Proyecto Grupal 2

Graph Mahine Learning

Alejandro Calderon Moreno 1[a,c] , Nicolas Andres Rey Ospina 2[a,c] , Ana Carolina Rubiano Manrique 3[a,c], Sebastian Zequeda Molina 4[a,c]

Sergio Alberto Mora Pardo[b,c]

[a]*Estudiante de MAESTRIA EN ANALITICA PARA INTELIGENCIA DE NEGOCIOS*
[b]*Profesor, Departamento de Ingeniería Industrial*
[c]*Pontificia Universidad Javeriana, Bogotá, Colombia*

1. Contexto

En el entorno empresarial de Amazon, se ha reconocido la importancia de comprender las relaciones entre productos a través de la perspectiva visual, ya que los seres humanos tienden a desarrollar un sentido de estas conexiones basadas en la apariencia. La capacidad de entender cómo los productos se complementan o compiten entre sí es crucial para guiar las decisiones de compra de los clientes y mejorar su experiencia de compra en la plataforma.

Amazon se encuentra en la búsqueda de modelar esta percepción humana de las relaciones entre productos basados en la apariencia. A diferencia de enfoques detallados que dependen de anotaciones de usuarios, la estrategia de Amazon se basa en recopilar una gran cantidad de datos y desarrollar métodos escalables. Esto se enmarca como un problema de inferencia Big Data de redes definido en gráficos de imágenes relacionadas, y Amazon ha creado un conjunto de datos a gran escala para entrenar y evaluar su sistema.

El sistema que Amazon está desarrollando tiene la capacidad de recomendar qué productos y accesorios combinan bien visualmente, lo que mejora la experiencia de compra de los clientes en la plataforma. Esto no se limita a la ropa, sino que se extiende a una amplia gama de aplicaciones en el negocio, lo que lo convierte en una herramienta poderosa para aumentar la satisfacción del cliente y las oportunidades de venta cruzada.

Los modelos de grafos y los sistemas de recomendación desempeñan un papel crucial en el contexto de Amazon y su enfoque en comprender las relaciones entre productos y satisfacer las necesidades de sus clientes. Los modelos de grafos, como los presentados anteriormente, permiten representar visualmente las conexiones entre productos y usuarios, lo que facilita la comprensión de cómo se relacionan y complementan los elementos del catálogo de Amazon. Estos modelos de grafos sirven como la base sobre la cual se construyen los sistemas de recomendación. Al comprender las conexiones visuales entre productos, Amazon puede desarrollar sistemas inteligentes que recomiendan productos complementarios o alternativas a los clientes, mejorando así su experiencia de compra y aumentando la satisfacción del cliente.

La recomendación de productos basada en el interés que el comprador ha mostrado por otros productos es una tarea de aprendizaje automático importante para cualquier e-commerce dado que es una forma sacar provecho de los grandes volúmenes de datos que acarrea el tener una tienda virtual para brindar un servicio personalizado y competitivo [1]. Algunos beneficios para el comercio son:

- Aumento de las ventas: Si los compradores ven productos adicionales que les interesan, es más probable que agreguen más elementos a su carrito de compra.
- Mejora de la experiencia del cliente: Ayudar a los clientes a encontrar lo que están buscando de manera más eficiente.
- Aumento del valor promedio de la orden: Al alentar a los clientes a comprar productos adicionales o de mayor valor.

- Fidelización de clientes: Ofrecer recomendaciones relevantes puede mejorar la retención de clientes, lo cual se podría evidenciar en compras futuras.
- Competitividad: En un mercado en línea altamente competitivo, las tiendas virtuales que ofrecen recomendaciones de productos pueden destacarse y proporcionar un valor agregado a los clientes, lo que puede ayudar a retener y atraer a nuevos compradores.
- Recopilación de datos valiosos: Se puede almacenar preferencias de los clientes y calcular el rendimiento de los productos. Información útil para la estrategia de marketing y el inventario.

Los datos en un e-commerce suelen tener una estructura de grafo natural. Los productos, usuarios, transacciones y las relaciones entre ellos se pueden representar como nodos y enlaces en un grafo. Utilizar un modelo de grafos permite una representación efectiva de estos datos, lo que facilita la captura de patrones y relaciones complejas.

La tarea de Link Prediction se centra en predecir las conexiones futuras en el grafo. En un e-commerce, esto puede traducirse en predecir qué productos serán comprados por un usuario en el futuro o identificar nuevas relaciones que pueden ser relevantes para el negocio, como la recomendación de productos que nos son inmediatamente evidentes.

2. Análisis Exploratorio de Datos

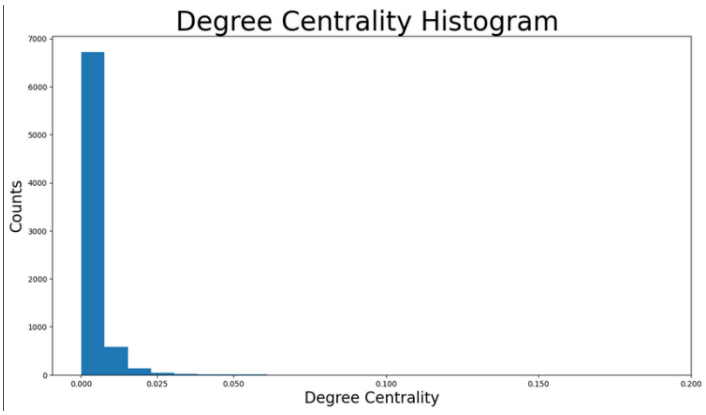
En una primera instancia se conoció la información general del grafo, analizando y entendiendo las propiedades básicas del grafo en cuestión. Donde se encuentra que tenemos 7650 nodos (Productos disponibles en Amazon y los enlaces entre nodos representan productos que son comprados en conjunto frecuentemente), cada uno con 745 features diferentes y 8 clases diferentes de productos disponibles. De igual manera, es importante conocer los tipos de enlaces y nodos que contiene el grafo, para el caso de este se tiene lo siguiente:

- No se cuentan con enlaces dirigidos
- Se cuentan con nodos aislados, 115 nodos aislados
- Y por ultimo si se tienen selfloops, el cual si tiene este grafo.

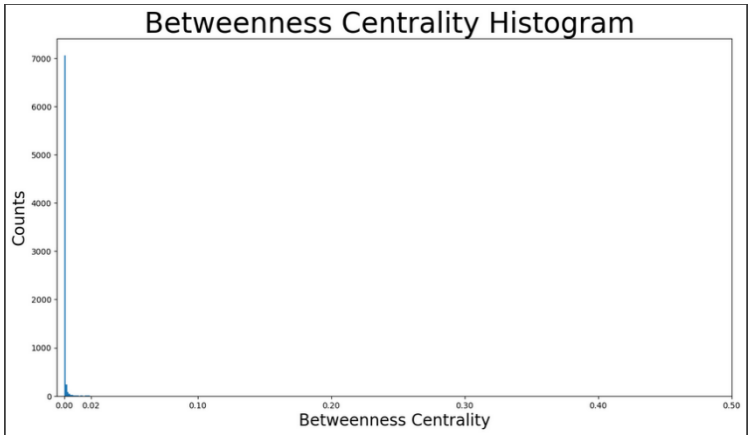
```
# Print information about the dataset
print(f'Dataset: {dataset}')
print('---------------')
print(f'Numero de grafos: {len(dataset)}')
print(f'Cantidad de Nodos: {data.x.shape[0]}')
print(f'Cantidad de features: {dataset.num_features}')
print(f'Cantidad de clases: {dataset.num_classes}')

# Print information about the graph
print(f'\nGraph:')
print('------')
print(f'Se tienen links dirigidos: {data.is_directed()}')
print(f'Grafo tiene nodos aislados: {data.has_isolated_nodes()}')
print(f'Cantidad de nodos aislados: {node_degrees[0]}')
print(f'Cantidad de nodos con grado = 1: {node_degrees[1]}')
print(f'Grafo tiene self-loops: {data.has_self_loops()}')
```

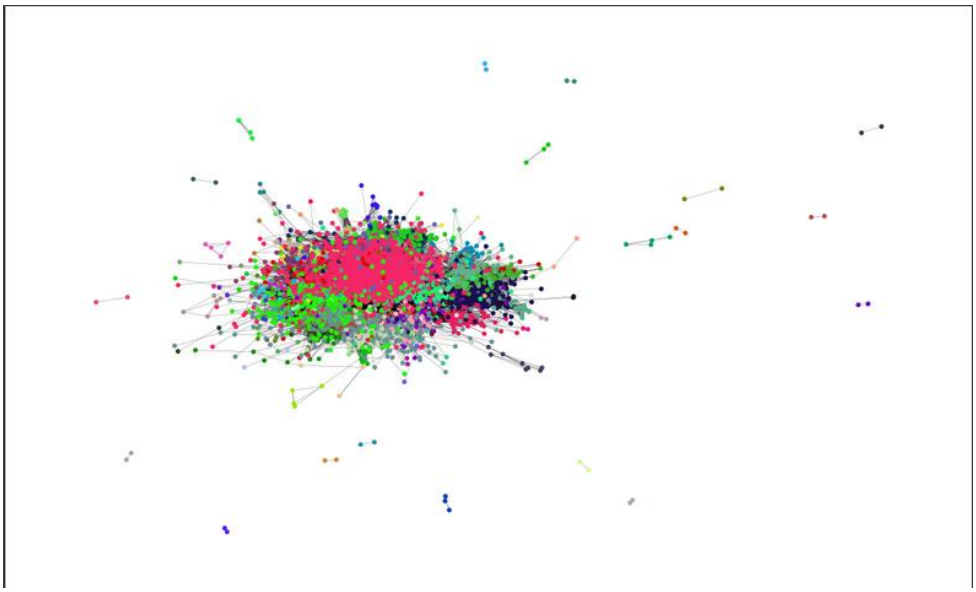
Además del análisis descriptivo inicial, se llevaron a cabo estudios de centralidad en el grafo, proporcionando una comprensión más profunda de la dinámica y estructura de este.



De acuerdo con la gráfica, la mayoría de los nodos tienen una centralidad de grado baja y pocos nodos tienen una centralidad de grado alta. En otras palabras, la mayoría de los nodos están conectados a un número reducido de otros nodos, mientras que solo unos pocos nodos están altamente conectados.



El histograma de centralidad intermedia identifica que la mayoría de los nodos en la red tienen una baja centralidad de intermediación, mientras que solo unos pocos nodos tienen valores altos de esta métrica. Por tanto, la mayoría de los nodos no actúan frecuentemente como puentes en los caminos más cortos entre otros nodos. Adicional, pocos nodos que tienen un papel crítico en términos de conectividad, ya que muchos caminos pasan por ellos.



Finalmente, al llevar a cabo un análisis de comunidades, pudimos distinguir ciertas agrupaciones densamente formadas, particularmente aquellas en tonos verdes y rosados. Además, observamos numerosos nodos que mantienen conexiones exclusivas entre sí, sin establecer vínculos con aquellos situados en el centro del grafo.

3. Metodología Analítica

3.1 Modelos Planteados

- **Modelo 1- Graph Convolutional Network:** El modelo es un autoencoder de grafos simple que tiene como objetivo aprender una representación latente de los nodos en un grafo. El autoencoder de grafos consta de dos partes principales:
  Encoder (Codificador): Esta parte del modelo se encarga de mapear los nodos del grafo original a una representación latente de menor dimensionalidad. En este caso, el codificador consiste en dos capas de convolución de grafos (GCNConv) que reducen gradualmente la dimensionalidad de las características de los nodos.
  Decoder (Decodificador): La función del decodificador es reconstruir el grafo original a partir de la representación latente aprendida por el codificador. El decodificador toma la representación latente y calcula un puntaje para cada par de nodos, indicando la probabilidad de que haya una arista entre ellos. Se utiliza un producto escalar entre las representaciones latentes de los nodos para calcular estos puntajes.

```python
class SimpleGraphAutoencoder(torch.nn.Module):
    def __init__(self):
        super(SimpleGraphAutoencoder, self).__init__()
        #Activacion 2 capas convolucionales
        self.conv1 = GCNConv(in_channels=dataset.num_features, out_channels=128)
        self.conv2 = GCNConv(in_channels=128, out_channels=64)


class SimpleGraphAutoencoder(torch.nn.Module):
    def __init__(self):
        super(SimpleGraphAutoencoder, self).__init()
        self.conv1 = GCNConv(in_channels=dataset.num_features, out_channels=128)
        self.conv2 = GCNConv(in_channels=128, out_channels=64)

    def encode(self):
      #Convolution 1 data.x, nodos y vocabularios de los nodos, clase positiva
      #replica nodos positivos
        x = self.conv1(x=data.x, edge_index=data.train_pos_edge_index)
        x = x.relu()
        #Convolution 2
        return self.conv2(x=x, edge_index=data.train_pos_edge_index)

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=-1)
        return logits

    def decode_all(self, z):
      #Matriz de adyacencia del nuevo grafo= espacio latente * T de espacio latente
        prob_adj = z @ z.t()
        return (prob_adj > 0).nonzero(as_tuple=False).t()
```

Ilustración 1 Definición Modelo 1

- **Modelo Graph Sage:** Es un modelo de redes neuronales profundas para grafos que utiliza una función de incrustación (embedding) para seleccionar y combinar información de los vecinos de manera inductiva. Las principales caracteristicas son:

  Muestreo de vecinos: Dado que los nodos pueden tener un número variable de vecinos, GraphSAGE introduce una estrategia de muestreo para seleccionar un subconjunto fijo de vecinos para cada nodo.

  Agregación de vecinos: GraphSAGE define varias estrategias para agregar las características de los vecinos de un nodo. Algunos de los agregadores propuestos en el artículo original son:

- Mean Aggregator: Calcula la media de las características de los vecinos.
- LSTM Aggregator: Utiliza un LSTM (Long Short Term Memory) para secuenciar y agregar las características de los vecinos.
- Pooling Aggregator: Aplica una función no lineal seguida de un max-pooling.

Actualización del nodo: Una vez que se ha realizado la agregación, la incrustación agregada se combina con las características originales del nodo para generar una nueva incrustación para el nodo.

Aprendizaje por capas: Al igual que otras redes neuronales para grafos, GraphSAGE opera en múltiples capas. En cada capa, un nodo agrega información de sus vecinos más cercanos. Así, en capas sucesivas, un nodo tiene acceso a información de nodos más alejados en el gráfico.

Inductividad: Una vez que el modelo ha sido entrenado, puede generar incrustaciones para nuevos vértices que no estaban presentes en el conjunto de entrenamiento, usando solo sus características y la estructura local del gráfico.

- **Modelo Graph Attention:** es un modelo diseñado en para auto-codificar grafos utilizando (Graph Attention Networ). Este modelo consta de dos capas convolucionales que transforman las características de entrada del grafo para aprender representaciones de los nodos. Durante el entrenamiento, se minimiza una función de pérdida de entropía cruzada binaria, diferenciando enlaces positivos de enlaces negativos. Para evaluar el rendimiento del modelo, se utiliza tanto la precisión (Accuracy), basándose en las probabilidades de borde predichas en comparación con las etiquetas verdaderas. Para este modelo no se especifica el número de cabezas de atención por simplicidad, y se deja el valor predeterminado de 1 (Lo que significa que no hay múltiples cabezas operando en paralelo)

```
class GATv2GraphAutoencoder(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels1, hidden_channels2):        #Activacion 2 capa
        super(GATv2GraphAutoencoder, self).__init__()
        self.conv1 = GATv2Conv(in_channels, hidden_channels1)
        self.conv2 = GATv2Conv(hidden_channels1, hidden_channels2)

    def encode(self):
        x = self.conv1(x=data.x, edge_index=data.train_pos_edge_index)          #Convolution 1 dat
        x = x.relu()
        return self.conv2(x=x, edge_index=data.train_pos_edge_index)            #Convolution 2

    def decode(self, z, pos_edge_index, neg_edge_index):                        #Only pos and neg
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)        #Concatenate pos a
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=-1)              #dot product
        return logits

    def decode_all(self, z):
        prob_adj = z @ z.t()                                                    #get adj NxN Matri
        return (prob_adj > 0).nonzero(as_tuple=False).t()                       #get predicted edg
```

```
model, data = GATv2GraphAutoencoder(dataset.num_features, 128, 64).to(device), data.to(device)
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.01)
```

3.2 Métrica

Acurracy: El Accuracy en la tarea de link prediction mide la proporción de predicciones correctas (tanto verdaderos positivos como verdaderos negativos) en relación con el número total de predicciones realizadas.

$$\text{Accuracy} = \frac{Número\ de\ Predicciones\ Correctas}{Número\ Total\ de\ Predicciones}$$

Es importante destacar que, en el contexto de link prediction, el Accuracy se utiliza para evaluar qué tan bien el modelo es capaz de predecir correctamente los enlaces en el grafo.

4. Solución Propuesta

Para evaluar el rendimiento de los modelos mencionados en la sección 3.3, realizamos la siguiente partición de datos:

```
# use train_test_split_edges to create neg and positive edges
data.train_mask = data.val_mask = data.test_mask = data.y = None
#Retorna mismo dataset con argumentos diferentes
#train_pos_edge_index= clase positiva de los ejes,
#test_pos_edge_index ejes de test de clase positiva,
#train_neg_adj_mask matriz de adyacencia, esta sera la que se samplea
#data = train_test_split_edges(data)
print(data)
```

Ilustración 2. Partición de Datos

En las tablas 1 y 2 presentamos los resultados obtenidos. La Tabla 1 contiene las métricas de evaluación obtenidas al aplicar los modelos a los datos de test. Estas métricas nos brindan una comprensión del rendimiento de los modelos en los datos con los que se testeo el entrenamiento.

MODELO	ACCURACY
GRAPH CONVOLUTIONAL NETWORK	74,3%
GRAPH MODE SAGE	72,9%
GRAPH ATTENTION NETWORK	**75,20%**

Tabla 4: Métricas de Evaluación en Test
Negrilla mejor resultado.

La Tabla 2 presenta los resultados de la evaluación de los modelos en el conjunto de validación que representa datos independientes que los modelos no habían visto durante el entrenamiento. Estas métricas son cruciales para medir la capacidad de generalización de los modelos a nuevos datos.

MODELO	ACURACCY
GRAPH CONVOLUTIONAL NETWORK	74,1%
GRAPH MODE SAGE	73,2%
GRAPH ATTENTION NETWORK	**75,0%**

Tabla 5: Métricas de Evaluación en Validación.

Negrilla mejor resultado.

Todos los modelos superan el umbral del Accuracy del 70%, lo que indica un rendimiento prometedor en la tarea de evaluación. Sin embargo, es importante destacar que **GRAPH ATTENTION NETWORK** se destaca con el mejor desempeño tanto en test como en validación.

5. Conclusiones

A lo largo de este estudio de grafos, hemos implementado cuatro modelos de aprendizaje automático en Python, a saber: **Graph Convolutional Network**, **GRAPH MODE SAGE** y **GRAPH ATTENTION NETWORK**. Tras exhaustivas pruebas y evaluaciones, observamos que **GRAPH ATTENTION NETWORK** se destacó como el modelo más eficaz en nuestra tarea de link prediction. Este modelo obtuvo un impresionante valor de Accuracy del 75%, lo que confirma su excelente desempeño.

El Accuracy del 75% indica que **GRAPH ATTENTION NETWORK** es altamente competente en la capacidad establecer conexiones entre los diferentes productos que los usuarios de Amazon han comprado lo que se puede reflejar en un excelente sistema de recomendación para compras futuras.

Es importante destacar que si bien **GRAPH ATTENTION NETWORK** superó a los otros modelos en esta tarea específica, la elección del modelo adecuado puede depender en gran medida del problema y los datos específicos. En consecuencia, esta investigación proporciona una base sólida para futuros desarrollos y mejoras en el sistema de recomendación de productos a los usuarios de cualquier e-commerce.

6. Bibliografía

1. Julian McAuley, Christopher Targett , Qinfeng (‘Javen’) Shi , and Anton van den Hengel “Image-based Recommendations on Styles and Substitutes” 2015. Tomado de https://arxiv.org/pdf/1506.04757.pdf
2. Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, Stephan Günnemann “Pitfalls of Graph Neural Network Evaluation” Technical University of Munich, Germany 2019. Tomado de https://arxiv.org/pdf/1811.05868.pdf
3. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *arXiv (Cornell University)*. https://arxiv.org/pdf/1706.02216.pdf