HSN Code Validation and Suggestion Agent: Project Documentation

HSN Code Validation and Suggestion Agent: Project Documentation

1. Project Overview

This document details the design, implementation, and functionality of the HSN Code Validation and Suggestion Agent, developed using the Google Agent Developer Kit (ADK) framework. The agent's primary purpose is to assist users in validating Harmonized System Nomenclature (HSN) codes and suggesting relevant HSN codes based on product or service descriptions, leveraging a master HSN dataset provided in an Excel file.

1.1 Objective

The agent aims to provide a robust and intelligent solution for:

- HSN Code Validation: Verifying the format, existence, and hierarchical structure of HSN codes.
- HSN Code Suggestion: Offering accurate HSN codes based on natural language descriptions of goods or services.

1.2 Problem Description

HSN codes are a globally standardized system for classifying traded products, varying in length (typically 2 to 8 digits) with each level representing a more specific classification. The challenge involves building an agent that can:

- Determine the validity of a given HSN code against a master dataset.

- Provide hierarchical context for valid codes.
- Suggest appropriate HSN codes when a user provides a textual description of a product or service.

2. Architecture and ADK Framework Integration

The agent's architecture is designed for modularity, maintainability, and leverages key components of the Google ADK framework to facilitate intelligent interaction and tool utilization.

2.1 Overall Agent Structure

The agent follows a layered architecture:

- ADK LlmAgent: The core orchestrator, responsible for interpreting user input and deciding which underlying tool to invoke (validation or suggestion).
- Function Tools: ADK's FunctionTool is used to expose the validate_hsn_code and suggest_hsn_codes functionalities of the HSNCodeAgent class to the LlmAgent.
- HSNCodeAgent (Custom Logic Layer): A custom Python class that encapsulates the business logic for HSN validation and suggestion. It acts as an intermediary, utilizing specialized modules for data loading, validation rules, and suggestion algorithms.
- Utility Modules:
  - HSNDataLoader: Handles the loading and initial processing of the HSN master Excel file.
  - HSNCodeValidator: Implements the specific rules for HSN code format, existence, and hierarchical validation.
  - HSNCodeSuggestor: Implements the logic for matching textual descriptions to HSN codes using TF-IDF and cosine similarity.
  - HSNErrorHandler: Provides centralized error handling, input validation, and logging across the application.
- ADK Runner & Session Service: Used for running the agent in an interactive loop and managing conversational sessions.

```
graph TD

User -->|Input (Text/Numeric)| ADK_Runner

ADK_Runner -->|New Message| LlmAgent

LlmAgent -->|Tool Selection| HSNCodeAgent

HSNCodeAgent -->|Calls| HSNCodeValidator

HSNCodeAgent -->|Calls| HSNCodeSuggestor
```

HSNCodeValidator -->|Uses| HSNErrorHandler

HSNCodeValidator -->|Uses| HSNDataLoader

HSNCodeSuggestor -->|Uses| HSNErrorHandler

HSNCodeSuggestor -->|Uses| HSNDataLoader

HSNDataLoader -->|Reads| HSN_Master_Data.xlsx

HSNCodeAgent -->|Returns Result| LlmAgent

LlmAgent -->|Final Response| ADK_Runner

ADK_Runner -->|Output| User

2.2 Handling User Input

The LlmAgent is configured with an instruction that guides its behavior:

- If the user provides a numeric input (e.g., "01011010"), the agent is instructed to use the validate_hsn_code tool.
- If the user provides a textual description (e.g., "live horses"), the agent is instructed to use the suggest_hsn_codes tool.

This intelligent routing is handled by the gemini-1.5-flash-latest model integrated within the LlmAgent, which interprets the user's intent and selects the appropriate FunctionTool.

2.3 Providing Output

The agent provides its validation and suggestion outputs as formatted strings.

- Validation Output: For a valid HSN code, it returns a confirmation message including the HSN code, its description, and any identified parent codes. For invalid codes, it provides an error message with the reason (e.g., "not found," "invalid format").
- Suggestion Output: For suggestions, it returns a list of top-k (default 5) suggested HSN codes along with their descriptions, or a "No strong matches found" message if similarity is below a threshold.

3. Module Breakdown

3.1 adk_hsn_agent.py

This is the main entry point for the agent.

- HSNCodeAgent Class:
  - __init__(self, excel_file_path="HSN_SAC.xlsx"): Initializes the data loader, validator, and suggestor components.
  - validate_hsn_code(self, code: str) -> str:
    - Takes an HSN code string as input.
    - Calls HSNCodeValidator.validate_code to check format and existence.
    - If valid, calls HSNCodeValidator.validate_hierarchy to find parent codes.
    - Returns a user-friendly string indicating validity, description, and parent codes, or an error message.
  - suggest_hsn_codes(self, description: str) -> str:
    - Takes a product/service description string as input.
    - Calls HSNCodeSuggestor.suggest to get relevant HSN codes.
    - Formats the suggestions into a readable string.
- create_hsn_agent() Function:
  - Instantiates HSNCodeAgent.
  - Creates FunctionTool instances for validate_hsn_code and suggest_hsn_codes.
  - Initializes and returns an LlmAgent with a descriptive name, instruction, model (gemini-1.5-flash-latest), and the defined tools.
- run_hsn_agent() Function:
  - Sets up the ADK Runner and InMemorySessionService.
  - Provides an interactive command-line interface for the user to input HSN codes or descriptions.
  - Uses runner.run_async to send user input to the agent and prints the agent's final response.

3.2 hsn_data_loader.py

Responsible for loading the master HSN data.

- HSNDataLoader Class:
  - __init__(self, file_path):
    - Uses HSNErrorHandler.safe_load_excel to load the Excel file into a Pandas DataFrame.
    - Strips whitespace from column names to ensure consistency.
    - Validates the presence of 'HSNCode' and 'Description' columns.
    - Converts 'HSNCode' and 'Description' columns to string and lowercase, respectively, for consistent processing.
  - get_dataframe(self): Returns the loaded and pre-processed Pandas DataFrame.

3.3 hsn_code_validator.py

Implements the HSN code validation logic.

- HSNCodeValidator Class:
  - __init__(self, df): Stores the DataFrame and pre-populates a set of valid HSN codes for efficient lookup.
  - validate_code(self, code):
    - Performs format validation using HSNErrorHandler.validate_hsn_input.
    - Performs existence validation by checking if the code is in the valid_codes set.
    - Retrieves the description using HSNErrorHandler.handle_data_access.
    - Returns a tuple (is_valid, message/description).
  - validate_hierarchy(self, code):
    - Generates potential parent codes (2, 4, 6 digits) from the input code.
    - Checks if these parent codes exist in the valid_codes set.
    - Returns a list of valid parent codes.

3.4 hsn_code_suggestor.py

Implements the HSN code suggestion logic.

- HSNCodeSuggestor Class:
  - __init__(self, df):
    - Initializes a TfidfVectorizer and fits it to the 'Description' column of the HSN master data to create a TF-IDF matrix. This pre-processing is crucial for efficient similarity calculations.
  - suggest(self, query, top_k=5):
    - Validates the input description using HSNErrorHandler.validate_description_input.
    - Transforms the user's query into a TF-IDF vector.
    - Calculates cosine_similarity between the query vector and the pre-computed TF-IDF matrix of descriptions.
    - Filters suggestions based on a cosine_sim.max() < 0.3 threshold to avoid weak matches.
    - Returns the top top_k matching HSN codes and their descriptions, or a message if no strong matches are found.

3.5 hsn_error_handler.py

A static utility class for robust error handling and logging.

- HSNErrorHandler Class (Static Methods):

- validate_hsn_input(code): Validates HSN code format (non-empty, string, numeric, length 2, 4, 6, or 8 digits).
- handle_data_access(df, code): Safely retrieves the description for a given HSN code from the DataFrame, handling cases of missing data or lookup errors.
- validate_description_input(description): Validates description input (non-empty string, minimum length).
- handle_suggestion_response(results, threshold=0.3): Standardizes the return format for suggestion results, handling empty results or error messages.
- log_warning(message), log_info(message): Centralized logging functions.
- safe_load_excel(file_path): Provides robust Excel file loading with error handling for FileNotFoundError and general exceptions.

3.6 test_hsn_agent.py

An automated testing script for the HSN agent.

- run_automated_tests() Function:
  - Sets up the agent and runner similar to adk_hsn_agent.py.
  - Loads the HSN_SAC.xlsx file to iterate through sample HSN codes and descriptions.
  - For each entry, it simulates two types of user queries:
    - "Validate HSN code <hsn_code>"
    - "Suggest HSN codes for '<description>'"
  - Prints the agent's responses for each test.
  - Includes an asyncio.sleep delay to prevent hitting API rate limits during automated testing.

4. Data Handling

The agent relies on a master Excel file (HSN_SAC.xlsx) containing 'HSNCode' and 'Description' columns.

- Access and Processing: The HSNDataLoader module is responsible for accessing this file. It uses the pandas library, specifically pd.read_excel(), for efficient data loading.
- Efficiency for Large Datasets:
  - Pre-processing: The data is loaded and pre-processed once during the HSNCodeAgent initialization. This includes stripping column names, converting data types (HSNCode to string, Description to lowercase string), and building a set of valid_codes for O(1) lookup in the validator. The TfidfVectorizer in the suggestor is also fitted once on the entire description corpus.
  - Trade-offs:
    - Pre-loading (Chosen Approach):
      - Pros: Faster lookup and suggestion generation during runtime as the data is already in memory and pre-indexed/vectorized. Reduces latency for user queries.
      - Cons: Higher memory consumption, especially for very large datasets. Initial startup time might be longer.

- Loading on Demand (Alternative):
  - Pros: Lower memory footprint.
  - Cons: Slower response times for each query as data would need to be read and potentially processed for every request. Not suitable for the suggestion logic which requires the entire corpus for TF-IDF.

Given the nature of HSN code validation and suggestion (requiring quick lookups and similarity comparisons across the entire dataset), pre-loading and pre-processing are the optimal choices for performance and user experience.

5. Validation and Suggestion Logic

5.1 HSN Code Validation Rules (hsn_code_validator.py)

- Format Validation:
  - Ensures the input is a non-empty string.
  - Verifies that the string consists only of digits (isdigit()).
  - Checks if the length of the code is one of the acceptable lengths: 2, 4, 6, or 8 digits. This is handled by HSNErrorHandler.validate_hsn_input.
- Existence Validation:
  - After format validation, it checks if the exact HSN code exists in the valid_codes set, which is populated from the 'HSNCode' column of the master dataset.

- Hierarchical Validation:
  - For an 8-digit HSN code (e.g., 01011010), the validate_hierarchy method systematically extracts its potential parent codes (e.g., 01, 0101, 010110).
  - It then verifies if each of these parent codes also exists in the valid_codes set.
  - Value Added: This adds significant value by providing context. If a specific 8-digit code is valid, knowing its valid 2, 4, or 6-digit parents confirms its place within the HSN structure and provides broader classification information, which can be very helpful for users understanding the classification system.

5.2 Suggestion Logic (hsn_code_suggestor.py)

- The suggest method uses TF-IDF (Term Frequency-Inverse Document Frequency) and Cosine Similarity for robust textual matching.
  - TF-IDF: This technique converts text descriptions into numerical vectors, where each dimension represents a word and its value reflects the word's importance in a document relative to the entire corpus. This helps in identifying key terms.
  - Cosine Similarity: This metric calculates the cosine of the angle between two non-zero vectors in a multi-dimensional space. A cosine similarity close to 1 indicates high similarity, while a value close to 0 indicates low similarity.
- Process:
  - The TfidfVectorizer is pre-fitted on all 'Description' entries in the master dataset.
  - When a user query (description) is received, it's transformed into a TF-IDF

- Cosine similarity is calculated between the query vector and all description vectors in the TF-IDF matrix.
- The descriptions with the highest similarity scores are identified.
- A threshold (0.3) is applied to cosine_sim.max() to ensure that only "strong matches" are returned, preventing irrelevant suggestions for vague queries.
- The top top_k (default 5) HSN codes and their corresponding descriptions are returned.

6. Agent Responses

The agent's responses are designed to be informative and clear, whether for validation or suggestion.

6.1 Valid HSN Code Response

For a valid HSN code, the agent returns a message confirming its validity, providing the code's description, and listing any identified hierarchical parent codes.

Example: Valid HSN Code: 01011010 — live horses, asses, mules and hinnies. Parent Codes: ['01', '0101', '010110']

6.2 Invalid HSN Code Response

For an invalid HSN code, the agent provides a specific error message indicating why the code is invalid.

Examples:

- Invalid HSN Code: HSN code must be numeric. (Format Validation Error)
- Invalid HSN Code: HSN code not found in master dataset. (Existence Validation Error)
- Invalid HSN Code: HSN code must be 2, 4, 6, or 8 digits long. (Length Validation Error)
- Invalid HSN Code: Input cannot be empty. (Empty Input Error)

6.3 HSN Code Suggestion Response

For suggestions based on a description, the agent returns a formatted list of the most relevant HSN codes and their descriptions.

Example: Suggestions: 01011010 — live horses, asses, mules and hinnies 01011020 — pure-bred breeding animals 01012100 — horses for slaughter

If no strong matches are found (e.g., similarity score below threshold), a specific message is returned: Suggestion: No strong matches found.

7. Setup and Running Instructions

7.1 Prerequisites

- Python 3.x
- pandas library (pip install pandas openpyxl)
- scikit-learn library (pip install scikit-learn)
- Google ADK (Agent Developer Kit) - Ensure you have the ADK installed and configured. Refer to the official ADK documentation: https://google.github.io/adk-docs/
- An Excel file named HSN_SAC.xlsx in the same directory as your Python scripts, containing 'HSNCode' and 'Description' columns.
- An env.json file in C:\python\projects\HSN_using_adk\ (or adjust path in adk_hsn_agent.py) with your Google API Key:
- "store": "YOUR_GOOGLE_API_KEY_HERE"

7.2 Installation

- Clone the repository (or ensure all .py files and HSN_SAC.xlsx are in the same directory).
- Install required Python packages:
- pip install pandas openpyxl scikit-learn google-generativeai

(Note: google-generativeai is required for google.genai.types and google.adk components).

7.3 Running the Agent

To run the interactive HSN Code Validator and Suggestion Agent:

```
python adk_hsn_agent.py
```

Follow the prompts in the terminal to enter HSN codes or descriptions. Type exit to quit.

7.4 Running Automated Tests

To execute the automated test suite:

```
python test_hsn_agent.py
```

This script will iterate through entries in your HSN_SAC.xlsx file and print the agent's validation and suggestion responses. There is an intentional delay between tests to manage API rate limits.