PREDICTING TRANSIT RIDERSHIP FOR A CITY

Capstone Project 1: Final Report Siri Surabathula

Problem Statement

As urban population levels rise, they lead to increasing traffic congestion, which in turn worsens mobility and carbon-emissions problems for cities. Public transit systems can be key in managing the urban mobility and emissions crises. However, the success of transit systems depends on their ability to scale and their responsiveness to regional variations in demand. Both short-term and long-term prediction models are hence vital to effectively solving problems of urban mobility.
This project aims at predicting transit ridership at different stations across a large city over short-term time frames using weather, traffic, taxi/cab and gas price data. The spatial aspect of transit, taxi/cab and traffic data will be taken into account while building the model for prediction.

Prospective clients for the model could be various city transportation authorities (like NYC DOT, NYC MTA) and city planning authorities (like City of New York)
City planning authorities could use the ridership predictions to efficiently allocate resources and modulate the frequency of trains and buses along specific routes.

Description of Data-set and Data Wrangling

Data Pipelines with Dask, Minio and Docker:

Considerable amount of time and effort was spent to load and wrangle the data. Most of the datasets (like the NYC cabs data) were massive in size and called for parallel processing techniques. Data Pipelines were setup using Dask, Minio, Redis and Docker. The code-base for this setup is on github at sirisurab/transpred

Dask:

The python API Dask was used for most of the data-loading and wrangling work for the transit, cabs and traffic data.

Minio:
The parallel processing environment called for more robust (multiprocessor/thread-safe) back-end and object store for storing and retrieving data. The Minio python API was used for this.

Docker:

The build and deploy of services like Minio and Redis (Redis used for basic task-queuing) as well as multiple parallel instances of Dask processes and jupyter notebooks was managed with Docker, Compose and Swarm

Transit Data

Stations:

Source : NYC MTA (Subway Stations Data)

Description : Description of all the subway stations in NYC. Useful fields are 'STOP_NAME' (station name) and 'GTFS Latitude' and 'GTFS Longitude' (geographic coordinates of the station)

Processing : This dataset has been processed by the python file refbase_loader.py and saved to minio bucket ref-base as a GIS shapefile stations.zip

Following data issues have been addressed :

1. There is no unique identifier that represents stations across the NYC MTA database : The 'STATION' column of the MTA turnstile dataset is the only identifier for the station in that set. The contents of this column differed significantly from the 'STOP_NAME' column of the stations dataset. For example, the station named 'TIMES SQ-42 ST' in one set was represented as 'Times Sq - 42 St' in the other. Although, issues like this were easy to fix, there were a fair number of cases where a station named 'Astoria - Ditmars Blvd' did not have any obvious match in the other data set. A possible cause for cases like this is the use alternate names for the same station ('Astoria - Ditmars Blvd' station was earlier known as 'Second Avenue'). Cases like this are hard, if not impossible to match. A python string-matching library called 'fuzzy-wuzzy' was used to find the best matches using 3 Levenshtein closeness methods (normal ratio, partial ratio and token sort ratio). The match was accepted only if one of the three matching methods returned a ratio of 88% or higher. The matching station names from the turnstile dataset were added to the 'STATION' column of the stations dataset.
2. The columns 'GTFS Latitude' and 'GTFS Longitude' required further processing in order to be readily consumable for joins (geographic) across different datasets (for example with the traffic and taxi/cab datasets) : The python geopandas library was leveraged for this purpose (This library in turn depends on shapely, fiona and rtree). 'GTFS Latitude' and 'GTFS Longitude' were merged into a single 'Point' geometry

(shapely.geometry.Point) and the entire dataset converted to a geopandas GeoDataFrame. This allows for fairly easy (though sometimes computationally expensive) joins across datasets using the geometry attributes like, points, lines and polygons. A circle of customizable radius, centered at each station, was also drawn and added to a new geometry column containing the circles as polygons. These circles represent 'circles of influence' or zones for each station and will be used to find intersection with traffic and taxi/cab data.

Turnstile Data:

Source : NYC MTA (Subway Stations Turnstile Data : 4-hour frequency, 2016 and 2017)

Description : Transit ridership (turnstile entry and exit counts) of all the subway stations in NYC. Useful fields are 'STATION' (station name), 'DATE' (date) , 'TIME' (time) , 'ENTRIES' (entry count) , 'EXITS' (exit count)

Processing : This dataset has been processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket cl-transit in parquet format

Following data issues have been addressed :

1. 'DATE' and 'TIME' occur as separate string columns : These two were merged and converted to type 'datetime64[ns]'. This column was also used as the index (after the rest of cleaning was complete)

2. 'EXITS' (and 'ENTRIES') columns have cumulative reading of the turnstile unit : the pandas.Series.diff method was used to calculate the change from the previous reading.

3. Turnstile units would reset randomly once in a while, resulting in outliers in the 'EXITS' (and 'ENTRIES') columns (abnormally high values or negative values) : These outliers were identified and filtered out by calculating the interquartile range and rejecting all rows with 'EXITS' (or 'ENTRIES') with values greater than 3 times the interquartile range or with negative values.

Taxi/Cab Data

Source : NYC TLC (Taxi and Cab Trip Data : every taxi/cab trip in NYC for 2016 and 2017)

Description : Every taxi/cab trip in NYC. Useful fields are 'dropoff_datetime', 'dropoff_latitude', 'dropoff_longitude' , 'pickup_datetime', 'pickup_latitude', 'pickup_longitude', 'passenger_count'

Processing : This dataset has been processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket cl-gcabs and cl-ycabs in parquet format

Following data issues have been addressed :

1. The dataset for 2016 and 2017 is too large and called for parallel processing techniques: The python API Dask was leveraged for this. This partitions large datasets into multiple pandas DataFrames and allows for parallel processing on them.

2. The columns 'dropoff_latitude', 'dropoff_longitude' (and 'pickup_latitude', 'pickup_longitude') required further processing in order to be readily consumable for joins (geographic) with the Stations dataset : The python geopandas library was leveraged was this purpose (This library in turn depends on shapely, fiona and rtree). 'dropoff_latitude', 'dropoff_longitude' were merged into a single 'Point' geometry (shapely.geometry.Point) and the entire dataset converted to a geopandas GeoDataFrame. This allows for fairly easy (though computationally expensive in this case due to the size of the dataset) joins across datasets using the geometry attributes like, points, lines and polygons. A circle of customizable radius, centered at each station, representing the 'circles of influence' or zones for each station will be used to find intersection with taxi/cab data. (each trip will be associated with a station for the pickup point, as well as a station for the drop-off point, by finding which station-zone the points fall in)

Traffic Data

Traffic Links :

Source : NYC Open data (Traffic Links Data)

Description : Traffic Links (sets of geographic coordinates) each representing a stretch or road/street over which the average speed of traffic is recorded. Useful fields are 'LINK_ID','LINK_POINTS','BOROUGH'

Processing : This dataset has been processed by the python file refbase_loader.py and saved to minio bucket ref-base as a GIS shapefile traffic_links.zip

Following data issues have been addressed :

1. The column 'LINK_POINTS' required further processing in order to be readily consumable for joins (geographic) with the Stations dataset : The python geopandas library was leveraged for this purpose (This library in turn depends on shapely, fiona and rtree). The contents of 'LINK_POINTS' were merged into a single 'LineString' geometry (shapely.geometry.LineString) and the entire dataset converted to a geopandas GeoDataFrame. This allows for fairly easy joins across datasets using the geometry attributes like points, lines and polygons.
2. Association of each 'LINK_ID' with a Station : A circle of customizable radius, centered at each station, representing the 'circles of influence' or zones for each station (represented as a Polygon geometry in the Stations dataset) was used to find intersection with traffic link data. (each link was associated with a station by finding which station-zone the link-line intersects with)

Traffic Speed Data :

Source : NYC Open data (Traffic speed data Data recorded at various locations in NYC for 2016 and 2017)
Description : Traffic speed data Data recorded at various locations in NYC. Useful fields are 'LINK_ID','DATETIME','SPEED'

Processing : This dataset has been processed by the python file data_clean/tasks.py (using dask) and saved to minio bucket dl-traffic in parquet format

No issues found with this series.

Weather Data

Source : National Climatic Data Center (daily temperature, rainfall and snowfall data for NYC for 2016 and 2017)

Description : Daily temperature, rainfall and snowfall data for NYC. Useful fields are 'DATE','PRCP','SNOW','TMAX','TMIN'

Processing : This dataset has been processed by the python file refbase_loader.py and saved to minio bucket ref-base as weather.csv

Following data issues have been addressed :

1. 'DATE' in string format : This was converted to type 'datetime64[ns]'. This column was also used as the index (after the rest of cleaning was complete)

2. Average daily temperature : This was calculated by finding the mean of 'TMIN' and 'TMAX' and added as 'TAVG'

Gas Price Data

Source : National Climatic Data Center (daily temperature, rainfall and snowfall data for NYC for 2016 and 2017)

Description : Daily temperature, rainfall and snowfall data for NYC. Useful fields are 'DATE','PRCP','SNOW','TMAX','TMIN'

Processing : This dataset has been processed by the python file refbase_loader.py and saved to minio bucket ref-base as gas.csv

Following data issues have been addressed :

1. 'DATE' in string format : This was converted to type 'datetime64[ns]'. This column was also used as the index (after the rest of cleaning was complete)

2. Average daily temperature : This was calculated by finding the mean of 'TMIN' and 'TMAX' and added as 'TAVG'

Exploratory Analysis

Time Series Analysis

The seasonality and trend for each time series was studied using time plots, Dickey-Fuller stationarity test, Autocorrelation and Partial Autocorrelation plots as well as frequency plots (histograms)
The test results and plots are at EDA_Modeling.ipynb (section EDA and Stationarity Tests) and are further described below.

Transit Data

Station Exit Count:

Seasonality:

The Autocorrelation and Partial Autocorrelation plots indicate that the data has seasonality with a period of 7 days (7 order moving average (MA) term)

Trend:

The Local Trend model indicates that the data does not have a trend

Station Entry Count:

Seasonality:

The Autocorrelation and Partial Autocorrelation plots indicate that the data has seasonality with a period of 7 days (7 order moving average (MA) term)

Trend:

The Local Trend model indicates that the data does not have a trend

Taxi/Cab Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots indicate that the data has seasonality with a period of 7 days (7 order moving average (MA) term)

Trend:

The Local Trend model indicates that the data has a varying trend, where it decreases from the January to September of 2016; after which it rises, then drops abruptly in January 2017 and again rises until March 2017. It exhibits a downward trend again until September 2016, after which it rises once again. It appears that the trend itself has a seasonality of 1 year.

Traffic Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots indicate that the data has a lag of both 1 and 7 days

Trend:

The Local Trend model indicates that the data has a general downward trend from January 2016 to October 2017, after which it begins to exhibit an upward trend

Gas Price Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots indicate that the data has significant autocorrelation at lags 1 (positive correlation) and 260 (negative correlation)

Trend:

The Local Trend model for the differenced price (first order difference with lag=1) shows a local level with significant variance and an initially decreasing trend (slope) which stabilizes after July 2016.

Weather Data

Seasonality:

The Dicker-Fuller stationarity test results as well as the Autocorrelation and Partial Autocorrelation plots indicate that the data has significant autocorrelation at lag 1 (positive correlation) and has an obvious seasonality of 365 days (annual)

Trend:

The Local Trend model shows a local level with significant variance and annual seasonality as well as a very gentle overall decreasing trend (slope).

Studying Variable Relationships

Relationships between the various exogenous variables (taxi passengers, traffic speed, gas price and mean daily temperature) with the endogenous variables (transit station exit and entry counts) were studied. This study ignored the time component for all the series.
The plots are at EDA_nonts.ipynb (section EDA for each of the 5 selected stations)
and are further described below.

Transit counts vs Cab-rides:

For most stations, there is a negative correlation between transit station (exit and entry) counts and the number of cab rides in the vicinity of the station. For the Wall Street and Court Square stations there is a positive correlation between transit entry counts and cab rides in the vicinity of the station.

Transit counts vs Traffic speed:

For most stations, there is a negative correlation between transit station (exit and entry) counts and the traffic speed in the vicinity of the station. For the Wall Street and Court Square stations there is a slight positive correlation between transit entry counts and traffic speed in the vicinity of the station.

Transit counts vs Gas price:

For most stations, there is a positive correlation between transit station (exit and entry) counts and gas price. For the Wall Street and Court Square stations there is a negative correlation between transit entry counts and gas price.

Transit counts vs Weather (daily mean temperature):

For some stations, there is a positive correlation between transit station (exit and entry) counts and mean daily temperature, while for some others there does not seem to be much correlation between weather and transit usage.

Inferences

1. The time series analysis indicates that all the series have autocorrelation and/or moving average components which need to be removed from the data before applying a statistical model for prediction.
2. Most series (transit, cabs, traffic) have Autoregressive and Moving Average components with lag = 1 and lag = 7
3. The gas price and weather data have AR component with lag=1 and a higher order MA component
4. The non-time series analysis indicates that there is significant cross-correlation (time independent) between the target variables (transit entry and exit counts) and the predictor variables (cabs, traffic, gas and weather)
5. After **making all the variables stationary**, and verifying that each of the **transformed series has an approximate normal distribution**, a **Multivariate ARMA Model like VARMA (Vector Autoregressive Moving Average)** can be applied for making predictions.

Results and In-depth analysis

Transforming the Time Series data and Normality tests

The exogenous variables (cab, traffic, gas and weather) were transformed to remove trend and seasonality. The endogenous variables were left untransformed as the models (VARMAX and VECM) took care of this. The processing steps are at EDA_Modeling.ipynb (section Transforming the Time Series)

Transit Data

The data was left untransformed as the models used later (VARMAX and VECM) took care of the transformation of endogenous variables.

Taxi/Cab Data

Difference smoothing was carried out using lags 1 and 7. The resulting double differenced data has an approximate normal distribution.

Traffic Data

Difference smoothing was carried out using lag 1. The resulting single differenced data has an approximate normal distribution.

Gas Price Data

Second-order polynomial smoothing followed by difference smoothing with lag 1, was carried out. The resulting data has an approximate normal distribution.

Weather Data

Second-order polynomial smoothing followed by difference smoothing with lag 1, was carried out. The resulting data has an approximate normal distribution.

Modeling

The processing steps are at EDA_Modeling.ipynb (section Modeling)
The modeling was carried out for the station 'Wall Street'

Train-Test Split

The data was split into training and hold-out sets using data upto 30th June 2017 as the training data and from 1st July 2017 onwards as the hold-out set.

VARMAX Model

The statsmodels model VARMAX was used.

Model Selection

5-fold cross-validation was used to find the optimum order of the VARMAX model (p, q)
The python class TimeSeriesSplit from scikit-learn was used while splitting the data for cross-validation.
The optimum order for the model using (aic score) was determined as p=3, q=3

Model Residuals

The model residuals were plotted to check for normality. The plots indicate an approximate normal distribution
The residuals were further modeled using an ARMA model (to remove residual heteroskedasticity)

Forecasting

The forecasted values for transit entry and exit counts for the station for the period 1st July 2017 - 31st December 2017 were computed using the tuned VARMAX model (with ARMA applied to residuals). Plots were created to compare actual and predicted values

Model Accuracy Score

The accuracy score (RMSE) for the model was computed as 7021.5 for transit exits and 8901.05 for transit entries.

VECM Model

The statsmodels model VECM was used.

Model Selection

5-fold cross-validation was used to find the optimum parameters of the VECM model (k_ar - number of lagged differences and coint_rank - cointegration rank)

The parameter for the number of periods in a cyclical season was set to 7, due to the mostly weekly seasonality in the data

The python class TimeSeriesSplit from scikit-learn was used while splitting the data for cross-validation.
The optimum parameters for the model using (log-likelihood score) was determined as k_ar=3, coin_rank=1

Model Residuals

The model residuals were plotted to check for normality. The plots indicate an approximate normal distribution
The residuals were further modeled using an ARMA model (to remove any residual heteroskedasticity)

Forecasting

The forecasted values for transit entry and exit counts for the station for the period 1st July 2017 - 31st December 2017 were computed using the tuned VECM model (with ARMA applied to residuals). Plots were created to compare actual and predicted values

Model Accuracy Score

The accuracy score (RMSE) for the model was computed as 3459.3 for transit exits and 4314.59 for transit entries.

Conclusion

It is obvious from the prediction plots and the accuracy scores that the VECM model performs significantly better than the VARMAX model for both transit entries and exits.

Further Exploration

Further exploration of time series regression models was performed using Dynamic Linear Regression, GAS Regression and Beta-t-EGARCH Regression models. The results and plots are at EDA_Modeling.ipynb (section Further experimentation with other Time Series Forecasting Models)