Photo Tag Recommendation System
Multimedia Systems and Applications 4/M Coursework Submission

Magda Kowalska
2037342K

---

1. *Introduction*

For this assignment I decided to use Java, because I'm the most familiar with I/O methods and data structures (especially Maps) in this language. When running the program, it creates two outputs. First is the coocurrencePhotoTags.csv file containing co-occurrence matrix, second is the standard output of Java Console containing top 5 recommendations with and without IDF for 'water', 'people' and 'london' tags.

2.

a. *Tag Recommendation Strategy*

The most popular tag recommendation technique gives you tags that co-occur the most with other tags within a given collection. However, this technique tends to recommend unrelated popular tags. The reason for this is because popular tags highly co-exist with almost all tags in a given collection.

i. *Pseudo-code (in Appendix1):*

b. *Tag Suggestion with popularity and significance*

This technique takes into account inverse document frequency (IDF) to get rid of unrelated popular tags. The syntax is the same as for Tag Recommendation Strategy however values of co-occurrence are multiplied by IDF factor. IDF is calculated as follows $IDF = \log(\frac{I}{I(X)})$, where I is the number of images in the collection and I(X) is number of images tagged with tag X. In fact, in my solution I used the same Map from Task 1 to calculate values for Task 3, only updating values with the IDF.

i. *Pseudo-code (it's based on methods and data structures used in 2.a.i:*

```
/Create a map of tags and their popularity values
//Using the map of photos and their tag co-occurrence values
     //Update every value using IDF factor for the specified tag
//Extract 5 top results from the updated map
```

3. *Code description*

   a.

      My code is divided into 3 sections – Task 1, Task 2 and Task 3.
      For the Task 1 `getCoocurrence()` is the most important one. It reads the photos_tags.csv file, and create a following data structure `Map<String, Map<String, Double>>` that contains co-occurrence values of tags. Later this map is printed in a CSV file using `coocurenceToFile(…) method.`
      For the Task 2 `findFiveHighestValues(…)` is a method that returns a list of top 5 entries (entries with highest values) from a specified map. Method `topFive(…)` prints out the results as standard output.
      For the Task 3 `tagsPopularityMap()` reads the tags.csv file and returns a map of tags and their popularity value. Method `calculateTopFiveWithIDF(…)` updates values of tags co-occurrence using IDF value, and a list of top 5 highest values using the `findFiveHighestValues(…)` I created for Task 2. Helper function `roundNumber(…)` rounds up numbers to 3 decimal digits and `getIDF(…)` function calculates IDF value of a tag.

   b. Link to the code in the appendix

4. *Top 5 tags without IDF*
   a. *Suggestions*
      i. water
         1. nature, 74.0
         2. blue, 71.0
         3. reflection, 63.0
         4. lake, 62.0
         5. landscape, 62.0
      ii. people
         1. portrait, 28.0
         2. street, 27.0
         3. bw, 24.0
         4. 2007, 23.0
         5. explore, 21.0
      iii. london
         1. explore, 32.0
         2. geotagged, 15.0
         3. graffiti, 15.0
         4. architecture, 14.0
         5. street, 13.0

5. *Top 5 tags with IDF*
    a. *Suggestions*
        i. water
            1. lake, 270.215
            2. reflection, 238.48
            3. nature, 236.189
            4. landscape, 224.4
            5. blue, 208.026
        ii. people
            1. street, 99.171
            2. portrait, 87.143
            3. bw, 70.818
            4. 2007, 66.626
            5. man, 53.69
        iii. london
            1. explore, 72.335
            2. graffiti, 56.715
            3. geotagged, 54.179
            4. architecture, 52.042
            5. street, 47.749
    b. *Reflections*
        i. Tags suggested for the tag 'water' seems quite good. They're all quite general but still apply to 'water' theme. I'm not sure about 'reflection', I wouldn't think it's a popular tag for photos tagged 'water'. However when I typed 'reflection' in Google Images I did get photos of lakes and seas, and overall water.
        Tags suggested for the tag 'people' seem very specific. I personally wouldn't link those up with tag 'people'. It seems like photos of streets are usually full of people, and that most black and white photos have people in them. Tag '2007' seems very random.
        Tags suggested for the tag 'london' are again quite random and general. It seems like graffiti is a popular thing to photograph in London, that's probably Banksy's fault. I don't think people actually use tag 'geotagged', this seems a bit fake. Tags 'architecture' and 'explore' make more sense. Tag 'street' seems relevant, after all there's is a few popular streets in London.
        ii. The IDF factor changed values of tag suggestions and therefore new results are more relevant for the given tags. Tag 'lake' is much more relevant than 'nature' for tag 'water'. Tag 'nature' is very popular and using IDF managed to reduce its importance. Tag 'people' is quite general so I can't really say anything about the IDF and non-IDF suggestions. Suggestions and order for tag 'london' are the same (both 'geotagged' and 'graffiti have value 15.0 so can be swapped) with and without IDF factor.

6. *Improving recommendations*

a. Possible strategies using the timestamp:
Using timestamp system could suggest time of the day related tags. Photos taken at morning hours could have ‘morning’, ‘sunrise’, ‘coffee’ as tags suggestions, photos taken at afternoon time could have ‘day’, ‘work’, ‘school’, ‘lunch’ as tag suggestions. And finally photos taken in the evening would have ‘sunset’, ‘evening’, ‘darksky’, ‘dark’ as recommended tags.

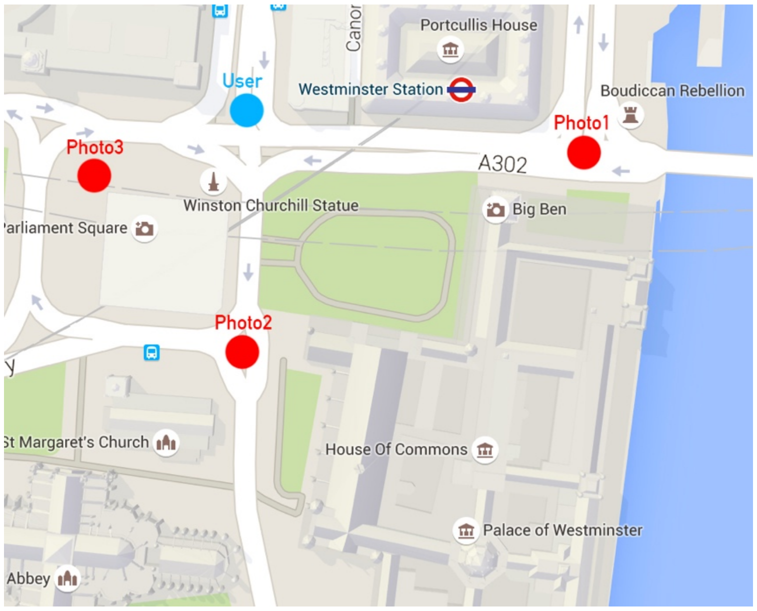
b. Possible strategies using the coordinates:
Using coordinates data we could suggest tags based on the location of a photo. For example, there exist photos with tags in these locations:

Photo 1:
Tags: london, bigben, thames, river, clock
Location: 51.500969, -0.124061

Photo 2:
Tags: london, westminster, architecture, church, palace
Location: 51.500186, -0.126113

Photo 3:
Tags: london, churchillstatue, street, people, construction
Location: 51.500819, -0.126946

Then a user of some photo sharing app with a tags recommendation system that is in location 51.501036, -0.126183 would get ‘london’, ‘bigben’, ‘westminster’ and ‘churchillstatue’ and ‘street’ as recommended tags. It would work as “What can you possibly see from your current location?”, “What can possibly in a photo that you just took in this location?”. The system would possibly need to use some AI engine too. But I believe it would be very useful thing have.



7. Appendix

a. Appendix 1: Pseudo-code for Tag Recommendation Strategy

```
//Iterate over every tag in the set
      //For every photo in the map of photos with co-existing tags
            //Extract the list of co-existing tags in the photo
            //Create a map to store tag names and values
            //For every tag in the list of co-existing tags
                  //If the current tag we're iterating over (see top of this algorithm) is in
                  the list
                        //And the main Map doesn't contain this tag yet
                              //Add the tag with initial value 1.0 to the inner map
                              //Add inner map to the main map using the current tag we're
                              iterating over now as the key
                        //If the main Map contains this tag already
                              //Extract the existing map from the key
                              //If the existing map contains the current tag
                                    //It means another co-occurrence occurred. Add 1 to the
                                    current co-occurrence value
                                    //Update the map at this tag
                              //If the existing map does not contain the current tag
                                    //Put 1.0 as initial value
                  //If the current tag we're iterating over (see top of this algorithm) is
                  not in the list
                        //And if the main Map doesn't contain the current tag
                              //Add the tag with initial value 0.0 to the inner map
                              //Add inner map to the main map using the current tag we're
                              iterating over now as the key
                        //If the main Map contains the current tag
                              //Extract the inner map of the current tag
                                    // If extracted map contains the tag from the list of co
                                    existing tags we're iterating over
                                          //Continue. Nothing happens
                                    // If extracted map does not have the tag from the list
                                    of co-existing tags
                                          //Initialize its value to 0.0
```

b. My code can be found here: http://codeshare.io/ZYCAD and as Java file I submitted along with the report. Copying and pasting code to Word gave horrible results.