New York City Taxi Trip Regression

github-TiFalcom-ny-taxi

Motivation

Predicting taxi demand in urban areas plays a crucial role in optimizing transportation services and enhancing passenger experiences. In a city characterized by intense economic activity and high population density, accurate demand forecasting enables taxi operators and ride-hailing platforms to adjust logistics, reduce waiting times, and maximize vehicle utilization. With advancements in machine learning methods and time series analysis, it is possible to leverage large volumes of historical and contextual data, such as weather conditions and local events, to achieve more precise and reliable forecasts.

Objective

This project aims to develop predictive models to estimate the **hourly number of taxi trips** in New York City. Using the publicly available dataset from nyc.gov, various machine learning methods and time series models will be explored to identify patterns and generate demand forecasts. Finally, we will analyze the results to determine which solution performs best for this problem and explore how other approaches can be optimized to improve their performance.

Model Development

Bellow you will find all the steps to replicate the experiments, the decisions and results we got.

Replication

To replicate the environment, ensure you are using Python 3.12 and run the code above.

```
. install.sh
```

0.Exogenous Data Capture

Getting exogenous data that can help prediction.

Two datasets were used to get exogenous data:

- Weather Data from NYC Central Park
- NYC Taxy Zone

To treat the data access the notebook.
00-Exogenous-Data-Capture.ipynb

1.Data Basic Process

Cleaning data, merging travels with weather information and data point knowledge base, fixing features to the right type. You can find all the process on this script:

```
python src/data/basic_process.py --config_file=features --dataset_name=full
```

If you computed train and test splited tables, you can union with the script bellow:

```
python src/data/concat_data.py --datasets_list=data/interim/train --
datasets_list=data/interim/test --dataset_name_output=data/interim/full
```

2.Split Data

Split train, test and validation datasets to avoid leak on feature engineering. Using [20240101, 20240131) for training, [20240201, 20240207] for validation, and [20240208, 20240229] for test.

```
python src/data/split_train_test.py --dataset_name=full --
ymd_train=20240101 --ymd_valid=20240201 --ymd_test=20240208
```

3.Feature Engineering

Features were created using pickupdate like 'hour', 'week', period of day and others.
To execute this part, run the code bellow:

```
python src/features/create_features.py --dataset_prefix=full
```

4.Encoding

Categorical encoding applied to features with less than 15 categories. Used one hot encoding, because we want to aggregate this data and get trends about each categorie.

To create the encoders run this script:

```
python src/features/create_encoders.py --dataset_prefix=full
```

And to encode dataset and save a checkpoint, run this:

```
python src/features/create_encoded_features.py --dataset_prefix=full --
encoder_type=2
```

5.Aggregate Features

Condense analytical features to hour features and fix target with lag 1

Features we decided to condense for this experiment was:

- Minimum and Maximum Temperature - Max
- Qty. Passengers - Count
- Day of Week - Count
- Period of Day - Count

```
python src/features/create_groupby_features.py --dataset_prefix=full
```

6.Scale Data

Scaled features for linear models.

To fit the encoders run:

```
python src/features/create_scalers.py --dataset_prefix=full
```

And to scale dataset and save a checkpoint, run this:

```
python src/features/create_scaled_features.py --dataset_prefix=full
```

7.Feature Enginnering - Lag

Create Lag Features for tabular models, created lag 3 for:

- Maximum and Minimum Temperature
- Qty Passengers
- Day of week
- Period of Day

The features will have a sufix '_l1', '_l2', '_l3' corresponding to lag

```
python src/features/create_lag_features.py --dataset_prefix=full
```

8.Feature Selection

A manually feature selection were done, to acelerate the process, on future works we can implement a robust method.

Features selected for Decision Tree and Boosting:
['PULocationID', 'Maximum_max', 'Minimum_max','passenger_count_sum', 'day_of_week_max',
'period_of_day_dawn_sum','period_of_day_morning_sum',
'period_of_day_afternoon_sum','period_of_day_evening_sum', 'Maximum_max_l3','Minimum_max_l3',
'passenger_count_sum_l3', 'day_of_week_max_l3','period_of_day_dawn_sum_l3',
'period_of_day_morning_sum_l3','period_of_day_afternoon_sum_l3',
'period_of_day_evening_sum_l3','qty_travels_l3', 'Maximum_max_l2',
'Minimum_max_l2','passenger_count_sum_l2', 'day_of_week_max_l2','period_of_day_dawn_sum_l2',
'period_of_day_morning_sum_l2','period_of_day_afternoon_sum_l2',
'period_of_day_evening_sum_l2','qty_travels_l2', 'Maximum_max_l1',
'Minimum_max_l1','passenger_count_sum_l1', 'day_of_week_max_l1','period_of_day_dawn_sum_l1',
'period_of_day_morning_sum_l1','period_of_day_afternoon_sum_l1',
'period_of_day_evening_sum_l1','qty_travels_l1']

Trends selected for ARIMA and SARIMAX:
['qty_travels']

9.Tunning

There was no tunning implemented yet, on future works we can implement some method.

Params for Decision Tree:
{random_state : 777, max_depth : 40, min_samples_leaf : 15}

Params for Boosting:
{boosting_type : gbdt, max_depth : 5, n_estimators : 500, learning_rate : 0.01, random_state : 777,
min_child_samples : 3}

Params for ARIMA:
{order : [3, 1, 3], freq : h}
With last 24 hours to fit the metrics

Params for SARIMAX:
{order : [3, 1, 3], freq : h, seasonal_order : [3, 1, 3, 5]}

10.Train Model

On this experiment we decided to work with only two locations (163, 79), because ARIMA/SARIMAX are expensive to fit, after building the entire pipeline we will work with more locations.

For training the Boosting model run the code bellow:

```
python src/model/train_boosting.py --config_file=features --
folder_dataset_prefix=lag/full --hyperparams_file=hyperparams --
model_suffix=reg_with_lag
```

For training the Tree model run the code bellow:

```
python src/model/train_decision_tree.py --config_file=features --
folder_dataset_prefix=lag/full --hyperparams_file=hyperparams --
model_suffix=reg_with_lag
```

For predicting the ARIMA model run the code bellow:

```
python src/model/predict_arima.py --folder_dataset_prefix=aggregated/full -
-model_suffix=2locations
```

For predicting the SARIMAX model, run the code bellow - Still not working

```
python src/model/predict_sarimax.py --folder_dataset_prefix=aggregated/full
--model_suffix=2locations
```

WIP:

- Logistic Regression
- Linear Regression
- Basic MLP
- LSTM

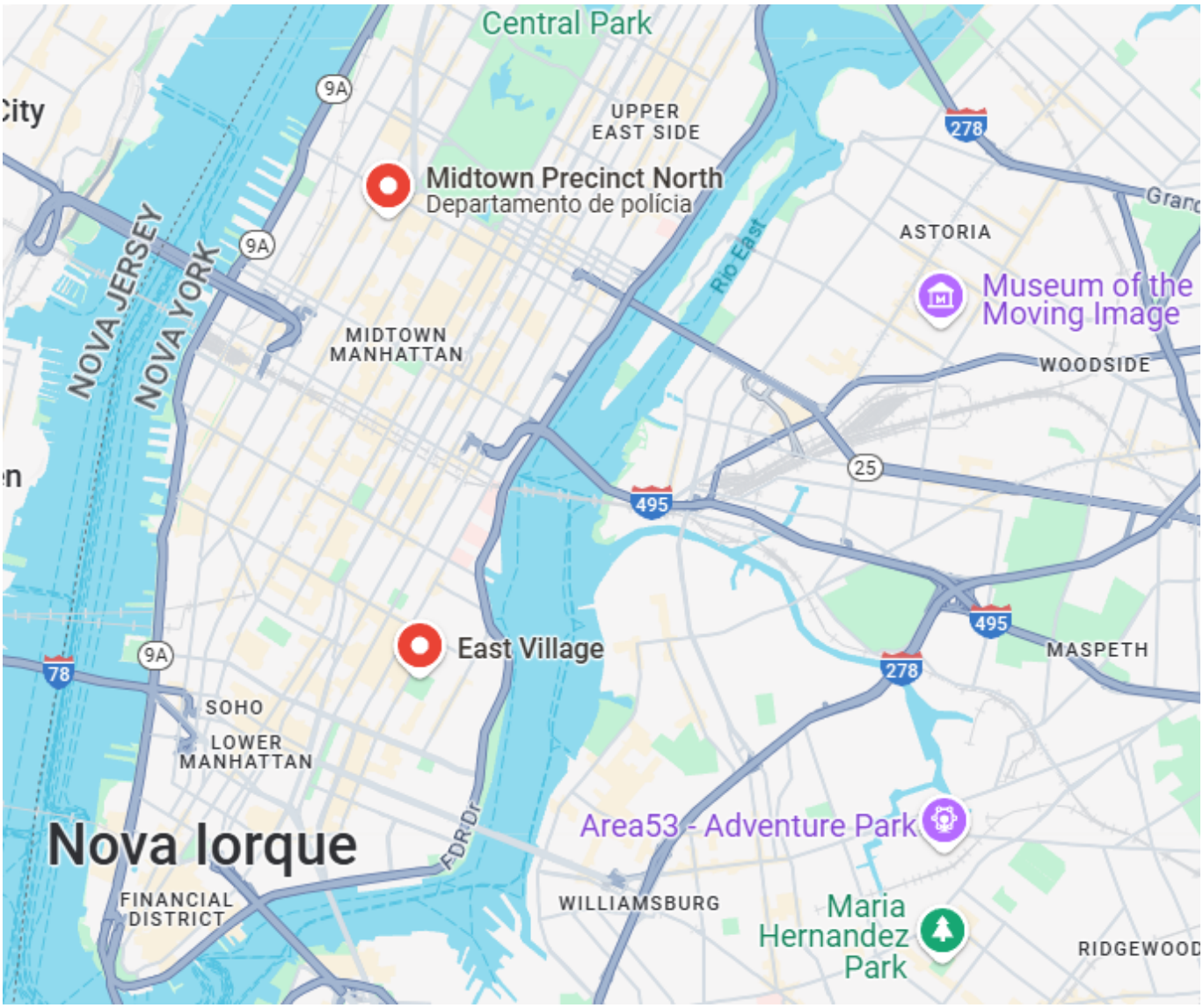
11.Register Experiment (mlflow?)

Use a framework to register experiments (maybe a folder structure)

12.Results

For this first attempt we decided to use only two regions:

- 163 - Manhattan, Midtown North, Yellow Zone
- 79 - Manhattan, East Village, Yellow Zone



We decided to use only two regions due to the fitting process of ARIMA. For each prediction, we need to refit the parameters for the next hour and compare the results with other models.
For the same reason, to reduce computational time, we are using the validation dataset [20240201, 20240701] to evaluate the models. Since we did not use the validation dataset for tuning or feature selection, there is no data leakage.

After training the DecisionTree and Boosting models using only two locations (with location as a feature) and fitting ARIMA separately for each location, we achieved the following results.

Model	Description	RMSE Train	RMSE Valid	Diff RMSE	MAE Train	MAE Valid	Diff MAE
Default	Mean ± std	45.56 ± 58.67	49.60 ± 65.00	0.00	45.56 ± 58.67	49.60 ± 65.00	0.00
DecisionTreeRegressor	with lag features	14.19	18.91	4.72	7.94	10.24	2.30
LightGBMRegressor	with lag features	6.91	16.77	9.85	4.69	9.22	4.53

Model	Description	RMSE Train	RMSE Valid	Diff RMSE	MAE Train	MAE Valid	Diff MAE
TemporalModels (ARIMA)	arima top2	35.28	35.28	0.00	19.18	19.18	0.00

All three models performed better than a mean predictor, although DecisionTree and Boosting showed some overfitting. To reduce this overfitting, we can try to:

Increase the training dataset by using a longer time period.

- Implement feature selection and tuning methods to reduce variance.
- These tests will be carried out in future work.

Analyzing ARIMA against tree-based models, ARIMA performed worse across all metrics, although it still outperformed the mean predictor. To reduce ARIMA's error, we could:

- Exclude weekends from trends and predictions to remove weekend seasonality.
- Use one model with a prediction horizon of more than 24 hours and another with less than 24 hours to evaluate how much history is needed to predict the next hour.
- Incorporate additional trends with SARIMAX, such as neighboring regions' trends and weather data, to enhance the model's information.
- Tune the model's parameters using autocorrelation methods to decrease prediction errors.

At this stage, we have achieved promising results with tree-based and time series methods, but there are still many ways to improve these results. That is what we will focus on in PGC II.

Project Organization

CCDS Project template

```
├── LICENSE            <- Open-source license if one is chosen
├── Makefile           <- Makefile with convenience commands like `make
data` or `make train`
├── README.md          <- The top-level README for developers using this
project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default mkdocs project; see www.mkdocs.org for
details
│
├── models             <- Trained and serialized models, model predictions,
or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number
(for ordering),
│                         the creator's initials, and a short `-` delimited
```

```
description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── pyproject.toml     <- Project configuration file with package metadata
for
│                         src and configuration for tools like black
│
├── references         <- Data dictionaries, manuals, and all other
explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in
reporting
│
├── requirements.txt   <- The requirements file for reproducing the
analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.cfg          <- Configuration file for flake8
│
└── src   <- Source code for use in this project.
    │
    ├── __init__.py             <- Makes src a Python module
    │
    ├── config.py               <- Store useful variables and configuration
    │
    ├── dataset.py              <- Scripts to download or generate data
    │
    ├── features.py             <- Code to create features for modeling
    │
    ├── modeling
    │   ├── __init__.py
    │   ├── predict.py          <- Code to run model inference with trained
models
    │   └── train.py            <- Code to train models
    │
    └── plots.py                <- Code to create visualizations
```