Photo Tag Recommendation System

Lim Bing Shun
2228131L

1 Introduction

Social networking application is something that everyone will use in their daily life. They used this application to share their moments with friends and even with anyone over the internet. Application like Instagram and Flickr allows user to share their moment of everyday life, and even Facebook and Twitter has functions to upload pictures into the user's account and the user are able to share this albums with people or even with the public. Public members may also use these pictures as reference for their studies or even inspiration for their work. For members of the public to find related pictures, there must be a certain keyword that is tag for the picture. However, when a user uploads a large amount of picture, they will tend to spend a long time tagging each individual photo, and as a result, some user will rather not tag their photo. This assignment demonstrate how to suggest relevant tags on each photo and it is actually to ease the photo contributor in tagging on photo.

2 Tag Recommendation Strategy

In general, some of the photo tags will be appearing more than the other tags due to its popularity within all the tags, and there may be some photo tags which appears lesser than the other photo but is quite popular within its own category. One of the tag recommendation strategy is to add a counter in every photo tag combination in a photo within a photo collection. The counter is also known as terms frequency, TF(t,j) where t and j are a set of tags. Refer to Appendix (i) for the pseudo-code

2.1 Tag Suggestion with popularity and significance

This strategy uses Inverse Document Frequency (IDF) to populate out the more popular tags in a collection. IDF score are being calculated as $IDF(t) = \log \frac{I}{I(t)}$, where I = no of photos and I(X) = no of photo tag with X. After which, we can also compute the tags with highest occurrence using TF-IDF score. TF-IDF score are calculated as TF-$IDF(t, j) = TF(t, j) \times IDF(t)$. Refer to Appendix (ii) for pseudo-code.

3 Code Description

The code is being divided to 8 static methods and 1 main method. Each different tasks use different methods.

```
// This method is used for copying the CSV contents and populate
   them into a hashmap.
public static HashMap<String, ArrayList<String>> csvToHM(String
   csvFile)

// This method is used to populate TF(t,j) value, where t is the
   key of outer hashmap and j is the key of inner hashmap. The TF
   value is being stored as a double in the inner hashmap.
public static Map<String, HashMap<String, Double>>
   hashMapData(HashMap<String, ArrayList<String>> csvToHM)
```

```
// This method is used to translate the TF value to matrix as such
   to print out the matrix table into csv file.
public static String[][] hashToMatrix(Map<String, HashMap<String,
   Double>> hashMapData)

// This method is used to print out the TF value in the matrix to
   CSV file.
public static void printMatrixToCSV(String[][] matrix)

// This method is used to sort the order of the TF value, in
   ascending / descending order
private static Map<String, Double> sortByComparator(Map<String,
   Double> unsortMap, final boolean order)

// This method is used to populate top 5 TF tags in the photo
   collection
public static void printTop5Tags(Map<String, HashMap<String,
   Double>> outerMap, String tags)

// This method is used to calculate the IDF value and populate it
   into a hashmap
public static HashMap<String, Double> tagMapWithIDF (String tagURL)

// This method is used to calculate and populate out top 5 TF-IDF
   tags in the given collection.
public static void populateTop5TagsWithTFIDF (HashMap<String,
   Double> tagHashMap, Map<String, HashMap<String, Double>>
   hashMapData, String outerKey)
```

Main method is used to execute Task 1 to Task 3. Refer to appendix (iii) for actual code implementation with comments.

4 Top 5 tags

a. Water

- nature: 74.0
- blue: 71.0
- reflection: 63.0
- lake: 62.0
- landscape: 62.0

b. people

- portrait: 28.0
- street: 27.0
- bw: 24.0
- 2007: 23.0
- explore: 21.0

c. london

- explore: 32.0
- graffiti: 15.0
- geotagged: 15.0
- architecture: 14.0
- street: 13.0

5 Top 5 tags based on Section 2.1

5.1 Tag Suggestions

a. Water

- lake: 117.31
- reflection: 103.538
- nature: 102.136
- landscape: 97.229
- blue: 89.124

b. people

- street: 42.959
- portrait: 37.588
- bw: 30.69
- 2007: 28.871
- man: 23.274

c. london

- explore: 30.536
- graffiti: 24.502
- geotagged: 23.523
- architecture: 22.579
- streetart: 20.689

5.2 Reflections

By finding the TF score does not determine how popular a tag combination should be as it may be that the tag has less general occurrence. For example the tag "water" and "nature" has the highest TF score of 74 which is actually more than the combination of "water" and "lake" the TF score of 62. However, when IDF value is computed with TFIDF score, "water" and "lake" has higher TFIDF score than "water" and "nature". As the "lake" tag is exists lesser in the photo as in comparison, "nature" tag has more, the popularity per combination should be determine by TFIDF score as it is not calculated based on the combination tags but in comparison with the number of tags associated in general.

6 Improving Recommendation

Using of timestamp may be a good strategy to determine if the photo should be in the day or in the night, and by giving a more relevant tags like "sunrise", "breakfast" for the day or "sunset", "dinner" at night. Using the coordinates may also help in identify the activity of the photo. E.g. The photo co-ordinates is at one of the restaurant, the tag might give suggestions like "restaurant" and other food related tags. By combining both strategies together, the photo tag suggestion may be smart enough to determine if the food serve is during "lunchtime" or "dinnertime" or things like that. Other possibility would be by using the location and the calendar, the photo tag can suggest if the photo uploaded is a celebration or etc. For example, one photo can be uploaded at a countdown party location on 31st Dec or 1st January, the possible tag may be "countdown" etc.

7 Appendix

i section 2.1

```
create (CSV hashmap)
read (photo tag csv file)
arrayList.create()
if (tempKey != first_data_from_filereader)
        tempKey = first_data_from_filereader
        arrayList.add(second_data_from_filereader)
        CSV hashmap.put(tempKey, arrayList)
else
        if (CSV hashmap has no key)
                arrayList.add(second_data_from_filereader)
                CSV hashmap.put(tempKey, arrayList)
        else //if CSV hashmap got key
                retrieve arrayList
                add value into arrayList
                CSV hashmap.put(tempKey, arrayList)

for (every unique tags in the CSV hashmap){
        add the unique tags as key into outerMap
        add an empty innerMap into outerMap
}

for (every outerKey in outerMap){
        for (every tag in the CSV hashmap){
                if (csv hashmap-> arraylist contain outerKey){
                        for (every String in the arrayList){
                                if (map is empty and not equal to outerKey)
                                        put into innerMap with counter
                                else
                                        if innerMap contains arrayList String
                                                add the counter in the innerMap and update value
                                        else
```

```
                                                    if arrayList not the same as outerKey
                                                            put into innerMap with counter
                        }
                }
                update outerMap with key and innerMap
        }
}
```

ii section 2.2

```
read(csv file contains total number of tag in a collection)
create(IDFhashmap)
        score = Math.log10(1000 / tag_counter from CSV) //IDF(X) = log (I/I(X))
IDFhashmap.add(tag, score)

innerMap = hashMap.get(outerKey)
for (every string in innerMap){
        innerMap.put(String, TFIDF score)
        //TFIDF = TF * IDF score
}
innerMap.sort()
for (every key in the innerMap.sort())
        print out key and score
        if (loop < 6)
                loop++
        else
                break out of loop and end method
```

iii TagOccurrenceMatrix.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.util.ArrayList;
```

```
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

public class TagOccurenceMatrix {

        // populate the CSV HashMap from specific CSV file location
        public static HashMap<String, ArrayList<String>> csvToHM(String csvFile) throws Exception {
                BufferedReader br = null;
                String line = "", token = ",", tempKey = "";
                HashMap<String, ArrayList<String>> tagMap = new HashMap<String, ArrayList<String>>();
                // read csvfile
                br = new BufferedReader(new FileReader(csvFile));
                // read per line in a csv file, delimit the token
                // place them in the hashmap, return hashmap
                while ((line = br.readLine()) != null) {
                        String data[] = line.split(token);
                        ArrayList<String> alValue = new ArrayList<String>();
                        if (!tempKey.equals(data[0])) {
                                tempKey = data[0];
                                alValue.add(data[1]);
                                tagMap.put(tempKey, alValue);
                        } else {
                                if (tagMap.get(tempKey) == null) {
                                        alValue.add(data[1]);
                                        tagMap.put(tempKey, alValue);
                                } else {
                                        alValue = tagMap.get(tempKey);
                                        alValue.add(data[1]);
                                        tagMap.put(tempKey, alValue);
```

```
                        }
                }
        }
        br.close();
        return tagMap;
}

// compute the value in the hashmap to find the co-occurence matrix
public static Map<String, HashMap<String, Double>> hashMapData(HashMap<String, ArrayList<String>>
    csvToHM)
                throws Exception {
        Map<String, HashMap<String, Double>> outerMap = new HashMap<String, HashMap<String, Double>>();
        HashMap<String, Double> innerMap = new HashMap<String, Double>();
        // populate all the unique value in csv hashmap as a key to co-occurence hashmap
        for (String i : csvToHM.keySet()) {
                ArrayList<String> tagValue = new ArrayList<String>();
                tagValue = csvToHM.get(i);
                for (String tag : tagValue) {
                        if (!outerMap.containsKey(tag)) {
                                outerMap.put(tag, innerMap);
                        }
                }
        }
        // populate the inner hashmap that has association with the outerkey
        // if innerkey does not exist, add innerkey into the keyset
        // if innerkey exist, then value in the innerkey + 1
        for (String outerKey : outerMap.keySet()) {
                innerMap = new HashMap<String, Double>();
                for (String tmKey : csvToHM.keySet()) {
                        ArrayList<String> tmValue = new ArrayList<String>();
                        tmValue = csvToHM.get(tmKey);
                        if (tmValue.contains(outerKey)) {
                                for (String tval : tmValue) {
                                        innerMap = outerMap.get(outerKey);
                                        if (innerMap.isEmpty() && !tval.equals(outerKey)) {
```

```
                                        double counter = 1;
                                        innerMap.put(tval, counter);
                                    } else {
                                        if (innerMap.containsKey(tval)) {
                                            double counter = innerMap.get(tval);
                                            counter = counter + 1;
                                            innerMap.put(tval, counter);
                                        } else {
                                            if (!tval.equals(outerKey)) {
                                                double counter = 1;
                                                innerMap.put(tval, counter);
                                            }
                                        }
                                    }
                                }
                            }
                            outerMap.put(outerKey, innerMap);
                        }
                    }
                    return outerMap;
                }

                // populate the co-occurrence hashtable into matrix for printing
                public static String[][] hashToMatrix(Map<String, HashMap<String, Double>> hashMapData) throws
                    Exception {
                        String[][] matrix = new String[hashMapData.size() + 1][hashMapData.size() + 1];
                        int matCounter = 1;
                        // set up the row and column of the matrix with the tags
                        for (String outerKey : hashMapData.keySet()) {
                            matrix[matCounter][0] = outerKey;
                            matrix[0][matCounter] = outerKey;
                            matCounter++;
                        }
                        // do a matrix with the counter, if tag combination does not exist = 0
                        // if similar tag = 0
```

```
        for (int i = 1; i < matrix.length; i++) {
                for (int j = 1; j < matrix.length; j++) {
                        if (matrix[i][0].equalsIgnoreCase(matrix[0][j])) {
                                matrix[i][j] = Double.toString(0);
                        }
                        if (hashMapData.keySet().contains(matrix[i][0])) {
                                if (hashMapData.get(matrix[i][0]).containsKey(matrix[0][j])) {
                                        matrix[i][j] =
                                           Double.toString(hashMapData.get(matrix[i][0]).get(matrix[0][j]));
                                } else {
                                        matrix[i][j] = "0";
                                }
                        } else {
                                matrix[i][j] = "0";
                        }
                }
        }
        return matrix;
}

//printing of the matrix to csvfile
public static void printMatrixToCSV(String[][] matrix) throws Exception {
        // initialise the printwrite method with csv file name
        PrintWriter pw = new PrintWriter(new File("tag_occurence.csv"));
        StringBuilder sb = new StringBuilder();
        // add the value and token back into the stringbuilder and
        // append next line if go to next row
        // print into csv file after finishing the for loop
        for (int i = 0; i < matrix.length; i++) {
                for (int j = 0; j < matrix.length; j++) {
                        if (j != (matrix.length - 1)) {
                                sb.append(matrix[i][j]);
                                sb.append(",");
                        } else {
                                sb.append(matrix[i][j]);
```

```
                        }
                }
                sb.append("\n");
        }
        pw.write(sb.toString());
        pw.close();
        System.out.println("CSV has generated successfully. CSV file name : tag_occurence.csv");
}

// sort the Map by value, with asc (true) /desc (false) order
private static Map<String, Double> sortByComparator(Map<String, Double> unsortMap, final boolean
    order) {

        List<Entry<String, Double>> list = new LinkedList<Entry<String, Double>>(unsortMap.entrySet());

        Collections.sort(list, new Comparator<Entry<String, Double>>() {
// place hashmap data into LinkedList
// compare the list based on values, swap if the value is higher / lower
                public int compare(Entry<String, Double> o1, Entry<String, Double> o2) {
                        if (order) {
                                return o1.getValue().compareTo(o2.getValue());
                        } else {
                                return o2.getValue().compareTo(o1.getValue());
                        }
                }
        });
        // Use LinkedHashMap to maintain the insertion order
        Map<String, Double> sortedMap = new LinkedHashMap<String, Double>();
        for (Entry<String, Double> entry : list) {
                sortedMap.put(entry.getKey(), entry.getValue());
        }

        return sortedMap;
}
```

```
// print top 5 TF tags
public static void printTop5Tags(Map<String, HashMap<String, Double>> outerMap, String tags) {
        // use the sorting method and print out the top 5 tags of the specify outerkey
        Map<String, Double> sortedMap = sortByComparator(outerMap.get(tags), false);
        int sortedMapCounter = 0;

        for (String x : sortedMap.keySet()) {
                System.out.println(x + ": " + sortedMap.get(x));
                if (sortedMapCounter < 4) {
                        sortedMapCounter++;
                } else {
                        break;
                }
        }
}

// compute the IDF score of each tag
public static HashMap<String, Double> tagMapWithIDF (String tagURL) throws Exception{
        HashMap<String, Double> tagHM = new HashMap<String, Double> ();
        // read the tags.csv file for the total no of tags per photo collection
        // then compute the IDF score = log10 (I / I(X))
        BufferedReader br = new BufferedReader(new FileReader(tagURL));
        String line = "";
        while ((line = br.readLine()) != null){
                String[] tag = line.split(",");
                String key = tag[0];
                double value = Math.log10(10000 / Integer.parseInt(tag[1]));
                tagHM.put(key, value);
        }
        return tagHM;
}

// populate out the TFIDF with IDF and the value of each counter, returning top 5 TFIDF tags (top 5
   popular tags)
public static void populateTop5TagsWithTFIDF (HashMap<String, Double> tagHashMap, Map<String,
```

```
    HashMap<String, Double>> hashMapData, String outerKey){
        HashMap<String, Double> innerMap = hashMapData.get(outerKey);
        int testCounter = 0;

        for (String x : innerMap.keySet()){
                // using the outerkey to retrieve the inner hashmap
                // update TFIDF score into the inner hashmap (3.d.p.)
                DecimalFormat df = new DecimalFormat("#.###");
                innerMap.put(x, Double.valueOf(df.format(innerMap.get(x) * tagHashMap.get(x))));
        }
        // sort the hashmap and print out top 5 TFIDF per outerkey specify
        Map<String, Double> tmpMap = sortByComparator(innerMap, false);
        for (String x : tmpMap.keySet()){
                System.out.println(x + ": " + tmpMap.get(x));
                if (testCounter < 4){
                        testCounter++;
                } else {
                        break;
                }
        }
}

public static void main(String[] args) throws Exception {
        // Task 1
        System.out.println("Start of Task 01: ");
        String csvFile = "photos_tags.csv";
        String tagFile = "tags.csv";
        System.out.println("Copying csv file to hashmap...");
        HashMap<String, ArrayList<String>> csvToHM = csvToHM(csvFile);
        System.out.println("Copying complete... Populating matrix map...");
        Map<String, HashMap<String, Double>> hashMapData = hashMapData(csvToHM);
        System.out.println("Matrix map populated. Printing map to CSV... ");
        String[][] hashToMatrix = hashToMatrix(hashMapData);
        printMatrixToCSV(hashToMatrix);
        System.out.println("Task 01 Complete...");
```

```
                // end of Task 1
                System.out.println("-----------------------------------------------------------------------");
                // Task 2
                System.out.println("Start of Task 02: ");
                System.out.println("Water Tags: ");
                printTop5Tags(hashMapData, "water");
                System.out.println("People Tags: ");
                printTop5Tags(hashMapData, "people");
                System.out.println("London Tags: ");
                printTop5Tags(hashMapData, "london");
                System.out.println("Task 02 Complete...");
                // end of Task 2
                System.out.println("-----------------------------------------------------------------------");
                // Task 3
                System.out.println("Start of Task 03: ");
                HashMap <String, Double> tagHashMap = tagMapWithIDF(tagFile);
                System.out.println("Water Tags: ");
                populateTop5TagsWithTFIDF(tagHashMap, hashMapData, "water");
                System.out.println("People Tags: ");
                populateTop5TagsWithTFIDF(tagHashMap, hashMapData, "people");
                System.out.println("London Tags: ");
                populateTop5TagsWithTFIDF(tagHashMap, hashMapData, "london");
                System.out.println("Task 03 Complete...");
                // end of Task 3
        }
}
```