Data Transformation

GAHAN SARAIYA (18MCEC10), PRIYANKA BHATI (18MCEC02)

18mcec10@nirmauni.ac.in,18mcec02@nirmauni.ac.in

I. INTRODUCTION

Data transformation is the process of converting/transforming data from one form[1] or structure into another form or structure. Data transformation is critical to activities such as data integration and data management. Data transformation can include a range of activities such as: converting data types, data cleaning by removing null values or duplicate data, enrich the data, or perform aggregations, depending on the requirements of the project.

A typical data transformation can be classified in two stages:

- Stage 1
  - Discovering the data and identify various sources and data types.
  - Find out the structure of data and data transformations that need to be performed.
  - Map data to define how individual fields are mapped, modified, joined, filtered, and aggregated.
- Stage 2
  - Extract data from the original source.
    The range of sources can vary from structured sources like databases to non-structured sources such as streaming sources.
  - Do the transformation
    i.e. aggregating values, changing date formats, editing text or joining attributes.
  - Migrate this transformed data to data warehouse.

II. TRANSFORMATION OF DATA

I. Data Discovery

We have taken a dataset as described in Section IV.I for which we need to perform knowledge gain of user mobility with correlation between locations.

II. Data Structure

Data is a raw data of multiple text files (one file per taxi id) containing records of booking of taxi with timestamp and latitude and longitude of pickup location.

III. Extracting Data

To perform operation we will first combine these multiple data files into one single file containing all the record of `taxi-id, latitude, longitude and timestamp`

[1] i.e. format for different data mining tools

IV. Performing Transformation

The two attributes in dataset IV.I, latitude and longitude are concatenate together to form unique area with method known as geohash described in Section IV.II.

This will generate a file (as in IV.II.2) containing `taxi-id, geohash and timestamp`

III. Data Visualization

Now we need to compute correlation between locations from our transformed data IV.II.2.

To compute correlation we simply create dissimilarity between two records (which is considered as the timestamp).

Visualization result is obtained as in Section IV.IV.1 which contains `geohash, geohash and dissimilarity`.

IV. Experiment

I. Dataset

For the experiments we have taken a dataset of user mobility which contains data attributes as described below:

Attribute	Data-type	Description
Taxi-id	Numerical (Integer)	unique id of each taxi
Latitude	Discrete	latitude of pickup location
Longitude	Discrete	longitude of pickup location
Timestamp	date-time	time-stamp when taxi booked

Table 1: Description of Dataset

I.1 Chunk 1

```
2, 116.36422, 39.88781, 2008-02-02 13:33:52
2, 116.37481, 39.88782, 2008-02-02 13:37:16
2, 116.37677, 39.88791, 2008-02-02 13:38:53
2, 116.38033, 39.88795, 2008-02-02 13:42:18
2, 116.39392, 39.89014, 2008-02-02 13:43:55
2, 116.41149, 39.89152, 2008-02-02 13:47:20
2, 116.42105, 39.89194, 2008-02-02 13:48:57
2, 116.4215, 39.89823, 2008-02-02 13:52:21
2, 116.4215, 39.89902, 2008-02-02 13:53:58
2, 116.42972, 39.90726, 2008-02-02 13:57:23
```

I.2 Chunk 2

```
3, 116.50029, 39.91458, 2008-02-03 23:18:57
3, 116.51617, 39.91126, 2008-02-03 23:23:57
3, 116.55026, 39.90787, 2008-02-03 23:28:56
3, 116.57449, 39.90895, 2008-02-03 23:33:56
3, 116.59587, 39.9006, 2008-02-03 23:43:56
3, 116.61065, 39.91218, 2008-02-03 23:48:56
3, 116.6347, 39.92558, 2008-02-03 23:53:56
3, 116.652, 39.93273, 2008-02-03 23:58:56
3, 116.65518, 39.95388, 2008-02-04 00:03:56
3, 116.69494, 39.98554, 2008-02-04 00:08:56
3, 116.75385, 40.02126, 2008-02-04 00:18:57
```

II. Geohash

The Geohash is the method for encoding regions with specific precision of latitude and longitude.

Geohashes offer properties like arbitrary precision and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision).

For example, the coordinate pair `57.64911,10.40744` (near the tip of the peninsula of Jutland, Denmark) produces a slightly shorter hash of `u4pruydqqvj`.

Implementation of Geohash is described below:

II.1 Implementation

```
from math import log10


#  Note: the alphabet in geohash differs from the common base32
#  alphabet described in IETF's RFC 4648
#  (http://tools.ietf.org/html/rfc4648)
__base32 = '0123456789bcdefghjkmnpqrstuvwxyz'
__decodemap = {}


for i in range(len(__base32)):
    __decodemap[__base32[i]] = i
del i


def decode_exactly(geohash):
    """
```

```
    Decode the geohash to its exact values, including the error
    margins of the result.  Returns four float values: latitude,
    longitude, the plus/minus error for latitude (as a positive
    number) and the plus/minus error for longitude (as a positive
    number).
    """
    lat_interval, lon_interval = (-90.0, 90.0), (-180.0, 180.0)
    lat_err, lon_err = 90.0, 180.0
    is_even = True
    for c in geohash:
        cd = __decodemap[c]
        for mask in [16, 8, 4, 2, 1]:
            if is_even:  # adds longitude info
                lon_err /= 2
                if cd & mask:
                    lon_interval = ((lon_interval[0] + lon_interval[1]) / 2,
                     ↪  lon_interval[1])
                else:
                    lon_interval = (lon_interval[0], (lon_interval[0] +
                     ↪  lon_interval[1]) / 2)
            else:       # adds latitude info
                lat_err /= 2
                if cd & mask:
                    lat_interval = ((lat_interval[0] + lat_interval[1]) / 2,
                     ↪  lat_interval[1])
                else:
                    lat_interval = (lat_interval[0], (lat_interval[0] +
                     ↪  lat_interval[1]) / 2)
            is_even = not is_even
    lat = (lat_interval[0] + lat_interval[1]) / 2
    lon = (lon_interval[0] + lon_interval[1]) / 2
    return lat, lon, lat_err, lon_err


def decode(geohash):
    """
    Decode geohash, returning two strings with latitude and longitude
    containing only relevant digits and with trailing zeroes removed.
    """
    lat, lon, lat_err, lon_err = decode_exactly(geohash)
    # Format to the number of decimals that are known
    lats = "%.*f" % (max(1, int(round(-log10(lat_err)))) - 1, lat)
```

```
    lons = "%.*f" % (max(1, int(round(-log10(lon_err)))) - 1, lon)
    if '.' in lats:
        lats = lats.rstrip('0')
    if '.' in lons:
        lons = lons.rstrip('0')
    return lats, lons


def encode(latitude, longitude, precision=10):
    """
    Encode a position given in float arguments latitude, longitude to
    a geohash which will have the character count precision.
    """
    lat_interval, lon_interval = (-90.0, 90.0), (-180.0, 180.0)
    geohash = []
    bits = [16, 8, 4, 2, 1]
    bit = 0
    ch = 0
    even = True
    while len(geohash) < precision:
        if even:
            mid = (lon_interval[0] + lon_interval[1]) / 2
            if longitude > mid:
                ch |= bits[bit]
                lon_interval = (mid, lon_interval[1])
            else:
                lon_interval = (lon_interval[0], mid)
        else:
            mid = (lat_interval[0] + lat_interval[1]) / 2
            if latitude > mid:
                ch |= bits[bit]
                lat_interval = (mid, lat_interval[1])
            else:
                lat_interval = (lat_interval[0], mid)
        even = not even
        if bit < 4:
            bit += 1
        else:
            geohash += __base32[ch]
            bit = 0
            ch = 0
    return ''.join(geohash)
```

II.2 Data File after applying geohash

II.3 Chunk 1

```
2, uzurg, 2008-02-02 13:33:52
2, uzurg, 2008-02-02 13:37:16
2, uzurg, 2008-02-02 13:38:53
2, uzurg, 2008-02-02 13:42:18
2, uzurg, 2008-02-02 13:43:55
2, uzurg, 2008-02-02 13:47:20
2, uzurg, 2008-02-02 13:48:57
2, uzurg, 2008-02-02 13:52:21
2, uzurg, 2008-02-02 13:53:58
2, uzuru, 2008-02-02 13:57:23
```

II.4 Chunk 2

```
3, uzuru, 2008-02-03 23:18:57
3, uzuru, 2008-02-03 23:23:57
3, uzuru, 2008-02-03 23:28:56
3, uzuru, 2008-02-03 23:33:56
3, uzurg, 2008-02-03 23:43:56
3, uzuru, 2008-02-03 23:48:56
3, uzuru, 2008-02-03 23:53:56
3, uzuru, 2008-02-03 23:58:56
3, uzurv, 2008-02-04 00:03:56
3, uzurv, 2008-02-04 00:08:56
3, uzury, 2008-02-04 00:18:57
```

III. Implementation of Utilities for parsing data

```
1 import os
2 import geohash
3
4 datasets_dir = os.path.abspath(os.path.join(os.path.dirname(__file__),
  ↪  'datasets'))
5 dataset_files = [os.path.join(datasets_dir, file) for file in
  ↪  os.listdir(datasets_dir)]
```

```
6
7
8  def parseline(line, precision=5):
9      if line[-1] == '\n':
10         line = line[:-1]
11     taxi_id, timestamp, latitude, longitude = line.split(',')
12     latitude, longitude = float(latitude), float(longitude)
13     gh = geohash.encode(latitude, longitude, precision=precision)
14     return {
15         'taxi_id': taxi_id,
16         'geohash': gh,
17         'latitude': latitude,
18         'longitude': longitude,
19         'timestamp': timestamp
20     }
21
22
23 def nextline(dataset_file, precision):
24     taxi_id = dataset_file.split('.')[0]
25     taxi_id = taxi_id.split('/')[-1]
26     with open(dataset_file, mode='r') as fp:
27         for line in fp.readlines():
28             yield parseline(line, precision)
29     yield None
30
31
32 def get_random_data(precision):
33     file_pointers = [nextline(dataset_file, precision) for dataset_file in
       ↪ dataset_files]
34     while len(file_pointers) > 0:
35         for fp in file_pointers:
36             line = fp.__next__()
37             if line is None:
38                 file_pointers.remove(fp)
39             else:
40                 # print(line)
41                 yield line
42
43
44 def get_sequential_data(precision):
45     file_pointers = [nextline(dataset_file, precision) for dataset_file in
       ↪ dataset_files]
```

```
    for fp in file_pointers:
        line = fp.__next__()
        while line is not None:
            yield line
            line = fp.__next__()

# Print iterations progress
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals = 1,
 ↪  length = 100, fill = '='):
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete
 ↪  (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration /
     ↪  float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print('\r%s |%s| %s%% %s' % (prefix, bar, percent, suffix), end = '\r')
    # Print New Line on Complete
    if iteration == total:
        print()
```

IV. Implementation of storing and transforming data

```
import os
import simulator
import itertools
from time import sleep
from datetime import datetime

PRECISION = 5

def generate_file_1():
```

```
    with open('file1.txt', 'w') as fp:
        for df in simulator.get_sequential_data(precision=PRECISION):
            fp.write("{}, {}, {}, {}\n".format(
                df['taxi_id'], df['latitude'], df['longitude'], df['timestamp']))

def generate_file_2():
    with open('file2.txt', 'w') as fp:
        for df in simulator.get_sequential_data(precision=PRECISION):
            fp.write("{}, {}, {}\n".format(
                df['taxi_id'], df['geohash'], df['timestamp']))


def parseline(line):
    if line[-1] == '\n':
        line = line[:-1]
    taxi_id, geohash, timestamp = line.split(',')
    epoch_time = datetime.strptime(timestamp, ' %Y-%m-%d %H:%M:%S').timestamp()
    return geohash, epoch_time, taxi_id

def generate_file_3(prevent_skip=False):
    records_to_process = int(os.popen("wc -l
    ↪  file2.txt").read().strip().split()[0])
    print("Processing Dissimilarity computation for {}
    ↪  records..".format(records_to_process))
    # Initial call to print 0% progress
    simulator.printProgressBar(0, records_to_process-1, prefix = 'Progress:',
    ↪  suffix = 'Complete', length = 50)
    with open('file3.txt', 'w') as write_file:
        skip_n = 0
        with open('file2.txt', 'r') as file1:
            count = 0
            for file1_line in file1.readlines():
                skip_n += 1
                count += 1
                dataset1 = parseline(file1_line)
                simulator.printProgressBar(count, records_to_process, prefix =
                ↪  'Progress:', suffix = 'Complete', length = 50)
                with open('file2.txt', 'r') as file2:
                    line_no = 0
                    for file2_line in file2.readlines():
                        line_no += 1
                        if line_no > skip_n or prevent_skip:
```

```
                            dataset2 = parseline(file2_line)
                            if dataset1[2] == dataset1[2] and dataset1 !=
                             ↪  dataset2 and dataset1[0] != dataset2[0]:
                                cmp_str = "{}-{}".format(dataset1[2],
                                 ↪  dataset2[2])
                                write_file.write("{}, {}, {}, {}
                                 ↪  \n".format(cmp_str, dataset1[0], dataset2[0],
                                 ↪  dataset1[1] - dataset2[1]))


if __name__ == '__main__':
    generate_file_1()
    generate_file_2()
    generate_file_3()
```

IV.1 Result

```
2-2,  uzurg,  uzuru, -1411.0
2-2,  uzurg,  uzuru, -1508.0
2-2,  uzurg,  uzuru, -1713.0
2-2,  uzurg,  uzuru, -1810.0
2-2,  uzurg,  uzuru, -1810.0
2-2,  uzurg,  uzuru, -2111.0
2-2,  uzurg,  uzurf, -10466.0
2-2,  uzurg,  uzurf, -10723.0
2-2,  uzurg,  uzurf, -10768.0
2-2,  uzurg,  uzurf, -11025.0
```

⋮

```
2-4,  uzurg,  uzuru, -397746.0
2-4,  uzurg,  uzuru, -398347.0
2-4,  uzurg,  uzuru, -398947.0
2-4,  uzurg,  uzuru, -399546.0
2-4,  uzurg,  uzuru, -400146.0
2-4,  uzurg,  uzuru, -400746.0
2-4,  uzurg,  uzuru, -401946.0
2-4,  uzurg,  uzuru, -402546.0
2-4,  uzurg,  uzuru, -403146.0
2-4,  uzurg,  uzuru, -403745.0
2-4,  uzurg,  uzuru, -404345.0
```

```
2-4,  uzurg,  uzuru, -404945.0
2-4,  uzurg,  uzuru, -405546.0
2-4,  uzurg,  uzuru, -406146.0
2-4,  uzurg,  uzuru, -406745.0
```

⋮

```
5-5,  uzuru,  uzurf, -229832.0
5-5,  uzuru,  uzurf, -230432.0
5-5,  uzuru,  uzurf, -231032.0
5-5,  uzuru,  uzurf, -231632.0
5-5,  uzuru,  uzurf, -232232.0
5-5,  uzuru,  uzurf, -232832.0
5-5,  uzuru,  uzurf, -233431.0
5-5,  uzuru,  uzurf, -234031.0
5-5,  uzuru,  uzurf, -234631.0
5-5,  uzuru,  uzurf, -235231.0
```

Attribute	Data-type	Description
taxi-id-taxi-id	string	ids of two taxis separated by hyphen
Area	geohash	hashed location value
Area	geohash	hashed location value
Dissimilarity	Float value	Float value containing difference of epoch timestamp

Table 2: Description of final result: dissimilarity between location

V. Conclusion

The technique geohash which we have used for data transformation gives the dissimilarity with greater precision. But this precision leads to higher computation cost which is the major drawback. So to reduce the computation cost the value of precision needs to be decreased to reduce computational cost.