BABEŞ BOLYAI UNIVERSITY, CLUJ-NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

HOLIDAY PLANNER: FIND THE BEST LOCATION WHICH SATISFIES THE INPUT

– MIRPR report –

Team members
Bangău Andrei-Ciprian
Beltechi Paul
Buciu Ştefan
Şurubariu Răzvan

Abstract

Finding the best location based on any input such as text/voice descriptions, custom filters or images is important because it can be abstracted for more complex searches like browsing the internet based on more specific input, like finding your favourite tune just by recording yourself singing it or finding your pet's breed by taking a photo. We consider that in order to find a specific content there must be also some form of specific input which can be given to find the best match, thus making the searching process more efficient. This article presents the steps made, so far, on implementing a search system for holiday recommendations

Chapter 1

Introduction

1.1 What? Why? How?

Find the best location based on user's input which can be anything from text, filters to images. The purpose of this is to offer best results to a specific search problem such as finding the best location for your vacation.

This problem was a little bit challenging in the way of choosing the best algorithm to perform the search based on the input which can be numerical (price, no of bedrooms), string (description of the vacation, location), jpg image (user wants to visit something close with this photo) or voice (user can record the desired description of the holiday).

So, taking them one by one, we decided to use basic filtering for numerical input, a content-based recommendation system for string input, which recommends vacations to a user by taking the similarity of the descriptions of a vacation and for images we chose the pre-trained DELF (DEep Local Feature) module which was trained with Google-Landmarks data-set and basically describes each noteworthy point in a given image with 40-dimensional vectors known as feature descriptor.

1.2 Related work

Something related to our problem is building a city recommender which shows where you should travel next. The approach is: collect data on different cities, do topic modelling on Wikipedia entries and city guides using Gensim, then use clustering algorithms to group similar cities together. So the cities grouped in the same clusters as the cities already visited are the ones which the user should travel to next.

How does this recommender make recommendations? It calculates the similarity between the user's taste and each of the users in the data-set, based on the cities they traveled to, by paying more attention to the users who have similar taste and less attention to those with different taste, it makes a list of cities the user might like, and it gives each city a score that sums up how often that city would have been recommended by the users with similar taste.

These types of algorithms recommend some things which are similar to the ones that a user has liked/viewed in the past.

For example, if we take any types of data like movies, if a person has liked the movie 'Inception', then this algorithm will recommend movies that fall under the same genre. But how does the algorithm understand which genre to pick and recommend movies from?

Consider the example of Netflix. They save all the information related to each user in a vector form. This vector contains the past behavior of the user, i.e. the movies liked/disliked by the user and the ratings given by them. This vector is known as the profile vector. All the information related to movies is stored in another vector called the item vector. Item vector contains the details of each movie, like genre, cast, director, etc.

This algorithm is called content-based filtering and finds the cosine of the angle between the profile vector and item vector, i.e. cosine similarity. Suppose A is the profile vector and B is the item vector, then the similarity between them can be calculated as:

1. Based on the cosine value, which ranges between -1 to 1, the movies are arranged in descending order and one of the two below approaches is used for recommendations:

- Top-n approach: where the top n movies are recommended
- Rating scale approach: where a threshold is set and all the movies above that threshold are recommended

2. Other methods that can be used to calculate the similarity are:

- Euclidean Distance: Similar items will lie in close proximity to each other if plotted in n-dimensional space. So, we can calculate the distance between items and based on that distance, recommend items to the user.
- Pearson's Correlation: It tells us how much two items are correlated. Higher the correlation, more will be the similarity.

A major drawback of this algorithm is that it is limited to recommending items that are of the same type. It will never recommend products which the user has not bought or liked in the past. So if a user has watched or liked only action movies in the past, the system will recommend only action movies. It is a very narrow way of building an engine.

1.3 Our work

So our problem does not store information related to users, we take only the user's preferences. For numerical and string inputs we used an airbnb data-set which contains the price per night, bedrooms count, a description of the place, a title, the location, and a airbnb link for booking. For the image inputs we used Google's Landmarks data-set v2.

Our first approach was to use a decision tree classifier. Decision trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. So the numerical features were just plugged in, but for the string features we decided to use Google's pretrained model of word2vec. Simply put, the word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. So for each description sentence we took each word and we got the vectorized string from word2vec, which in our case would be a 300 float element vector, and then we made the average sum of all word vectors from the current sentece. Finally, for one string feature we would have a vector of length 300. We decided the output label should be the city.

After that, we wanted to evaluate the algorithm but this was kind of problematic too because take the price feature, for example, if the user puts the price 100$ we cannot evaluate this only by comparing if the predicted result has the same price, we would want to take one of the results with price lower than 100$ and give a higher importance to the ones that have a smaller price. So our decision was for the moment just to compare the predicted output (the city) and see if they are equal. Finally, we normalised the features and got a 0.14 accuracy.

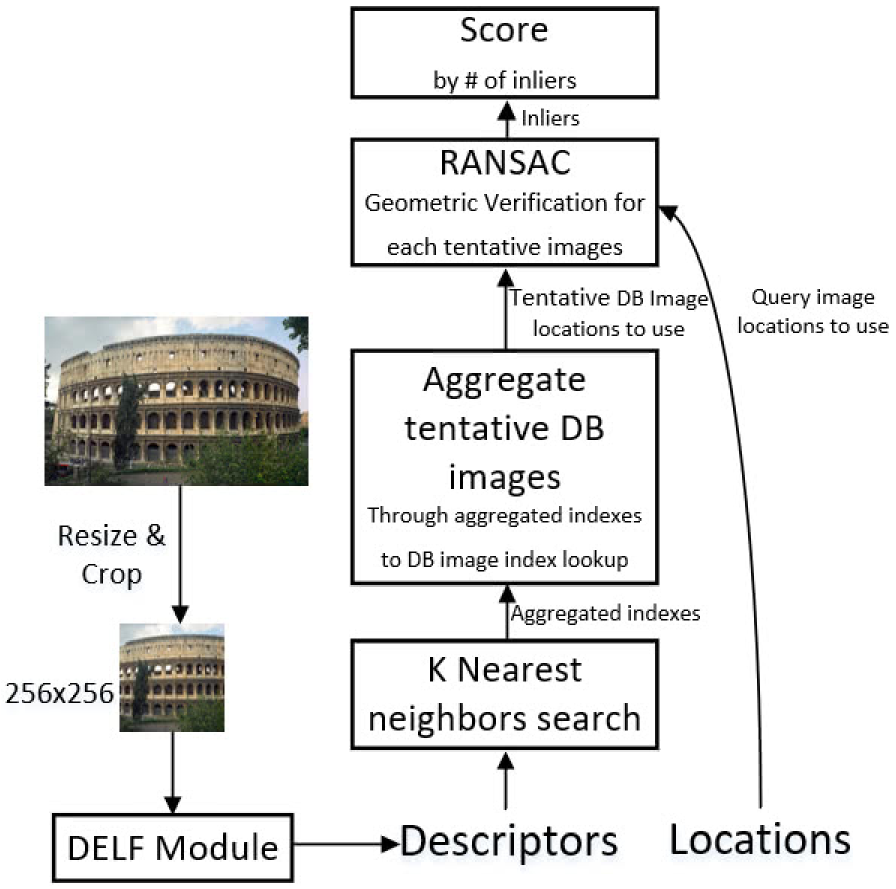
Our second approach was to exclude the decision tree classifier and include the TF-IDF technique. TF-IDF is a term frequency-inverse document frequency. It helps to calculate the importance of a given word relative to other words in the document and in the corpus. It calculates in two quantities, TF and IDF. Combining two will give a TF-IDF score. So we combined the average Word2Vec and TF-IDF and we got again a 300 float element vector. Then, for each string feature we decided to use cosine similarity to find the most similar results.

For the image feature, we used the pre-trained DELF (DEep Local Feature) module, which can be used for image retrieval as a drop-in replacement for other keypoint detectors and descriptors. It describes each noteworthy point in a given image with 40-dimensional vectors known as feature descriptor.

The DELF Image retrieval system can be decomposed into four main blocks, which will be emphasized later:

- Dense localized feature extraction
- Keypoint selection
- Dimensionality reduction
- Indexing and retrieval

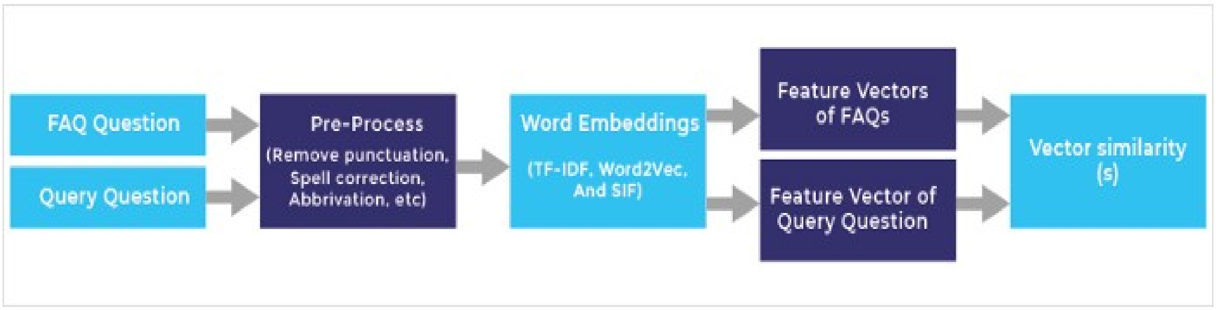
We started by extracting feature descriptors from the Google's Landmarks data-set images and aggregate their descriptors and locations. Our image retrieval system is based on nearest neighbor search, to facilitate this, we used a KD-tree with all aggregated descriptors. At runtime, the query image was first resized and cropped to 256x256 resolution followed by the DELF module computing its descriptors and locations. Then we query the KD-tree to find K nearest neighbors for each descriptor of the query image. Finally, we perform geometric verification using RANSAC and employ the number of inliers as the score for retrieved images.


Score
by # of inliers
Inliers
RANSAC
Geometric Verification for each tentative images
Tentative DB Image locations to use
Query image locations to use
Aggregate tentative DB images
Through aggregated indexes to DB image index lookup
Aggregated indexes
K Nearest neighbors search
Resize & Crop
256x256
DELF Module
Descriptors
Locations

Random sample consensus is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. Therefore, it also can be interpreted as an outlier detection method.

Putting it altogether, for the image feature we predicted a label which in our case is the location. Therefore, this will have an impact when calculating the cosine similarity, along with the description features, making it more precise because more information is given by the user.

The text processing flow is well explained in the image below.


FAQ Question
Query Question
Pre-Process
(Remove punctuation, Spell correction, Abbrivation, etc)
Word Embeddings
(TF-IDF, Word2Vec, And SIF)
Feature Vectors of FAQs
Feature Vector of Query Question
Vector similarity (s)

Chapter 2

Scientific Problem

2.1 Cosine similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a term-frequency vector. For example, in Table 2.5, we see that Document1 contains five instances of the word team, while hockey occurs three times. The word coach is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Table example: (see Table 2.1)

Table 2.1: Document Vector or Term-Frequency Vector.

Document	**team**	**coach**	**hockey**	**baseball**	**soccer**	**penalty**	**score**	**win**	**loss**	**season**
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

Term-frequency vectors are typically very long and sparse (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, biological taxonomy, and gene feature mapping. The traditional distance measures that we have studied in this chapter do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents do have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let x and y be two vectors for comparison. Using the cosine measure as a similarity function,we have

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||},$$

where ||x|| is the Euclidean norm of vector

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_p),$$

, defined as

$$\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}.$$

Conceptually, it is the length of the vector. Similarly, ||y|| is the Euclidean norm of vector y. The measure computes the cosine of the angle between vectors x and y. A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

2.2 Average Word2Vec

Lets take an example which has 23 words. We will denote the words as w1, w2, w3, w4, ...w23. (w1 = william, w2 = bernstein, ..., w23 = oregon). Calculate Word2Vec for all the 23 words. Then, sum all the vectors and divide the same by a total number of words in the description (n).

$$\frac{W2V(w_1) + W2V(w_2) + \ldots\ldots + W2V(w_{23})}{N}$$

average word2vec

Here, vectors are in d -dimensional (used 300 dimensions) N = number of words in description 1 (Total: 23) v1 = vector representation of description 1.

2.3 TF-IDF Word2Vec

Again, the description has 23 words. We will denote the words as w1, w2, w3, w4, ...w23. (w1 = william, w2 = bernstein ..., w23 = oregon). Steps to calculate the TF-IDF Word2Vec: Calculate the TF-IDF vector for each word in the above description. Lets call the TF-IDF vectors as tf1, tf2, tf3 ... tf23. Please note TF-IDF vector won't give d-dimensional vectors. Calculate the Word2Vec for each word in the description Multiple the TF-IDF score and Word2Vec vector representation of each word and, total the same. Then, divide the total by sum of TF-IDF vectors. It can be called as v1 and written as follow:

$$\frac{tf1 * W2V(w_1) + tf2 * W2V(w_2) + \ldots\ldots + tf23 * W2V(w_{23})}{tf1 + tf2 + tf3 + \ldots tf23}$$

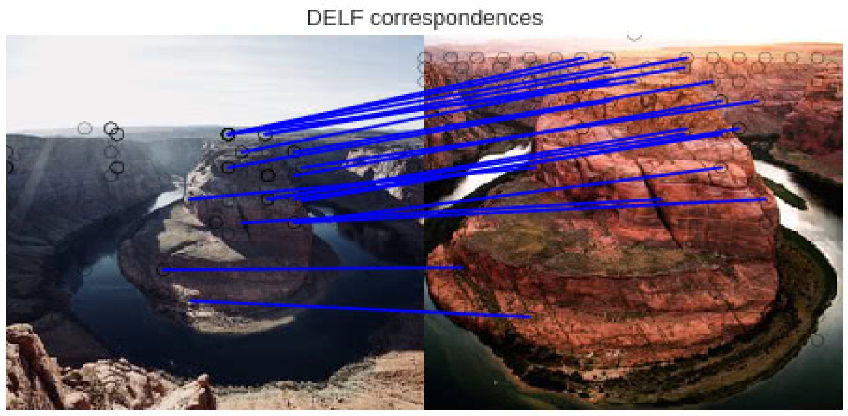
tf-idf word2vec

v1 = vector representation of book description 1. This is the method for calculating TF-IDF Word2Vec. In the same way, we can convert the other descriptions into vectors.

2.4 DELF(DEep Local Feature)

The pre-trained DELF module describes describes each noteworthy point in a given image with 40-dimensional vectors known as feature descriptor.

The image below shows the DELF correspondences of two images.



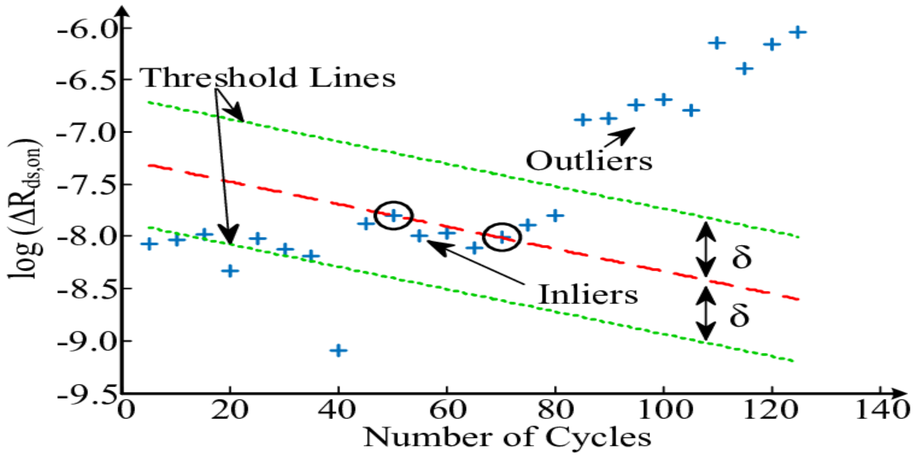
This result is optained in two steps

2.4.1 K Nearest neighbour search

Using a cKDTree built on the stored descriptors, we can then query relatively fast a new image and get the closest K neighbours, shrinking our search space to K images.

2.4.2 Random sample consensus (RANSAC)

The query result of the cKDTree is then filtered using RANSAC. We through each neighbours' image locations, along the query locations, and by using this algorithm, we can filter some more and find the best result based on the number of inliers.



Chapter 3

Application

Due to the nature of the problem, obtaining a mathematical result that evaluates the algorithm is another problem in itself.

Having as output a score given by cosine similarity the easiest solution would be to set a treshold value for the chosen text descriptions, for price filters the lower the value the better, for the bedrooms filter it should be an exact value.

Regarding the accuracy for image-based algorithms, the classification of the landscape is still an open problem, which sounds like based on a given image find if the output location is really the location of the picture. For the landmark, the features are matched using cKDTree, and then these results are filtered using RANSAC, choosing the result that contains the most inliers.

3.1 Methodology

- The criteria used to evaluate our experiment was how much we liked the recommendation and how close it was to what we expected.
- The hypotheses our experiment tests is that people would like to get recommendations for a holiday.
- The training test data is composed of real rented homes from Airbnb, this data is pretty much as close as you can get to what a customer would use when searching for a place to rent while on a vacation.

3.2 Data

- https://www.kaggle.com/google/google-landmarks-dataset
- https://www.kaggle.com/ponrajsubramaniian/vacational-places-image-dataset
- https://www.kaggle.com/rusiano/madrid-airbnb-data?select=listings-detailed.csv
- https://www.kaggle.com/tylerx/melbourne-airbnb-open-data?select=cleansed-listings-dec18.csv
- https://www.kaggle.com/liubacuzacov/stockholm-sweden-airbnb-listings?select=listings-detailed.csv
- https://www.kaggle.com/airbnb/seattle?select=listings.csv
- https://www.kaggle.com/labdmitriy/airbnb?select=listings.csv

- https://www.kaggle.com/oindrilasen/la-airbnb-listings
- https://www.kaggle.com/leonardopena/rio-de-janeiro-brazil-airbnb-data?select=listings.csv
- https://www.kaggle.com/fuyutaro/tokyo-airbnb-detailed-open-data?select=listings.csv
- https://www.kaggle.com/datafiniti/palm-springs-vacation-rentals
- https://www.kaggle.com/puneet6060/intel-image-classification

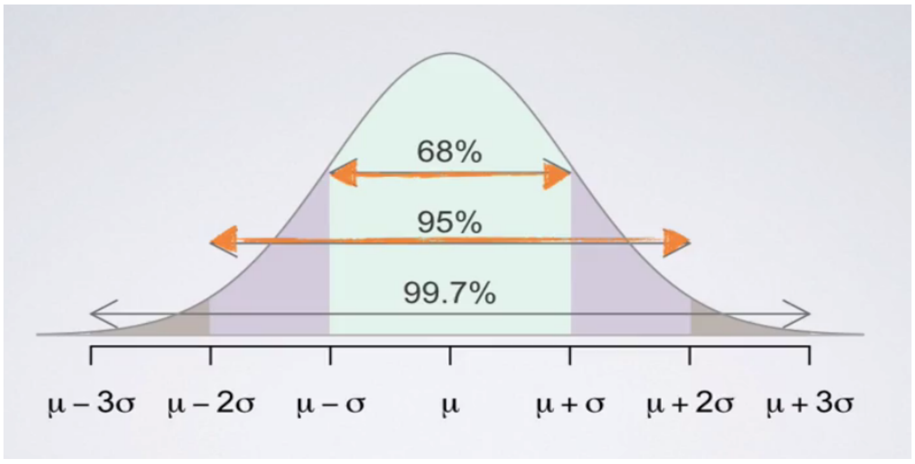
3.3 Discussion

A huge impediment was DELF. It is a ML algorithm that is trained to find the most important features from an image. This gives you the features, but it doesn't give you a label. To get the label you have to somehow match against known images by using these features.

Our approach is based on K Nearest Neibourgh Search and RANSAC, but this solution is not robust enough for the Google Landmark Dataset which contains hundreds of thousands of images that account for over 100GB of data.

By using cKDTree and RANSAC, how is it possible to fit that much amount of data in RAM? We thought of a way that comes with some trade offs.

First we need to understand the Location and Description features. The location is a tuple (y, x) in the range of [0, imageHeight], [0, imageWidth]. The descriptor is a feature of shape (40, x) where x is the number of focus points found by DELF. These 40 values range between [-1, 1] describing the importance of that point.



These values are continuous and we thought of transforming them in discrete values. We achieved that by defining interval ranges which are binned by some granular value and then we check if a feature value goes in a range, then selected the left part of the interval.

For example, a location feature (12.9162, 56.81942) would transform in (12.0, 56.0), thus shrinking the cardinality of the values.

After that, we built up a database where we set unique constraints on the values and on collisions we didn't save the values twice. This helped us making the dataset smaller and easier to fit in RAM.

Chapter 4

Conclusion

Finally, following the development of this application, we learned to combine word processing techniques such as TF-IDF and Word2vec and classify the results obtained using a score. Also methods of image processing and classification.

The datasets used were significant in size and taught us to question the robustness of the programs we write.

Bibliography

[1] https://www.datarevenue.com/en-blog/building-a-city-recommender-for-nomads/
[2] https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/
[3] https://www.dlology.com/blog/easy-landmark-image-recognition-with-tensorflow-hub-delf-module/
[4] https://en.wikipedia.org/wiki/Random_sample_consensus/
[5] https://medium.com/towards-artificial-intelligence/content-based-recommendation-system-using-word-embeddings-c1c15de1ef95
[6] https://arxiv.org/pdf/1612.06321.pdf