Project Write-Up

1. System Explanation

Overview:

Our system predicts the Harmonized System (HS) tariff code for shipping products using structured metadata, including item description, category tags, gift status, origin/destination, declared value, and weight. As manual classification is slow and error-prone, our project builds a machine-learning pipeline that automates classification, performs error analysis, and outputs confidence scores to assist operators.

AI Methods Used:

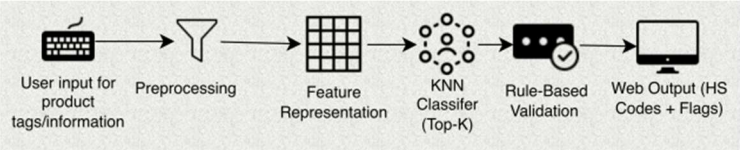
We implemented two supervised learning models:

1) **K-Nearest Neighbors (KNN):** Uses cosine similarity on TF-IDF text vectors combined with normalized numerical and categorical features. Through validation, we found that **k = 3**, using the cosine distance, achieved the best overall performance.

2) **ID3 Decision Tree:** Captures attribute-based decision patterns (e.g., weight thresholds).

Both models utilize the same engineered feature vector, which combines text, numeric, and categorical fields. Performance is evaluated using **cross-validation**, a **held-out test set**, **accuracy**, **macro-F1**, and **top-K accuracy**.

The AI Pipeline:



1. Data Input:

- Raw CSV containing fields: title, tags, price, weight, gift flag, origin, destination, HS label.

2. Preprocessing & Feature Engineering:

- **Cleaning:** Missing numeric fields (price, weight) are filled with column medians, and outliers are clipped at the 99th percentile.
- **Normalization:** Numerical fields are Z-score normalized to account for scale differences.
- **Text Vectorization:** Product titles are tokenized and converted into sparse vectors using TF-IDF weighting. This down-weights common words ("pack", "set") and highlights predictive terms ("lipstick", "battery").
- **Features Matrix:** All features are concatenated into a unified sparse matrix using a *ColumnTransformer*, as shown in the code snippet below:

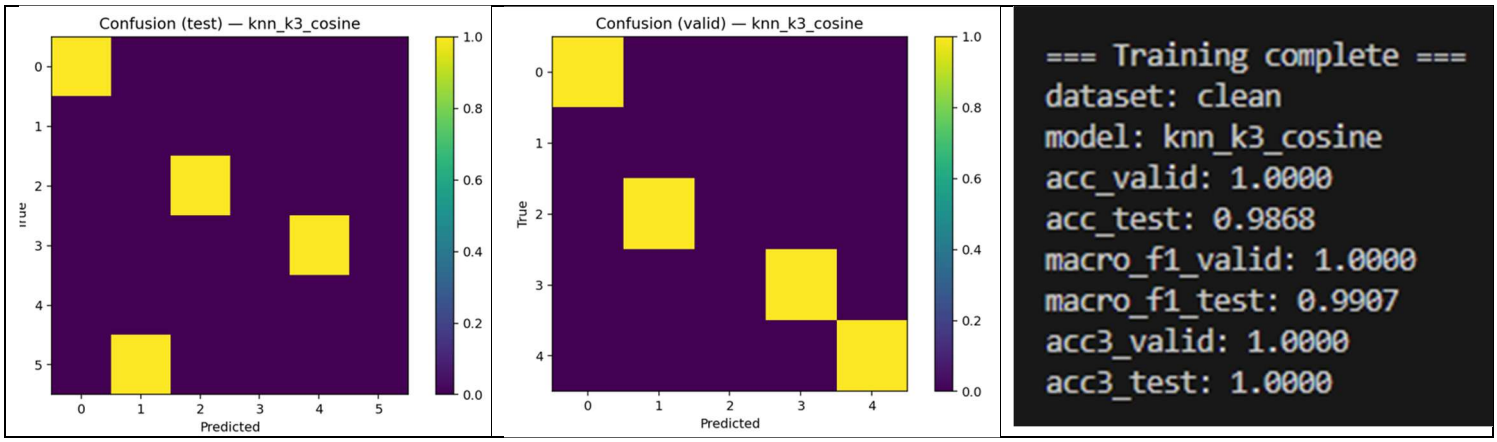
```
pre = ColumnTransformer(
    transformers=[
        ("tags", tag_vec, "tags"),
        ("desc", desc_vec, "description"),
        ("price_bin", cat_enc, ["price_bin"]),
        ("weight_bin", cat_enc, ["weight_bin"]),
        ("odg", cat_enc, ["origin", "dest", "gift"]),
    ],
    remainder="drop",
    sparse_threshold=1.0,
)

# Fit/transform
X_train = pre.fit_transform(train_df[used_cols])
X_valid = pre.transform(valid_df[used_cols])
X_test = pre.transform(test_df[used_cols])
```

3) Modeling: The processed arrays are fed into the KNN or ID3 models, trained on a split dataset (Train/Validation/Test):

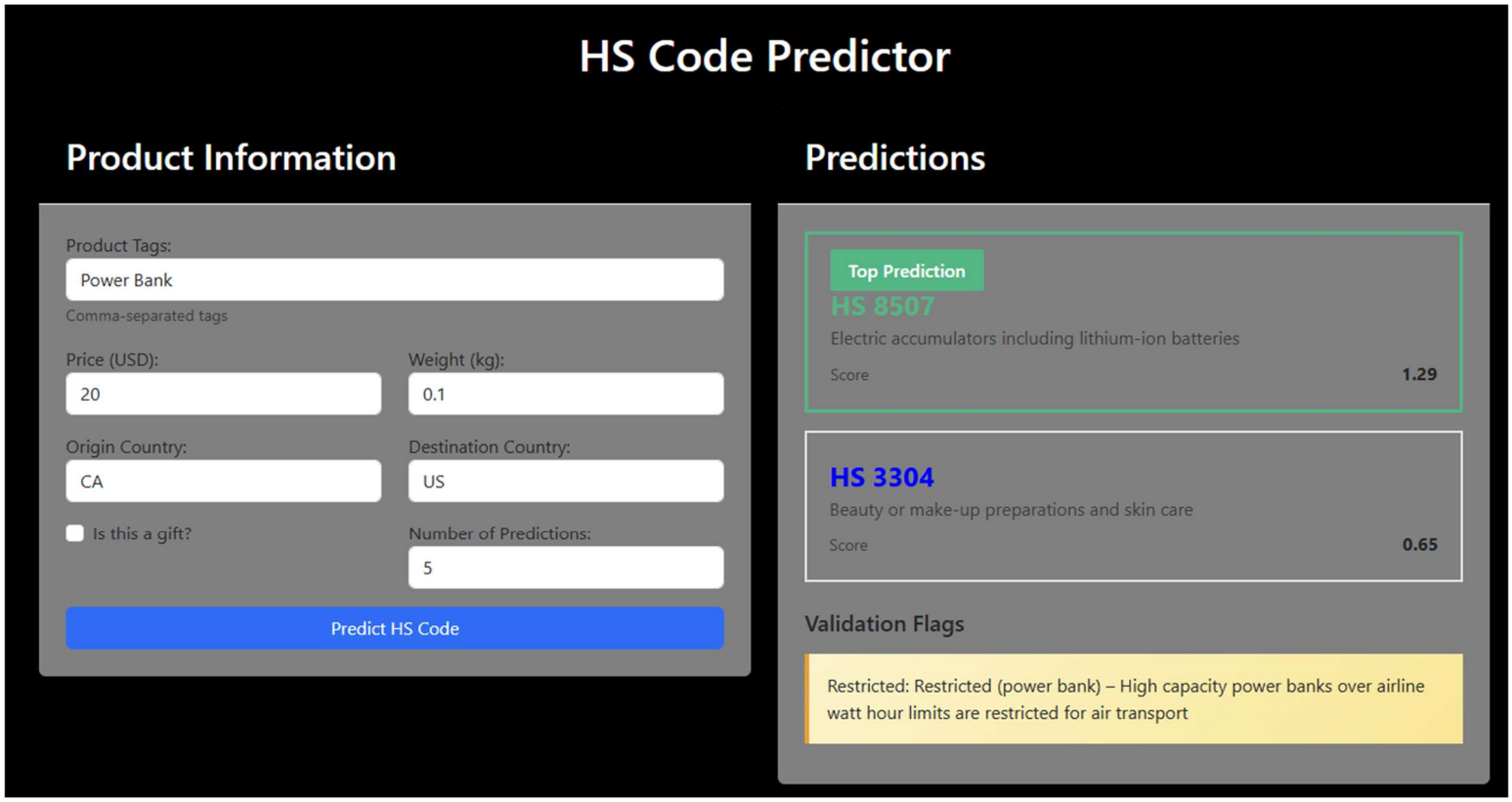
We evaluated both models using validation and held-out test sets. The confusion matrices below summarize the KNN classifier's performance on clean vs. noisy (real-world) data:

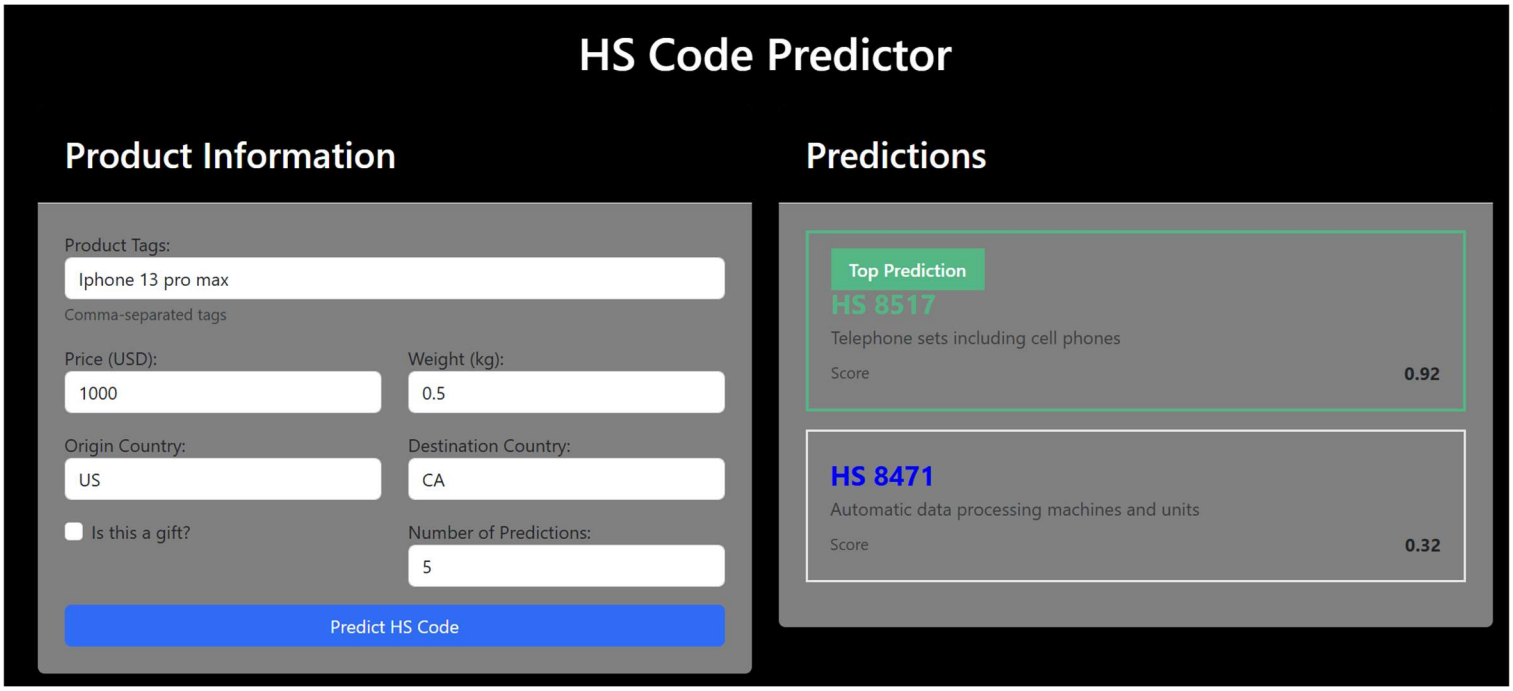
- **Validation Set:** Strong diagonal accuracy → The model performs better on structured, consistent product descriptions.
- **Test Set:** Accuracy drops when product titles are noisy, short, or ambiguous, revealing real-world classification challenges.



These evaluation outputs helped us to verify the model behavior and highlight where additional preprocessing or model robustness is needed.

4) Output: The system outputs the predicted HS code, a confidence score, and the top 3 most likely predictions to help user's decision-making. Some examples of the output:





Limitations & Mitigation:

- **Class Imbalance:**

  Certain HS codes appear rarely (1-2 times), making them difficult to predict. We addressed this by adjusting splitting strategies to ensure rare classes were represented or warned against.

- **Noise Sensitivity:**

  To understand robustness, we implemented a comparative analysis between "Clean" and "Noisy" datasets. The noisy dataset performed significantly worse due to spelling errors, highlighting the need for robust text preprocessing.

- **High Dimensionality:**

  The TF-IDF process generates 300–900 features, which can impact tree-based models. We mitigated this by using Cosine Distance in KNN, which is effective for high-dimensional sparse vectors.

- **Dependency on Labeled Data & Planned BERT Extension:**
  Our system relies on labeled training data, since KNN and ID3 require each product to include a known HS code. This limits performance on ambiguous or user-generated product descriptions. We originally planned to integrate a BERT text encoder to enhance semantic understanding (e.g., mapping terms such as “phone case,” “protective cover,” and “mobile accessory” to similar representations). Although not fully implemented due to time constraints, BERT remains the most promising next step for handling noisy or unlabeled inputs for our project’s future.

2. Feature Table

Description	Platform	Completeness	Code	Author	Notes
Data Cleaning & Preprocessing	Local	5	Python	Ertugrul Yurtseven	Implemented full cleaning, binning, one-hot encoding, text vectorization; Saves .npy feature matrices.
Feature Extraction Pipeline	Local	5	Python	Kazi Boni Amin	Modular ColumnTransformer pipeline; Supports swapping in future ML/NLP models.
KNN Classifier (Cosine Distance)	Local	5	Python	Kazi Boni Amin	Custom KNN implementation with top 3 accuracy; Performs best on a clean dataset.
ID3 Decision Tree Classifier	Local	3	Python	Kazi Boni Amin	Custom ID3 tree; Works but is less accurate than KNN; Limited by sparse high-dimensional data.
Cross-Validation Evaluation	Local	5	Python	Brayden Yee, Ertugrul Yurtseven	Computes accuracy, macro-F1, top-3 accuracy, and generates confusion matrices; Exports metrics JSON.
Dataset Handling	Local	4	Python	Ertugrul Yurtseven	Contributed to CSV organization and label-id consistency across datasets.
Frontend Web Interface	Local	3	Python/Django HTML/CSS	Brayden Yee	Early prototype page; partial Django integration; not connected to ML pipeline yet.
Model Saving & Artifact Management	Local	5	Python	Ertugrul Yurtseven, Kazi Boni Amin	Saves models, preprocessors, confusion matrices, and metrics in structured artifact folders.
Modular ML Pipeline	Local	4	Python	Kazi Boni Amin	Clean separation of preprocessing and modeling; ready for future BERT/TF-IDF upgrades.

3. External Tools, Sources, and Dependencies

3.1 External Python Libraries:

Library	Purpose
scikit-learn	Used for TF-IDF vectorization, KNN classifier, Decision Tree structure, and metric evaluation
pandas	CSV loading, data cleaning, and data frame manipulation
NumPy	Numerical operations, array handling, and vector transformations
matplotlib	Visualization of confusion matrices and data analysis
argparse	CLI interface for the training pipeline
os / pathlib / json	Managing file paths and saving metrics/model metadata

3.2 Data Sources:

1. HS Code Dataset (Provided Dataset)

- The dataset comprises product descriptions, category tags, weights, values, and the corresponding HS code.
- No external datasets were imported.

2. No external online data

- No scraping, external APIs, or public datasets were used.
- All data cleaning and feature generation were done strictly from the provided CSV files.

3.3 External Documentation Consulted:

Source	Link
scikit-learn Documentation	scikit-learn: machine learning in Python — scikit-learn 1.7.2 documentation
NumPy Documentation	NumPy Documentation
Pandas Documentation	pandas documentation — pandas 2.3.3 documentation
Python Argparse Documentation	argparse — Parser for command-line options, arguments and subcommands — Python 3.14.0 documentation

These resources were used only for syntax and correct usage of standard functions (e.g., TfidfVectorizer, KNeighborsClassifier, DecisionTreeClassifier, train_test_split, etc.).

No external ML models, pretrained transformers, or LLMs were used in the core system.

3.4 Additional References Consulted:

- Canada Border Services Agency (CBSA). *Tariff Classification Guidelines*.
- Exporteers. *HS Code Lookup Tool*. https://exporteers.com/hs-code/

Used for understanding the HS code structure, no data or API calls were imported into the project.

4. Contributions:

Kazi – Built and trained the machine learning model, implemented evaluation logic, and integrated it with the overall pipeline.

Ertugrul – Prepared and cleaned the datasets, structured the data splits, and ensured compatibility with the feature extraction and model training pipeline.

Brayden – Developed the web interface, connected the frontend to the model backend, and ensured the system functions end-to-end for user interaction.