Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 042

Traffic Prediction Module Documentation

D. Kehagias, T. Diamantopoulos

September 2013

Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 042

Traffic Prediction Module Documentation

D. Kehagias, T. Diamantopoulos

September 2013

Traffic Prediction Module

DOCUMENTATION

The purpose of this document is to properly accompany the implementation of the traffic prediction module of the ECOMPASS project. This document presents the main parts of the traffic prediction module and explains their functionality. These parts are divided to the following:

- Road Network
- Data Input and Output
- Algorithm Execution

The aforementioned parts of the system are explained in the following paragraphs.

Road Network

The road network is loaded in the form of a shapefile (shp). The class `CNetworkParser` handles loading a network shapefile and constructing a `CNetwork` instance as well as the links and the nodes of the instance. `CNetworkParser` uses the GDAL library to load data features and map them to the `CNetwork` instance. Upon loading links and nodes, the `CNetwork` instance creates the roads of the network. Note that any straight network line is a link, whereas a road is defined as a segment between two intersections (i.e. containing arbitrary number of links). Links, nodes, and roads have their respective classes `CLink`, `CNode`, and `CRoad`. Each `CRoad` contains the links it covers, and two `CNode` instances that denote its starting and ending point. In addition, it contains its neighboring roads.

Data Input and Output

Input and output is handled by the `fpars` namespace. The namespace has two main functions, `LoadData` and `SaveKMLData`.

`LoadData` is used to parse the speed probe data and assign the speed records to the roads according to the link ID that each record has as well as the given link direction. The parsing is similar for both the training and the input (testing) dataset. In any case (training or testing data), the timestamp-speed pair is provided to the respective `Intervals` instance of the appropriate road. The `Intervals` object contains one

`Interval` instance per 5-minute interval. `Interval` instances contain arithmetic average, harmonic average etc. of speed for the 5-minute interval. The record is also given to `CDataStatistics` instance, that contains accumulative statistics for the whole file.

The `SaveKMLData` function handles saving the results of any algorithm in KML files. The function instantiates a `CKMLFileWriter` for each prediction interval that was asked. In specific, assuming the number of intervals to predict ahead is e.g. 2 and the testing data consists of intervals from 3 to 20, the function writes a KML file for interval 3 and 1 intervals ahead of 3, interval 3 and 2 intervals ahead of 3, interval 4 and 1 intervals ahead of 4, interval 4 and 2 intervals ahead of 4, interval 5 and 1 intervals ahead of 5, etc. The `CKMLFileWriter` handles saving the data in the appropriate KML format. Thus, the `SaveKMLData` function iterates over all roads for all predicted intervals and provides the predicted speed value to the appropriate `CKMLFileWriter`.

Algorithm Execution

The `CTrafficPrediction` class serves as an intermediate API between the user (or any) interface and the `CTrafficPredictionAlgorithm` class. All algorithms inherit the `CTrafficPredictionAlgorithm` class. The latter contains useful functions that allow algorithms to receive mean (or harm, etc.) speed values without using other objects. All algorithms are trained by implementing the virtual `TrainAlgorithm` function and are tested by implementing the virtual `RunAlgorithm` function. Each of the algorithms is analyzed in the following paragraphs.

***k*-Nearest Neighbors** The kNN algorithm (`CKNN`) selects certain neighboring roads of each road by finding the Coefficient of Determination (CoD) between them and the road to predict. Thus, for each road, the algorithm initially uses the `CreateAllRoadsForRoad` function which in turn calls the `GetCoD` function. The latter provides the CoD for 2 road series x and y at lag k which is computed as follows [1]:

$$CoD_{xy}(k) = 100 \left[\frac{E\left[(x_t - \mu_x)(y_{t+k} - \mu_y)\right]}{\sigma_x \sigma_y} \right]^2 \tag{1}$$

`CreateAllRoadsForRoad` returns the most well correlated neighboring roads to the road at hand. After that the algorithm calls the `CreateAllTrainingVectorsForRoad` function which creates the vectors for road r defined as:

$$x(t) = [V_r(t), V_r(t-1), V_r(t-2), V_{n_1}(t), V_{n_1}(t-1), V_{n_1}(t-2), V_{n_2}(t), \dots] \tag{2}$$

and the respective output value:

$$y(t) = V_r(t+1) \tag{3}$$

where $n_1, n_2, \dots$ are the well correlated neighboring roads and $V_r(t)$ is the speed of road r at time t.

The aforementioned vectors are saved to and loaded from disk (for training and testing respectively) using a black-box `CKKNFileWriter`.

Running the algorithm (using `RunAlgorithm`) iterates over all desired intervals and calls the `CreateInputVectors` function that creates an input vector similar to (2) for the testing data at time t. After that, the function `RunForIntervalForRoad` is called to find the k nearest neighbors of the created vector and output the predicted value by averaging their respective output values (see (3)).

STARIMA Space-Time Auto-Regressive Integrated Moving Average (STARIMA) is implemented by the `CStarima` class. Training the algorithm for a road involves calling the `CreateTrainingZArrayForRoad` function. The latter finds and retrieves the Z (and neighboring W) terms of the following equation [3]:

$$Z_{t+1} = \phi_{00} \cdot Z_t + \phi_{10} \cdot Z_{t-1} + \phi_{20} \cdot Z_{t-2} + \phi_{11} \cdot W_1 Z_t + \phi_{12} \cdot W_2 Z_t + \dots \tag{4}$$

where Z_t represents road speed(s) at time t, W_o is the neighbor matrix of order o and ϕ_{to} is the parameter(s) for road(s) of order o at interval t. After that, the algorithm finds the optimal ϕ parameters:

$$\phi = [\phi_{00}\ \phi_{10}\ \phi_{20}\ \phi_{11}\ \phi_{12}\ \dots] \tag{5}$$

using the function `ApproximateFUsingLeastSquares` which calls external library functions (OpenCV) and saves the approximation.

The approximated parameters are saved to and loaded from disk (for training and testing respectively) using a black-box `CStarimaFileWriter`.

`RunAlgorithm` iterates over all desired intervals and calls the `CreateInputVectors` function in order to create the Z (and neighboring W) terms of equation (4) for the respective testing set interval. After that, the function `RunForIntervalForRoad` applies equation (4) using the loaded ϕ values, and outputs the predicted value.

RandomForest The Random Forest (RF) is trained using the `TrainAlgorithm` function. The function creates 3 arrays, corresponding to the features of the algorithm, the responses, as well as a missing mask denoting. Each missing mask value denotes if the corresponding response is known or not. The functions `prepLocalInputs` and `prepResponses` are called to create the feature and responses tables respectively, which conform with the selection shown in [2]. After that the data is cleaned by discarding the vectors where the missing mask indicates there are no response values. The algorithm is trained using an external library function (OpenCV).

The models of the algorithm are saved to and loaded from disk (for training and testing respectively) using a black-box `CRandomForestFileWriter`.

Running the algorithm is similar. `RunAlgorithm` iterates over all desired intervals and calls the `prepLocalInputs` function to create the features of the road for the interval to predict. After that, the features are given to the respective RF that computes the output.

References

[1] Tao Cheng, James Haworth, and Jiaqiu Wang. Spatio-temporal autocorrelation of road network data. *Journal of Geographical Systems*, 14(4):389–413, 2012.

[2] Benjamin Hamner. Predicting travel times with context-dependent random forests by modeling local and aggregate traffic flow. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ICDMW '10, pages 1357–1359, Washington, DC, USA, 2010. IEEE Computer Society.

[3] Yiannis Kamarianakis and Poulicos Prastacos. Space-time modeling of traffic flow. *Comput. Geosci.*, 31(2):119–133, March 2005.