Internet of Things Report - Air Pollution-Based Traffic Management System

May 10, 2016

Personal Contribution

My main task was to take care of the development of the cloud platform which would retrieve sensor data, feed them into a predictive analytics system, and output a webpage with the predicted traffic and routing information. The IBM Bluemix application development falls into 2 distinct parts. The first is the data gathering part which consisted of building our own database with existing data to build a statistical model that could provide real-time predictive analysis. The second part is the predictive model which is the actual working Traffic Prediction System based on Air pollution levels. This model is able to make the prediction based on the statistical model built using our "home-made" database.

1 Data Gathering & Statistical Modeling in Bluemix

We envisioned to build our own statistical model based on NO_2 levels, location, and time of the day, in order to predict the level of congestion given these parameters. In order to build our own database and classifier, we needed to gather data about the NO_2 levels at particular locations of London in real time along with real-time traffic information at these exact locations.

1.1 Gathering Air Pollution Information & NO_2 Levels

To gather air pollution information of the targeted locations in South Kensington, I integrated the London Air API developed by King's College University[1] which provided real-time (See Discrepancies [3]) information about various gases in various parts of London. As a team, we chose South Kensington because the API offered 3 locations that were relatively close to each other in that area, hence providing good grounds for the traffic management system. Additionally, we settled on NO_2 instead of CO gas because Kensington locations did not provide Carbon Monoxide readings. This API was integrated in NodeRed with the HTTP GET function that pulled filtered for each of the three locations every 15minutes using the polling node (See Appedinx A [3.5]).

The REST URL polled for each location resembled the URL shown in Figure 1.

```
msg.url = "http://api.erg.kcl.ac.uk/AirQuality/Data/Site/SiteCode=KC5/StartDate=2016-04-14/EndDate=2016-12-31/Json";
msg.method = "GET";
```

Figure 1: URL and Method of the London Air API Request for NO_2at KC5 Location.

1.2 Gathering Traffic Information & "Jam Factor"

Now that I had the NO_2 Gas information for each Kensington location every 15 minutes, I could gather the corresponding traffic data using a traffic API. It is interesting to note that the London Air API returned the coordinates of the monitored locations. I used these coordinates as parameters in the Flow endpoint of Traffic API provided from "HERE" to retrieve traffic information for these locations [3]. The HERE Traffic API has a "bbox" parameter which uses the top left corner coordinate of the map (for the targeted location) along with its bottom right corner. The REST URL polled for each location resembled the URL shown in Figure 2

```
msg.url = "http://traffic.cit.api.here.com/traffic/6.1/flow.json?bbox=51.492384%2C%20-0.193234%3B51.491912%2C%20-0.192263&app_code=LF2rQzk
msg.method = "GET";
```

Figure 2: URL and Method of the HERE Traffic API Request for Jam Factor at KC5 Location.

The "HERE Traffic API" calculates the "Jam Factor", which is the level of traffic, by building an index based on Congestion Index (% congestion of the crossing) and vehicle speed. This Jam Factor falls in a range between 1 and 10. I polled this Jam Factor every 15 minutes for each of the locations.

1.3 Pushing to Cloudant Database

Now that I had all the information necessary to build a database, I modeled a JSON object called "Object Creator" that consisted of the location, NO_2 level, Jam Factor, lattitude, longitude, and timestamp of the reading. The object structure is shown in Figure 3. This JSON object was pushed to a Cloudant Database at the polling rate of 5 minutes for a period of 2 weeks. It is currently still running and all elements can be found in our public database [4]. Additionally all of the NodeRed model code for the data gathering phase can be found in the UCL Dropbox under the name: "nodered-flow.json" (See Appendix A [3.5]).

```
msg = {
        "location": msg.gas.location,
        "NO2Level": msg.gas.NO2,
        "TSGAS": msg.gas.TSGAS,
        "JamFactor": msg.jam.jam,
        "TSJAM": msg.jam.TSJAM,
        "lat": msg.jam.lattitude,
        "lon":msg.jam.longitude,
        "ts": msg.timestamp.timestamp
    }
```

Figure 3: Object Creator JSON Object for Cloudant Use.

1.4 Manipulating the Data to Build Classes

After two weeks, I downloaded the data (See sensordata.csv file in UCL IOT Dropbox) from Cloudant using a modified version of the Python extractor code provided by UCL (See CSVCreator.py file in UCL IOT Dropbox). With the help of Lorrie, we created a classifier (See data-classifier.txt file in UCL IOT Dropbox) that could be fed into a statistical modeler such as MATLAB or SPSS. The classifier consisted of the headers: Location Class (KC3, KC4, or KC5 transformed to 3,4 & 5), NO_2 Level (Continuous data), Timestamp Class (from 0 to 23 for each "hour"), and a Jam Factor Class. For the Jam Factor Class, we tranformed the index to a range of 1 to 5 - 1 being very low and 5, very high. This was done by assigning Jam Factor ranges from 0 to 2 to a class of 1, range 2 to 3.5 to a class of 2, and so on until 10.

1.5 Building the Real Time Prediction Model

IBM Bluemix offers a Predictive Analysis Service[5] which offers an API that enables me to send it the real time NO_2 and Time Class readings and it will return a prediction of the Jam or traffic level for each area based on the statistical model uploaded to it. IBM offers their own SPSS Modeler software which allows us to build the statistical model to upload to the Predictive Analysis Service. Using an excel version of the classifier built with Lorrie, I uploaded the data as an input and pointed out the various headers as continuous or nominal along with the target data set, in this case, the Jam Factor (See Appendix B [3.5]). When running, the modeler creates various models such CHAID or Neural Net along with their accuracy (See Appendix C [3.5]) and creates a distribution chart that counts the occurances of Predicted Jam factor compared to the actual (See Appendix D [3.5]). This flow model (See Appendix E [3.5]) is then uploaded to IBM Bluemix's Predictive Analysis Service that will then be used as a REST endpoint to send the real time $NO_2$and Time data to.

2 Prediction Model in Bluemix

The prediction model in Bluemix is the actual application that retrieves NO_2 and timestamp data from the sensors and pushes them to the Prectivie Analysis API to retrieve the predicted Jam Factor, or traffic level.

Once done, the routing algorithm takes care of assigning reroute information based on the different traffic levels at each location and outputs everything on a traffic management dashboard. To provide a more realistic view, this report's demonstration of the Bluemix flow uses the data retrieved from the London Air API. However, we can remove the London Air API endpoints and instead hook up our Gas sensors to the platform. Example of this is shown in code provided by Syed in the UCL IOT Dropbox (serialRead.py) which includes the sensor authentication and gas data.

2.1 Retrieving the Predictions

The sensors send their NO_2 readings along with a time class and coordinates to NodeRed and these are in turn sent to the Predictive Analysis Service endpoint using an HTTP POST node. IBM's PA API returns a prediction of the Jam factor (or traffic level) along with a confidence rate for each location. The API call requires to point to the classifier table's name, which resides in the SPSS model, along with the appropriate headers being Location, NO2 Level, Time, and estimated Jam Factor (which is actually unused by the API). The API call's structure is shown in Figure 4a and the API's response is shown in Figure 4b.

```
msg.url = "https://palblyp.pmservice.ibmcloud.com/pm/v1/score/jam?accesskey=llndXBllwR
msg.payload = {
                "tablename":"sdata3.xlsx",
                "header":["Location", "NO2", "Time", "JamFactor"],
                "data":[[5, NO2KC5, ts, 1]]
              };
msg.method = "POST";
```

(a) URL and Method of the Predictive Analysis API Request for Predicted Jam Factor at KC5 Location.

[{"header":["Location","NO2","Time","JamFactor","$XS-JamFactor","$XSC-JamFactor"],"data":
[[5.0,175.0,14.0,1.0,4.0,0.18533288582813345]]}]

(b) Response from the Predictive Analysis API Request for Predicted Jam Factor at KC5 Location.

2.2 Routing Algorithm

Having the predicted traffic level for each location, I built a rudimentary routing algorithm that would redirect incoming cars to an area with lower traffic density. The algorithm is based solely on the predicted Jam factor class which is a manipulation of the predicted jam factor (see section 1.4 for more details). For cases where the traffic jam factor classes are 4,4,2 for the locations (KC5, KC4, and KC3) then a route is randomly selected between the high traffic nodes (KC5 and KC4) and the low one (KC3). Obviously this routing algorithm would have to be much more complex if the number of nodes were to grow. Using 3 nodes was a viable amount of nodes to work with to develop a proof of concept. The routing algorithm code can be found within the Routing Algorithm node of the NodeRed Prediction Model Flow (See Appendix F [3.5]). Additionally, the screenshot in Appendix G [3.5] illustrates the routing algorithm's outcome for traffic level prediction of 4, 3, and 2.

2.3 Mapping the Data

Now that I have the NO2 readings, the predicted Jam Factor Class, the route, and a timestamp I can map all this on a simple web page in order to show the routing based on the predicted traffic levels in real time. I built a simple AngularJS dashboard application that uses Firebase - a real-time data transmission framework to which the data is pushed to throught Firebase Node in Appendix F [3.5], ChartJS for building a bar graph out of that data, and Google Maps API to map the locations, route, and illustrate the traffic levels. This application is hosted at **ucliot.github.io** . Additionally, all of the data created by the Traffair system is available as an API to extend the use of the system - the data can be seen on https://traffair.firebaseio.com/data.json

2.4 Pushing it to Cloudant

These data sets were then placed into data objects (eg. KC5-Extract) and pushed to another Clouding Database for Lorrie to build a Matlab rendition of a 1 week timeframe of this data. Additonally, the data is available as an API to extend the use of the system, an example of the Traffair API output can be found in the UCL IOT Dropbox under the file name "traffair-api-data.json".

2.5 Twitter

I attempted to build a Twitter feed to create an alert system with the rerouting information. The Twitter account @Traffair along with the NodeRed Twitter node are set up however the Twitter integration with NodeRed is faulty and returned authentication errors.

3 Issues, Discrepancies & Improvements

Though the IBM integration proved to be a viable proof of concept, there have been many discrpancies and issues along the way that affected the quality of the results. If this project were to be scaled these issues would have to be addressed.

3.1 "Not So Realtime" NO_2 Readings

Unfortunately, the London Air API did not provide consistent data readings for us to build an accurate database/Classifier. For example, the readings of a location's gas level would sometimesbe taken every 15 minutes by the London Air organization, and sometimes every two days. Since the NodeRed application polled both the Air and Traffic APIs every 15 minutes regardless, the traffic levels were often matched with gas readings from another day and time. For example, traffic level measured at 8 AM on Sunday would see the same NO_2level as if it were taken on Monday 3PM. Therefore, the statistical model provided high error rates and low confidence levels. I attempted to increase the size of the database to 1 month worth of readings but this actually increased this discrepancy to the point that IBM's predictive analysis API could not even provide an estimation anymore and gave an error message. Additionally, the time factor is considered a more important predictor than the NO2 readings. This is because of the fact that time readings were always different and constant unlike NO2 readings. Therefore, SPSS is able to build better models based on Time rather than NO2 (See Appendix H [3.5]). This phenomenon can be seen in particular, late at night when NO2 levels seem to be high but predicted traffic is relatively low (See Appendix I [3.5]).

3.2 Approximate Traffic Readings For the Locations

When implementing the HERE Traffic API, the locations of the traffic readings were approximative to the location of the NO2 Sensor of the London Air API. The API only provided traffic measurements for segments of roads between two crossings - which sometimes were very long. On the other hand, the air API provided measurements for a fixed geolocalized point. Therefore the traffic readings probably did not accurately match the NO_2 reading for these locations.

3.3 Timestamp Discrepancy

Late in the process, I realized that the timestamp I was retrieving came from the IBM server which gave GMT without daylight savings resulting in a difference of 1 hour between the time displayed in the dashboard and the actual UK time This is not dramatic as this discrepancy is kept constant across the database/classifier and the data manipulation and display. However, moving forward this discrepancy should be addressed.

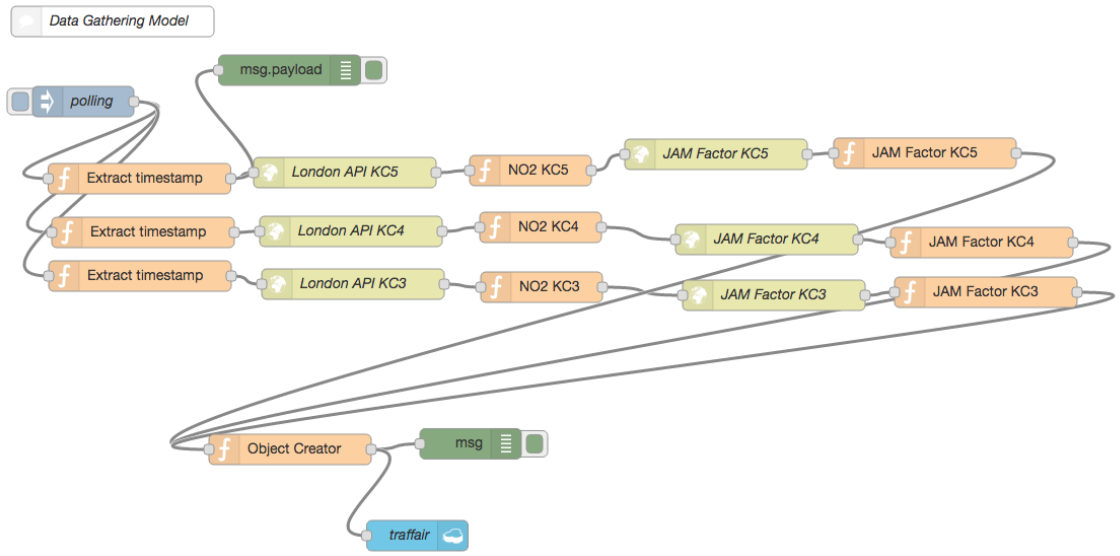
3.4 Choosing NO_2 vs. CO

The London Air API did not offer CO readings on a regular and consistent basis for the locations that we chose in South Kensington, however, we had initially ordered CO sensor. This meant that we had to devise some kind of conversion from our CO sensor readings to data that matched NO_2 readings. We based ourselves on the usual ratio of NO_2 to CO levels in nromal non-polluted air and converted the CO reading to NO_2based on that ratio.

3.5 Omitting Day of the Week

An important factor that was not taken into consideration in this project is that the day of the week will affect the traffic level. For example, 8 AM on Mondays will result in much different traffic levels than on Sundays. Days of the week could be added as an other class in the classifier if this project mvoes forward.

Appendix A - NodeRED Data Gathering Flow



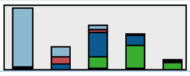
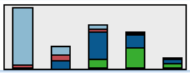
Code available on UCL IOT Dropbox under "nodered-flow.json"

Appendix B - Field Classification

Field	Measurement	Values	Missing	Check	Role
Location	Nominal	3.0,4.0,5.0		None	Input
NO2	Continuous	[22.8,212.2]		None	Input
Time	Continuous	[0.0,23.0]		None	Input
JamFactor	Nominal	1.0,2.0,3.0,...		None	Target

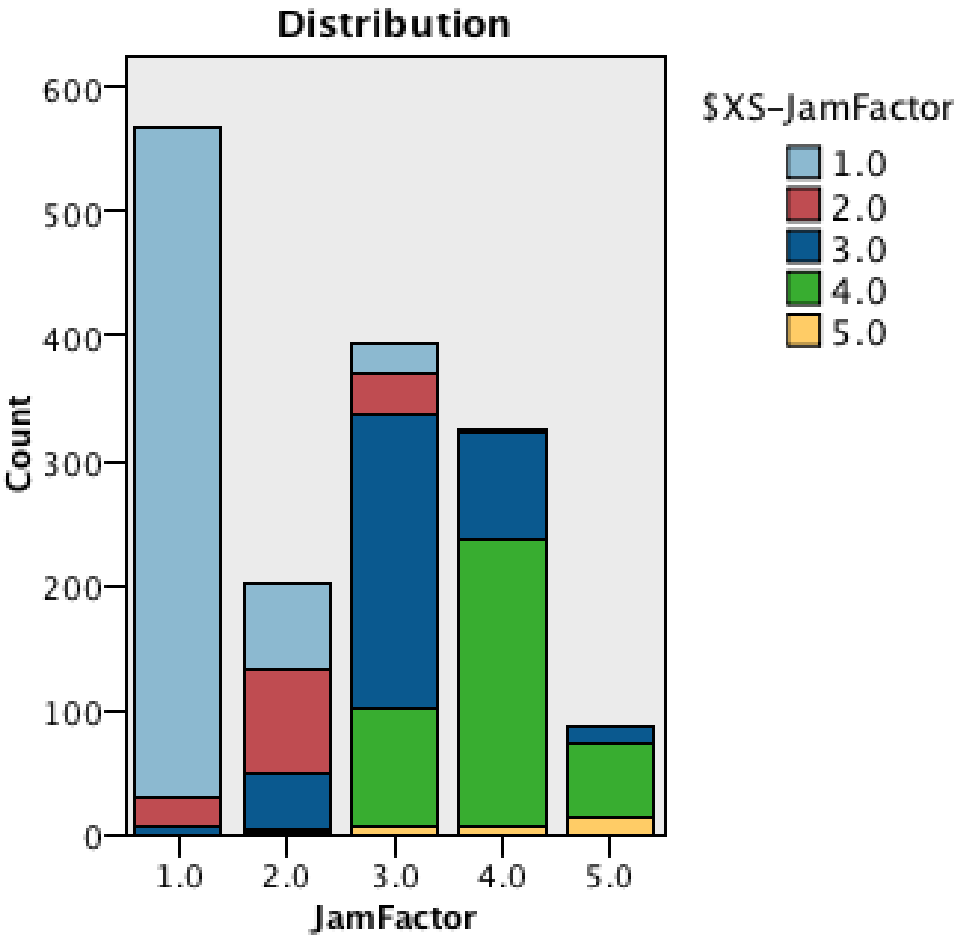
Flow file available on UCL IOT Dropbox under "traffair.str"

Appendix C - Model Generation & Accuracy

Use?	Graph	Model	Build Time (mins)	Overall Accuracy (%)	No. Fields Used
☑		C5 1	< 1	70.540	3
☑		Neural Net 1	< 1	66.921	3
☑		CHAID 1	< 1	66.032	3

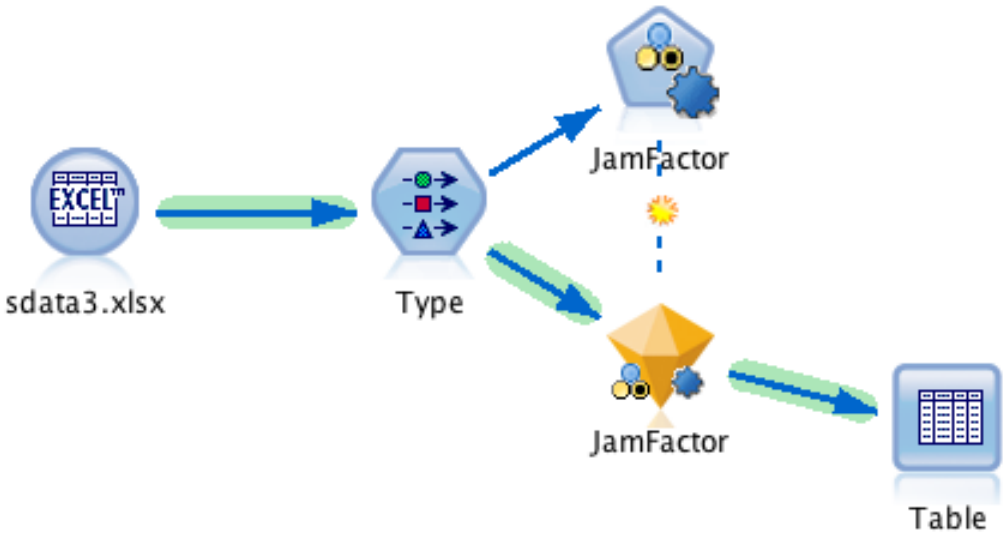
Flow file available on UCL IOT Dropbox under "traffair.str"

Appendix D - Jam Factor Distribution



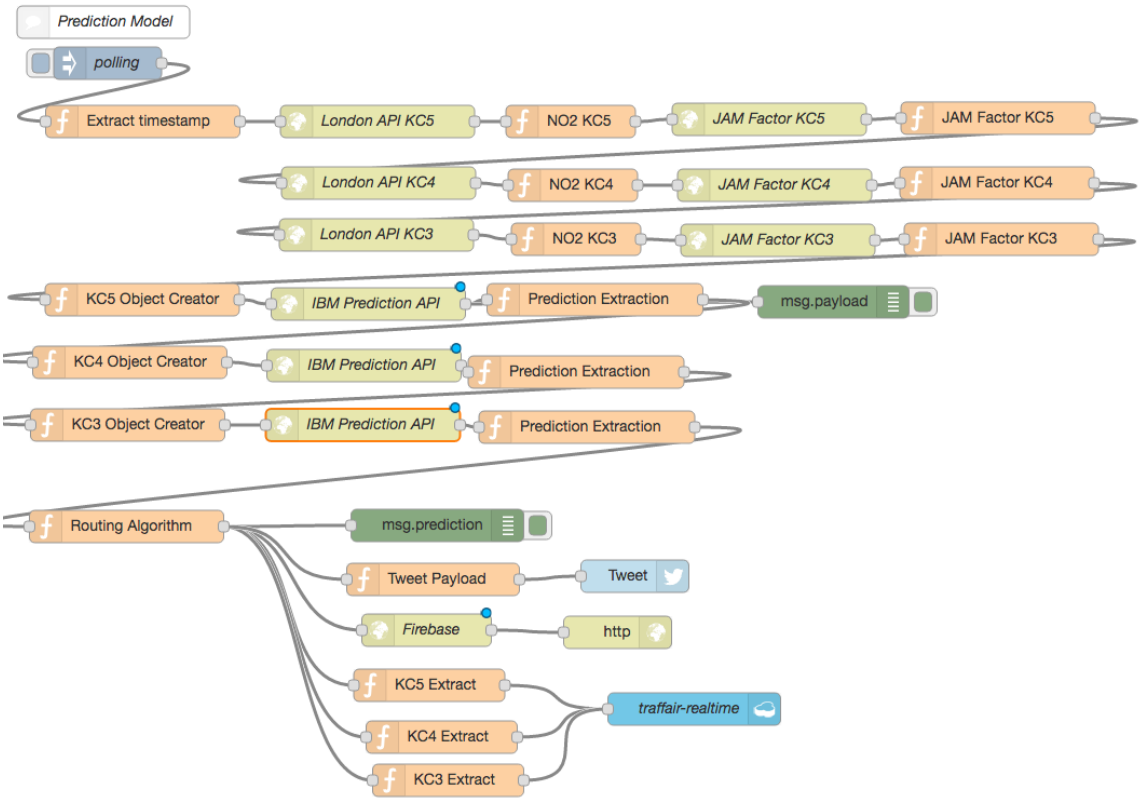
Flow file available on UCL IOT Dropbox under "traffair.str"

Appendix E - SPSS Model Flow



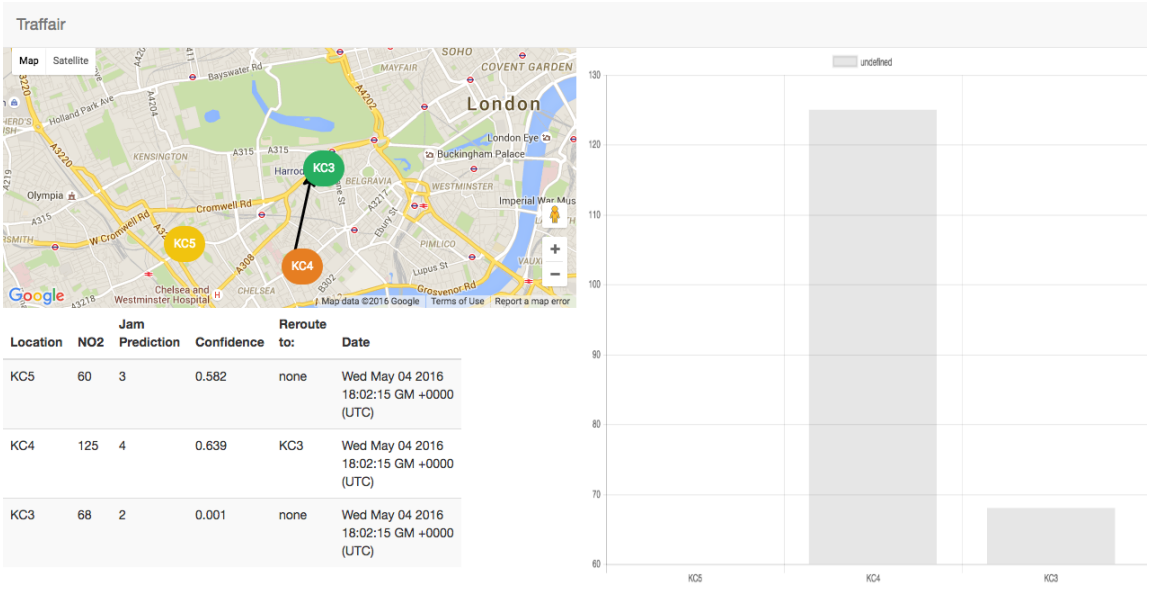
Flow file available on UCL IOT Dropbox under "traffair.str"

Appendix F - NodeRed Prediction Model Flow



Code available on UCL IOT Dropbox under “nodered-flow.json”

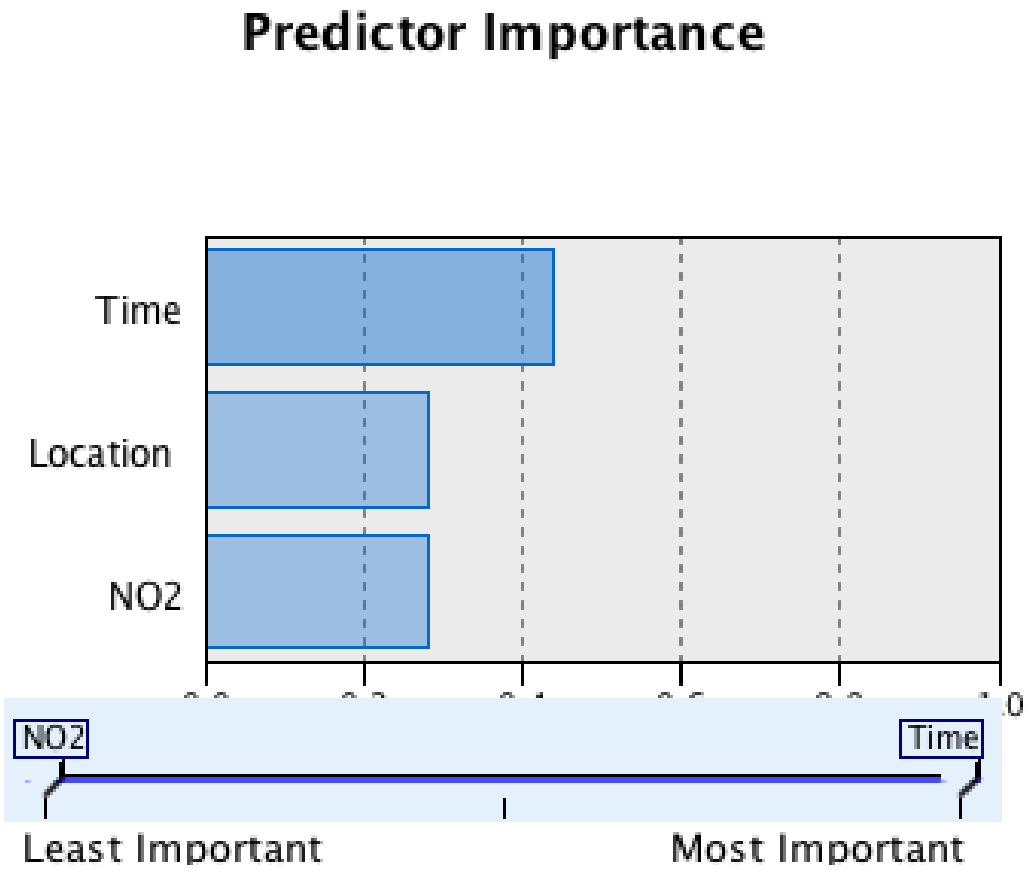
Appendix G - “Traffair” Traffic Management System Dashboard



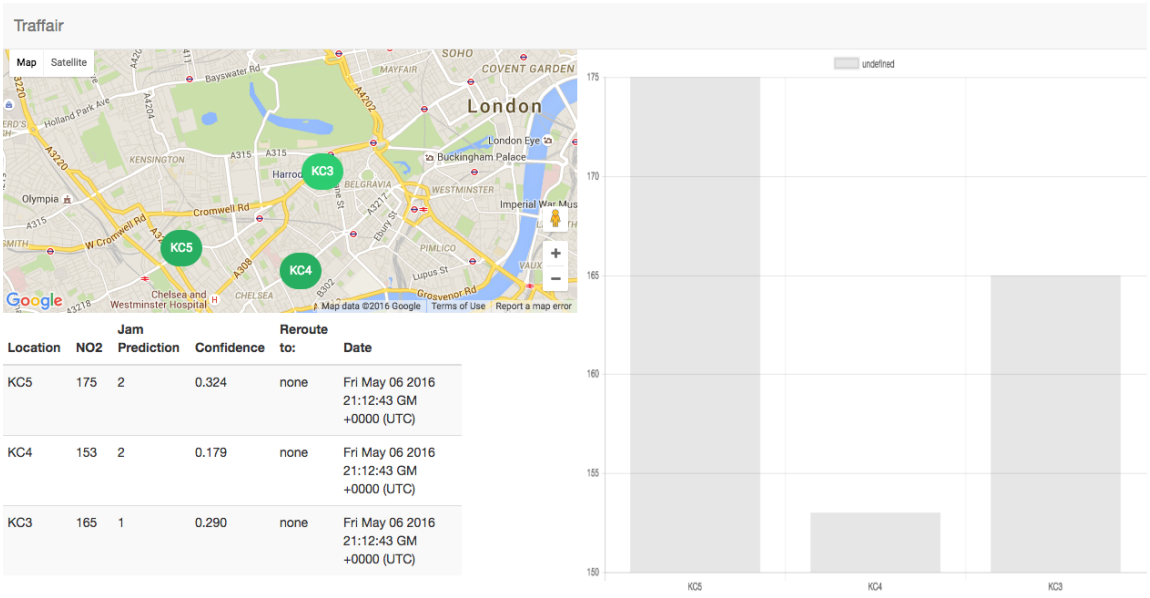
Location	NO2	Jam Prediction	Confidence	Reroute to:	Date
KC5	60	3	0.582	none	Wed May 04 2016 18:02:15 GM +0000 (UTC)
KC4	125	4	0.639	KC3	Wed May 04 2016 18:02:15 GM +0000 (UTC)
KC3	68	2	0.001	none	Wed May 04 2016 18:02:15 GM +0000 (UTC)

Code available on UCL IOT Dropbox under “traffair.zip” & ucliot.github.io

Appendix H - Predictor Importance



Appendix I - Late Night Dashboard View



Location	NO2	Jam Prediction	Confidence	Reroute to:	Date
KC5	175	2	0.324	none	Fri May 06 2016 21:12:43 GM +0000 (UTC)
KC4	153	2	0.179	none	Fri May 06 2016 21:12:43 GM +0000 (UTC)
KC3	165	1	0.290	none	Fri May 06 2016 21:12:43 GM +0000 (UTC)

References

[1] London Air. "London Air Quality Network || API." London Air Quality Network || API. King's College London, n.d. Web. 04 May 2016. <http://www.londonair.org.uk/LondonAir/API/>.

[2] London Air API. "List of Endpoints." London Air API Help. N.p., n.d. Web. 04 May 2016. <http://api.erg.kcl.ac.uk/AirQuality/help>.

[3] HERE. "Flow." Traffic API Developer's Guide. N.p., n.d. Web. 04 May 2016. <https://developer.here.com/rest-apis/documentation/traffic/topics_v6.1/resource-parameters-flow.html>.

[4] Nacar, Sacha. "UCL IOT Group 2 - Cloudant Database." N.p., n.d. Web. 4 May 2016. <https%3A%2F%2Fec5f2a2a-d88e-4358-8690-8349621cb9bd-bluemix.cloudant.com%2Ftraffic2%2F_all_docs>.

[5] IBM. "API Documentatio - Predictive Analytics Service API." IBM Bluemix Docs. N.p., n.d. Web. 04 May 2016. <https://console.ng.bluemix.net/docs/services/PredictiveModeling/index-gentopic1.html#pm_service_api>.