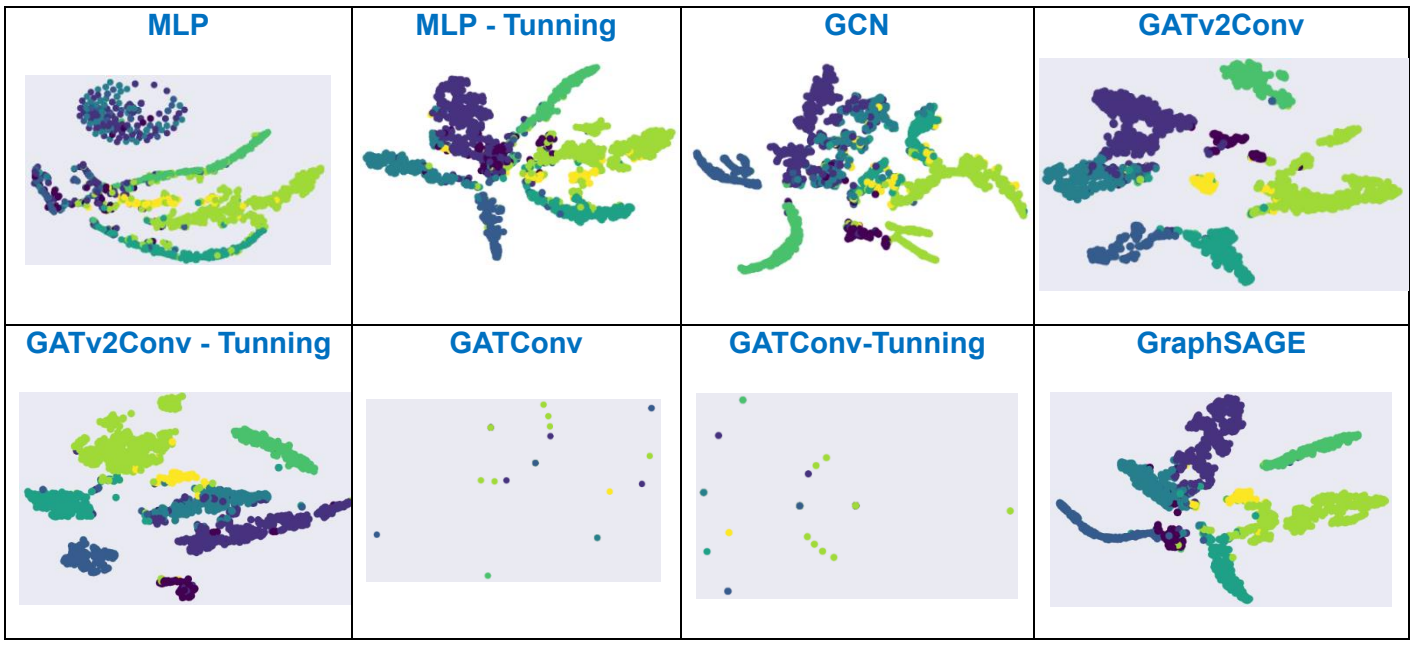
3.5 Modelo Graph Sample and Aggregate (GraphSAGE)

Este modelo es una variante de redes neuronales de grafos que permite realizar aprendizaje en nodos de un grafo de manera inductiva. Este modelo tiene la capacidad para generar embeddings para nodos no vistos durante el entrenamiento.

- Este modelo consta de dos capas de convolución GraphSAGE (SAGEConv). La primera capa transforma las características de entrada de los nodos (dim_input) en representaciones intermedias (dim_hidden) utilizando la agregación de las características de los nodos vecinos.
- La segunda capa toma estas representaciones intermedias y las convierte en las dimensiones de salida (dim_output), adecuadas para la tarea de clasificación.
- El uso de la función de activación ReLU y de dropout con una probabilidad del 10% entre las capas ayuda a introducir no linealidad y a prevenir el sobreajuste.
- El modelo está optimizado utilizando el algoritmo Adam con una tasa de aprendizaje de 0.01 y un decaimiento de peso de 5e-4.
- Se incluyo como parámetro de regularización un early stopping usando un parámetro de paciencia de 5.

3.6 Optimizacion de hiperparametros

Para todos los modelos base (MPL, GCN, GATv2Conv, GATConv, y GraphSAGE) se hizo un proceso de tunning a los hiperparámetros buscando la optimización de los modelos. Los principales hiperparamentros considerados en este ejercicio fueron: (i) Learning rate, (ii) Weight decay, (iii) Dim_hidden (iv) hidden_chanel. A continuación, se presenta la visualización del grafo con los resultados de clasificación en los modelos:



4 Evaluación

Para la construcción de cada uno de los modelos presentados en la sección 3, se evaluaron un modelo con hiperparámetros aleatorios y otro con hiperparámetros optimizados (grid search) con la intención de analizar la mejora cada modelo en comparación de su versión vanilla. En la mayoría de los modelos donde se aplicó optimización de hiperparámetros tienen un mejor desempeño comparativamente con aquellos en los que no se realizó optimización.

Los modelos trabajados reflejan los siguientes resultados:

#	Nombre del modelo	Accuracy
1	Perceptrón Multicapa (MLP)	43.52%
2	Perceptrón Multicapa (MLP) con tunning	85.76%
3	GCN (Graph Convolutional Network)	87.04%
4	GCN (Graph Convolutional Network) con tunning	87.04%
5	GATv2Conv	94.60%
6	GATv2Conv con tunning	94.96%
7	GATConv	25.17%
8	GATConv con tunning	25.17%
9	GraphSAGE	95.22%
10	GraphSAGE con tunning & early stopping	**94.03%**

Tabla 4 – Tabla de resultados en términos de accuracy para cada modelo de predicción

En el caso del modelo GATv2Conv se aplicó la optimización de hiperparámetros logrando una precisión de 94.96%, mientras que en el modelo GATConv la búsqueda de hiperparametros no proporcionó mejoría alguna obteniendo la misma baja calidad, además de resultar demasiado intensivo en cómputo ya que el tiempo de ejecución supera los seis minutos para 100 épocas. Finalmente, el modelo con mejor desempeño fue GraphSAGE utilizando los parámetros optimizados y regularizados con un accuracy de 94.03%:

```
GraphSAGE(
  (sage1): SAGEConv(745, 64, aggr=mean)
  (sage2): SAGEConv(64, 16, aggr=mean)
  (sage3): SAGEConv(16, 8, aggr=mean)
)
Epoch   0 | Train Loss: 2.195 | Train Acc:  15.94% | Val Loss: 2.13 | Val Acc: 18.21%
Epoch  10 | Train Loss: 0.740 | Train Acc:  68.27% | Val Loss: 0.78 | Val Acc: 68.09%
Epoch  20 | Train Loss: 0.391 | Train Acc:  83.93% | Val Loss: 0.47 | Val Acc: 83.73%
Epoch  30 | Train Loss: 0.280 | Train Acc:  89.53% | Val Loss: 0.35 | Val Acc: 89.27%
Epoch  40 | Train Loss: 0.180 | Train Acc:  91.94% | Val Loss: 0.32 | Val Acc: 88.27%
Epoch  50 | Train Loss: 0.176 | Train Acc:  92.42% | Val Loss: 0.42 | Val Acc: 88.16%
Early stopping triggered. No improvement in validation loss for 5 epochs.

GraphSAGE test accuracy: 94.03%

CPU times: user 1min 12s, sys: 793 ms, total: 1min 13s
Wall time: 1min 22s
```

Ilustración 5 - Resultados mejor modelo GraphSAGE

Se generó una matriz de confusión de clases para el mejor modelo “GraphSAGE” y se observa que la clase # 7 presentan el mayor número de casos con error de clasificación:

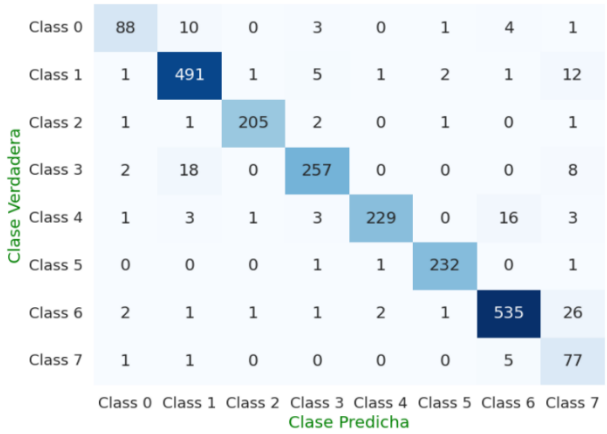


Ilustración 6 - GraphSAGE Matriz de Confusión con early stopping

5 Conclusiones

- Los modelos de mejor desempeño fueron GraphSAGE (94.03%) y GATv2Conv (94.96%) los cuales tienen la capacidad de capturar y procesar patrones complejos de los grafos y sus propiedades. En el caso de GraphSAGE al realizar un muestreo de la cantidad de vecinos es computacionalmente mas eficiente, o por otra parte, en los GATv2Conv se explota su arquitectura de apilamiento de múltiples capas permitiendo capturar la estructura local y global del grafo.

- El ajuste de hiperparámetros mejora significativamente la precisión del MLP, pero incluso con ajustes, su rendimiento es inferior al de los modelos basados en grafos, claramente debido a que los MLPs no están diseñados para manejar la conectividad y la estructura de los grafos y hay pérdida de información contextual. Los modelos GATv2Conv superan tanto a los MLP como a los GCN en términos de precisión. GraphSAGE tiene la mayor precisión entre todos los modelos probados, pero curiosamente, el uso de early stopping y el ajuste de hiperparámetros reducen ligeramente la precisión. Esto podría sugerir que el modelo vainilla ya estaba bien adaptado a los datos, o que los ajustes introdujeron cierta regularización que impactó negativamente la precisión.

- Dada la naturaleza del problema estudiado, tiene sentido que uno de los modelos con mejor desempeño sea un modelo basado en atención, considerando que estos son capaces de capturar patrones complejos en un grafo en el que existe co-ocurrencia de productos. En el contexto de la clasificación de productos, la co-ocurrencia de productos puede ser importante porque los productos que se compran juntos con frecuencia pueden tener características o propiedades similares, lo que podría ser útil para agruparlos en la misma categoría o clase. Por lo tanto, capturar y modelar estas relaciones de co-ocurrencia en el grafo puede ser crucial para el éxito de la tarea de clasificación de productos.
- Los productos que se compran juntos normalmente tienen reseñas parecidas, por lo que tiene sentido que el modelo GraphSAGE sea capaz de clasificar tan bien, puesto que este agrega las características de todos los vecinos a las del nodo original haciendo que no se generen diferencias grandes en grupos de nodos que son parecidos, pero sí se generen diferencias significativas entre nodos que se encuentren en un borde con otro grupo de nodos que no son tan parecidos.

En resumen, la atención global y la flexibilidad en la representación de relaciones en GraphSAGE, pueden ser especialmente beneficiosas para capturar patrones complejos en el grafo de co-ocurrencia de productos, como los del caso expuesto.

6 División de Tareas

Sonia Ramírez	Anguie García
• Búsqueda y exploración de diferentes datasets y análisis de viabilidad para el proyecto. • Análisis descriptivo de los datos sobre el dataset seleccionado. • Definición de métricas del grafo. • Análisis de resultados. • Elaboración de Presentación	Modelos: • Perceptrón Multicapa (MLP) • GCN (Graph Convolutional Network) • GATv2Conv • GATConv • GraphSAGE • Técnica para evitar sobreajuste y mejorar la generalización de modelo (Early stopping, Dropout). • Análisis de resultados.
Mili Galindo	**Lourdes Rodil**
• Descripción del dataset base seleccionado para el desarrollo del proyecto. • Descripción de los diferentes modelos evaluados en el proyecto para elaboración de informe. • Análisis de los resultados alcanzados en cada modelo para presentación de conclusiones en el informe. • Análisis de resultados. • Elaboración de Presentación	• Búsqueda y exploración de diferentes datasets y análisis de viabilidad para el proyecto. • Exploración y optimización de hiperparámetros de cada modelo • Análisis de resultados • Conclusiones