Design Specification document for Product Categorization

Version 1.0

1. Introduction

This document provides a comprehensive design for a machine learning system to predict product categories (e.g., "Sports & Outdoors," "Fashion") from its associated structured and unstructured product metadata.

1.1 Problem Statement

Objective: Given product attributes (title, description, features, manufacturer, SKU, details), classify each product into the correct category before building a proof-of-concept pipeline.

Challenges:

- Mixed data types (text, numerical, categorical, nested JSON).
- Key fields like 'Price,' 'Description,' or 'Features' may be missing.
- Some categories may dominate (in terms of number of occurrences) and can bias the model.
- Descriptions/titles contain unnecessary jargon, special characters, or inconsistent formatting.
- Need to derive structured features (e.g., weight, volume) from unstructured 'Details' while accounting for unit mismatches.
- Grouping and mapping metadata such as weight and volume is not straightforward due to inconsistent naming conventions.
- Need to accommodate added categories, if any, in the future.

1.2 Assumptions

- Features such as 'Description,' 'Details,' and 'Manufacturer' are mostly populated.
- The algorithm should prioritize accuracy over speed (can exceed ideal real-time end-to-end time).

- 'Price,' 'Weight,' 'Volume,' 'Material,' and 'Size' can be sparse but useful if imputed.
- The text fields and 'Manufacturer' field contain discriminative keywords (e.g., "Brake" → 'Automotive').
- The model should handle ~10k product categories with moderate resource usage and be scalable to ~1 million categories.
- Text fields contribute more significantly to predictions than numerical fields.

1.3 Open Questions

Question	Possible Approaches
How to handle highly skewed numerical variables?	Log-transform & standardize, bin into quantiles, or drop as outliers.
Is NER worth the overhead?	Test spaCy NER using text based data
Best way to merge dimension fields?	Use regex to extract and standardize for units
How to handle similar categories like sports and fitness	Club them under a single category
Which model has ideal trade off between accuracy and memory requirements	Iterate through different models, starting from the base models and compare

2. Functional Requirements

- Extracting and preprocessing necessary fields, including normalization and standardization.
- Identifying appropriate features for model training.
- Tokenization, lemmatization, and stop word removal.

- Grouping or generating features for training.
- Handling missing values, outliers, and data skewness.
- Addressing data imbalances.
- Hyperparameter tuning.
- Developing a classification model using ML algorithms.

3. Non-Functional Requirements

- Scalability considerations.
- Efficient memory management.
- Logging and tracking models and versions.
- Security and privacy.

4. Feature Engineering

Feature Type	Examples	Encoding
Numerical	Price, Weight, Volume	StandardScaler
Categorical	Manufacturer, SKU	One-Hot (if <50 cats), else Label
Text	Combined text	TF-IDF or BERT
Derived (via NER)	text ner, manufacturer NER	One-Hot

4.1 Approaches

- **Pattern Matching:** Extract structured information from JSON using tools like regex.

- **NER (Named Entity Recognition):** Extract brand names and product materials from all text fields and the manufacturer field.
- **Dimensionality Reduction:** Use PCA (Principal Component Analysis) or ICA (Independent Component Analysis).
- **Feature Selection:** Use techniques like correlation matrix analysis, dimensionality reduction, and feature importance from base models.

5. Model Development

5.1 Class Imbalance Handling

- **Data-Level:** Use SMOTE for minority classes.
- **Algorithm-Level:** Utilize class weights in XGBoost and Random Forest.

5.2 Text Processing Methods

- **Basic Approaches:** TF-IDF or Bag-of-Words for lightweight solutions.
- **Advanced Approaches:** Use BERT embeddings for better contextual understanding.
- **NER:** Utilize spaCy's built-in models (e.g., `en_core_web_trf`) to derive structured features.

5.3 Baseline Models

- **Random Forest:** Handles heterogeneous features well.
    - Key hyperparameters: `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`
- **XGBoost:** Optimized for imbalanced data.
    - Key hyperparameters: `n_estimators`, `max_depth`, `subsample`
- **SVM:** Suitable for small and complex datasets.
    - Key hyperparameters: `kernel`, `gamma`

5.4 Advanced Models

- **BERT:**

    - Use `distilbert-base-uncased` for faster training.
    - Full BERT model for improved accuracy.
    - Key hyperparameters: `learning_rate`, `batch_size`, `embedding_size`
- **Neural Networks:**
    - LSTMs or RNNs as a fallback if BERT is too resource-intensive.
    - Key hyperparameters: `learning_rate`, `batch_size`

6. Evaluation

Metric	Purpose
Accuracy, Balanced Accuracy	Overall performance
F1-Score (per-class)	Better for imbalanced data.
Confusion Matrix	Identify misclassifications
Precision & Recall	Can prioritize one over the other based on the deployment considerations
Cross validation accuracy	Stratified K-Fold to ensure category distribution balance.

7. Deployment & Monitoring

7.1 API

- **FastAPI Endpoint:**

- `POST /predict` (Input: JSON, Output: predicted category)

7.2 Containerization

- **Technologies:** Docker + Kubernetes for scalability.
- **Inference Optimization:** Convert models to ONNX or TensorRT for efficiency.

7.3 Maintenance & Monitoring

- **Retraining Strategy:** Evaluate continuously with added data and retrain as needed.
- **Logging:** Store predictions for continuous improvement.

8. Risks & Mitigation

Risk	**Severity**	**Possible Mitigations**
Poor text feature quality	High	Augment with NER/regex features.
GPU out-of-memory with BERT	Medium	Use knowledge distillation techniques, decrease batch size and divide the dataset into chunks etc
Category drift over time	High	Regular retraining
Overfitting	High	Regularization, cross validation
Imbalanced datasets	Medium	Use upsample or downsample techniques
Order induced Bias	Medium	Randomize the data rows/samples

9. Alternative/additional Approaches

9.1 Hyperparameter Tuning

- Use grid search or random search for optimization.

9.2 Feature Engineering Enhancements

- Incorporate additional features like material and size.
- Use FastText for handling out-of-vocabulary words if speed is a higher priority than performance.

9.3 Others

- Explore alternative techniques for imbalance handling
- Experiment with different embedding strategies beyond TF-IDF and BERT.
- Experiment with zero shot learning models
- Try a combination of models, if appropriate
- Check SHAP for better model interpretability

10. Deployment Strategies

- Consider serverless deployment using FastAPI + Docker for for cost-effective scaling
- AWS Sagemaker for better control and ease of deployment

11. Implementation Plan

N/A

12. Tools & Libraries

- **Data Processing:** Pandas, spaCy, Regex.
- **Machine Learning:** Scikit-learn, XGBoost, Hugging Face, TensorFlow, torch, Transformers

- **Model Versioning & Tracking:** MLflow, Weights & Biases.
- **Deployment:** FastAPI, Docker.

This structured document provides a clear roadmap for developing and deploying a robust product categorization model.