TAG SEARCH APP

Abstract

This document discusses installation steps, algorithm and improvements.

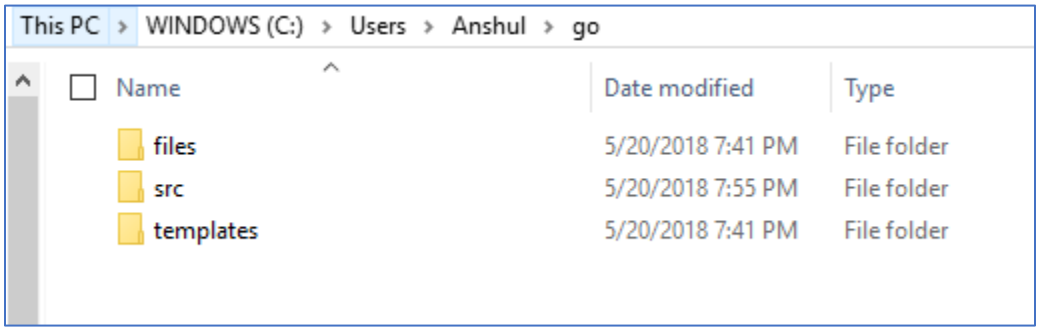
Anshul Prakash

Anshul_prakash@tamu.edu

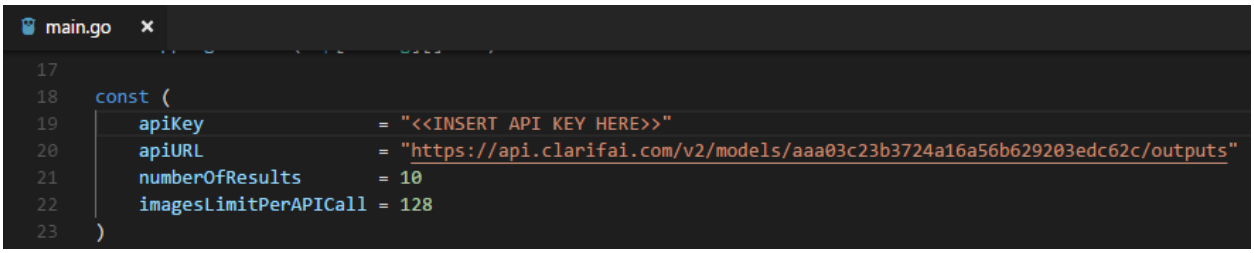
Table of Contents

To run the application:

* Copy contents of project folder – “src”, “files”, “templates” into go directory i.e. %USERPROFILE%\go in default installation of go



* paste Clarifai’s API key in src\tagSearchApp\main.go at line 19



* run the following commands from GOPATH set in the environment variables i.e. %USERPROFILE%\go in default installation of go

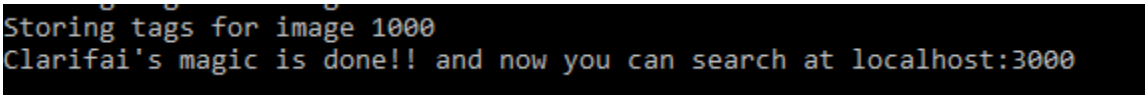
* `go install tagSearchApp`

```
C:\Users\Anshul\go>go install tagSearchApp
```

* `.\bin\tagSearchApp.exe`

```
C:\Users\Anshul\go> .\bin\tagSearchApp.exe
API Call Number = 1 of 8
Storing tags for image 1
Storing tags for image 2
```

* Calls will be made to Clarifai's API to tag the images and once all images get tagged a success message will be displayed in the console and application can be accessed at localhost:3000



Solution Design:

The main driving force behind the design was to provide quick search results. So, all the preprocessing for tagging images and creating a mapping for tags and corresponding images in order of probabilities is done when the app is started. So, when the user searches through the webpage then AJAX calls are made to the server to get image URLs instantly. The steps of algorithm are as follows:

1. Read all image urls from text file and store them in an array structure
2. Iterate over the urls array 128 elements at a time to minimize the number of calls made to Clarifai's API as we can send max 128 images in a single API call
3. For each response returned from API store the tags corresponding to each image in Map with tag as key and an array of struct containing image URL and probability as values.
4. Sort all the values corresponding to a tag by the probability of that for an image

The mapping obtained using this algorithm is used to return at most top 10 images corresponding to a tag entered by the user in the web page.

Algorithmic Efficiency

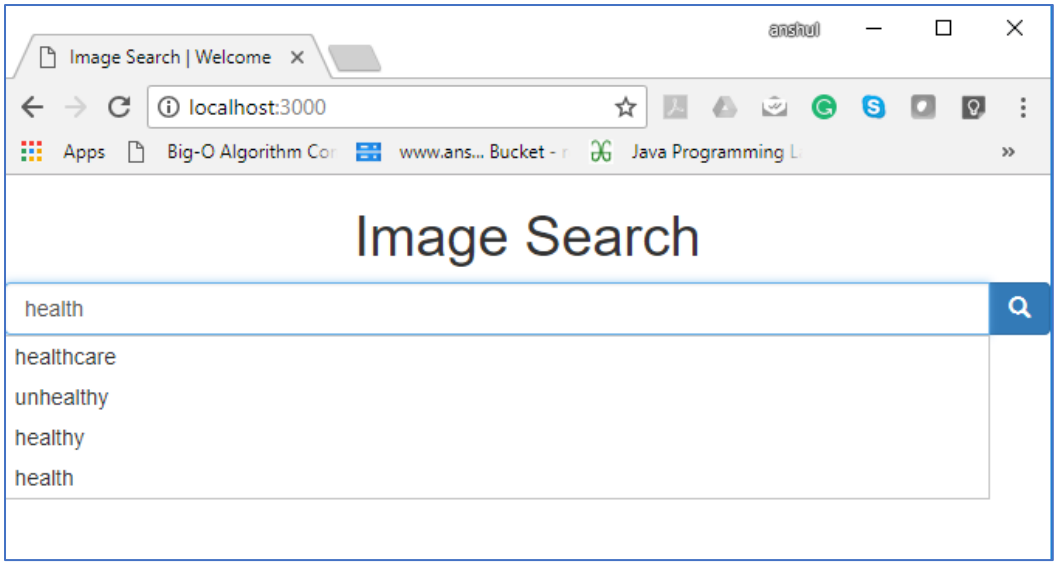
Not considering the time lapsed between API calls the efficiency of the solution is defined by two factors:

1. Iterating over all image urls and storing tags corresponding to each image in the map. If there are m images then this operation is O(m * number of tags of each image) as updating the map will be O(1) for each tag
2. Sorting the arrays corresponding to each tag in the map. This operation will be O(total number of tags * (number of images corresponding to each tag)* log(number of images corresponding to each tag))

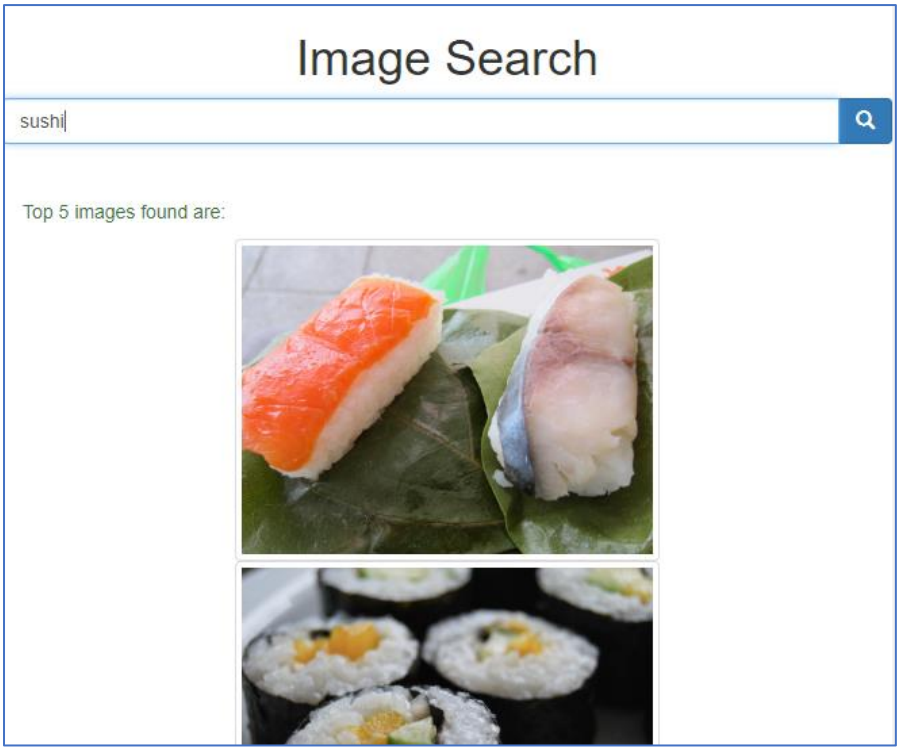
Another option is to maintain a heap corresponding to each tag key in the map. But adding a new image corresponding to a tag would have costed O(log(number of elements in the heap)) and ultimately solution would have become O(m * number of tags of each image * log(number of elements in the heap))

Web Interface:

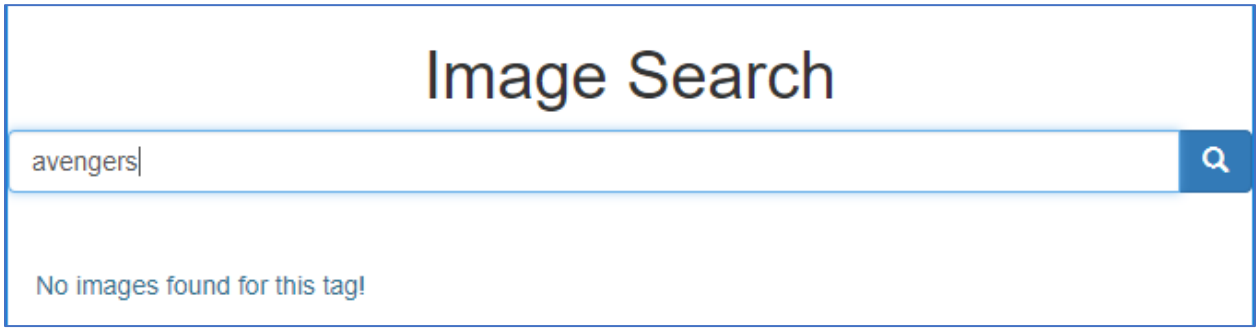
Web page comprises of an input field to search a tag which also has an autocomplete functionality. The web page is also responsive as Bootstrap has been used for creating it.



Search results and tags for autocomplete are fetched by making AJAX calls to the server. All tags are fetched during the page load for autocomplete functionality.



If a tag is entered that is not present in the mapping, then “No Images found for this tag” is shown to the user.



Improvements

1. The current solution doesn’t return result for a combination of tags. It only returns results for a single tag. This can be improved by splitting the input search string and then searching for images corresponding to each tag and then finding intersection of results. Ordering of results would require calculating probability of intersection using individual probabilities
2. Current Solution also doesn’t consider typing errors or misspelled words. This can be improved by maintaining a map of frequently misspelled words for quick lookup.
3. Currently there are only 1000 images and mapping of corresponding 2127 tags is being stored in memory. As the app scales and the number of images increase we would need to persist this mapping in a database and may be maintain most frequently searched tags using 80:20 principle in a cache solution such as Redis.