Amazon Product Classification System Design Specification

1. Problem Statement

This project addresses the challenge of automatically classifying product listings into appropriate categories based on their attributes. The system uses semi-structured data containing product details to perform multi-class classification across 28 predefined categories.

Key Challenges:

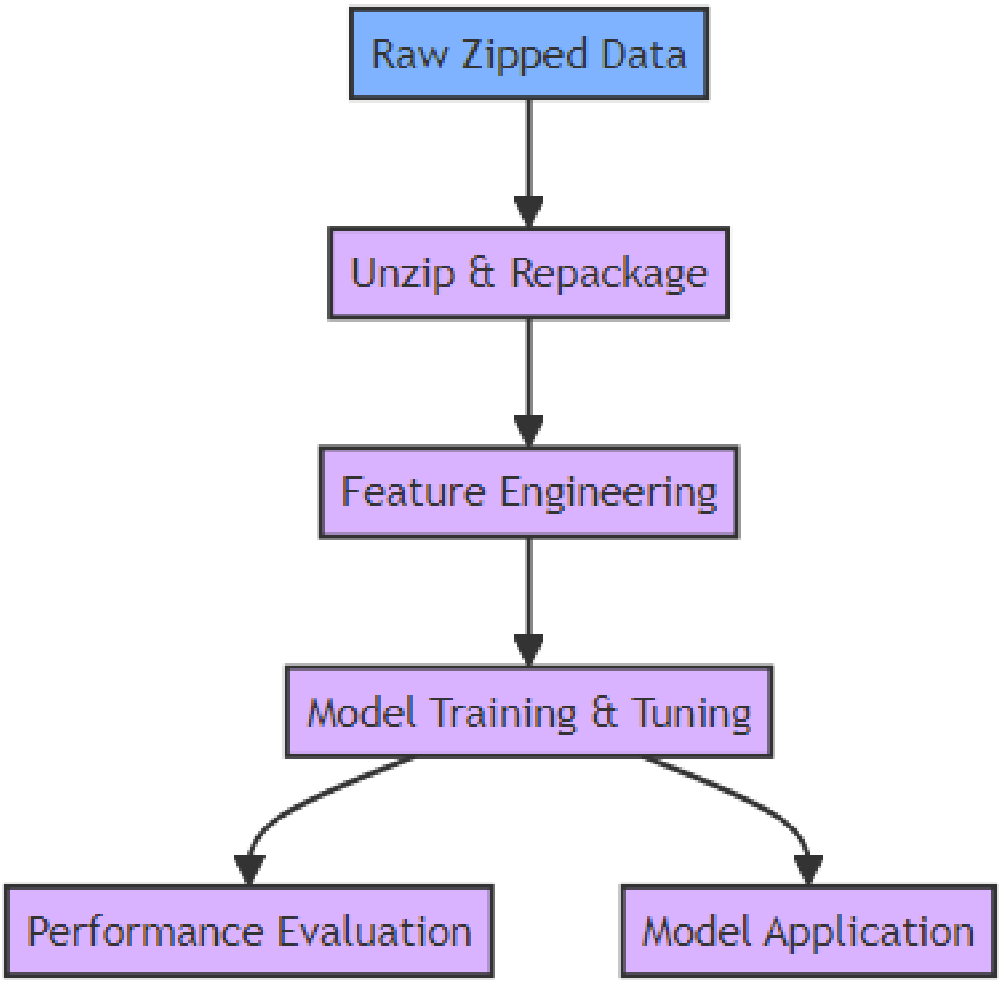
- Handling missing data (58.58% null values in price field)
- Processing heterogeneous data structures (varied fields across product types)
- Understanding semantic meaning in text data
- Managing sparse feature matrices with high dimensionality

2. Assumptions

1. Text fields (Title, Features, Description) contain valuable semantic information
2. Missing values, particularly in price fields, reflect dataset reality rather than data quality issues
3. Details field contains useful classification signals despite variability across products
4. Dataset distribution reflects expected production data
5. Tree-based models will handle heterogeneous data effectively
6. Processing requires balancing model complexity with performance

3. Solution Design

3.1 Data Pipeline



3.2 Data Preprocessing

Numeric Fields (Price)

- Log transformation normalizes the skewed distribution
- Min-max normalization preserves null values as distinct signals
- Float32 type conversion for memory efficiency

Text Fields (Title, Features, Description)

- Text preprocessing removes special characters
- List field concatenation into single strings
- DistilBERT embeddings capture semantic meaning
- TruncatedSVD reduces dimensionality from 768 to 50 components per field

Details Field (Nested Structure)

- Field name standardization (lowercase, special character replacement)
- Duplicate column merging for consistency

- Binary encoding (0/1) indicating presence/absence of each attribute
- Sparse PCA reduces dimensionality from 42,429 columns to 50 components

3.3 Model Selection & Training

XGBoost was selected as the classification algorithm for these reasons: - Effective handling of heterogeneous data types - Native support for missing values without imputation - Computational efficiency compared to deep learning approaches - Strong performance on tabular data - Regularization parameters to prevent overfitting

3.4 Training Process

1. **Data Splitting**
    - Stratified train/validation/test split (60%/20%/20%)
    - Subsample of data used for efficient hyperparameter tuning
2. **Hyperparameter Optimization**
    - RandomizedSearchCV explores parameter space efficiently
    - Parameters tuned: learning rate, tree depth, estimator count, regularization
    - 5-fold cross-validation for robust parameter selection
3. **Final Model Training**
    - Training on full dataset with optimized hyperparameters
    - Early stopping to prevent overfitting
    - ~90-91% accuracy achieved on hold-out test set

3.5 Evaluation Metrics

- Overall: accuracy, precision, recall, F1 (macro and weighted)
- Per-class metrics to identify category-specific performance
- Top-k accuracy (k=3, k=5)
- Confusion matrix visualization
- SHAP values for feature importance analysis

3.6 Deployment & Application

- Trained model saved with parameters and metadata
- Command-line interface for batch processing of unlabeled data
- Prediction outputs saved in both Parquet and CSV formats
- System designed for reproducibility and easy re-execution

4. Alternative Solutions

1. **Deep Learning Approaches**
    - Multi-modal neural networks could potentially achieve higher accuracy
    - Would require significantly more training time and resources
    - Advantage: Could better model complex relationships in textual data
2. **Ensemble Methods**
    - Combining multiple models (XGBoost, Random Forest, Neural Networks)
    - Advantage: Diversity of learning approaches could improve accuracy
    - Disadvantage: More complex to maintain and deploy
3. **Advanced Text Processing**
    - Entity recognition or topic modeling for text fields
    - Advantage: More nuanced feature extraction
    - Disadvantage: Additional complexity without guaranteed performance gains
4. **Feature Selection Techniques**
    - More rigorous feature selection or feature importance filtering

    - Advantage: Simpler model with faster inference time
    - Disadvantage: Potential loss of predictive power

5. Maintenance & Monitoring

Model Monitoring

- Track prediction distribution shifts over time
- Monitor class-specific performance metrics
- Implement alerting for significant accuracy drops

Retraining Strategy

- Periodic retraining with fresh data (monthly/quarterly)
- Compare new model performance against baseline
- Version control for models and training datasets

Error Analysis

- Automated logging of misclassified products
- Regular review of high-confidence incorrect predictions
- Identification of problematic categories or feature patterns

6. Open Questions

1. How would the model perform on new products from categories not represented in training?
2. Could a more sophisticated approach to the Details field further improve performance?
3. Would incorporating image data (if available) significantly improve classification accuracy?
4. How stable is model performance over time as product descriptions and attributes evolve?
5. What is the minimum set of features needed to maintain current accuracy?
6. Could the system be extended to recommend new categories when products don't fit existing ones?

7. Resource Requirements

- **Storage**: ~25% of raw data size due to efficient parquet compression
- **Compute**:
    - Feature Engineering: GPU recommended for BERT embeddings
    - Model Training: Multi-core CPU sufficient, GPU optional
    - Inference: Single CPU adequate for batch prediction
- **Memory**: 8-16GB RAM for standard dataset sizes