Traffic Flow Prediction Using Uber Movement Data

Daniele Cenni[1][0000−0002−9761−9111] and Qi Han[2][0000−0002−5856−383X]

[1] University of Florence, Florence, Italy
daniele.cenni@unifi.it
[2] Department of Computer Science, Colorado School of Mines, Golden, CO, USA
qhan@mines.edu

Abstract. The smart city paradigm is closely related to the orderly and sustainable use of the services it provides, on the efficiency of interconnections and communications that take place in an urban context. In this regard, one of the biggest challenges for smart city development relates to the prediction of traffic conditions. In fact, the city's road system has a decisive impact on air pollution, the management of public events, and in general on the efficiency of services offered to people, and thus strongly affects the city's economic development. In recent years, the development of increasingly effective machine learning and deep learning techniques has made a significant contribution to the definition of predictive models in the smart city domain. Deep learning techniques provide efficient results, but need significant computational resources to deal with huge and constantly updating datasets. Very often, however, the traffic data provided by cities are incomplete and insufficient to implement effective deep-learning models. In this paper, a novel solution for defining predictive models of traffic conditions is presented, based on road segmentation and urban traffic-related data, with the aim of dealing with the inherent complexity of geographical datasets. The obtained model has an average accuracy of 94.8%. The proposed architecture is able to reduce the inherent complexity of traffic related data, is easily scalable, can be quickly applied to any urban context.

Keywords: Crowdsourcing · Urban Traffic Dataset · Traffic Prediction · Data Processing.

1 Introduction

Traffic congestion refers to an excess of vehicles on a portion of the roadway at a particular time, resulting in speeds that are lower than normal (or *free flow*) speeds. Analyzing the primary causes of traffic congestion, we find that it depends on seven factors that often condition on each other, which can be divided into traffic incidents, work zones, weather; fluctuations in normal traffic, special events; physical road features, traffic control devices, physical bottlenecks.

The rapid emergence of deep learning techniques has enabled the definition of increasingly sophisticated models for traffic prediction, and numerous services

have been proposed in this regard. Commercial providers of traffic-related services include Google Maps Platform, Waze, TomTom's Traffic RESTful APIs, HERE Real-Time Traffic, ArcGIS Traffic Service REST APIs, and PTV Traffic Data. Google Maps analyzes historical traffic patterns for roads over time, but do not provide direct traffic forecasts for specific time horizons in terms of congestion metrics [1]. Waze collects information on traffic, accidents, and jams from its users every 2 minutes, which can be accessed via XML, JSON and Geo RSS data feeds [2]. However, this information is only useful if there is a sufficient number of vehicles in the area of interest, and traffic predictions are not provided. TomTom provides access to historical and real-time traffic incidents and traffic flow data that is updated every 30 seconds and provides predictions up to 24-hours ahead. HERE collects data every minute from various incident monitoring services and provides predictions for the next 12 hours, based on historical and real-time data [3]. ArcGIS allows visualization of traffic conditions that is updated every 5 minutes and provides traffic predictions for the next 4 hours shown in web maps [4]. PTV provides short-term traffic predictions, up to 60 minutes [5]. Each of these solutions has some advantages, but none can provide real-time traffic predictions with different time horizons, and more specifically they do not provide predictions with medium to long time horizons. In order to build efficient predictive models that can generalize well to any geographic scale with highly detailed time resolutions, it is necessary to deal with large amounts of data and the issues that come with it (e.g., noise from spurious data, data imputation, data reconciliation, use of data from different sources with different time resolutions).

This paper presents an effective solution for dealing with huge amounts of traffic data, to overcome all the above-mentioned issues, with the aim of providing fast and reliable traffic predictions at large geographical scales. Section 2 discusses the main techniques implemented for solving the traffic prediction task. Section 3 introduces the dataset used for this study, and various pre-processing strategies implemented to mitigate the complexity of the problem, clean the data, speed up the training time, and build a scalable solution. Section 4 describes the proposed model's architecture and setup. Section 5 describes the experiments and the obtained results, with relevant performance metrics. Section 6 describes a web application implemented to visualize traffic predictions in an easy and usable manner. Section 7 reports the conclusions with a sketch of future development.

2 Related Work

Traffic prediction is a problem that has benefited greatly from the introduction of low-cost deep-learning techniques. Generally, traffic prediction models are based on classical or deep learning methods. Classical methods include statistical methods and traditional machine learning methods, while deep learning methods include CNN, RNN, GCN, and attention mechanism. Some approaches make use of graph neural networks (GNNs) to model spatial dependencies, or

recurrent neural networks (RNNs) to capture temporal dependencies. However, in general RNNs are not effective tools in the case of temporal sequences, as they cannot capture their sequentiality, but are capable of modeling the periodicity and seasonality of the series. GNNs have limitations in terms of traffic prediction, since they model spatial dependencies statically, and thus have a limited capacity to learn dynamic traffic patterns. Often, they only consider short-range spatial dependencies and cannot capture long-range ones. In general, GNNs do not consider that traffic conditions propagate with a time delay between locations. To overcome some of these problems a novel Propagation Delay-aware dynamic long-range transformer was proposed, for accurate traffic flow prediction, designed with a self-attention module to capture the dynamic spatial dependencies [6].

Since the spatial dependence is usually treated as a static graph, a Dynamical Spatial-Temporal Graph Neural Network model (DSTGNN), creating a spatial dependence graph based on the stability of node's spatial dependence, to capture the dynamical relationship. The changing demand process is modeled using a Poisson process to address the interpretability and build a spatial-temporal embedding network (integrating the diffusion convolution neural network and a modified transformer) that can infer the intensity [7]. Other algorithms often used for traffic prediction are Historical Average (HA), Auto-Regressive Integrated Moving Average (ARIMA), and Vector AutoRegressive (VAR). However, these techniques are only applicable to datasets of small size, while time-varying traffic data is huge. In order to avoid the problem of dealing with high dimensionality and to capture complex nonlinear relationships, other approaches such as Support Vector Regression (SVR) [8] and Random Forest Regression [9] have been proposed. Using geospatial dependencies introduces high complexity into the model, which makes it difficult to scale. State-of-the-art solutions for processing geospatial dependencies require not inconsiderable computational resources and, despite the application of distributed computing techniques (e.g., Dask and multi-GPU multi-node architectures), do not allow the training phase of the model to scale effectively. Our solution consists of a platform for processing geospatial data, which aims to reduce their complexity, so that models can be built to perform the learning phase in reasonable time, and accurate predictions, enabling the big data domain in the field of traffic data.

3 Data and Architecture

With the aim of avoiding the use of complex datasets that do not provide data over time and that require using metrics that are not easily measured, we chose to model traffic intensity using the average speed of vehicles in each road segment. A road segment (*osm_way_id* in the OpenStreetMap's notation) is the specific representation of a road's portion with uniform characteristics. Vehicle speed, being a direct metric for measuring traffic intensity, is representative when calculated as an average value over a given time interval. For this reason, traffic related data used in this study were retrieved from Uber Movement, (c) 2023

Uber Technologies, Inc. [10], for the city of San Francisco, US. Uber provides hourly time series data with historical speeds (mph), for various cities, and travel times, and allows for a detailed dataset to produce traffic predictions with high spatial resolution. This dataset provides the average hourly speed on a given road segment for each day of January 2019, including only road segments with at least 5 unique trips in that hour. Preliminarily a osm file for the region of interest was retrieved with the aim of OpenStreetMap API [11], and then converted to a ESRI shapefile. The bounding box used was [-122.514, 37.716, -122.372, 37.859]. In OpenStreetMap, bounding boxes are expressed as four comma-separated numbers, in this order: left, bottom, right, top ([*min longitude, min latitude, max longitude, max latitude*]). Latitude and longitude are expressed in decimal degrees (north latitude is positive, south latitude is negative, west longitude is negative, east longitude is positive). The dataset is reported in Table 1, with the geometry column using EPSG:4326 (WGS84) as a geodetic system. Feature *osm_way_id* was renamed from to *osm_id*, in order to merge this dataset with the above built shapefile. Then, the speed movements' dataset was merged with the road segments' dataset, based on the common column *osm_id*. In this way we got a dataset with the fields reported in Table 2 including the geometries of each road segment. After merging, data rows with *null* values were removed from the resulting dataset. The geometry column was then reprojected to the EPSG:3857 projected system, and a new column was created, representing the weekday for each row.

The use of a projected system does not result from the need to map coordinates in the plane to deal with euclidean distance metrics, but only for calculating the centroid of each road segment, for visualization purposes. The advantage of using road segments instead of road geometries is that the traffic model can also be trained for large geographical areas, covering different projected systems, and traffic data are analyzed at a high geographical resolution. Timestamp was also converted to the local timezone of interest (US/Eastern), and a datetime-index was built on the dataset, to further resample the data with time slots of [2, 3, 4, 6, 8, 12, 24] hours. For this goal, data were grouped by *segment_id* and resampled by *datetime*, taking the mean of *speed_mph_mean* in the time period of interest, and removing *null* speed values. Columns [*day*, *hour*, *week*, *weekday*] were also extracted from the timestamp, converted to categorical (as well as *segment_id*), and included in the dataset. In this way the temporal component is taken into account, without the need to handle the inherent complexity of time sequences. Once completed this pre-processing phase, each dataset's row was reporting a traffic event, including the average speed value (mph) for a particular road segment, at a particular time (*year*, *month*, *day*, *hour*). With the goal of reducing the task to a classification problem, speed values were converted to classes, labeling as *low speed* values in the range [0, 25] mph, *medium speed* values in the range (25, 45] mph, and *high speed* values in the range above 45 mph. Low speed maximum threshold is derived from the definition of low speed vehicle, as reported in the rule 63 FR 33194 (Federal Motor Vehicle Safety Standards), published by the National Highway Traffic Safety Administration. Columns [*day*, *hour*, *week-*

day, *segment_id*] were considered as dataset features. Data were split into train and test sets with a 80/20 ratio, preserving the majority/minority classes ratio. To avoid the curse of dimensionality and to further reduce the complexity of the problem, features were encoded using target encoding [12], since using one-hot encoding would create a sparse matrix and increase the number of dimensions. Using target encoding, the labels are directly correlated with the target. For instance, in mean target encoding, the feature label is calculated by averaging the values of the target variables, for each category. In the case of the categorical target, features are replaced with a blend of posterior probability of the target, given a categorical value and the prior probability of the target over all the training data. This technique is particularly useful in the present case, because it does not increase the size of the data and allows for shorter training time. Regularized target encoding outperforms traditional encoding methods, in the case of high cardinality features [13]. For target encoding, different combinations of *smoothing* and $min\_samples\_leaf$ parameters were tested with no significant changes in the accuracy of the resulting model. The general architecture of the proposed solution is depicted in Fig. 1.

Table 1: Road Segments Data

Column Name	**Description**	**Type**
osm_id	Year (local city time)	int
name	Road's name	text
highway	Highway's type (e.g, residential, motorway, service)	text
waterway	Waterway's type (e.g., stream, dam, ditch, drain)	text
aerialway	Aerialway's type (e.g., gondola)	text
barrier	Barrier's type (e.g., retaining_wall, fence, wall, kerb, hedge)	text
man_made	e.g. breakwater, reservoir_covered, pipeline, tower, sign	text
railway	e.g. rail, subway, light_rail, tram	text
z_order	attribute to model the above/below relat. btw elements	int
other_tags	other unused tags	text
geometry	the roads' geometry with a LINESTRING shape	object

4 Traffic Model

Prediction tasks are related to flow (i.e., number of vehicles passing through a given point on the road at a certain time), vehicle speed, demand (i.e., the number of start or end in a region at a certain time), travel time, occupancy (i.e., the extent to which vehicles occupy the road). A system for computing traffic metrics based on road segments saves time in dataset setup and training, because it avoids computing geographic clusters within which to measure traffic, and it

Table 2: Speed Movements Data

Column Name	**Description**	**Type**
year	Year (local city time)	int
month	Month 1-12 (local city time)	int
day	Day 1-31 (local city time)	int
hour	Hour 0-23 (local city time)	int
utc_timestamp	Unix time for start of hour (UTC)	int
segment_id	Replaced by OSM value "osm_way_id." Movement ID that maps to a specific road segment.	text
start_junction_id	Replaced by OSM value "osm_start_node_id." Movement ID that maps to start intersection of traversal.	text
end_junction_id	Replaced by OSM value "osm_end_node_id." Movement ID that maps to end intersection of traversal.	text
osm_way_id	Corresponding OpenStreetMap Way ID for this segment. Note that one OpenStreetMap Way may contain multiple Movement segments.	bigint
osm_start_node_id	Corresponding OpenStreetMap Node ID for this junction.	bigint
osm_end_node_id	Corresponding OpenStreetMap Node ID for this junction.	bigint
speed_mph_mean	Average speed of Uber vehicles on this road segment in mph.	float
speed_mph_stdev	Standard deviation of speeds on this road segment in mph.	float

makes it unnecessary to work with geographic features that introduce nonlinear patterns and are difficult to manage. Statistical techniques are not suitable for very large datasets, while deep learning models are often used to deal with big data. Though, in many cases you have to deal with some cities providing large amounts of data (i.e., for many years with a daily time resolution), and others providing few data with a different time range. Deep learning models are computationally expensive, often require the use of GPU clusters, and have high memory requirements that increase rapidly with dataset size, which could make the problem unmanageable. The development of scalable solutions capable of handling large amounts of data (i.e., implementing efficient traffic prediction for large geographical areas and time intervals), and able to update quickly, requires reducing the dimensional complexity of the dataset and using an approach that eliminates the criticality of the various techniques reported. For these reasons, in order to build a scalable solution we chose to implement both a Random Forest model and a gradient boosting model, using the XGBoost library [14], with a *multi:softprob* objective function (to get class probabilities as model's predictions). Model's outputs are probabilities for each traffic congestion related class (i.e., *low*, *medium* and *high speed*), which are suitable to show the prediction's confidence with a color gradient scale on the map. Tree-based models are particularly well suited to be used in this context, since they outperform deep learning approaches on tabular data [15]. The model works as a probabilistic classifier that generalizes the notion of classifiers. Thus, for a given $x \epsilon X$, it assigns prob-

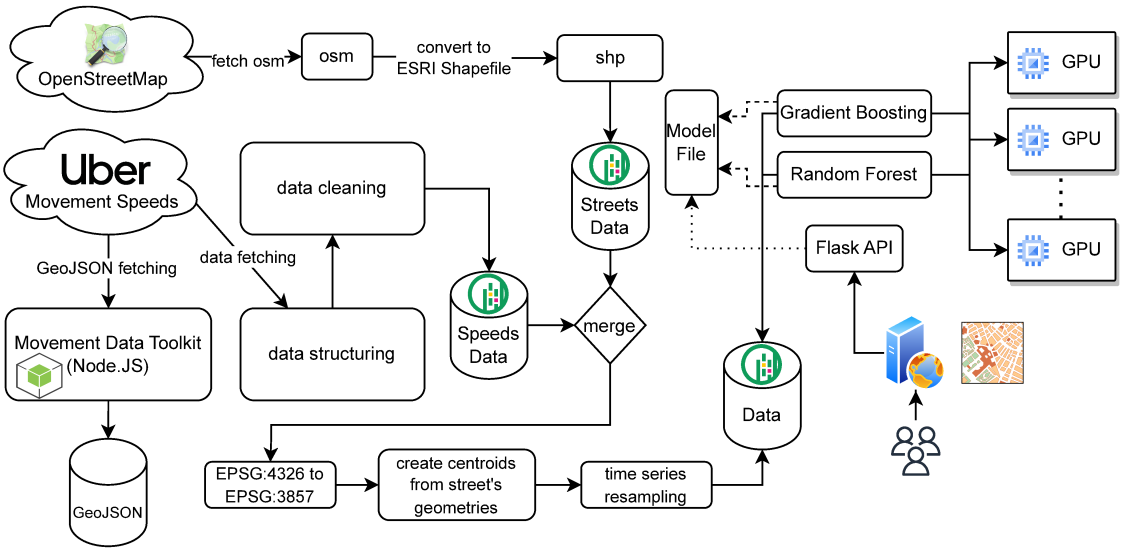

Fig. 1: General Architecture

abilities to all $y \epsilon Y$, and probabilities sum to one. Standard classification can be obtained using the optimal decision rule $\hat{y} = \arg\max_y Pr(y = Y|X)$, where X and Y are respectively the features' set and the model's output. Model's training took place on a Nvidia Titan XP GPU, featuring 3840 CUDA Cores, 12 GB GDDR5X. XGBoost model has the advantage, over a Random Forest model, of being incrementally updatable, for example with traffic data from previously untracked areas. In this regard, an incremental scheme was implemented, to further train the model with new data without the need of training from scratch every time. At the end of training the model is saved to a file and can be loaded later for further training. In this way the proposed architecture is able to seamlessly scale up without the need of training from the beginning in case of new data. The model can automatically cope with class imbalance, by taking into account class weights if appropriate. At this regards, it was preferred not to implement strategies to mitigate class imbalance (e.g., SMOTE [16]), since such approaches are computationally expensive, increase the size of the dataset and consequently the training time, and most importantly add additional noise to the dataset. The described solution supports using parallel computing frameworks (e.g. Dask) to leverage multi-nodes and multi-GPUs for faster training.

5 Experiments and Discussion

Training was performed for various time slots (i.e., [1, 2, 3, 4, 6, 8, 12, 24] hours). A *n-hour* time slot is defined as a time period spanning all the day with time slots of *n-hours* each. For example, 8-hour includes time slots [00:00-08:00),

[08:00-16:00), [16:00-00). Relevant training parameters for gradient boosting include learning rate (0.3), maximum number of discrete bins to bucket continuous features (256), grow policy (depthwise), L2 regularization term on weights (1), minimum loss reduction required to make a further partition on a leaf node of the tree (0), maximum depth of a tree (6), maximum number of nodes to be added (6). Random Forest model was trained defining the fraction of randomly selected features used to train each tree (80%), learning rate (1), maximum depth (5), number of parallel trees (100), the fraction of the randomly selected training samples used to train each tree (80%). For each training, performance metrics were calculated for the best iteration number, and they are reported in Table 3. By averaging each model's accuracy, calculated for each time slot, we get an accuracy of 94.78%, with the XGBoost model, which generally outperforms other machine learning or deep learning approaches (e.g., CNN, RNN) [17–19], and 94.81% with the Random Forest model. Training times for each model highlight the efficiency of the model, which is able to scale easily with large amounts of real-time data. (31.1 s for 1H model, 13.5 s for 2H model, 10.4 s for 3H model, 7.06 s for 4H model, 5.32 s for 6H model, 4.18 s for 8H model, 3.05 s for 12H model, 1.74 s for 24H model). Classification metrics for the worst and best cases are reported in Table 4. Confusion matrixes for the worst and best cases are reported respectively in Fig. 2. In the worst and best cases, training times were respectively 31.1 s and 1.74 s, with the traffic dataset of the whole area of San Francisco (4,771,410 rows in the training set and 1,192,853 rows in the test set).

These metrics are reported only for the XGBoost model, since Random Forest showed comparable results. Features' importances, calculated for 1-hour and 24-hours models (XGBoost), show the features' impact on the model's output magnitude, with road segment's by far the most important feature). The importance was calculated using the cover strategy, thus calculating the average coverage of splits that use the feature, where coverage is defined as the number of samples affected by the split. Results indicate that *segment_id* has the biggest impact on the model's output, and *hour* is of course useless for the 24-hour model (i.e., the bigger the model's time slots, the lesser the importance of time-dependent features). The impact on city traffic is thus affected more by the structure of arterial roads than by the time related features, which can exacerbate critical issues arising from bottlenecks generated by poorly sized roads. Among the temporal components, the greater importance of *hour* is highlighted, as opposed to *day* or *weekday*. In general, temporal features with fine-grained resolution (i.e., hour, minutes) have a greater impact on the traffic predictions, but models with such features are noisier and have worse performance in terms of accuracy. Despite its relative importance, *weekday* is an important feature that deserves consideration. In the present experiments, the relevance of this feature is limited, having considered one month, whereas for larger time intervals it is an indicator of the seasonality of traffic trends.

Table 3: Model's performance metrics XGBoost (Random Forest)

Accuracy Score	**Balanced Accuracy Score**	**F1 Score**	**Geom. Mean Score**	**Cohen Kappa Score**	**Matth. Corr. Coeff.**	**Model**
0.971 (0.971)	0.931 (0.931)	0.940 (0.940)	0.930 (0.930)	0.902 (0.903)	0.902 (0.903)	24H
0.961 (0.962)	0.936 (0.937)	0.937 (0.937)	0.934 (0.936)	0.880 (0.881)	0.880 (0.881)	12H
0.954 (0.955)	0.922 (0.922)	0.924 (0.925)	0.920 (0.921)	0.865 (0.866)	0.865 (0.866)	8H
0.948 (0.949)	0.917 (0.919)	0.917 (0.919)	0.915 (0.918)	0.849 (0.852)	0.849 (0.852)	6H
0.944 (0.945)	0.915 (0.915)	0.915 (0.917)	0.914 (0.914)	0.846 (0.848)	0.846 (0.848)	4H
0.941 (0.941)	0.909 (0.909)	0.913 (0.913)	0.908 (0.908)	0.842 (0.842)	0.842 (0.842)	3H
0.936 (0.936)	0.910 (0.909)	0.909 (0.909)	0.908 (0.907)	0.831 (0.832)	0.832 (0.832)	2H
0.926 (0.927)	0.902 (0.902)	0.900 (0.901)	0.900 (0.900)	0.813 (0.813)	0.813 (0.813)	1H

Table 4: Model's Classification report 1-hour (24-hour) (XGBoost)

	precision	**recall**	**f1-score**	**support**
low speed	0.94 (0.98)	0.96 (0.99)	0.95 (0.98)	901925 (63914)
medium speed	0.84 (0.92)	0.78 (0.90)	0.81 (0.91)	257554 (12414)
high speed	0.89 (0.95)	0.93 (0.91)	0.91 (0.93)	33374 (1457)
accuracy			0.92 (0.97)	1192853 (77785)
macro avg	0.89 (0.95)	0.89 (0.94)	0.89	1192853 (77785)
weighted avg	0.92 (0.97)	0.92 (0.97)	0.92	1192853 (77785)

6 Web Map

To visualize traffic predictions in an easy way, we developed a web application by converting the dataset with predictions in a GeoJSON format. GeoJSON is a text-based format that uses a simple data structure and is ideal both for accessing and creating geospatial data. The conversion process was automated with a Python script that reads the original dataset and builds the GeoJSON structure, for each date (i.e., day, month, year). The dataset was built using the features *day*, *hour*, *weekday*, *segment_id*, *geometry*, with three class labels (0: *low traffic*, 1: *medium traffic*, 2: *high traffic*). Predictions provide an array with [*y_prob*_0, *y_prob*_1, *y_prob*_2], where *prob_x* represents the probability of class x . Each traffic layer is represented as a GeoJSON *FeatureCollection*, with *LineString* geometries representing the roads. Speed related classes were converted to colors, *dark red* for low speed (*high traffic*), *red* for medium speed (*medium traffic*), *orange* for high speed (*low traffic*). For the purpose of visualizing traffic data on the map, the road's geometries were converted to the EPSG:4326 geodetic system. The web application is structured as an interactive map, built with OpenStreetMap and the Javascript library Leaflet, allowing to zoom and drag.

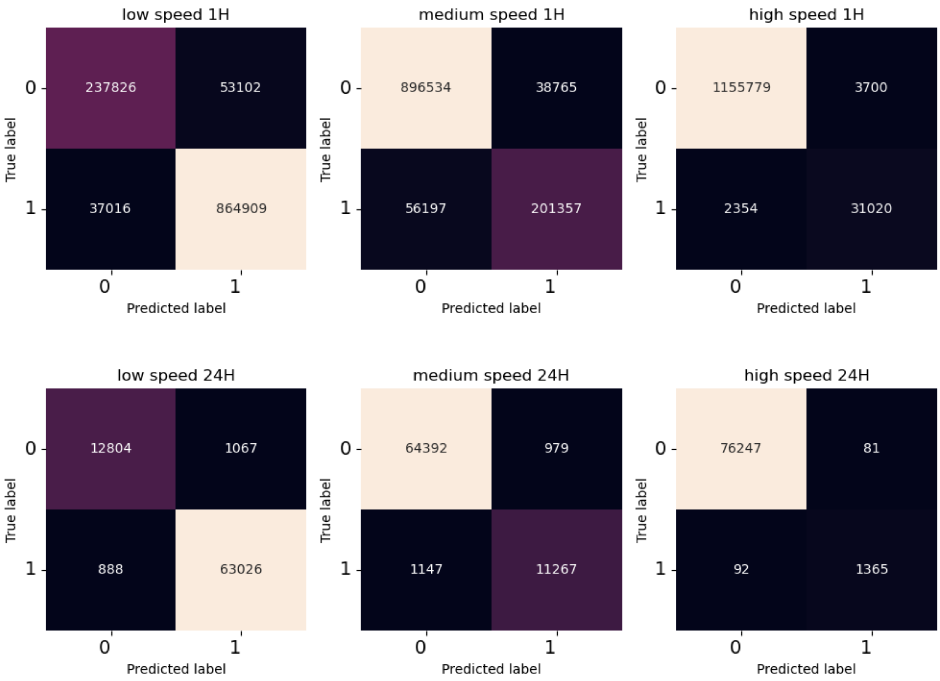

Fig. 2: Confusion Matrixes [1, 24 hours] (XGBoost)

The user is able to select a date and then the traffic data is asynchronously loaded and shown on the map, using a color for each road segment, depending on the traffic intensity. In this way, it is possible to give an insight into traffic conditions at first glance. Users can select and switch between dates and time resolutions with their respective select menus, and browse different traffic maps in a seamless way, by clicking on appropriate arrow buttons. Since large amounts of data can be handled with GeoJSON, depending on the portion of the displayed map, a spinning wheel is shown as the layers with traffic data are loaded, to warn the user to wait for the completion of asynchronous data loading. Clicking of each road segment will pop up the probabilities of each predicted class. Predictions can be obtained from a Flask REST API that exposes the traffic model, or are read from a batch generated file. The use of a traffic heatmap is a valuable aid in getting an immediate perception of traffic conditions, and allows for easy identification of traffic jams.

7 Conclusion

In this paper we introduced a solution for processing and managing traffic related data, with the goal of providing accurate predictions of city-related traffic intensities. In general, traffic predictions can be derived from statistical, machine

learning or deep learning based models, but dealing with geographical features is often a complex task that prevents the model's architeture to scale up efficiently. Statistical approaches allow to identify traffic patterns at different time scales, and are easier to implement, though they are less accurate and are not best suited to deal with multivariate data. Deep learning models are more complex, but they are difficult to handle when the size of the dataset grows exponentially, which is the case with real-time traffic data covering large geopraphical areas. With the goal of mitigating the curse of dimensionality, without reducing the information content of the traffic datasets, this paper implemented a strategy for creating a compact predictive model that can be trained on clusters of GPUs, even incrementally so as to update as new traffic data arrives (both historical and real-time traffic data can be used to generate valuable traffic related predictions with our platform). The proposed solution turns out to be easily scalable, upgradeable with new data to take into account larger geographical areas, and has an average accuracy of 94.7% (i.e., calculated by averaging the models' accuracies at various time slots). These results outperform traditional machine learning or deep learning approaches, with the clear advantage of constituting a streamlined and efficient solution that can be easily implemented at large scales with reasonable hardware resources. Cities can make use of this platform to curb critical traffic event issues, with a clear benefit from a pollution and energy-saving perspective, benefiting the productivity and economic development of urban settings. Future directions include taking into account external data (e.g., weather data, social media) and using transfer learning techniques to perform spatio-temporal predictions.

Acknowledgment

Traffic data used for this study were retrieved from Uber Movement, (c) 2023 Uber Technologies, Inc. This work is supported in part by the NSF project CNS-1932482.

References

1. "Google Maps Platform," https://developers.google.com/maps/documentation/javascript/trafficlayer, 2023, [Online; accessed 25-Jun-2023].
2. "Waze," https://support.google.com/waze/partners/answer/10618035?hl=en, 2023, [Online; accessed 25-Jun-2023].
3. "HERE," https://www.here.com/platform/traffic-solutions/real-time-traffic-information, 2023, [Online; accessed 25-Jun-2023].
4. "ArcGIS," https://developers.arcgis.com/rest/network/api-reference/traffic-service.htm, 2023, [Online; accessed 25-Jun-2023].
5. "PTV," https://www.ptvgroup.com/en/solutions/products/ptv-maps-data/traffic/real-time-traffic-data/, 2023, [Online; accessed 25-Jun-2023].
6. Jiang, J., Han, C., Zhao, W. X., and Wang, J., "Pdformer: Propagation delay-aware dynamic long-range transformer for traffic flow prediction," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, pp. 4365–4373, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/25556

7. Huang, F., Yi, P., Wang, J., Li, M., Peng, J., and Xiong, X., "A dynamical spatial-temporal graph neural network for traffic demand prediction," *Information Sciences*, vol. 594, pp. 286–304, 2022. [Online]. Available: https://doi.org/10.1016/j.ins.2022.02.031
8. Chen, R., Liang, C.-Y., Hong, W.-C., and Gu, D.-X., "Forecasting holiday daily tourist flow based on seasonal support vector regression with adaptive genetic algorithm," *Applied Soft Computing*, vol. 26, pp. 435–443, 2015. [Online]. Available: https://doi.org/10.1016/j.asoc.2014.10.022
9. Johansson, U., Boström, H., Löfström, T., and Linusson, H., "Regression conformal prediction with random forests," *Machine Learning*, vol. 97, no. 1, pp. 155–176, Oct 2014. [Online]. Available: https://doi.org/10.1007/s10994-014-5453-0
10. "Uber Movement," https://movement.uber.com, 2023, [Online; accessed 25-Jun-2023].
11. OpenStreetMap, "OpenStreetMap API," https://wiki.openstreetmap.org/wiki/API, 2023, [Online; accessed 25-Jun-2023].
12. Micci-Barreca, D., "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," *SIGKDD Explor. Newsl.*, vol. 3, no. 1, p. 27–32, jul 2001. [Online]. Available: https://doi.org/10.1145/507533.507538
13. Pargent, F., Pfisterer, F., Thomas, J., and Bischl, B., "Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features," *Computational Statistics*, vol. 37, no. 5, pp. 2671–2692, Nov 2022. [Online]. Available: https://doi.org/10.1007/s00180-022-01207-6
14. Chen, T. and Guestrin, C., "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785
15. Grinsztajn, L., Oyallon, E., and Varoquaux, G., "Why do tree-based models still outperform deep learning on typical tabular data?" in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. [Online]. Available: https://openreview.net/forum?id=Fp7__phQszn
16. Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P., "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, p. 321–357, jun 2002. [Online]. Available: https://dl.acm.org/doi/10.5555/1622407.1622416
17. Liu, Y. and Wu, H., "Prediction of road traffic congestion based on random forest," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 2, 2017, pp. 361–364. [Online]. Available: https://doi.org/10.1109/ISCID.2017.216
18. Kurniawan, J., Syahra, S. G., Dewa, C. K., and Afiahayati, "Traffic congestion detection: Learning from cctv monitoring images using convolutional neural network," *Procedia Computer Science*, vol. 144, pp. 291–297, 2018, iNNS Conference on Big Data and Deep Learning. [Online]. Available: https://doi.org/10.1016/j.procs.2018.10.530
19. Ma, X., Yu, H., Wang, Y., and Wang, Y., "Large-scale transportation network congestion evolution prediction using deep learning theory," *PLOS ONE*, vol. 10, no. 3, pp. 1–17, 03 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0119044