ML Challenge 2025 Problem Statement

Smart Product Pricing Challenge

1. Overview

The goal of this project was to estimate product prices using both textual and visual data available in the dataset. This dataset includes product descriptions, catalog metadata, structured numerical features, and associated images. To tackle this task, we integrated natural language processing methods for analyzing text, convolutional neural network embeddings for image representation, and employed a gradient boosting algorithm to perform regression.

2. ML Approach

a. Data Preprocessing

- Combined and normalized text fields (catalog_content, text) by converting them to lowercase.
- Replaced missing values in numeric columns (price, ipq) with zeros to maintain consistency.
- Extracted simple text-based metrics such as total word count, character count, average word length, and number of unique words.
- Verified image URLs, retrieved the corresponding files, and resized them to 224×224 pixels in RGB format for uniformity.

b. Feature Extraction

- **Textual Features**
  - Applied a TF-IDF vectorizer with a cap of 4000 features, using unigrams and bigrams to capture relevant text patterns.
  - Optionally incorporated semantic-rich sentence embeddings using the SentenceTransformer model for deeper contextual understanding.
- **Visual Features**
  - Leveraged MobileNetV2 with pre-trained weights from ImageNet, excluding the top classification layer and using average pooling to generate 1280-dimensional image embeddings.
- **Numerical Features**
  - Combined structured attributes like price, ipq, and derived text statistics with the extracted text and image embeddings to form a unified feature set.

c. Model Architecture

- Selected the CatBoost Regressor for its strength in managing mixed data types and delivering reliable results with minimal hyperparameter adjustments.
- The model was configured with the following parameters:
    - iterations = 800
    - learning_rate = 0.05
    - depth = 8
    - loss_function = 'MAE'
    - random_seed = 42

d. Training Strategy

- Employed 5-fold cross-validation to assess the model's consistency and generalization across different data splits.
- Reserved 10% of the dataset as a holdout set for final performance evaluation.
- Ensured consistent feature alignment during training and inference by concatenating features in a fixed sequence: TF-IDF vectors, followed by numerical attributes, and then image embeddings.

3. Experiments & Observations

Metric	K-Fold CV	Holdout Set
MAE	14.93 ± 1.00	5.58
R^2	0.12 ± 0.03	0.56

Observations: - Incorporating image-based embeddings enhanced the model's accuracy, particularly for products with strong visual characteristics. While TF-IDF features alone offered a solid starting point by capturing price-related signals from textual descriptions, the integration of textual, numerical, and visual data led to more reliable predictions across a diverse range of product types.

4. Conclusion

The solution demonstrates a multi-modal approach: - NLP (TF-IDF, optional sentence embeddings) captures textual nuances. - CNN image embedding captures visual cues. - Gradient boosting (CatBoost) integrates all features for robust regression.

This pipeline can be further improved by: - Fine-tuning CNN embeddings on the domain-specific dataset. - Using advanced transformers for text (BERT, RoBERTa) to capture context. - Hyperparameter tuning of CatBoost or alternative regressors like LightGBM or XGBoost.

5. Source Code Structure

1. Data Preprocessing: - `data_preprocessing.py`: load, clean, and merge datasets.

2. Text Feature Extraction: - `text_features.py`: TF-IDF vectorization, optional sentence embeddings.

3. Image Feature Extraction: - `image_features.py`: Download images, generate CNN embeddings using MobileNetV2.

4. Model Training: `model_training.py`: Generate combined features, train CatBoost regressor, save model.

5. Inference: - `model_inference.py`: Load saved model, vectorizer, image embeddings, predict prices.

6. Submission: - `generate_submission.py`: Preprocess test data, generate predictions, save CSV for submission.

Commenting & Documentation: Each script contains clear function-level comments describing input/output and processing steps. - Configurable parameters for paths, feature settings, and model hyperparameters are defined at the top of each script.