National and Kapodistrian University of Athens

Department of Informatics and Telecommunications

Large-Scale Data Analysis Techniques

Spatiotemporal Data Classification

Nikiforos Pittaras, M1422
Chrysoula Themeli, M1423

January 6, 2018

Contents

List of Figures

Todo list

Remove the null jids from the start

1 Introduction

This is the report related to the Python project for the course "Large-Scale Data Analysis Techniques". In the below sections, the implemented logic will be explicitly analyzed. The project is divided in two parts: in the first part the goals are: to preprocess and clean the given data as well as to visualize five of the given trips in the train_set file. In the second part, the tasks are: to find for all test trips the k nearest neabours from the given train trips, to find the train trips with the longest similar subroute for each test trip, to use 10-fold cross validation to the training data and classify them with three different methods (k-nearest neighbours, logistic regression and random forest) and finally to choose one of the above classification methods and improve the result.

In the Main.py file, it is required to provide the path to the input folder (containing the training and test files) and the output folder where all results will be stored. In this folder an additional directories are created, in order to keep all maps extracted from this application and also to store the charts showing the classification accuracy. There are two ways in order to run this project, either add these folders in the edit configurations choice from the menu, or use the run.sh in order to provide these args. Finally, the dependencies are checked and the folders and files are prepared before starting to run the application.

2 Exercise 1

In this section, further details are provided related to the preprocessing and cleaning of the training data as well as to the trip's visualization. All files implementing the requirements of this part are in the folder question1.

2.1 Question A

The requirement in this part is to preprocess the input training data and create a new file where will be in 3 columns. The first column is the trip_id incrementing by 1 for each trip, the second is the journey_id and the last one contains a list of lists, where each item contains a timestamp, a longitude and a latitude. In order to read the given csv file, pandas library was used transforming the csv file to a dataframe.

The first necessary thing to do, was to remove all data with null journeyPatternId and then using the groupby method on the columns vehicleID and journeyPatternId, three different dataframes were created. By concatenating them, the final dataframe contains five columns, where the three last columns contain a list of timestamps, a list of longitudes and a list of latitudes respectively.

By iterating to each dataframe's row, the three lists are now concatenated in one column and therefore the other two are dropped from the dataframe. Then, the column vehicleID is replaced by the tripId column which is the index of this dataframe, starting from 0 and incrementing by 1. Finally, in the column points, which now contains the list of lists with the timestamps and the poins, is sorted based on the timestamp of each list item. The final result is exported in a file, keeping the requested format.

2.2 Question B

The second goal was to clean the given data, based on the total trip distance and the max distance between two successive points. Total distance should not be less than 2km and max distance should not exceed 2km.

The above logic is implemented in the file CleanData.py in function filter_trips. The two lists (trips_too_small and trips_too_big) will keep the excluded trips due to either not valid total distance or to invalid max distance between successive points. For each trip in the training dataset (in the dictionary format from question A) the total distance is calculate, using the function calculate_lonlat_distance, which returns the distance between two successive points using the haversine formula. Following the same logic, the max distance between successive points of a trip is calculated and a distance is found more than 2km, the trip is excluded.

Finally, the cleaned trips are kept in a list under the same format that was used in Question A and they are extracted in the file cleanTrips.csv, but also in the file tripsClean.pickle which keeps the data structure used. The functions for the data export are in the file utils.py (write_trips_to_file and write_trips_using_pickle).

2.3 Question C

This task is implemented in the file DataVisualization.py and the created output is stored in the output folder in the gmplots directory. The requirement is to visualize five of the given train trips. For this purporse, after having excluded all trips with null journey_id and with less than 1 lists with coordinates, the trips list is shuffled

and then iterated only for 5 trips. Afterwards, a points list is created having two tuples with all the longitudes and latitudes(function idx_to_lonlat in the utils.py).

Finally, the function write_group_gml from the utils.py file is called. In this function, it is calculated the mean for both longitudes and latitudes and then gmplot is uded so as to visualize the selected trips. Below, there is an example of a trip shown on map is presented:

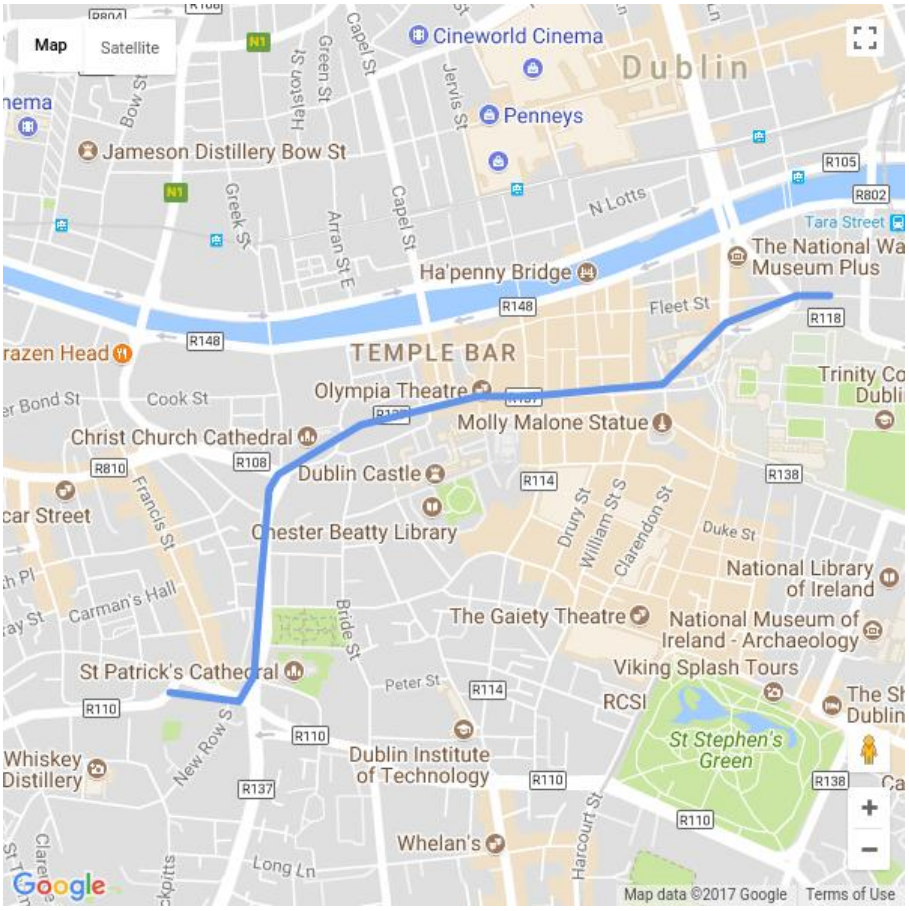

Figure 1: Example of trip visualization

3 Exercise 2

The files implementing the requirements of this part are in the folder SecondQuestion.

3.1 Question A1

For this part a test file is given (test_set_a1.csv, included in the input folder) containing additional trips. For each of these trips, the k nearest neighbours are targeted (in this exercise k=5) and an image file is produced, showing all six trips (the test trip and the trips from the training file) on six different maps. Under each map, the neighbour index, the journey_id, the dynamic time warping and the time needed to process the neighbours of this test trip in millisec.

The code for this question is in the file NearestNeighbours.py. Firstly, all test trips in the test file are extracted and stored in the same data structure as the trip list that contains the training dataset. Subsequently, for each test trip all nearest neighbours are calculated in the function find_nearest_neighbours_for_test_trip, keeping the time needed in a variable. At the beginning of the trips processing, the tic method from utils.py file is used (finds the current time in millisec using the datetime.datetime.now() method). After having found the nearest neighbours for the given test trip, the method tictoc() is called, returning the required time. In order to find the nearest neighbours, the following process is implemented: For the test trip and all the training trips, the longitudes and latitudes are stored in tuples which are provided to the function calculate_dynamic_time_warping in order to find the distance between two points. The calculated distances are stored in a list, which finally is sorted in ascending order and only the five nearest neighbours are kept.

Since now the nearest neighbour trips are found, the trips list is updated (function: get_updated_trips_list) containing now only those five trips. The final step now is to visualize the six trips. For this purpose, three list are created: a list with all the points (longitudes and latitudes), a list with the labels and a list with the colors (in this question the only color is the blue). All this data is provided to the function visualize_paths in the utils.py file in order to export the trips as map images.

In order to be able to create an image containing all the six maps each time, the rasterize.js file is required as well as phantomjs. In this function, both .html and .jpg files are created. First of all, the function write_group_gml is used so as to create the html files (this function is described above in question c). These html files are converted to images using the function html_to_png. File names are kept in a list and when the final image file is created (containing all six maps) these files are deleted. Afterwards, images as displayed as collection of plots and exported in the .jpg format.

An example of this question image output can be found below:

Figure 2: Example of nearest neighbours trips visualization

3.2 Question A2

Following the same logic as above, the given test file(test_set_a2.csv) is parsed storing the data in the usual format and the function find_similar_subroots_per_test_trip is used to determine the nearest sub-routes for each test trip.

Similarly to the above question, the points are stored as tuples and the required time is calculated using the relevant methods from the utils.py file. To begin with,

the method calc_lcss is called. In this function, the list of lonlat coordinate tuples are given as input. We keep all similar point values and indexes in two lists while iterating the two given lists. The distance between two points is calculated using the method calculate_lonlat_distance in the CleanData.py file. Based on the exercise requirements, two points are considered as equals if their distance is less than 200 metres. Therefore, if the points are equal, and the indexes are the first point of one of the given lists the cell is marked to unit cost and a new similar sequence is started. Otherwise, we continue to an existing similar sequence, increasing by 1 its length. On the other hand, if points are not equal, then a new subsequence should start, and therefore L[i][j] is set to zero.

explain z variable

Having now found the longest subsequences, we sort the indexes based on the length of the subsequences and we call the update_current_maxsubseq in order to update our result comparing previous subsequences. First of all, the new sequences found are iterated, checking if the length of the subsequence list has reached the given number (in our case 5). If there is still space to add the subsequence, then it is added, otherwise we check if the new subsequence is longer from those kept in the list and therefore it should replace it.

Finally, a similar data preprocessing is followed before visualize the data. This time the points are separated so as to show which of them belong to the nearest sub-route and color them with red color. The function for the visualization is the same that was used for the question 2A1.

3.3 Question B

The purpose of this task was to export features for classification for the training dataset. In short, a grid should be created and the points of each trip trace should be replaced by a specific square of the grid. The output file where the exported features are stored is the tripFeatures.csv and the tripFeatures.pickle, which was used in order to keep the final data structure with the features.

In order to be able to draw the grid, it was necessary to find the min and max longitudes and latitudes (function: find_min_max_latlong). Having now this information, the next function called is the create_grid giving as input the number_of_cells and the result of the previous method. The distance per cell in the grid is calculated as the difference between the max and the min lat/long divided by the total number of cells. The next step is to create the lines of the rows and the columns and then to provide a name for each cell.

Afterwards, all points in the trip list should be replaced by the cell names that each

point corresponds to. This logic is implemented in the method replace_points. For each trip in the list, all coordinates are transformed in tuples format and using the find_index function a name is mapped to this point. After having replaced all points with their new names, this list is exported in .csv and .pickle format. In order to be able to easily check the created grid, the method visualize_grid is implemented so as to plot the grid.

3.4 Question C

The file corresponding to this question is the JourneyClassification.py. First of all, the data structure is loaded from the .pickle file, created in above question as we have already mentioned. In the requirements of this question it is mentioned to use 10-fold cross validation, therefore the KFold class is used providing the required folds number. Then it is used in order to split our list to the training and test datasets.

explain pre-pro-cess-data func-tion

It worths to be mentioned here, that method asarrays from numpy was used to the necessary for this question lists. In addition, a list with the names of the three classifiers is created and for each of them, the relevant function is called in order to train the classifier and then count the accuracy.

In this question, the scikit learn library was used. Firstly, for the knn classifier the class KNeighborsClassifier was used, and more specifically the fit and predict methods. In order to calculate the accuracy of the classifiers we have used the accuracy_score method (this result is return from each classification function in the JourneyClassification.py). The same logic is followed for the other two classifiers(logistic regression and random forest). What should be mentioned, though, is that the LogisticRegression class implements the predict_proba method, which returns a list with probabilities and argmax from numpy was used in order to determine the result with the max probability.

3.4.1 Beat the Benchmark

References

[1] SciKit Learn: Nearest Neighbors
http://scikit-learn.org/stable/modules/neighbors.html

[2] SciKit Learn: Logistic Regression
http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[3] SciKit Learn: Random Forest
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

[4] SciKit Learn: Accuracy
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score

[5] A Complete Guide to K-Nearest-Neighbors with Applications in Python and R
https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[6] Dynamic time warping
https://en.wikipedia.org/wiki/Dynamic_time_warping

[7] Longest common subsequence problem
https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

[8] SciKit Learn: Classification report
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[9] ThreadPool
https://stackoverflow.com/questions/8533318/multiprocessing-pool-when-to-use-apply-apply-async-or-map