Catalog Enrichment

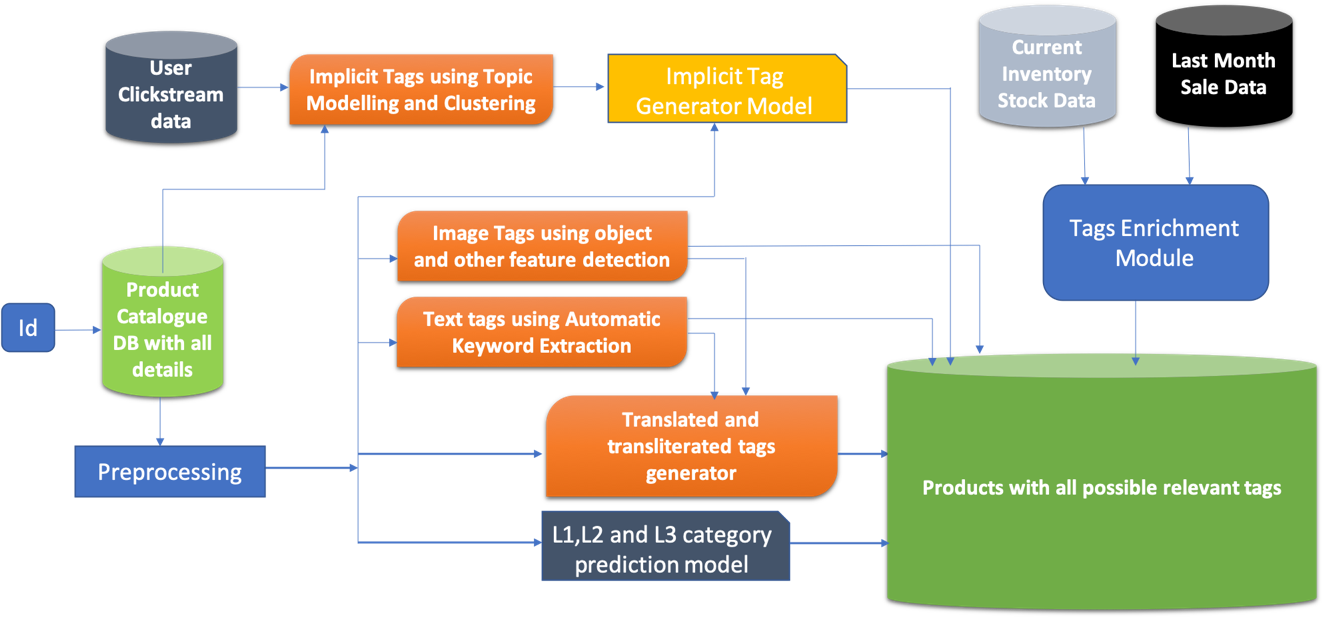
# **introduction**

The objective of catalog enrichment is to help businesses enrich catalogues by finding errors and updating them with actual attribute values. Missing attribute values are also added during the catalog enrichment process.

When all the attribute values are mentioned, it gets easier for customers to identify products they are looking for. Some key benefits of catalog enrichment:

* Improved Information Availability
* Better Filters for Product Search
* Improved Conversion during Sales & Seasons

## **Catalog Enrichment Pipeline**



# **Components** of **Catalog Enrichment**

## **Spell Check Module**

**Purpose**

Correct user typed queries to match the relevant products and eliminate the zero search results, thereby enhancing the search experience

**Architecture of Spell Correction as a part of the Query preprocessing module**

**Overview of the Query Pre-processing Pipeline**

**Data Cleansing**

The following are the steps involved in a typical Data cleansing

1. **Normalization**

Normalization involves text cleaning activities such as the following. This is available for consumption by any other Pipeline

* Converting the Text into lowercase
* Removing leading and trailing spaces
* Removing special characters and html tags from the text
* Converting all the numbers and dates to text format

**Normalization API**

Entire text is converted to Lowercase and apostrophe is removed along with html tag

**“mens running shoes”**

**“<href>Men’s running shoes”**

**Input Output**

1. **Tokenization**

After the Normalization the query goes through Tokenization. **This need not be the Natural flow as this can be consumed as a service within the Spell Correction.** Splitting the words into Individual tokens is a part of the Tokenization modules. This is available for consumption by any other Pipeline

**Note on Non-english tokens**

Non-english tokens (if any are identified) identified as a part of Tokenization will be flagged as per the respective language ID and not processed further as per the current design and there can be a fallback to ask user input in English only

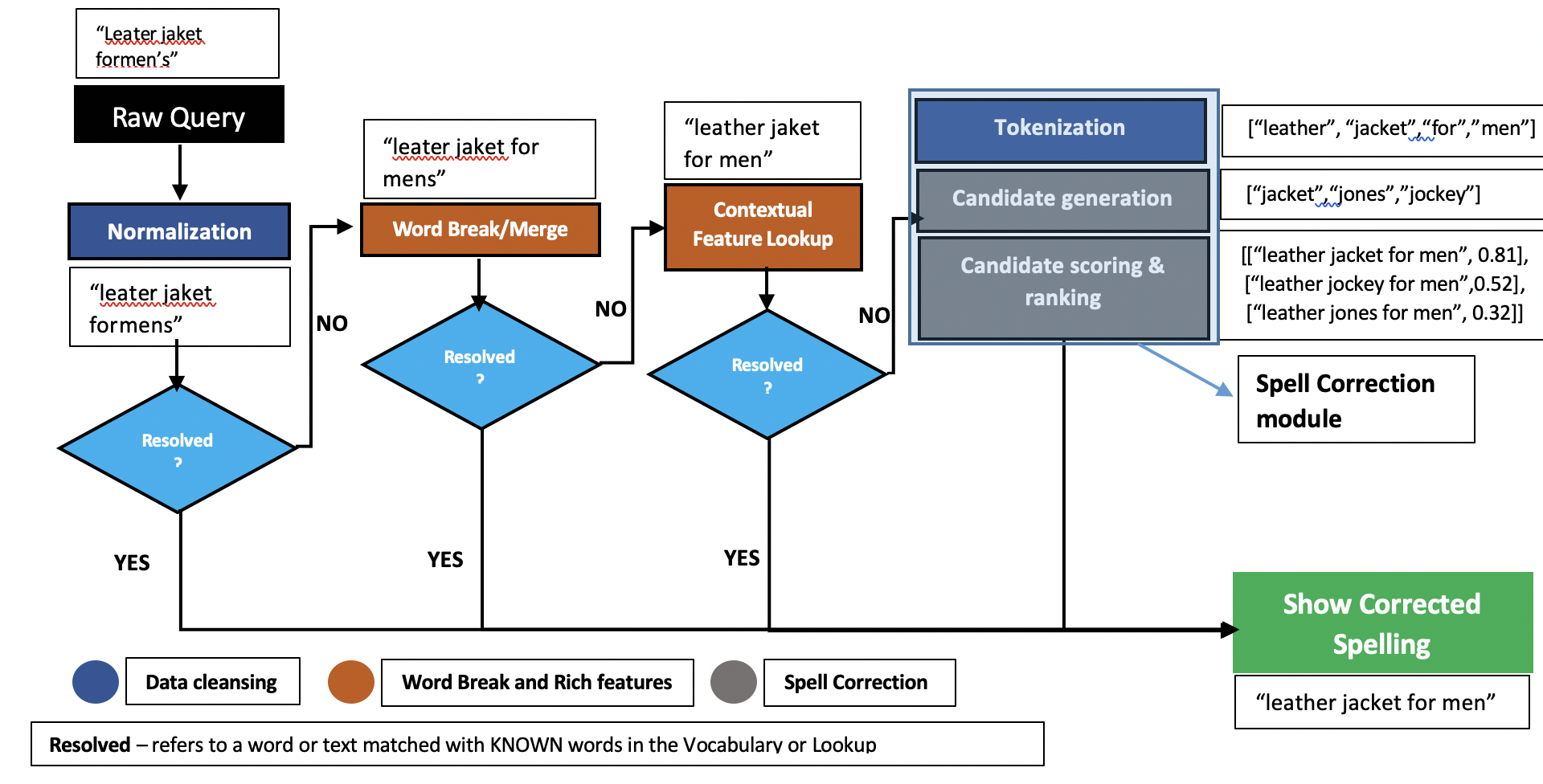
**Tokenization API**

Text is split into tokens

**“mens running shoes”**

**[“mens”,”running”,”shoes”]**

**Journey of a Query in the Preprocessing Module**

****

**Key Features of Spell Correction**

The Spell Correction Module powered by AICoE, comes with the following unique features –

1. **Context Specific enriched features**

Corpus containing common spell variants specific to a domain help in quick spell correction based on the mistakes usually committed while typing a query

pent 🡪 pant

tometo 🡪 tomato

cusion 🡪 cushion

1. **Phonetic aware Spell Correction**

Spell variants including similar sounding words and vowel elongations can be handled efficiently by the Spell Correction module. Below are a few examples –

Waalpaper 🡪 wallpaper

Phish 🡪 fish

Kream 🡪 cream

1. **Correction for spell errors with High Edit Distances**

Wherever there are spell errors with high edit distance, the context awareness aspect kicks in and helps retrieve close matches for spell mistakes with edit distance greater than 2

Bahroeb 🡪 bathrobe

1. **Word Break/Merge**

The text can either be broken into tokens or 2 or more different tokens can be merged into a single token based on the Vocabulary. Incorrect word break or join can result in Irrelevant or NO results from the Search Engine. Consider the following examples –

**Word Break** – Allen Solly is a popular fashion brand and when the space between the 2 words is missing, then the word break should happen to get the actual brand name

“allensolly pants”

“allen solly pants”

Word Break/ Merge API

**Word Merge** – Anarkali Dress is a popular style in Women’s wear and when this is broken into 2 words, then the words should be joined to get the actual style

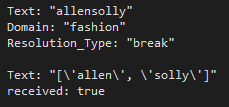
“anar kali dress”

Word Break/ Merge API

“anarkali dress”

**Types of Knowledge Nuggets used**

* Spell variations (Domain specific, shrt🡪shirt, sauc🡪sauce etc.)
* Indic entities (Domain specific, pyaaz🡪onion)
* Brand Names and variants (h and m🡪h&m)

**Spell Correction - Components**

**Response from the Server**

def GetServerResponse(self, request, context):

# get the string from the incoming request

message = request.Text

domain = request.Domain

if domain.lower() == “fashion":

res = wbg.word\_break(message)

print(res)

result = {'Text': str(res), 'received': True}

**The following are the Components of the Spell Correction module**

1. **\*Tokenization** – Shell Tokenization is performed to split the text to retain n-grams within Double quotes (identified from the Contextual/ Indic Lookup) as opposed to the splitting by white space.
2. **Candidate Generation** – The probable candidates are identified for Correction if they are NOT a part of Vocabulary and KNOWN words from the vocabulary which are within 2 edit distances are generated from the misspelt word.
3. **Candidate Scoring and Ranking** – the word Candidates corresponding to the misspelt words are generated and scored based on their frequencies and probability of mutual information (if the candidates form word-pairs) and sorted in descending order

\*This is a part of the Data cleansing module consumed in Spell Correction

**Spell Correction Module – Types of Implementation**

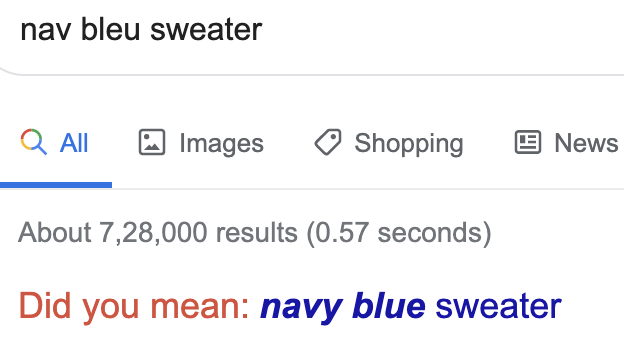
**Offline**

This module kicks in only after the entire text typed by the user is fed into the system. Predictions are made based on the trained vocabulary after generating the candidates wherever the Spell Correction is necessary. This considers the context of the words appearing in the given text. An example is given below for the same

**Features** – Context specific correction

**Downsides –** High latency, cannot handle generic correction

**Eg** – User types in nav bleu sweater, give the context of sweater, it corrects it to navy blue sweater confidently. In Google world, it asks if the user meant that



**Online**

This module has a cached version on type along Spell Suggestions, which works as the user starts typing using character sequences. It has a pre-trained model for common spell variations as shown in the below example. Cached versions of the models are popular in this particular mode of Spell Correction

The online spell correction differs from the Online version during the real time suggestions based on the following when it comes to candidate generation –

* Word suggestions based on the sequence of words typed by the user
* Correction based on word transition probabilities (Eg: **Sjirt** could be **shirt** or **skirt** depending on user search history which determines the transition probability between words)
* Correction based on phonetics (**Eg:** **“f”** sounds same as **“ph”** and vice-versa, which the system should be able to suggest as a correction based on the character sequence)

**Character transition as a filtering criteria for candidate generation**

**Identify candidates**

**Preprocessing**

**Raw Query**

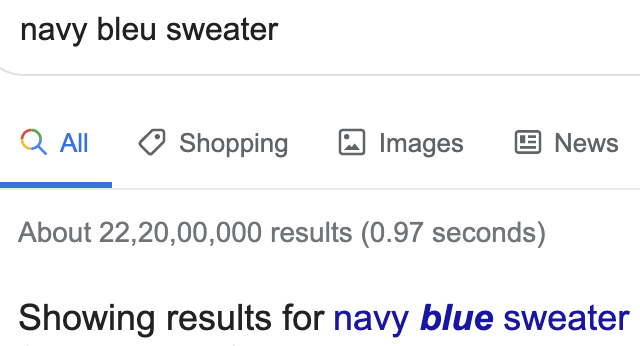
**Filter candidates which result in high character transition probabilities**

**Compute Character transition probabilities**

**Features** –

* Keeps users informed of potential errors as they type
* Spelling errors and the resulting ambiguities can be eliminated even before issuing the query
* The ability to suggest popular completions from corrected partial queries can improve the effectiveness of the suggestions.
* In instances when NO RESULTS are returned, character level transitions help in suggesting the correct spelling for a given variation

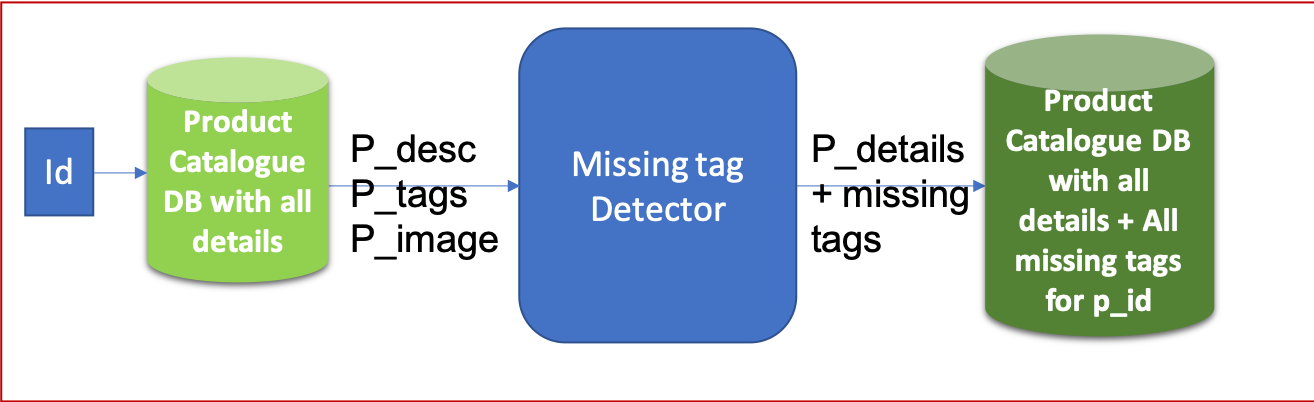
**Downsides –** Cannot handle contextual correction



## **Product Tagging** (tag types: missing, explicit, implicit, popularity and inventory based)

### **Missing tags**

* Identify any missing tag corresponding to any product id and then add them to make the product more searchable.
  + - * attributes like category, brand etc. are often missing for some products which makes them non-searchable which leads to lower sales
* Missing tags can be of various type like thematic tags (beach wear) or enriched tags (most sold) or Indic tags or can be normal product tags



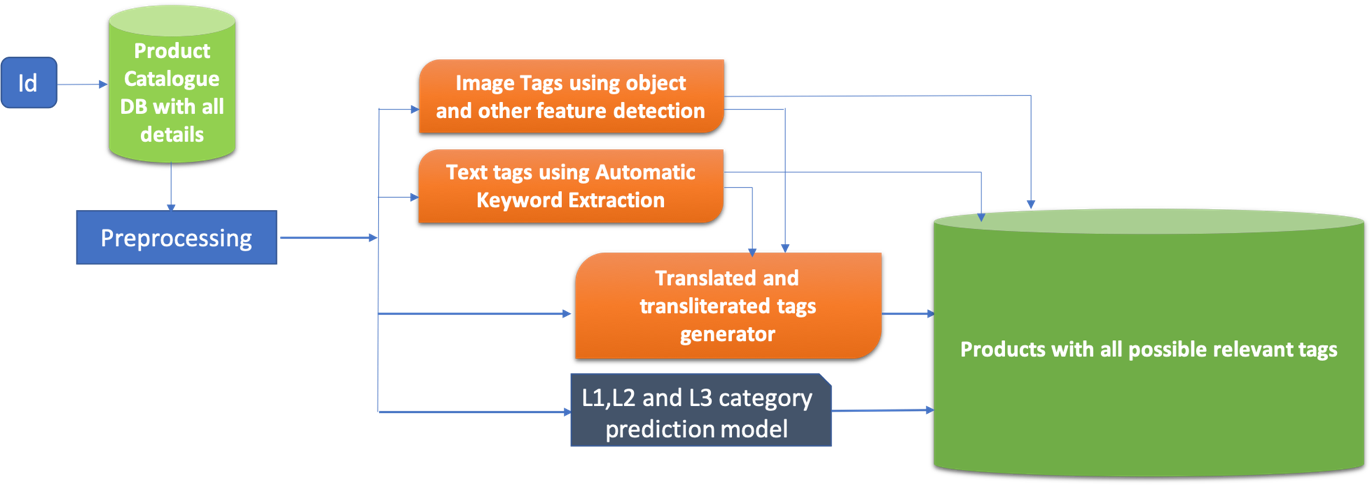
### **Explicit tags**

For product catalog, this will be a one-time process and hence this service can be run as an offline service or can be exposed as an API to be used for other services.

This service will generate 4 different type of tags as follows:

* Textual tags from product description
* Image tags (tags extracted from objects and other features in image)
* Indic-tags (translated and transliterated tags)
  + Ex: Aloo, Batata, Uralaikilangu, Urlagadda, आलू, અલૂ, बटाटा, ಆಲೂಗಡ್ಡೆ, ಉರಾಲೈಕಿಲಂಗು
* Category hierarchy, brands and quantity

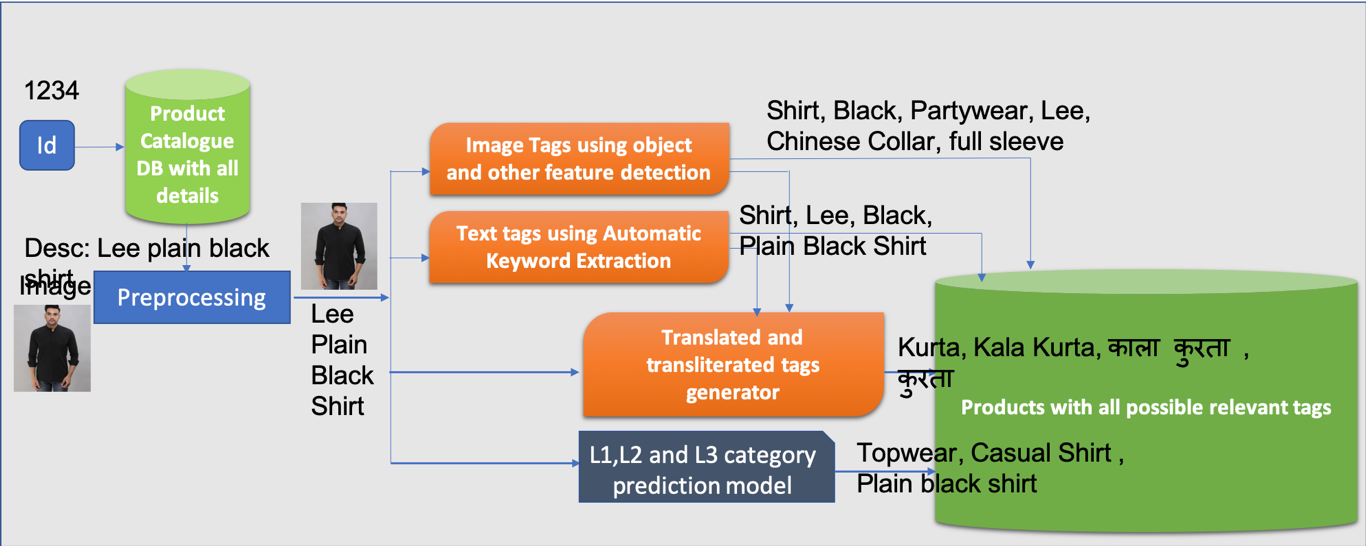
Below is the flow diagram of explicit tag generator.



For generating explicit tags, we need product description and image corresponding to product Id.

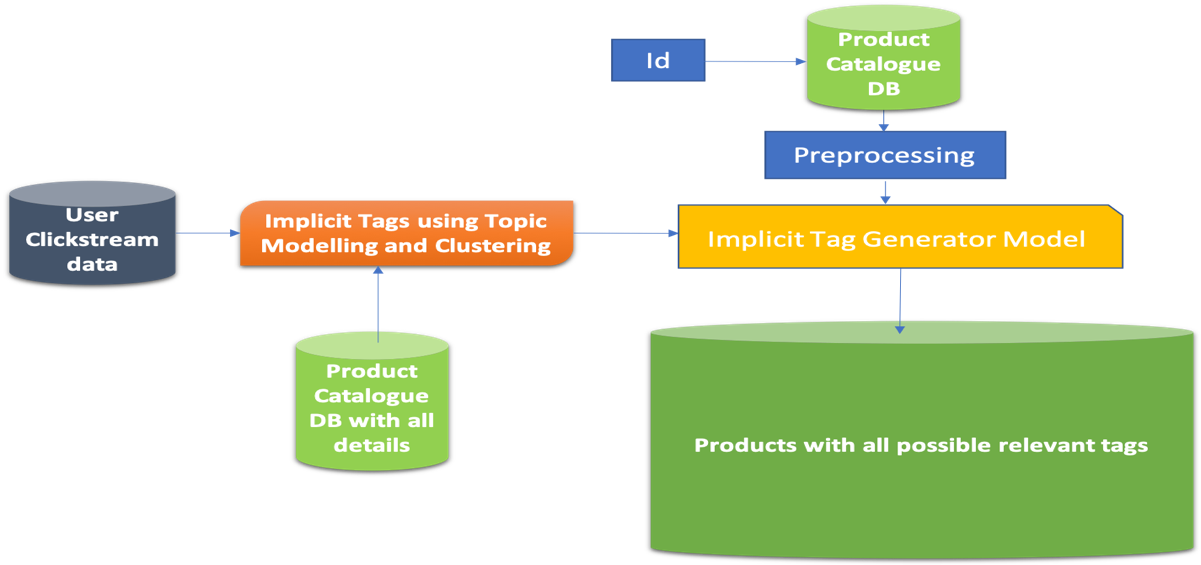
* Pass Product text to keyword Extraction service and product image to image detection service.
* Keywords generated from above step will be passed to translate and transliterate the translated keywords to form Indic-aware keywords.
* Current l1, l2 and l3 categories will also be a part of explicit tags.

Consider an example,

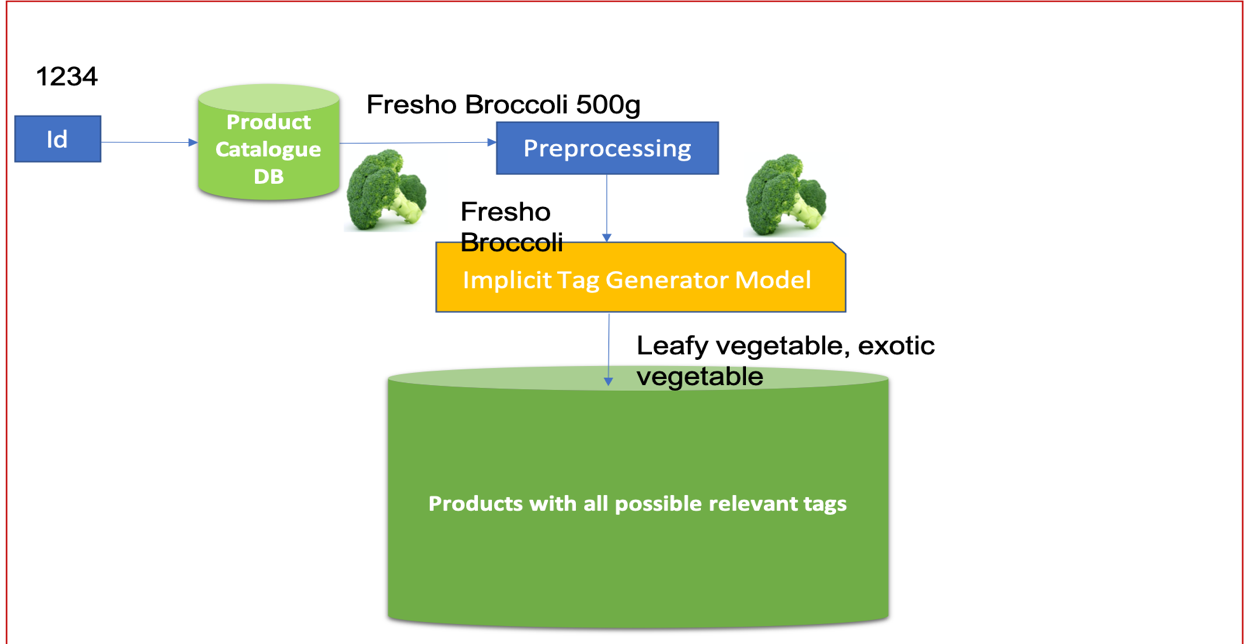


### **Implicit tags**

* Implicit tags are generated when products are grouped or clustered along different dimension. They can be occasional or thematic in nature.
* Implicit tags generation model can be trained using clustering after computation of embeddings of each product base on the click data of each session.
  + example: Jacket -> Winterwear
* This model can be updated in batch process using weekly or monthly cronjob.

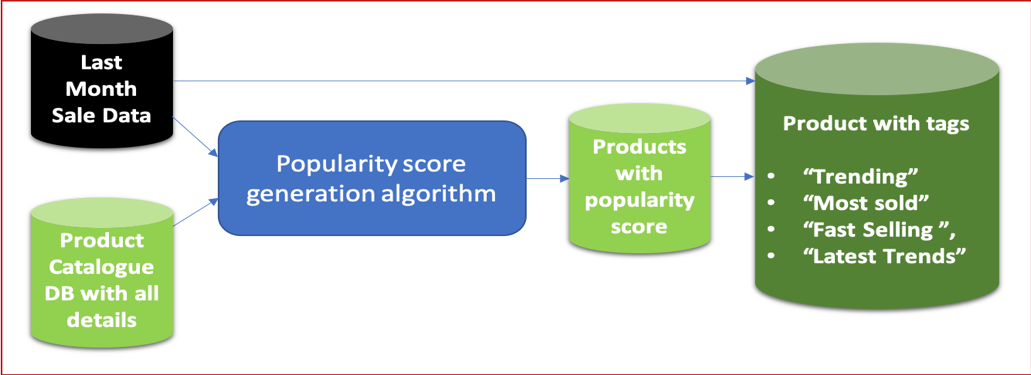


Consider an example,



### **Popularity based tags**

* Based on the sale data, **popularity score** for each product will be calculated and with other sale details, different tags will be assigned to the products.
* These tags will be updated **daily/ twice a week** depending upon the data.



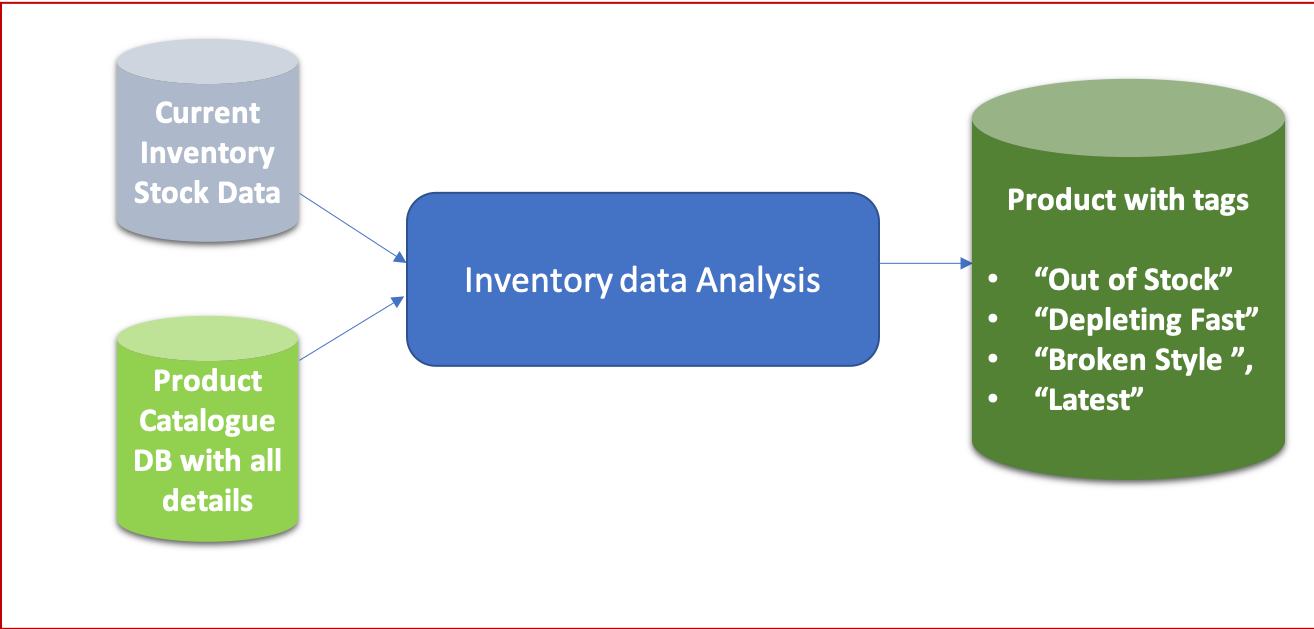
Common tags:

* **Most popular/Trending** products in category/overall. For example, “Plain sports Nike Shoes” sale rate increased exponentially can be marked “Most Popular”.
* **Fast selling** products (can be finalized based on the rate of sale).

“Ipad pro” whose sale was exponential recently can be marked as “Fast Selling”.

* **Most sold** products in category – Mainly based on the sale volume of products
* **Latest Trends**
* Mainly for fashion e-commerce.
* Will be given to products of some specific types whose CTR/conversion is maximum in last month (can be changed based on sale volatility).

### **Inventory based tags**



* **Out of stock**.

All the products having stock count 0 will be tagged as “Out of Stock”.

* **Depleting Fast**

Products whose rate of sale will be more than a threshold and whose stock count will be less than a threshold number can be tagged as “Depleting Fast”.

* **Broken Style (Fashion Ecommerce)**

Products whose stock count is 0 for one or more sizes but is available in some other size can be tagged as “Broken style”.

* **Latest products in category**

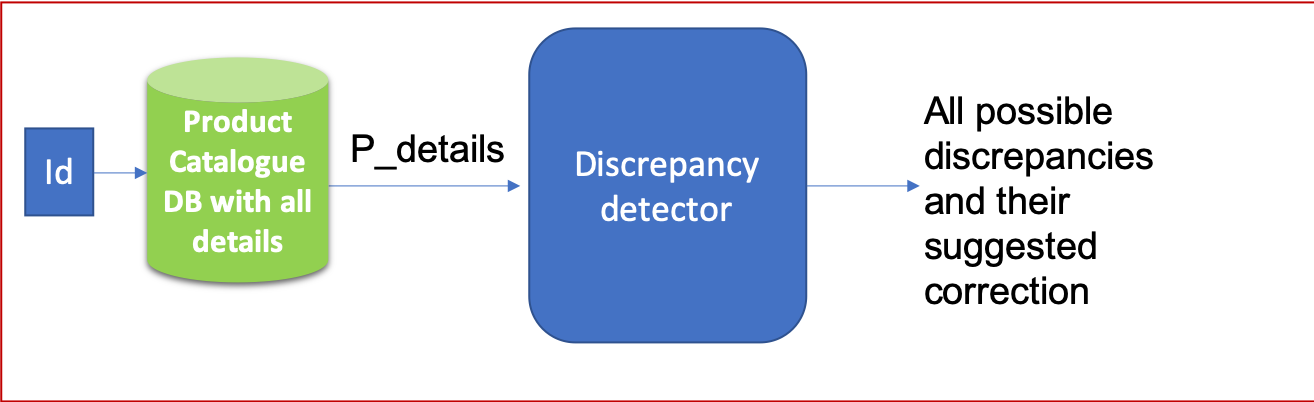
Most recent in inventory for any category can be marked as “Latest”.

## **Discrepancy detection in catalog**

* Different kind of discrepancies present in the catalog:  
  + wrong **brand**
    - “**Red Tape**” brand name for “**Nike shoes**”
  + wrong **categories** (l1, l2 or l3)
    - “**Fruits**” category tagged for “**Orange Juice**”
  + **merged** **categories** (l1, l2 or l3)  
    - categories with **&** between two different category types
      * ex: “**cucumber and capsicum**”
    - such categories need to be resolved to either cucumber or capsicum, **merged categories leads to** lot of **search relevance issues**
  + wrong **category hierarchy**
    - same category present at two different levels
    - or a category having two different parent categories
  + wrong /missing **image**
    - “**Shoe**” image for a product **“Half sleeve shirt**”
    - missing images can again reduce user trust and should be present
  + **duplicate tags** (with slight difference)
    - these two categories (**“dry fruits”, “dry fruits & nuts”**) should be merged, having them as two different categories can create problems in relevance

**Solution**: 

* For each product, there should be a validation algorithm/process which can check if all the data provided is correct/valid.



Many of the above observed problems like **category** **redundancy, missing categories, wrong category hierarchy** (multiple l2 for same L3) etc**.** mainly occur due to **Manual Tagging.**

We can fix this using below steps.

* An **auto-tagging trained model** (can be trained be for any domain) which can assign these categories based on description.
* So, in case of missing categories, those will be auto-filled from model prediction.
* Category redundancy will also be removed as model predicted category name will always be same for each category.

Now, In case of poor confidence of a particular prediction, we will have to proceed manually. For that case,

* There should be a fixed set of categories in the dropdown from which we have to select a particular category.
* For each L1, there should be a fixed set of L2 categories dropdown (only categories belonging to that L2).
* Similarly, for each L2, there should be a fixed set of L3 categories dropdown.

Data Requirements

# for **Catalog Enrichment Service**

## **Product Catalog**

* List of all available products that are being offered on the platform, along with the relevant information about each product such as mentioned below:
  + title
  + brand
  + complete category hierarchy
  + quantitative information about the product such as weight, volume, price, size etc.
  + Inventory and Discount that is being offered at different stores
* Product catalog data is dynamic, it’s attributes might change with different store/location and time. We need to have access to the latest catalog dump that is being used for POS machines, along with all the attributes as mentioned in the previous point.

## **Clickstream Data**

**query logs:**

* complete list of queries searched by users
* for each query we need
  + total search count:
    - # of times the query was searched by any user
  + clickthrough rate:
    - (# of times a query is searched and led to a click)/(total search count of that query)
  + add to cart rate:
    - (# of times a query is searched and led to a product being added to cart)/(total search count of that query)
  + order rate
    - (# of times a query is searched and led to a product being ordered)/(total search count of that query)
  + query to product mapping
    - list of all the products that were interacted with for the given query
    - for each product:
      * separate count of click, add to cart, and order

**session data:**

* for each session of a particular user on the search platform, we need the below data:
  + user metadata
    - gender, device type, session id, location (latitude, longitude) etc.,
  + transactions made by user such as
    - query search, page view, add to cart, order
    - all these transactions when done in a same session should be tagged with same session id

## **Synonym List**

This data enables the Spell check module to identify synonyms for search terms for example:

* **magenta** 🡪 **purple**
* **pillow** 🡪 **cushion** etc.