**Spell Correction 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

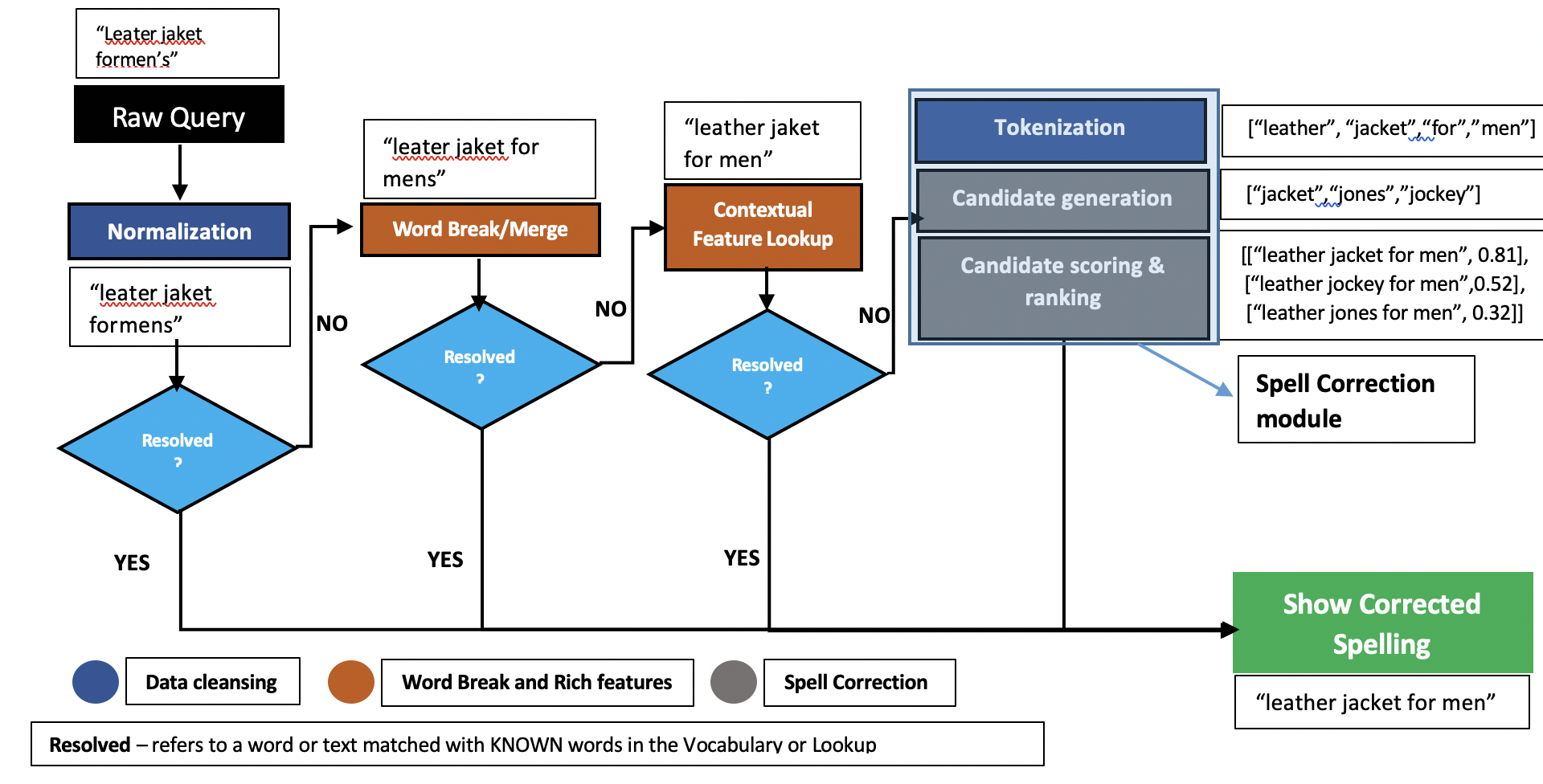
**Tokenization API**

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

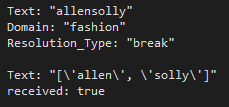
“anar kali dress”

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**

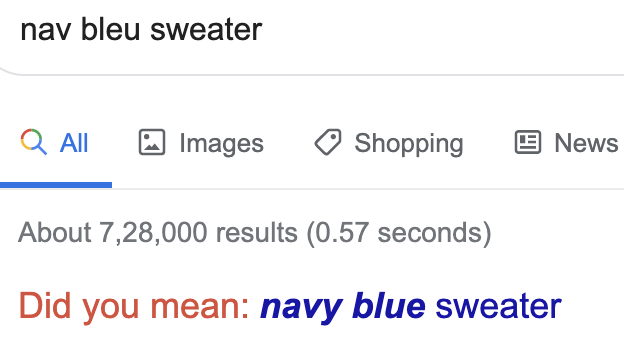
**Offline**

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**

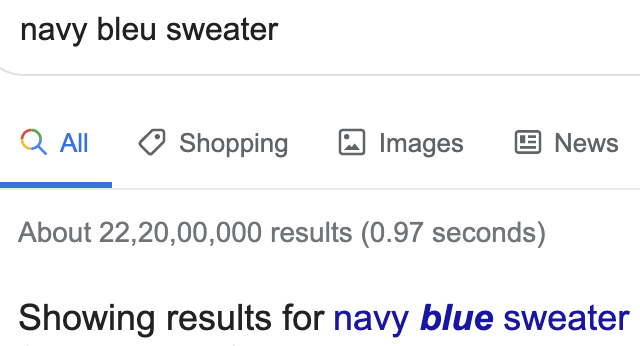
**Filter candidates which result in high character transition probabilities**

**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



**Data Sources for the Spell Correction**

The following sources of data as Input for better performance along with preserving context during Spell Correction

**Catalogue Data**

Contains catalogue and product definitions along with the corresponding attributes along with scraped 3rd party data sources

**Query Logs**

Contains user types query dump these can be extracted as per the domain. This would have the variations of most common words typed by the users

**Contextual Entities Data**

Contains catalogue and product specific Keywords and Domain specific entities and their types along with their common variations. This is usually a lookup list or a Dictionary **Eg:** **jenes** is the misspelt variation of **jeans** in Fashion Domain

**Indic Entities Data**

Contains the Indic or Local variations of a given word as compared to English language. **Eg:** Pyaz is the indic word for Onion in Hindi

**Session Data**

This data enables the Spell check module to suggest spelling specific to the user typing and search patterns also helps us identify the errors from a particular geography etc.

**Synonym list**

This data enables the Spell check module to identify synonyms for search terms for example **magenta 🡪 purple, pillow 🡪 cushion** etc.