Product Category Classification Using Machine Learning

Harsh Bhadania
College of Computing and Informatics
University of North Carolina at Charlotte
Charlotte, NC 28233
hbhadani@uncc.edu

Introduction

In the current era of digital commerce, online marketplaces often struggle to automatically categorize products based on noisy or inconsistent product titles. Manual categorization is labor-intensive and error-prone, especially at scale. This project aims to automate the classification of product listings into predefined categories using supervised machine learning models. The task is framed as a multi-class classification problem, where the goal is to predict the category of a product based solely on its title.

The report explores the full pipeline of a machine learning project, including data exploration, preprocessing, modeling with multiple algorithms, evaluation, and interpretation of results. By comparing Logistic Regression and a Neural Network model (MLPClassifier), the study aims to determine which model best fits the task and why.

This study also demonstrates how different components of the ML lifecycle — such as hyperparameter tuning, feature extraction, evaluation, and visualization — contribute to building a performant and explainable model. Ultimately, the results indicate that thoughtful preprocessing and model selection play a vital role in solving real-world classification tasks efficiently.

Data

Dataset Description

The dataset used for this project was obtained from the UCI Machine Learning Repository (https://archive.ics.uci.edu/datasets). It consists of 35,311 product entries. Each entry includes:

- **Product ID**: A unique identifier for each product.
- **Product Title**: A short textual description of the product.
- **Merchant ID**: The seller or retailer offering the product.
- **Cluster ID**: An ID grouping visually or descriptively similar products.
- **Cluster Label**: A human-readable label for the cluster.
- **Category ID**: A numerical representation of the product's category.
- **Category Label**: The actual category name (e.g., Mobile Phones).

This data was provided in CSV format and pre-verified to be clean in terms of structure. Data integrity was also assessed by examining duplicates and outliers.

Exploratory Data Analysis

Initial data inspection revealed important insights:

- **No missing values** were found, ensuring a clean base for preprocessing.
- **Uneven category distribution** showed that some product types appeared more frequently than others. For example, Mobile Phones had significantly more entries than Freezers.
- **Title lengths and word counts** varied widely, suggesting a need for standardization and text processing.

Visualizations:

- A **bar chart of category distribution** revealed which product categories were most and least represented.
- A **histogram of product title lengths** helped in identifying the typical size of product descriptions.
- A **histogram of word counts** gave insight into how verbose the titles were, which affects TF-IDF performance.

In addition, basic summary statistics of text-based features were computed. This included the mean, median, and standard deviation of title lengths and word counts. The longest titles had over 30 words, while the shortest had only one.

Preprocessing

Data preprocessing involved several steps to ensure the text data was clean, relevant, and machine-readable:

- **Lowercasing and cleaning**: All product titles were lowercased to maintain uniformity. Punctuation, numbers, and excess whitespace were removed using regex.
- **Label Encoding**: Category labels were converted into numerical values using `LabelEncoder`, which allowed models to process them.

- **TF-IDF Feature Extraction**: `TfidfVectorizer` was used to convert text into numerical vectors. Important parameters:
    - `stop_words='english'` to eliminate common but non-informative words like "the" or "and".
    - `max_df=0.95` to ignore extremely frequent words that could reduce model performance.
    - `min_df=5` to ignore very rare words.
    - `ngram_range =(1, 2)` to capture both individual words and bigrams (e.g., "iphone 8").
    - `max_features=5000` to limit the number of features and control model complexity.
- **Train-Test Split**: The dataset was split 80/20 using stratified sampling, ensuring all classes were proportionally represented.

Text preprocessing also included stop word removal, which significantly reduced the number of uninformative terms, and improved training time. The TF-IDF representation resulted in a sparse matrix of 35,311 rows and 5,000 columns, ideal for linear models.

Methods

Logistic Regression

Logistic Regression was selected as a strong, interpretable baseline model for multi-class classification. It calculates probabilities of a class label based on a linear combination of features. The model was tuned using `GridSearchCV`, which tries different combinations of hyperparameters:

- `C` controls the regularization strength (higher means less regularization).
- `penalty` indicates the type of regularization (L2 was used).
- `solver` specifies the algorithm to optimize the cost function (e.g., 'lbfgs', 'liblinear'). This tuning ensured optimal performance across the feature space.

Neural Network (MLPClassifier)

A Multi-Layer Perceptron (MLP) is a type of feed-forward artificial neural network. It was chosen because of its ability to model complex, non-linear patterns in the data. Our architecture included:

- One hidden layer with 100 neurons
- ReLU activation
- A softmax output layer for multi-class classification
- A training limit of 300 iterations
  This model was expected to perform better than Logistic Regression for complex text patterns.

Both models were saved using `joblib` for future reuse and prediction. Model training was modularized in the code for clarity and reusability.

Results

Evaluation Metrics

To measure performance, we used:

- **Accuracy**: Percentage of correct predictions
- **Precision**: Fraction of relevant instances among retrieved instances
- **Recall**: Fraction of relevant instances that were retrieved
- **F1-score**: Harmonic mean of precision and recall
- **Confusion Matrix**: Visual representation of classification errors

Performance Summary

Model	Accuracy	Notes
Logistic Regression	~96%	Excellent baseline, efficient to train

Neural Network	~96%	Comparable accuracy, better on rare classes

Visualizations

- Confusion matrices with category labels show where models performed well or struggled.
- EDA graphs helped understand the data distributions and supported preprocessing choices

Observations

- Both models performed well due to quality preprocessing and feature extraction.
- Neural Network outperformed Logistic Regression in minority class predictions.
- Logistic Regression trained faster and was easier to interpret.
- Bi-grams in TF-IDF added valuable contextual information.

The model evaluation was made more robust by running additional test runs with different random states. This provided confidence in the reproducibility and generalizability of results. All output was captured into a log file using a custom logging system.

Conclusions

This project demonstrated that with the right preprocessing and feature engineering, even relatively simple models can deliver high classification performance. Logistic Regression served as a reliable, interpretable baseline, while the Neural Network offered a more flexible alternative for modeling complex patterns in text.

Key Learnings:

- Data cleaning and feature selection significantly impact results
- Simple models like Logistic Regression can outperform complex ones if tuned well
- TF-IDF with n-grams enhances model's understanding of language
- Modular coding and pipeline structure improve reusability and reproducibility

- Logging results and visualizing confusion matrices helps interpret model behavior

References

- Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*.
- Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing*. 3rd ed.
- McKinney, W. (2012). *Python for Data Analysis*. O'Reilly Media.
- Python documentation. https://docs.python.org
- scikit-learn documentation. https://scikit-learn.org/stable/
- ChatGPT is used for debugging and concept understanding

Source Code

GitHub Repository: git@github.com:HarshBhadania05/Product-Classification.git