MSc Data Science AUEB

Data Science Challenge

PRODUCT CLASSIFICATION CHALLENGE

Professor: Ioannis Nikolentzos

Giagkos Stylianos | f3352410

Kazantzis Gerasimos| f3352406

Vougioukos Dimitris | f3352411

Table of Contents

1. Introduction

The rapid growth of e-commerce platforms has led to an explosion in the number of products available online, necessitating efficient and accurate methods for product organization and categorization. Assigning products to predefined categories is a fundamental challenge in retail, with significant implications for managing product hierarchies, enhancing product taxonomy, and improving user experience through better search and recommendations. This paper addresses a real-world classification problem focused on categorizing sports products available on the Amazon retail website.

The core task of this project is to study and apply machine learning and data mining techniques to assign each product sample into one of sixteen predefined categories (classes), such as cycling products or golf products. This challenge is unique in its leveraging of a multi-modal dataset, comprising both a graph, representing co-viewing relationships between thousands of sports products, and their associated textual descriptions. The problem is closely related to the well-studied fields of **text categorization** and **node classification**. Our objective is to build a robust model for predicting the class label of each product by learning parameters from a collection of training products with known class information, and then applying this knowledge to predict the categories of unlabeled products. This challenge is hosted on **Kaggle**, a prominent platform for predictive modeling competitions. The performance of our models will be rigorously evaluated using the **multi-class logarithmic loss** measure. The subsequent sections of this paper will detail the problem, dataset, methodology, experimental results, and concluding remarks with future work.

2. Problem Description and Dataset Analysis

2.1 Detailed Problem Statement

The central objective of this project is to accurately classify sports products from the Amazon retail website into one of 16 distinct categories. This is a supervised multi-class classification problem where each product is a sample, and the goal is to predict its correct class label. The challenge specifically advises the integration of both graph-theoretical and textual information to build effective predictive models.

2.2 Dataset Description

The methods developed in this project are evaluated on a comprehensive dataset of sports products, provided through several distinct files. The dataset is accessible via a Dropbox link: https://www.dropbox.com/scl/fo/5zeei61e9ljyxxzfz9sjz/AI0dDM4cHThws0jocGLE2NM?rlkey=z104yubjou8a0ibieom3she8d&dl=0

The dataset consists of the following files:

1. **edgelist.txt**: This file represents an undirected graph where **nodes correspond to Amazon sports products**. An edge between two products signifies that they are frequently co-viewed by users. The graph is substantial, comprising **276,453 vertices** (products) and **1,811,087 edges** in total.
2. **descriptions.txt**: This file contains the **textual content** for all 276,453 products. Each row includes a product ID, followed by its title and description (if available). The product ID is separated from the textual content by the string | = |.
3. **price.txt**: This file provides price information for **198,817 products**. Each row contains a product ID and its corresponding price, separated by a comma (,).
4. **y_train.txt**: This file contains the **class labels for 182,006 training samples**. Each row lists a product ID and its assigned class label, separated by a comma (,). These samples correspond to nodes within the provided graph.
5. **test.txt**: This file contains the **IDs of 45,502 products** for which class labels need to be predicted. These products also belong to one of the 16 available categories, and the final evaluation of the developed methods will be performed on these predictions.

2.3 Data Preparation and Preprocessing for Product Data

Data Ingestion and Structuring

The process begins by **reading raw data** from text files.

- **Product prices** are extracted from price.txt, converted into a DataFrame, and saved as product_prices.xlsx.
- **Product descriptions** from description.txt are parsed into a DataFrame. These descriptions then undergo a crucial cleaning process before being saved to product_descriptions.xlsx.
- **Training labels** (y_train.txt) and **test product IDs** (test.txt) are similarly extracted and formatted into DataFrames. All product IDs are consistently converted to integer types to facilitate accurate merging.

The Product Description Cleaning Function

To standardize and clean the raw product descriptions—often noisy due to web scraping or inconsistent formatting—we developed a preprocessing routine that uses regular expressions and HTML decoding. This ensures that the textual data is consistent and free from artifacts, making it more suitable for downstream tasks such as feature extraction and machine learning.

Cleaning methodology is presented below:

- **Decode HTML Entities**: Converts HTML character codes (like & for & or < for <) into their actual characters. This ensures proper interpretation of text, especially when sourced from web pages.
- **Remove HTML Tags**: Strips out all HTML tags (e.g., , <p>, <b>) using regular expressions. Replacing them with a space prevents words from being unintentionally merged.
- **Eliminate Non-Printable Characters**: Removes invisible or control characters from the ASCII range that may interfere with text processing or display.
- **Normalize Whitespace**: Replaces newline (\n), carriage return (\r), and tab (\t) characters with single spaces and then collapses multiple spaces into one to maintain clean and consistent formatting.
- **Trim Whitespace**: Removes any leading or trailing spaces to ensure that the cleaned text starts and ends properly.

Dataset Assembly and Graph Creation

The prepared data is then integrated to form complete datasets and define product relationships.

- **Training and Test Datasets** are created by **merging** the cleaned product descriptions with their respective training labels or test product IDs. This ensures each description is paired with its relevant information, forming training_dataset.xlsx and test_dataset.xlsx.

- Finally, **product relationships** from edgelist.txt are processed. For every connection between two products, a reciprocal connection is added, resulting in an **undirected graph** that's saved as undirected_products_graph.txt.

This systematic approach transforms raw, disparate data into clean, structured, and interconnected datasets, ready for advanced analytical tasks.

2.4 Data Characteristics

Distribution of Product Prices

The dataset contains **16 distinct sports product categories**. A significant characteristic of the training dataset is its **high class imbalance**; the number of samples per class ranges widely from **1,129 - 43,260**. This imbalance requires careful consideration during model training to prevent bias towards majority classes. The presence of both graph structure (co-viewing patterns) and rich textual descriptions (titles and descriptions) offers diverse feature extraction opportunities, while price information serves as an additional feature source— though it is less comprehensive or consistently available than the others.

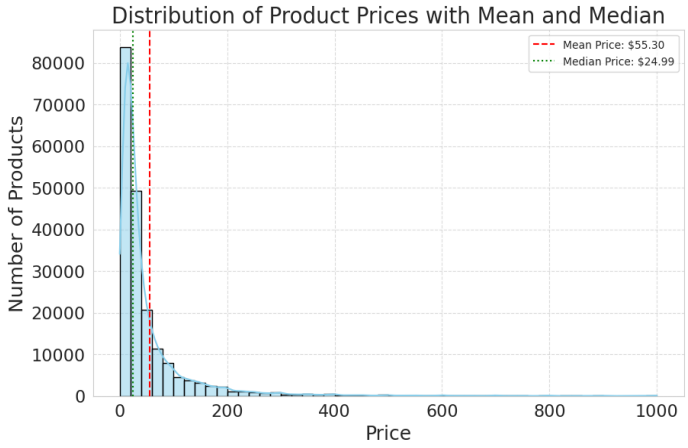

Figure 1: Distribution of Product Prices

The histogram in Figure 1 shows a **highly right-skewed** distribution of product prices. Most products are concentrated at **lower price points (between $0 and $100)**, with very few items priced above $400-$500.

The **mean price is $55.30**, while the **median price is $24.99**. This significant difference highlights the skewness, as the mean is pulled higher by the few expensive products. The **median is a better indicator of the typical product price** due to the influence of these high-priced "outliers." This distribution suggests a business model with a large volume of inexpensive products alongside a smaller number of premium items.

Product Price Distribution by Label

Across nearly all labels, there's a consistent presence of **high-priced outliers**, reaching up to $1000. While all labels show a wide price range, the **typical price range** (boxes and whiskers) **and median prices vary**, generally falling between $10 and $100.

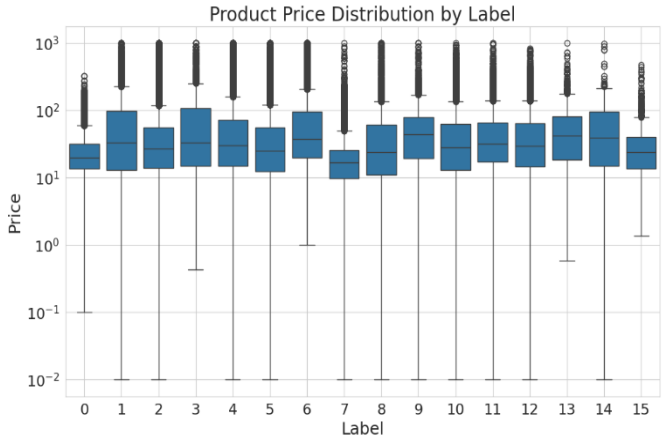

Figure 2: Product Price Distribution by Label

Outliers Are Abundant

- Every category shows many outliers, mostly on the high-price side, indicating the presence of a few very expensive products.
- This suggests a heavy-tailed distribution, which is typical in e-commerce datasets.

Medians Vary Across Categories

- Median price (horizontal line inside each box) is generally around $20–$40, but:
- Label 7 has a notably lower median (~$10).
- Label 9 and 13 show slightly higher medians than the rest.

IQR Differences

- The Interquartile Range (IQR) (the width of each box) varies, indicating that some categories (e.g., 1, 3, and 14) have more price variability than others.
- Label 7 again stands out with a narrower IQR, suggesting less price diversity in that category.

Minimum Values

- On the logarithmic y-axis, we see that 11 out of 16 categories have products priced as low as $0.01 (visible as the bottom whisker or outliers touching 10^{-2}).
- The remaining 5 categories (likely labels such as 1, 4, 7, 13, and 15) have higher minimum prices, suggesting that extremely low-priced products are not present in those classes.
- This indicates that some product categories inherently exclude very low-cost items, potentially due to their nature (e.g., equipment vs accessories).

Degree Distribution (Log-Scale)

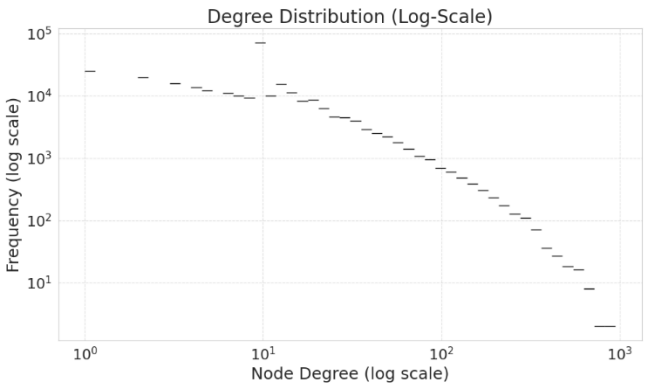

Figure 3: Degree Distribution (Log-Scale)

The plot (Figure 3) shows a clear decreasing trend: as the node degree increases, the frequency of nodes with that degree rapidly declines. In the mid-range (approximately between degrees 10 and a few hundred), the data follows a roughly downward trajectory on the log-log scale, though not perfectly linear. This suggests the degree distribution may exhibit heavy-tailed behavior, possibly resembling a truncated power-law or log-normal distribution, rather than a pure power-law.

At the lower end of the spectrum (degrees 1 to 5), there is a high concentration of nodes with very few connections — a common feature in real-world networks. At the upper end (degrees approaching 1000), the number of nodes drops sharply, indicating that high-degree "hub" nodes are rare, though present.

2.5 Evaluation Metric

The performance of the classification models will be assessed using the **multi-class logarithmic loss (LogLoss)**, a standard metric for probabilistic classifiers. LogLoss penalizes incorrect classifications, especially those made with high confidence. It is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij})$$

Where:

- N: Total number of samples (products) being evaluated.
- C: Total number of classes, which is 16 in this challenge.
- y_{ij}: Binary indicator, equal to 1 if sample i truly belong to class j and 0 otherwise.
- p_{ij}: Predicted probability that sample i belong to class j.

LogLoss indicates model performance; a lower value means predicted probabilities are closer to true labels. It's particularly sensitive to confident, incorrect predictions, making it a robust metric for probabilistic outputs.

3. Models Methodology

3.1 BERT Embeddings

The BERT + LoRA model architecture described here was developed to extract rich semantic representations from product descriptions for use in the classification task. This model focuses exclusively on leveraging textual information, drawing on the power of transformer-based language models to capture contextual meaning from raw product text.

At the core of this approach is **DistilBERT**, a lightweight, distilled version of BERT that retains most of BERT's language understanding capabilities while being computationally more efficient. The input to the model consists solely of product descriptions, which are tokenized using the pre-trained DistilBERT tokenizer. Each description is truncated or padded to a maximum length of 128 tokens, with attention masks indicating valid tokens in each input sequence.

To enhance the efficiency and flexibility of fine-tuning, we integrated **Low-Rank Adaptation (LoRA)** into the architecture. LoRA introduces trainable rank decomposition matrices into the attention mechanism of the transformer, specifically targeting the query (q_lin) and value (v_lin) projection layers. This allows the base model to remain largely frozen while learning a small number of task-specific parameters, significantly reducing the number of trainable weights and computational cost. The configuration we used employed a rank of 8, a scaling factor (α) of 32, and a dropout of 0.05 within the adapted modules.

To address the issue of label imbalance across the 16 product categories, we introduced **class-weighted loss** into the training pipeline. This was implemented through a custom WeightedTrainer module that wraps the standard model and modifies the forward pass to compute a class-weighted cross-entropy loss, giving higher importance to underrepresented classes.

The model was trained using Hugging Face's Trainer API with early stopping and evaluation performed at the end of each epoch. We monitored classification metrics including accuracy, precision, recall, and F1-score. Throughout training, both training and validation losses were logged to visualize learning dynamics and detect overfitting.

Once training was complete, we repurposed the model for **embedding extraction** by discarding the classification head and retaining only the base DistilBERT encoder. The [CLS] token embedding from the final hidden layer was used to represent each product description as a dense 768-dimensional vector. These embeddings were extracted in batches and saved for integration with graph-based models.

This BERT + LoRA pipeline served as a strong text-only baseline. It demonstrated that even without graph structure or price metadata, deep semantic representations derived from natural language could offer significant predictive power in the product classification task. Moreover, the use of LoRA provided a scalable and efficient fine-tuning mechanism that enabled fast experimentation and adaptation to the challenge's specific domain.

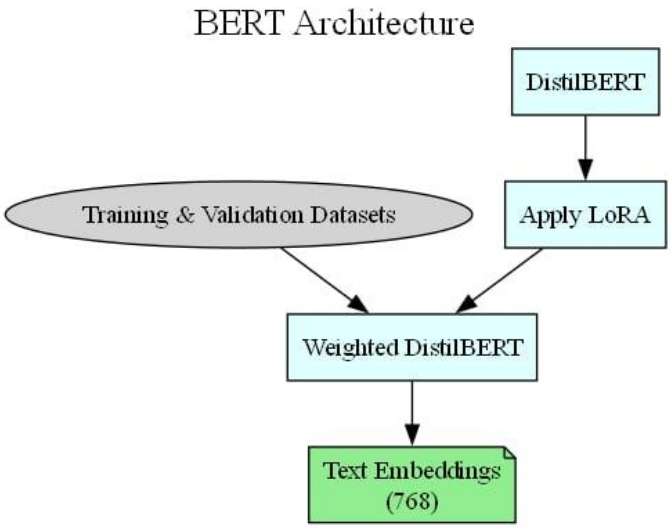

Figure 2: BERT Architecture

3.2 GAT

The Graph Attention Network (GAT) model described here was developed for the product classification task to combine textual, numerical, and graph-based relational information within a unified framework. In this setting, each node in the graph corresponds to a product, and edges represent co-view relationships — i.e., instances where two products were frequently viewed together by users. This graph structure enables the model to exploit relational patterns among products that extend beyond their standalone features.

For node features, we used a combination of semantic embeddings and numerical price data. Specifically, textual embeddings were extracted from a pretrained language model, combined with normalized product prices, resulting in a comprehensive input vector for each node. These vectors were used as node features in the GAT.

The GAT model itself consists of multiple layers. The first layer employs multi-head attention with several attention heads, generating concatenated feature vectors to enhance representation capacity. This attention mechanism allows the model to dynamically weigh the importance of neighboring nodes when aggregating information. Following this layer, Batch Normalization, ELU activation, and Dropout are applied for improved training stability and regularization. Subsequent hidden layers use single-head attention to refine learned features while maintaining computational efficiency. The final GAT layer outputs logits corresponding to the number of classification categories.

In parallel to the main GAT stream, we introduce a skip connection: the original input vector is projected directly to the output dimension through a linear layer and is added to the final GAT output. This skip connection helps preserve raw feature information and facilitates better gradient flow during training.

The final output serves as logits for a softmax classifier, producing the predicted class for each product. This GAT-based architecture effectively integrates textual, numeric, and relational signals and demonstrated strong performance. It provides valuable insights into how attention mechanisms can be leveraged for product classification within graph-based learning pipelines.

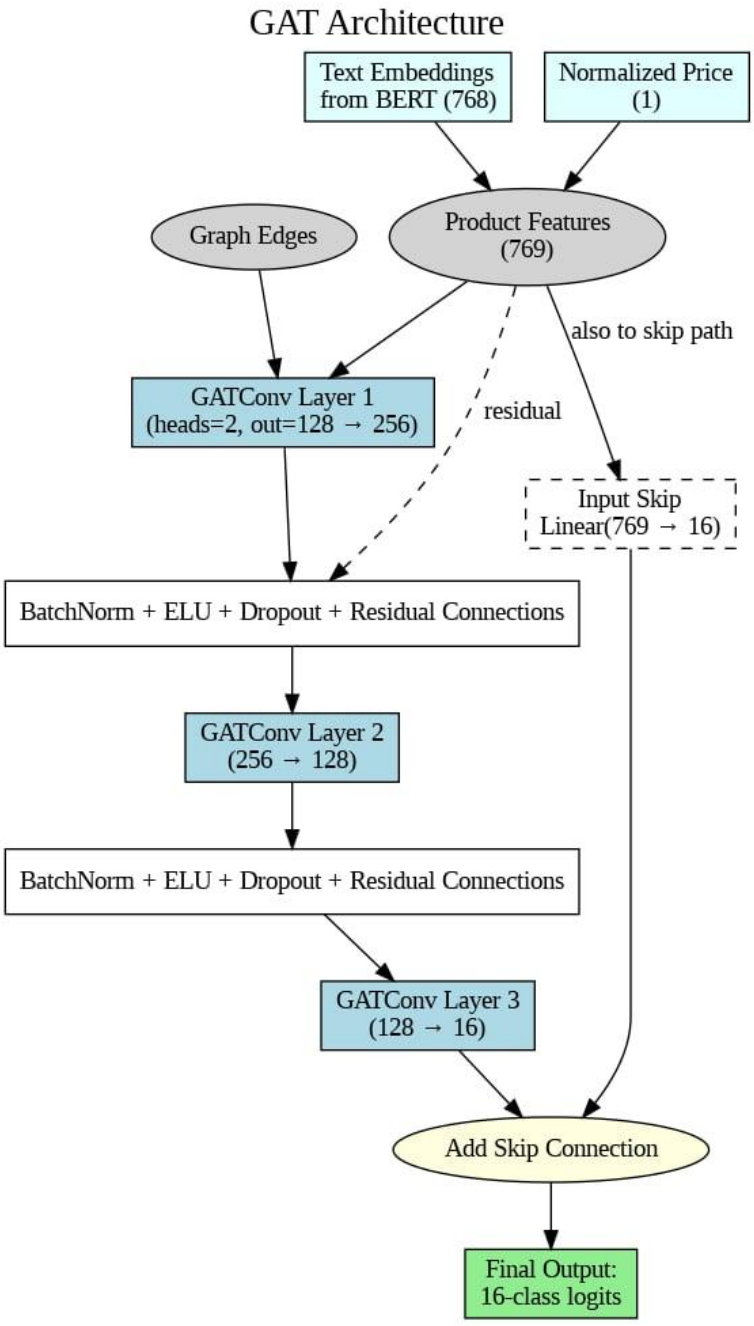

Figure 3: Flexible GAT Architecture

3.3 GraphSAGE

The GraphSAGE model implemented here serves as powerful graph neural network architecture for the product classification task, leveraging the relational structure of products co-viewed by users. In this formulation, each node in the graph corresponds to a product, and edges capture co-view patterns between products, enabling the model to aggregate and propagate information across connected nodes beyond the individual product features.

Node features are derived from a combination of text embeddings obtained from the implementation described in paragraph 3.1 and numeric attributes which are the product prices, resulting in an input feature vector that represents each product comprehensively. These feature vectors are then fed into the GraphSAGE network, which applies an inductive aggregation mechanism to learn meaningful node embeddings by sampling and aggregating feature information from each node's local neighborhood.

The GraphSAGE architecture consists of three consecutive layers. Each layer performs neighborhood aggregation through the SAGEConv operation, which computes the representation of a node by combining its own features with aggregated features from its neighbors. This strategy enables efficient learning on large graphs and better generalization to unseen nodes.

For the first two layers, the output dimensions are set to 256, allowing the network to capture high-dimensional intermediate feature representations. Each of these layers is followed by batch normalization, ReLU activation, and dropout regularization to stabilize training and prevent overfitting. Additionally, residual connections are incorporated whenever the input and output dimensions match, which helps preserve gradient flow and mitigates the vanishing gradient problem during deep training.

The final layer reduces the representation dimension to 16, corresponding to the number of classification categories, and outputs raw logits without activation or normalization, directly used for classification with cross-entropy loss.

The training procedure involves stratified 5-fold cross-validation, where for each fold the model is trained with early stopping based on validation loss, Adam optimization and weight decay regularization. After training across folds, predictions on the test set are averaged to produce robust final class probabilities.

This GraphSAGE-based approach demonstrates an effective and scalable method to incorporate graph neighborhood information for product classification, outperforming earlier attention-based models by leveraging generalized neighborhood aggregation and residual connections to improve feature learning and classification accuracy.

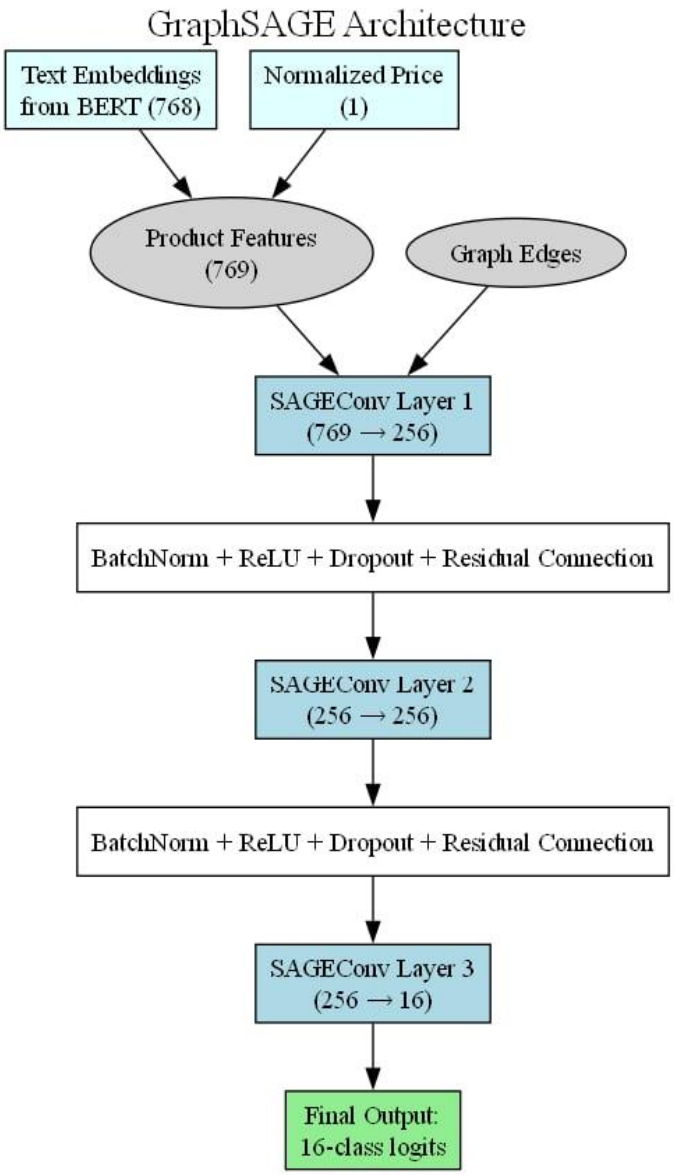

Figure 4: Graph SAGE Architecture

3.4 Ensembled Predictions

To further enhance classification robustness and leverage the complementary strengths of different graph-based models, we implemented a simple ensemble strategy by averaging the predicted class probabilities from the GraphSAGE and Graph Attention Network (GAT) models. Specifically, for each test product, we combined the softmax output probabilities produced by the 5-fold averaged GraphSAGE submission and the corresponding GAT submission, calculating the element-wise means of their class probability distributions.

This probability-level fusion allows the ensemble to integrate diverse learned representations and decision boundaries from both models, which can lead to improved predictive accuracy and stability compared to relying on a single model's output.

4. Experiments and Results

4.1 BERT Training Experiment

To train the **BERT + LoRA** model for product description classification, we adopted a fine-tuning approach on top of a pretrained **DistilBERT** model. Our goal was to extract high-quality semantic embeddings while ensuring efficient and balanced training across all 16 product categories. To this end, we configured a set of training hyperparameters and regularization strategies aimed at optimizing both performance and computational efficiency.

We fine-tuned the model using the Adam optimizer with a learning rate of 0.0002 and a batch size of 32, training for up to 10 epochs with early stopping (patience = 3) and checkpointing after each epoch. Weight decay (0.01) was applied to reduce overfitting. To handle class imbalance, we used scikit-learn's compute_class_weight and a custom WeightedTrainer to ensure balanced gradient updates. Performance was tracked using weighted F1-score, accuracy, precision, and recall.

A key feature of our approach was the application of **Low-Rank Adaptation (LoRA)**, which enabled us to inject trainable parameters into just the attention layers of the DistilBERT model. Specifically, LoRA was applied to the q_lin and v_lin projections with a rank r=8, an alpha scaling factor of 32, and a dropout of 0.05. This reduced the number of trainable parameters while still enabling effective domain-specific adaptation.

Throughout training, we tracked both **training and validation losses**, which showed a consistent downward trend before stabilizing. The early stopping criterion was triggered before reaching the maximum epoch count, indicating good generalization and helping to avoid overfitting.

The final model achieved strong validation metrics, demonstrating that even with a lightweight transformer and minimal fine-tuning via LoRA, it is possible to capture the semantic richness of product descriptions for accurate multi-class classification.

The training process is visualized below in a plot comparing training and validation loss across epochs. The training loss decreases consistently, while the validation loss initially follows a similar trend before plateauing with slight fluctuations. Despite this, the two curves remain relatively close, indicating a stable optimization process with no significant overfitting. The best-performing model was saved automatically and later used to extract high-dimensional embeddings from the full product description dataset.

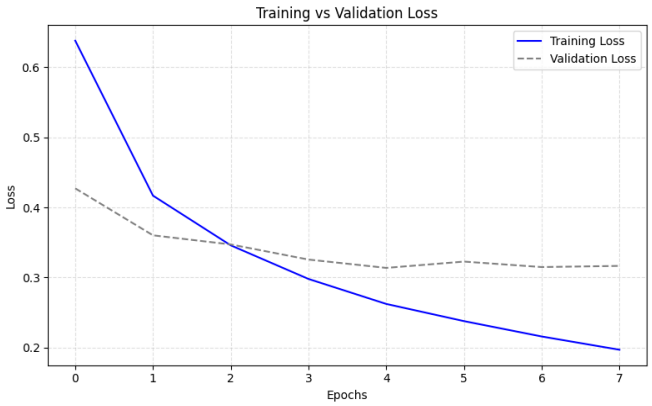

Figure 7: Training and Validation Losses for Finetuned BERT

The detailed **classification report** based on the validation set (20% split) is presented below:

Class	Precision	Recall	F1-score	Support
0	0.9636	0.969	0.9663	3033
1	0.8564	0.8925	0.8741	2372
2	0.9414	0.8484	0.8925	8652
3	0.925	0.9655	0.9448	1073
4	0.8868	0.9347	0.9101	3016
5	0.9323	0.9313	0.9318	3565
6	0.9007	0.9375	0.9187	1519
7	0.9562	0.9198	0.9376	3752
8	0.9263	0.946	0.9361	1316
9	0.9181	0.9181	0.9181	903
10	0.8081	0.8426	0.825	3589
11	0.8929	0.9011	0.897	1425
12	0.6957	0.8794	0.7768	1318
13	0.7955	0.8793	0.8353	323
14	0.9008	0.9646	0.9316	226
15	0.8739	0.9313	0.9017	320
Accuracy			0.9007	36402
Macro Avg	0.8859	0.9163	0.8998	36402
Weighted Avg	0.9044	0.9007	0.9014	36402

Table 1: BERT Classification Report

The results indicate strong overall model performance, with high precision and recall for most classes, reflecting accurate and reliable predictions. Some variation exists, such as lower precision in class 12, which suggests occasional false positives. The overall accuracy of about 90% and balanced macro and weighted averages demonstrate that the model generalizes well across different classes

4.2 GAT Training Experiment

To train the Graph Attention Network (GAT) model, we conducted a series of controlled experiments and selected hyperparameters that offered a good trade-off between model complexity and generalization. The network was trained using the Adam optimizer with a learning rate of 0.0005, which provided stable convergence over long training runs. We employed a maximum of 6000 epochs to allow sufficient room for optimization but implemented an early stopping mechanism with a patience of 200 epochs. This means training was halted if the validation loss did not improve for 200 consecutive epochs, helping to prevent overfitting and unnecessary computation.

The model also included a dropout rate of 0.4 after each GAT layer, which acts as a regularization technique to reduce overfitting. Additionally, we used batch normalization and ELU (Exponential Linear Unit) activations to stabilize training and introduce non-linearity. For the GAT-specific settings, we used 3 layers with 2 attention heads in the first layer, and a hidden dimensionality of 128, resulting in a 256-dimensional output from the first layer due to head concatenation. Class imbalance was addressed by computing class weights using the compute_class_weight function from scikit-learn, and these weights were incorporated into the cross-entropy loss function to ensure fairer learning across all 16 classes.

This combination of architectural and training hyperparameters contributed to a balanced and efficient learning process.

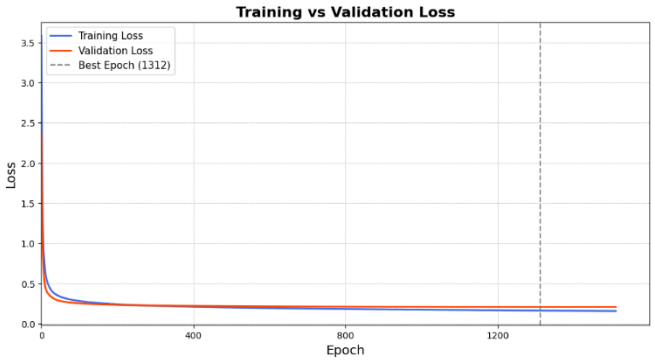

Figure 8: Training and Validation Losses

This plot illustrates the training progress of a machine learning model by showing **Training Loss** and **Validation Loss** against **Epochs**. Both losses decrease rapidly initially and then flatten out, indicating convergence. The **Best Epoch (1312)**, marked by a vertical dashed line, signifies the point where the validation loss was lowest, suggesting optimal generalization performance.

The close proximity of the training and validation loss curves throughout training indicates good generalization and minimal overfitting, making epoch 1312 an ideal stopping point for this model's training.

The detailed **classification report** based on the validation set is presented below:

Class	Precision	Recall	F1-score	Support
0	0.977	0.9796	0.9783	1517
1	0.9067	0.9182	0.9124	1186
2	0.9272	0.9505	0.9387	4326
3	0.956	0.9721	0.964	537
4	0.9516	0.9655	0.9585	1508
5	0.9723	0.9658	0.969	1782
6	0.9635	0.9368	0.95	760
7	0.9531	0.9424	0.9477	1876
8	0.9697	0.9726	0.9712	658
9	0.9529	0.9424	0.9476	451
10	0.8792	0.8724	0.8758	1794
11	0.9408	0.9157	0.9281	712
12	0.8854	0.8209	0.852	659
13	0.9338	0.8704	0.901	162
14	1	0.9558	0.9774	113
15	0.9742	0.9437	0.9587	160
Accuracy			0.9389	18201
Macro Avg	0.9465	0.9328	0.9394	18201
Weighted Avg	0.9389	0.9389	0.9388	18201

Table 2: Flexible GAT Classification Report

The model shows strong overall performance with a high accuracy of 93.89% and consistently high precision and recall across most classes. While classes like 10, 12 and 13 show slightly lower scores, indicating areas for improvement, the macro and weighted F1-scores above 0.93 confirm reliable and balanced performance across all categories.

4.3 GraphSAGE Training Experiment

To train the GraphSAGE model, we conducted a series of systematic experiments to identify hyperparameters that effectively balance model expressiveness and generalization performance. The training utilized the Adam optimizer with a learning rate of 0.0005 and a weight decay of 1e-4 to promote stable convergence and control overfitting. We allowed a maximum of 6000 training epochs, providing ample opportunity for optimization, while incorporating an early stopping mechanism with a patience of 100 epochs. This early stopping criterion terminated training if the validation loss did not improve for 100 consecutive epochs, thereby preventing overfitting and saving computational resources.

Each GraphSAGE layer was followed by batch normalization, ReLU activations, and a dropout rate of 0.5, which served to regularize the model and enhance training stability. Residual connections were employed where applicable to facilitate gradient flow and support deeper network training. The model architecture consisted of three layers, progressively reducing the feature dimensionality from the input size down to 16 logits corresponding to the product classes.

Training was performed using stratified 5-fold cross-validation to ensure robust evaluation across different subsets of the data. At each fold, training and validation masks were carefully constructed to maintain class distributions, and the model's performance was monitored on the validation set to determine the best checkpoint. Class imbalance was implicitly handled through stratification in the folds, and cross-entropy loss was used as the optimization objective.

This rigorous training setup enabled the GraphSAGE model to learn powerful node representations that effectively leveraged graph neighborhood information, resulting in improved classification accuracy and generalization on unseen test data.

In all five folds, both the training and validation losses decrease rapidly in the initial epochs, indicating that the model is learning effectively. After this initial phase, the losses stabilize and remain relatively low for the remainder of the training process, suggesting that the model has converged. Crucially, the validation loss closely follows the training loss in each fold without significant divergence, which indicates that the model is not overfitting to the training data and generalizes well to unseen data. The dashed vertical line in each plot marks the "Best Epoch," which is the point where the validation loss was at its minimum, and these best epochs occur consistently around the 450-500 epoch mark across all folds.

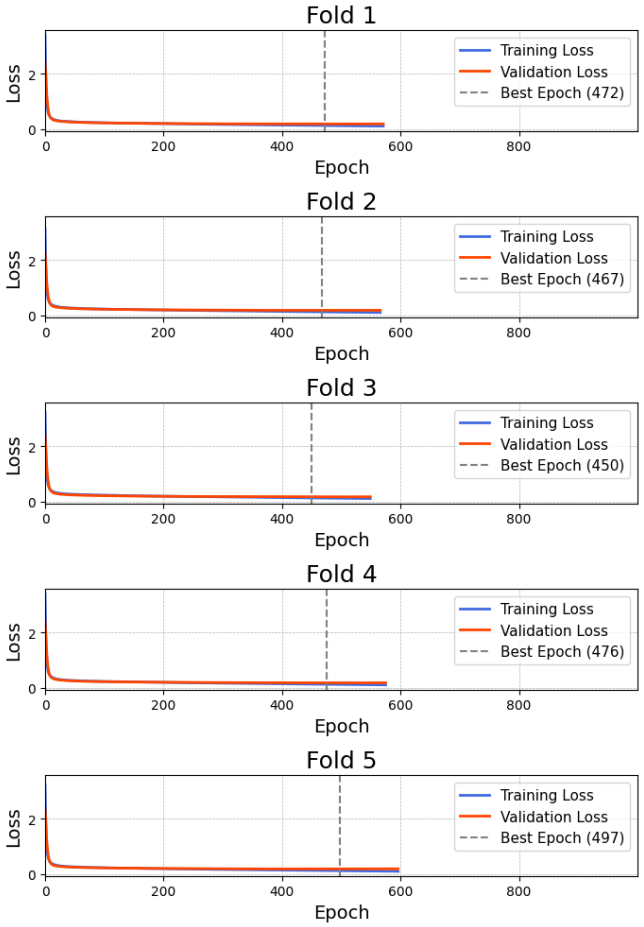

Figure 9: Train and Validation Losses for 5 Folds

Class	Precision	Recall	F1-score	Support
0	0.9773	0.9799	0.9786	3033
1	0.9172	0.9288	0.9229	2372
2	0.9278	0.9542	0.9409	8652
3	0.964	0.9721	0.968	1074
4	0.9562	0.9625	0.9594	3016
5	0.971	0.9669	0.9689	3565
6	0.9603	0.9546	0.9574	1519
7	0.9642	0.9414	0.9527	3752
8	0.9697	0.9719	0.9708	1316
9	0.9495	0.9568	0.9531	903
10	0.9013	0.8754	0.8882	3588
11	0.9465	0.9305	0.9384	1425
12	0.8812	0.8445	0.8625	1318
13	0.9441	0.8885	0.9155	323
14	0.973	0.9558	0.9643	226
15	0.9683	0.9531	0.9606	320
Accuracy			0.9433	36402
Macro Avg	0.9482	0.9398	0.9439	36402
Weighted Avg	0.9433	0.9433	0.9432	36402

Table 3: SAGE Fold 1 Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9787	0.9835	0.9811	3033
1	0.9228	0.9228	0.9228	2372
2	0.9311	0.9568	0.9437	8652
3	0.9579	0.9748	0.9663	1073
4	0.9539	0.9668	0.9603	3016
5	0.9686	0.9697	0.9692	3565
6	0.9592	0.9605	0.9599	1519
7	0.9709	0.9419	0.9562	3752
8	0.9728	0.978	0.9754	1316
9	0.9459	0.9491	0.9475	903
10	0.9059	0.8804	0.893	3588
11	0.9482	0.9382	0.9432	1425
12	0.8926	0.8506	0.8711	1319
13	0.9538	0.8947	0.9233	323
14	0.9858	0.9289	0.9565	225
15	0.9808	0.9594	0.97	320
Accuracy			0.9458	36401
Macro Avg	0.9518	0.941	0.9462	36401
Weighted Avg	0.9458	0.9458	0.9457	36401

Table 4: SAGE Fold 2 Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9781	0.9848	0.9814	3033
1	0.917	0.9275	0.9222	2372
2	0.9308	0.956	0.9432	8652
3	0.9667	0.9739	0.9703	1073
4	0.9581	0.9639	0.961	3016
5	0.9709	0.9733	0.9721	3564
6	0.9705	0.9546	0.9625	1519
7	0.9673	0.9459	0.9565	3752
8	0.9734	0.9742	0.9738	1316
9	0.9603	0.938	0.949	903
10	0.8988	0.8763	0.8874	3589
11	0.9516	0.9382	0.9449	1425
12	0.8903	0.8431	0.866	1319
13	0.9154	0.904	0.9097	323
14	0.9732	0.9646	0.9689	226
15	0.9684	0.9592	0.9638	319
Accuracy			0.9455	36401
Macro Avg	0.9494	0.9423	0.9458	36401
Weighted Avg	0.9454	0.9455	0.9454	36401

Table 5: SAGE Fold 3 Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9822	0.9825	0.9824	3033
1	0.9122	0.9283	0.9202	2372
2	0.9352	0.9528	0.944	8652
3	0.974	0.9758	0.9749	1073
4	0.9542	0.9612	0.9577	3016
5	0.9706	0.9714	0.971	3564
6	0.964	0.9519	0.9579	1519
7	0.966	0.9398	0.9527	3752
8	0.9736	0.9802	0.9769	1316
9	0.9471	0.9313	0.9391	903
10	0.8988	0.8905	0.8946	3589
11	0.9438	0.9305	0.9371	1425
12	0.8712	0.8414	0.856	1318
13	0.9518	0.9136	0.9323	324
14	0.9614	0.9912	0.976	226
15	0.9558	0.9498	0.9528	319
Accuracy			0.9448	36401
Macro Avg	0.9476	0.9433	0.9454	36401
Weighted Avg	0.9448	0.9448	0.9447	36401

Table 6: SAGE Fold 4 Classification Report

Class	Precision	Recall	F1-score	Support
0	0.9813	0.9862	0.9837	3033
1	0.9201	0.9216	0.9208	2373
2	0.9274	0.9542	0.9406	8652
3	0.9713	0.9786	0.9749	1073
4	0.9605	0.9672	0.9638	3015
5	0.975	0.9739	0.9745	3564
6	0.9614	0.95	0.9556	1519
7	0.9671	0.94	0.9534	3752
8	0.9674	0.9704	0.9689	1317
9	0.9627	0.9424	0.9524	903
10	0.9031	0.8698	0.8861	3588
11	0.9365	0.9522	0.9443	1424
12	0.8776	0.8543	0.8658	1318
13	0.9236	0.8951	0.9091	324
14	0.964	0.9469	0.9554	226
15	0.9564	0.9594	0.9579	320
Accuracy			0.9446	36401
Macro Avg	0.9472	0.9414	0.9442	36401
Weighted Avg	0.9445	0.9446	0.9445	36401

Table 7: SAGE Fold 5 Classification Report

5. Submission Results and Analysis

Final Two Submissions

As part of our participation in the Kaggle competition, we analyze two final submissions that focus on graph-based machine learning models. One of these submissions, submission_graphsage_5fold_avg_new.csv, employed a GraphSAGE model with 5-fold averaging and achieved log loss scores of 0.1890 on the private leaderboard and 0.1943 on the public leaderboard. This submission represents a relatively straightforward application of a single, albeit robust, graph neural network architecture.

Subsequently, we investigated an ensemble approach, ensemble_2.csv, which combined predictions from both GAT (Graph Attention Networks) and SAGE (GraphSAGE) models. Our analysis indicates that this ensemble, despite its increased complexity and the theoretical advantages often associated with ensembling, yielded performance metrics of 0.1922 and 0.1985 on private and public set respectively. Since Logloss Error represents a form of error where lower values are preferable, we can conclude that the simpler, singular GraphSAGE model demonstrated superior performance in this particular competition. This finding prompts us to consider the explainability of such results. While ensembles are generally expected to boost performance by leveraging diverse model strengths, their black-box nature can sometimes obscure why they might underperform compared to a well-tuned single model. Understanding the individual contributions and failure modes of each model within the ensemble, and why their combination did not lead to an improvement in this specific case, becomes crucial for future research and for enhancing the explainability of our modeling choices.

Other Submissions

Conversely, we also examined several other submissions that, comparatively, exhibited less optimal performance. Among these, a submission leveraging a BERT-base-uncased model registered metrics of 0.3515 and 0.3498. While BERT models excel in natural language processing, their application in this specific context appears to have resulted in higher error rates. Furthermore, a submission employing a combination of Graph Convolutional Networks (GCN) with a Multi-Layer Perceptron (MLP) showed metrics of 0.3021 and 0.3160, indicating that this particular hybrid model was less effective than the leading graph neural network solutions. Lastly, a submission based on an XGBoost algorithm achieved 0.2312 and 0.2433. While it is a strong traditional machine learning method, its performance, although better than BERT and GCN-MLP, remained inferior to that of the GraphSAGE model. These less successful models still offer valuable insights into how different architectures and techniques perform under the specific constraints of this challenge.

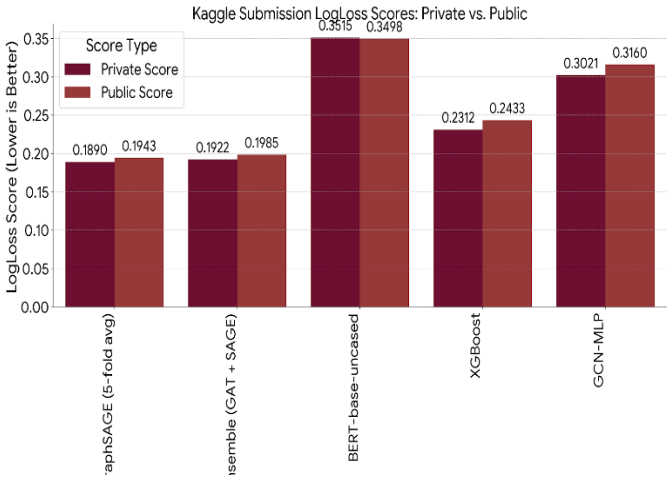

Figure 10: Best Submissions on Kaggle

6. Conclusion and Future Work

This report presented our comprehensive approach to the Amazon Product Classification Challenge hosted on Kaggle, leveraging both textual and graph-based representations of product data. Through a combination of modern machine learning techniques—including DistilBERT with LoRA for text embeddings, Graph Attention Networks (GAT), and GraphSAGE—we explored various modeling strategies to address the multi-class classification task. Our experiments confirmed the value of graph neural networks, with the GraphSAGE model achieving the strongest performance among all submissions. Interestingly, while ensemble methods are typically expected to enhance robustness and accuracy, our results showed that a single well-optimized GraphSAGE model outperformed the ensemble, highlighting the importance of model simplicity and targeted tuning in specific contexts.

The inclusion of detailed preprocessing steps, price normalization, class balancing strategies, and rigorous cross-validation further contributed to the reliability of our models. Additionally, by comparing our best submissions to a range of baseline models—including BERT, GCN-MLP hybrids, and XGBoost—we were able to better understand the limitations and strengths of different approaches when applied to multi-modal, real-world data.

Looking ahead, several avenues of future work could be pursued to improve performance and interpretability. First, advanced ensemble strategies, such as stacking with a meta-learner or incorporating uncertainty estimation, may provide more reliable aggregation of complementary models. Second, exploring self-supervised learning or contrastive methods to pre-train graph embeddings could lead to richer representations. Third, deeper analysis into failure cases—particularly for underperforming classes—could guide

targeted architectural refinements or data augmentation techniques.

Another promising direction involves the use of more powerful large language models (LLMs), such as LLaMA or Mistral, to generate higher-quality text embeddings. These models offer deeper contextual understanding and could better capture subtle semantic differences between product descriptions, especially in cases where smaller models like DistilBERT may fall short. Integrating such LLM-based embeddings into the current graph framework may further enhance the model's ability to distinguish between closely related product categories.

In summary, this challenge not only provided a valuable opportunity to apply cutting-edge machine learning models in a realistic setting but also underscored the critical role of graph-based learning in enhancing classification tasks that go beyond purely textual analysis.

References

1. G. Nikolentzos, “1_practical_aspects_of_data_science; 2_learning_on_graphs; 3_text_representation_learning,” lecture slides, eClass, Athens University of Economics and Business, 2025.

2. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

3. P. Veličković et al., “Graph attention networks,” in Proc. ICLR, 2018. [Online]. Available: https://arxiv.org/abs/1710.10903

4. W. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Adv. Neural Inf. Process. Syst. (NeurIPS), 2017. [Online]. Available: https://arxiv.org/abs/1706.02216

5. T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939785

6. T. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. ICLR, 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

****