“INF342: Product Classification Challenge”

Christos Mountzouris, Evangelos Rafail Kalemkeridis, Polytimi Doulou

1. Introduction

The “INF342: Product Classification Challenge” addresses a supervised learning task focused on sport product classification within the Amazon retail store. The challenge provides a vast dataset that comprises 227,508 distinct sport products from the Amazon retail store. Of these products, the 182,006 have been allocated to the training set, while the remaining 45,502 products have been allocated the test set, suggesting an 80/20 split.

2. Data imputation – Product prices

To address the issue of missing prices, two distinct imputation strategies were applied. The first approach imputed missing product prices by assigning the median price of their co-viewed products. The median was selected over the mean to reduce the influence of outliers, as users is assumed to view higher-priced products to compare the desired product specifications with more expensive ones, while the majority of co-viewed items are likely to be within a similar price range. Nevertheless, this method did not fully resolve missingness, as some products lack any co-viewed graph neighbors with known prices. The second, more advanced imputation strategy leveraged information from product descriptions. Product descriptions were vectorized using Term Frequency-Inverse Document Frequency (TF-IDF) without additional parameter fine-tuning, thereby capturing essential text features for each product. Subsequently, a k-Nearest Neighbors (kNN) algorithm with five neighbors and cosine similarity as the distance metric was applied to identify similar products in the TF-IDF space. For each product with a missing price, the mean price of its kNN-identified neighbors was used for imputation. This approach enabled the estimation of missing values based on textual similarity, even in the absence of co-viewed products with known prices.

2.3. Data cleaning – Product description

The data cleaning process primarily focused on preparing the product descriptions for effective TF-IDF vectorization. Initially, all descriptions were subjected to standard text preprocessing, which included lowercasing product descriptions, removing numerical values, special symbols, HTML markup, excessive whitespace, and filtering out words with fewer than three characters. This stage was essential for normalizing the text and minimizing the presence of noise that could adversely affect downstream vectorization. Subsequently, stop words were removed utilizing the built-in ENGLISH_STOP_WORDS set from the scikit-learn library, thereby eliminating common words that typically do not contribute meaningful information for text-based classification tasks. Lemmatization was then performed using the WordNetLemmatizer from the NLTK library with the ‘wordnet’ resource. This step reduced words to their canonical forms, enhancing the consistency of word representations. This lightweight lemmatization approach was intentionally selected to balance the need for semantic normalization with computational efficiency, as more exhaustive lemmatization techniques often incur significantly higher processing costs without proportional gains in performance for this application. Following lemmatization, removal of specific part-of-speech (POS) categories performed—specifically pronouns (PRON), determiners (DET), coordinating conjunctions (CCONJ), and subordinating conjunctions (SCONJ)—using POS tagging from the en_core_web_sm model of the spaCy library. The rationale for excluding these POS categories was to further refine the text to retain only the most informative content words, thereby reducing the dimensionality and potential redundancy in the feature space while focusing on terms that are more likely to carry discriminative power for product classification.

The final stage of text preprocessing involved the removal of the most frequently occurring words present in product descriptions across all the categories, as such terms are unlikely to contribute meaningful information for downstream modeling tasks. To determine an appropriate frequency threshold for word exclusion, a trial-and-error strategy was employed. During this process, the set of common words identified at various threshold levels was systematically

reviewed to ensure that only non-informative, generic terms were eliminated, while preserving potentially discriminative vocabulary. Some illustrative examples of the high-frequency, low-informative words excluded from the corpus included: 'best', 'black', 'color', 'come', 'comfort', 'design', 'designed', 'durable', 'easy', 'extra', 'feature', 'glove', 'great', 'grip', 'hand', 'head', 'help', 'high', 'inch', 'includes', 'large', 'length'. These terms, being generic in nature, do not facilitate the distinction between different product categories. Following this iterative analysis, the optimal cut-off frequency was empirically determined to be 60, above which words were systematically removed from the product descriptions.

3. Feature engineering

The feature engineering process comprised both the generation and extraction of an extensive set of derivative features from the available data. This was achieved by integrating domain-specific intuition regarding product classification with established methodological approaches tailored for both graph-structured and textual data. The primary objective of this step was to generate a diverse and comprehensive feature set, thereby enabling the models to capture intricate relationships and non-linear interactions within the data.

For tree-based models, the importance of each feature was systematically assessed using built-in feature importance metrics, which provide a robust framework for evaluating the predictive contribution of individual features. In the context of neural network models, a trial-and-error approach was employed to empirically determine the most effective features, acknowledging the model's inherent capacity to automatically learn complex, non-linear representations.

3.1. Feature generation for product prices

For the product prices, several derived features were engineered to enhance interpretability of the models and their predictive performance. The **price_na** feature was introduced to indicate the presence of missing price values in the original dataset, providing the model with explicit information about imputed entries. To address skewness and stabilize variance, the **price_log** feature captured the log-transformed price, a common practice for modeling monetary values. Additionally, the **price_decile** feature was created to denote the decile in which each product's price falls within the overall distribution, facilitating the model's ability to recognize relative price positioning. The **price_segments** feature further categorized prices into five ordinal bins (under 1, under 10, under 50, under 100, and over 100 units), thereby enabling the model to distinguish between broader price ranges that may correspond to different product types or market segments. The **price_ending** feature captured common psychological pricing patterns—such as prices ending in .99, .95, or .00—which can influence consumer perception and behavior. In accordance with their properties, the price_segments were encoded using ordinal encoding to preserve their inherent order, while nominal feature of price ending pattern was one-hot encoded to allow the model to utilize each pattern independently.

3.2. Feature generation for graph

The representation, in its raw form, could not be directly integrated into the tabular dataset; therefore, we extracted node-level features using the NetworkX library to generate meaningful graph-derived attributes for each product. Rather than relying on group-level or global graph statistics, only features directly associated with individual nodes (i.e., products) were considered to preserve the granularity required for product-level prediction. Specifically, the core_number feature was included to capture the node's participation in k-core subgraphs, thereby reflecting its structural resilience or centrality within the network. The clustering coefficient was also incorporated, quantifying the degree to which a product node tends to cluster with its immediate neighbors, indicating local connectivity. The avg_neighbor_degree feature measured the average degree of a node's neighbors, providing insights into the local network environment of each product. Additionally, pagerank was used to represent the global importance of each node within the product graph, reflecting its overall influence. Finally, the Louvain community detection algorithm (community_louvain) was applied to assign each product to a community partition, thereby encoding higher-order network structure and potential group memberships

as categorical features. This approach enabled the integration of rich structural information from the graph directly into the machine learning pipeline.

3.3. Feature generation for descriptions

To further enrich the feature space, product descriptions were systematically processed to derive category-specific textual features. To that end, for each product category, the most frequent and discriminative terms were identified, forming a curated vocabulary that characterizes that category. For instance, in the case of airsoft and paintball category, domain-relevant terms such as 'airsoft', 'paintball', 'gun', 'rifle', and 'pistol' were selected. Subsequently, for each product, features were computed by counting the occurrences of these category-specific terms within its description, thereby quantifying the textual alignment of a product with its assigned category. The selection process deliberately excluded both overlapping terms appearing in multiple categories (e.g., 'helmet' present in both 'bicycle' and 'ski') as well as generic or non-informative terms (e.g., 'product', 'amazing', 'offer'), thereby ensuring that the resulting features provided meaningful discrimination and reduced noise in the classification process.

3.4. Feature selection

To assess the relative importance of the engineered price and graph features, a Random Forest classifier with 100 estimators was constructed using the default scikit-learn configuration. The classifier was independently fitted on the sets of price-related and graph-derived features, providing an initial overview of their predictive contributions through the computed feature importances. While this analysis was not intended for direct feature elimination, it served as a valuable guide for subsequent experiments with feature selection in tree-based models.

Among the graph features, the community assignment emerged as the most influential predictor (importance > 0.70), followed by core number (0.14), avg_neighbor_degree (0.09), and pagerank and clustering with feature importance score around 0.05. In the price feature set, the presence of missing values (price_na) was most informative (0.49), followed by price decile (0.19) and log-transformed price (0.15), highlighting the significant role of missingness and relative price positioning in the classification task.

4. Modeling

The team's methodological approach for this challenge involved the development of two distinct classification models, each utilizing different feature sets, with their predictions subsequently combined using ensemble learning strategies. An XGBoost classifier was implemented to capture patterns from structured and categorical features (see Section 4.1), while an MLP was employed to model the textual information included in the product descriptions (see Section 4.2). The predictions from these base models were then integrated using two separate ensemble approaches: a stacked classification framework with soft voting, and a meta-learner scheme with Logistic Regression—each described in detail in Section 4.3.

The labeled data designated for model development was further partitioned into training and development (dev) subsets, following the standard practice of an 80/20 split. The dev set was reserved exclusively for validation purposes throughout the training process. To ensure that the class distribution in both subsets was accurately reflected the original dataset—particularly important in the presence of class imbalance—a stratified sampling strategy was employed.

The primary evaluation metric for this challenge was logarithmic loss (log loss), reflecting the objective of producing accurate probabilistic predictions for product classification. Logarithmic loss is formally defined as the negative log-likelihood of the true class labels given the predicted probability distributions generated by the classifier. This is mathematically expressed by the following equation:

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

where N denotes the total number of samples, C is the number of possible classes, y_{ij} is a binary indicator (0 or 1) specifying whether the true class label for sample ij, and p_{ij} is the predicted probability that sample i belongs to class j. The metric penalizes models for assigning high probabilities to incorrect classes, with larger penalties for more confident but wrong predictions.

4.1 XGBoost classifier

The objective of the first modeling approach was to capture patterns within the structured and categorical features, encompassing pricing, graph-derived, and description-based attributes. As an initial baseline, a logistic regression model with moderate regularization ($C = 10$) was employed, achieving a log loss of 1.36. This result surpassed the graph-only baseline of 2.364, yet did not outperform the text-only baseline of 0.3540, highlighting the relative strength of textual features. Subsequently, an XGBoost classifier with default hyperparameters was also trained, yielding a marked improvement with a log loss of 0.58. To further enhance the model's performance, a randomized grid search with five-fold cross-validation was conducted to optimize some key hyperparameters. The optimal configuration of 600 estimators with subsamples of 0.6, min_child_weight=3, max_depth=9, learning_rate=0.3, gamma=0.5, and colsample_bytree=0.8 further reduced the logloss to 0.3531, underscoring the importance of hyperparameter tuning in maximizing the predictive power of gradient-boosted tree models for structured data.

Table 1 presents the detailed classification report for the model, summarizing key performance metrics across all 16 product classes. The overall accuracy of the classifier reached 90%, with macro-averaged precision, recall, and F1-score values of 0.91, 0.90, and 0.90, respectively. The weighted averages are also high, indicating balanced performance despite variations in class frequencies. Most classes achieved F1-scores >= 0.90, reflecting the model's strong ability to correctly identify products within these categories. However, the model struggled to accurately classify products in C1, C10, C11 and C12, while demonstrating a strong predictive accuracy for products in minority classes.

4.2 MLP classifier

The second modeling approach centered on the development of an MLP classifier, specifically trained on high-dimensional, sparse TF-IDF vectors derived from the product descriptions. To ensure computational feasibility and prevent memory overload, the TF-IDF feature space was restricted to the top 20,000 most informative terms, as attempts to exceed this threshold resulted in memory allocation errors. The TF-IDF vectorization process was further refined by incorporating both unigrams and bigrams, discarding tokens that appeared fewer than five times in the entire corpus, and excluding terms with document frequency above the 99th percentile.

The architecture of the MLP was carefully constructed to balance model expressiveness and regularization. The network comprises an input layer matching the dimensionality of the TF-IDF features, followed by three hidden layers with 512, 256, and 128 units, respectively. Each hidden layer employs the ReLU activation function, batch normalization, and dropout with progressively decreasing rates (0.4, 0.3, and 0.2) to mitigate overfitting and stabilize the learning process. The final output layer utilizes the softmax activation function to produce class probability distributions for multi-class classification.

The MLP classifier was trained for up to 10 epochs with early stopping and class weighting to address class imbalance. The model demonstrated rapid convergence, achieving a training accuracy of 95.48% and a validation accuracy of 92.46% by the fifth epoch. The validation loss remained stable across epochs till the fifth epoch. The classification report for the MLP trained on TF-IDF features demonstrates strong overall performance, with an accuracy of 91.75% and a macro-averaged F1-score of 0.93. Precision, recall, and F1-scores are consistently high across the majority of classes, Notably, the model exhibited superior performance on minority classes, while significantly struggling with C12. Nevertheless, the weighted averages for all metrics remain high, reflecting balanced classification across the dataset.

4.3. Ensemble learning

Two ensemble learning approaches followed to combine the predictions of the two different classification methods; the first included a soft voting scheme, and the second included a meta-learner scheme.

4.3.1. Ensemble learning – Soft Voting

For the soft voting scheme, ensemble predictions were generated by combining the outputs of the XGBoost and MLP models using a weighted average, with weights set to 0.25 and 0.75, respectively. The optimal weights were determined using iterative adjustments in increments of 0.1 and evaluating the log loss. This approach led to a substantial improvement in predictive performance, reducing the overall log loss to 0.1313, however, this improvement was not reflected in the unseen data in Kaggle, potentially due to overfitting.

4.3.1. Ensemble learning – Meta-learner

For the meta-learner scheme, two diverse base classifiers employed—an XGBoost model trained on tabular features and an MLP trained on TF–IDF text vectors—each producing "honest" out-of-fold probability estimates for every training example. By splitting the training set into k=3 folds, model ensures that each example's probabilities come from a version of each base model that never saw that example during training. Then, those out-of-fold probability vectors were concatenated into a new feature matrix and train an XGBoost classifier (the meta-learner) on these combined probabilities to learn how best to weight and correct the base models' strengths and weaknesses. At inference time, we retrain each base model on the full dataset, generate their probabilities on new instances, and feed those probabilities into the meta-learner to produce the final predictions.

The meta-learning approach achieved a remarkably low log-loss of about 0.206, indicating that it's assigning high confidence to correct classes overall. Looking at the class-by-class breakdown, most categories (e.g. classes 0, 3, 5, 8) exhibit precision and recall above 0.95, and F1-scores around 0.97–0.98, even for quite large support counts (e.g. over 40 K samples for class 2). The harder classes—particularly class 10 (support ≈17,942) and class 12 (support ≈ 6,592)—show slightly lower recall (0.86 for class 10, 0.82 for class 12) and F1-scores around 0.87 and 0.86, respectively, but their precision remains in the high 0.80s–0.90 range. Minority classes like 13, 14, and 15 (with supports under 2 K) still achieve F1-scores of roughly 0.90–0.97, which is very good given their limited examples.

Stacked Meta-learner classifier				
Class	Precision	Recall	F1-score	Support
0	0.97	0.98	0.98	15165
1	0.90	0.90	0.90	11861
2	0.92	0.95	0.94	43260
3	0.97	0.98	0.97	5366
4	0.94	0.96	0.95	15079
5	0.97	0.96	0.97	17822
6	0.96	0.95	0.95	7595
7	0.97	0.94	0.95	18760
8	0.97	0.97	0.97	6581
9	0.95	0.94	0.94	4515
10	0.89	0.86	0.87	17942

11	0.93	0.92	0.93	7124
12	0.90	0.82	0.86	6592
13	0.91	0.88	0.89	1617
14	0.97	0.96	0.97	1129
15	0.96	0.95	0.96	1598

Table 5 – Classification average report for meta-learner classifier.