Problem Statement

Feature Extraction from Images

In this hackathon, the goal is to create a machine learning model that extracts entity values from images. This capability is crucial in fields like healthcare, e-commerce, and content moderation, where precise product information is vital. As digital marketplaces expand, many products lack detailed textual descriptions, making it essential to obtain key details directly from images. These images provide important information such as weight, volume, voltage, wattage, dimensions, and many more, which are critical for digital stores.

Data Description:

The dataset consists of the following columns:

1. **index**: A unique identifier (ID) for the data sample.
2. **image_link**: Public URL where the product image is available for download. Example link - https://m.media-amazon.com/images/I/71XfHPR36-L.jpg To download images, use the `download_images` function from `src/utils.py`. See sample code in `src/test.ipynb`.
3. **group_id**: Category code of the product.
4. **entity_name**: Product entity name. For example, "item_weight".
5. **entity_value**: Product entity value. For example, "34 gram".

Note: For `test.csv`, you will not see the column `entity_value` as it is the target variable.

Output Format:

The output file should be a CSV with 2 columns:

1. **index**: The unique identifier (ID) of the data sample. Note that the index should match the test record index.
2. **prediction**: A string which should have the following format: "x unit" where x is a float number in standard formatting and unit is one of the allowed units (allowed units are mentioned in the Appendix). The two values should be concatenated and have a space between them.
   For example: "2 gram", "12.5 centimetre", "2.56 ounce" are valid.
   Invalid cases: "2 gms", "60 ounce/1.7 kilogram", "2.2e2 kilogram", etc.
   Note: Make sure to output a prediction for all indices. If no value is found in the image for any test sample, return an empty string, i.e., "". If you have less/more number of output samples in the output file as compared to `test.csv`, your output won't be evaluated.

File Descriptions:

Source Files:

1. **src/sanity.py**: Sanity checker to ensure that the final output file passes all formatting checks.
   Note: The script will not check if fewer/more number of predictions are present compared to the test file. See sample code in `src/test.ipynb`.
2. **src/utils.py**: Contains helper functions for downloading images from the `image_link`.
3. **src/constants.py**: Contains the allowed units for each entity type.
4. **sample_code.py**: A sample dummy code that can generate an output file in the given format. Usage of this file is optional.

Dataset Files:

1. **dataset/train.csv**: Training file with labels (`entity_value`).
2. **dataset/test.csv**: Test file without output labels (`entity_value`). Generate predictions using your model/solution on this file's data and format the output file to match `sample_test_out.csv` (Refer to the "Output Format" section above).
3. **dataset/sample_test.csv**: Sample test input file.
4. **dataset/sample_test_out.csv**: Sample outputs for `sample_test.csv`. The output for `test.csv` must be formatted in the exact same way.
   Note: The predictions in the file might not be correct.

Constraints:

1. You will be provided with a sample output file and a sanity checker file. Format your output to match the sample output file exactly and pass it through the sanity checker to ensure its validity.
   Note: If the file does not pass through the sanity checker, it will not be evaluated. You should receive a message like `Parsing successful for file: ...csv` if the output file is correctly formatted.
2. You are given the list of allowed units in `constants.py` and also in the Appendix. Your outputs must be in these units. Predictions using any other units will be considered invalid during validation.

Evaluation Criteria:

Submissions will be evaluated based on the **F1 score**, which is a standard measure of prediction accuracy for classification and extraction problems.

Let **GT** = Ground truth value for a sample and **OUT** be the output prediction from the model for a sample. Then we classify the predictions into one of the 4 classes with the following logic:

1. **True Positives:** If `OUT != ""` and `GT != ""` and `OUT == GT`
2. **False Positives:** If `OUT != ""` and `GT != ""` and `OUT != GT`
3. **False Positives:** If `OUT != ""` and `GT == ""`
4. **False Negatives:** If `OUT == ""` and `GT != ""`
5. **True Negatives:** If `OUT == ""` and `GT == ""`

Then,
F1 score = `2 * Precision * Recall / (Precision + Recall)`
where:

1. **Precision** = `True Positives / (True Positives + False Positives)`
2. **Recall** = `True Positives / (True Positives + False Negatives)`

Submission File:

Upload a test_out.csv file in the portal with the exact same formatting as sample_test_out.csv.

Download Data Set

Participant Resource