Photo Tag recommendation system

Student name: Rajeevan Vijayakumar
Student ID: 2080123v

This report will outline the process used to create a photo tag recommendation system in Python. It will begin by explaining the method used for the photo tag recommendation as well as the method for tag suggestion with popularity and significance taken into consideration. Both of these explanations will be followed by a pseudocode representation of the python code where each section of pseudocode will begin with a small description of what the section will do The code will be explained in detail with comments in the source code file being attached to this report as a .py file. This will be be followed by the top 5 tags for the required list of tags along with the top 5 tags based on the tag suggestion with popularity and significance. The report will then round off by reflecting back on the tag recommendation strategy used and going on to talk about possible strategies to improve recommendations using timestamp and co-ordinates as well as other methods.

The photo tag recommendation system is all based around the functionality of when given a tag, recommending related tags from existing images previously annotated with the given tag. The main functionality is done through creating a co occurrence matrix and querying it for a required tag to return the most popular tags for it. This matrix tell us how many times the required tag has co occurred at the same time in an image with another tag and the higher the number, the more similar the tags are supposed to be. The pseudocode for creating such matrix is shown below:

This program takes in information from 3 CSV files where the first one, tags.csv, contains all the different tags in the collection of images as well as the number of times they are tagged in an image, photos.csv contain a collection of 10 000 photos each being referred by an image id and other details such as date added and a small description. The final csv file, photo_tags.csv contains all the photos by id and their tags. Note it is easier to understand if followed along with the code in matrix.py

//adding the contents of tags.csv to a dictionary where the tag is the key and a dictionary with all the other tags as the value. Ie {‘water’: {‘water’ :0, …}}

open tags.csv and read
loop through each line
add each tag to a dictionary as the key with the value as an empty dictionary
loop through the tags dictionary in parallel with two loops
The first loop loops through the tags dictionary
The second loop adds all the tags as keys into the dictionary value of the current tag in the first loop with 0 as the value.
close the file

//adding contents of photo_tag.csv to a dictionary with the imageid as the key and a list of tags as the value. Ie {1 :[‘water’, ‘people’, …]}

Open photos_tags.csv and read
loop through each line
check that the photo is not already in the dictionary
add each photo to a dictionary as the key with the value as an empty list
close the file

open photos_tags.csv and read

loop through each line
get the photo's empty list from the photos dictionary made above
if the current photos tag is not in list
add the tag to the list
close the file

//creation of the matrix

Loop through the tags dictionary
Loop through the photos dictionary
get the list for the photo in the photos dictionary
If the current tag in the tags dictionary is found in the list of tags
loop through the list
If the current index in the list is not equal to the current tag
add 1 to the index's tag value in the tags dictionary for the current tag
use panda to create a data frame of the tags dictionary
output dataframe to a csv file.

//gets the top 5 recommended(co occurs with it the most) tags for a given tag

create a new variable to store the dictionary of co occurrence values for the required tag
retrieve the top 5 tags in the dictionary based on their co occurrence value and store in a list
loop through the list and print out each tag along with it co occurrence value

The following tag recommendation technique is based on popularity and significance as the recommendation system above can give recommendations of unrelated popular tags due to popular tags co existing with almost all tags in a collection by nature. So to overcome this, a weighting factor is introduced called the inverse document frequency(IDF). It is used to capture the significant or popularity of a tag and computed as

$$IDF=\log(I/I(X))$$

Where I corresponds to the number of images in a collection and I(x) is the number images tagged by the tag x. Hence to overcome the popular tag problem, the IDF value for each tag is multiplied by the number of times it co occurs with the tag in question to get the TF-IDF value.

$$TF\text{-}IDF = TF(i,j) * IDF(i)$$

where TF is the number of times tag i co occurs with tag j.

The following is the pseudocode for finding the TF-IDF value. Note it is easier to understand if followed along with the code in matrix.py.

//adding the contents of tags.csv to a dictionary where the tag is the key and the number of images it has occurred in as the integer value

open tags.csv file and read
loop through each line
add each tag into a dictionary as the key with the value as the number of times it appear in an image
close the file

//calculates the tfidf value for all the tags and give the top 5 recommended tags for the required tag based on the tfidf values

copy the contents of the tags dictionary into a new dictionary
get the value for the tag in question from the new dictionary above
loop through the value from the above(the value is a dictionary)
get the total no of images that the current tag in the loop appears in
calculate the idf value by taking the $\log_e$ of the total images/value from previous line
multiply the idf value by the value of the current tag in the loop
retrieve the top 5 tags from the new dictionary created in the first line and store in a list
loop through the list and print out each tag along with it co occurrence value

The following are the most popular tags for the tags, water, people and london. The first list is without taking the IDF value into consideration and the second is after multiplying the tags by the given tag's IDF values hence the TF-IDF value.

Task 2:
Water: {nature: 74 , blue: 71, reflection: 63, lake: 62, landscape: 62}
People: {portrait: 28, street: 27, bw: 24, 2007: 23, explore: 21}
London: {explore: 32, geotagged: 15, graffiti: 15, architecture: 14, street: 13}

Task 3:
Water: {lake : 270.116, reflection : 238.404, nature : 235.176, landscape : 223.877, blue : 205.216}
People: {street : 98.916, portrait : 86.549, bw : 70.667, 2007 : 66.479, man : 53.591}
London: {explore : 70.311, graffiti : 56.418, geotagged : 54.164, architecture : 51.990, streetart : 47.638}

While these methods work well to give recommendations, they do have their issues. The main issue with the co-occurrence matrix is that when trying to recommend tags for popular tags, the returned set of tags are popular tags just like it so some of the recommended tags tend to be quite different from the original. This is proven when the top 5 popular tags for people are returned and 2007 and bw are included as recommended tags. The former is a year and has little to do with people. It is only recommended as it is very popular in lots of images that were taken in 2007 and lots of these photos will most likely have little to do with the subject of people.

Tag recommendations can be improved with the inclusion of timestamps and coordinates to images. When timestamps are included with images, it can be used to identify when the image was taken which in turn can be used to sort images by the date or time that they were taken and create a separate co occurrence matrix for those images. Since those images were taken at around the same time, their tags should be more similar hence more relevant when trying to recommend tags found on those images as the tags returned will be tags that were made around the same time frame of the images. This is also applied when coordinates are also included which means the location of where the images were taken can also be included in the sorting of the images before giving tag recommendations. This means the tags in an image will be relevant to a specific location leading to those images having tags relevant to the location. Other ways that these methods could be improved would be to sort based on keywords found in the description of images if included. The image will include a small description about their content and keywords could be extracted and images sorted by these keywords to create a cooc matrix for them. This would mean the tags that will be recommended will be very similar as the images have been sorted based on similar descriptions.

The matrix has been programmed in Python so a matrix.py file has been attached with this report that can be run to create the matrix. The code is also in the appendix below but copying python code into a IDE from a PDF may cause problems and not run because the formatting has been modified. The python code has been commented sufficiently and readable with the pseudocode found in this report. Note the python program requires the numpy and pandas package to run properly as the latter package is used to convert the matrix created in the program into a csv file. When calculating the TF-IDF values, natural log e was used instead of $\log_{10}$ to calculate the IDF value. This was due to the fact that it was done this way in the lecture notes. Reading online, lots of articles used $\log_{10}$ but the lecture notes example was used in the end. The CSV file containing the co occurrence matrix is also included with this report. All 3 files will be zipped as MMSAE.

The code found in matrix.py is below:

```
import csv
import heapq
import pandas as pd
import math
import copy
from pprint import pprint

tags = {}                       # stores the co occurrence matrix
photos_with_tags = {}           # stores all the photos and their tags as a list
tags_photos = {}                # stores all the tags and the no of images they are tagged in
idf_tags = {}                   # stores all the tags with their co occurrence values*idf values

# opening the tags.csv to add them to a dictionary using a csv reader to read the file line by line
# with each line, a new tag is added and a second for loop loops through the tags dictionary to
add
# all the tags to each dictionary value of the tags so each tag has a dictionary hold all the tags
# as the key with values being the number of co occurrences.

with open('tags.csv', 'rb') as csvfile:
  reader = csv.reader(csvfile)
  for i in reader:
     tags[i[0]] = {}
  for i in tags:
     for j in tags:
        values = tags.get(i)
        values[j] = 0
  csvfile.close()

# opening the photos_tag.csv file to add all the photos to a dictionary along with their tags in a list
as their value.
# the first loop adds the images to the dictionary and the conditional is used to eliminate
duplicate additions. The
# second file opening and for loops are used to add the tags for each image to a list and add
them as the photo's value
# in the dictionary.
```

```
with open('photos_tags.csv', 'rb') as csvfile:
  reader = csv.reader(csvfile)
  for i in reader:
     if i[0] not in photos_with_tags:
        photos_with_tags[i[0]] = []
  csvfile.close()

with open('photos_tags.csv', 'rb') as csvfile:
  reader = csv.reader(csvfile)
  for i in reader:
     x = photos_with_tags.get(i[0])
     if i[1] not in x:
        x.append(i[1])
  csvfile.close()

with open('tags.csv', 'rb') as csvfile:
  reader = csv.reader(csvfile)
  for i in reader:
     tags_photos[i[0]] = i[1]
  csvfile.close()

# x = tags.get('red')
# x['red']+=1


def create_matrix():
  for i in tags:                                  # loop through tags dictionary
     for j in photos_with_tags:                          # loop through dictionary with pics and their list
        x = photos_with_tags.get(j)                       # of tags. get the current photo's list of tags
        if i in x:                                # check if tag exists for the current pic
           for k in x:                            # if so loop through the list
              if k != i:                          # if the current index is not equal to the tag then
                 y = tags.get(i)                        # find tag in current tags dictionary of tags and add
                 y[k] += 1                        # 1 to its value
  data = pd.DataFrame(tags)
  data.to_csv("matrix.csv")

# returns the top 5 recommended tags for a given tag


def top_5(name):
  x = tags.get(name)                                  # get the co occurrence for the required tag
  y = heapq.nlargest(5, x, key=x.get)                       # get the top 5 tag co occurring tags and
puts them in
  for i in y:                                 # a list set to var y. The for loop loops through this
     print "%s: %d" % (i, x.get(i))                 # list and prints out the tag and co occurrence.
```

```
# calculate the idf value for a tag given the tag and the total number images
# in a collection. The idf value is then multipled by the co occurrence value
# for each of the tags in the given tag's dictionary.


def tfidf(tag, images):
  idf_tags = copy.deepcopy(tags)                    # create a copy of the tags dictionary
  tag_in_question = idf_tags.get(tag)               # get the dictionary of tags for the current
tag
  for i in tag_in_question:                         # in question from the new dictionary and loop
    images_tagged = tags_photos.get(i)              # get the number of images the current
tag in loop
    idf_value = math.log(images / int(images_tagged))   # appears in and calculate the idf
value using above
    tag_in_question[i] *= idf_value                 # value and total images. Then multiply the
current
  top5 = heapq.nlargest(5, tag_in_question, key=tag_in_question.get)  # tag's value in the copied
dictionary by the
  for j in top5:                                    # idf  value. get the top 5 tags and store in list
    print "%s :%.3f" % (j, tag_in_question.get(j))  # and loop through list and print

create_matrix()                                     # function call to create the co occurrence matrix
print "Task 2"
print "Water:"
top_5('water')                                      # calling the top5 function for each of the tags
print "People:"
top_5('people')
print "London:"
top_5('london')

print "Task 3"
print "Water"
tfidf('water', 10000)                               # calling the tfidf value on all the tags
print "People"
tfidf('people', 10000)
print "London"
tfidf('london', 10000)
```