Di-Tech Challenge: Forecasting Supply and Demand Gap

Definition

Project Overview

Didi Chuxing, the dominant ride-hailing company covering more than 400 cities across China, processes over 11 million trips, plans over 9 billion routes and collects over 50TB of data per day. For this challenge, Di-Tech provides 600MB of its real world data on order information, traffic jam level, weather measurements and points of interest. The goal is to predict the supply and demand gap for a specific area over a certain period of time. With this knowledge on the gap between requested and unanswered order, the company can improve their driver-rider match and enhance service reliability[1].

Problem Statement

With the mobile app Didi, a passenger requests a ride by setting pickup location and destination address. When a ride is requested, a driver takes the order and provides the taxi service. When an order is not answered by any driver, it is left as unanswered. For this challenge, Didi segments its service area, City M, into 66 non-overlapping square districts $D = \{d_1, d_2, \cdots, d_{66}\}$ and divides one day into 144 uniform 10-minute slots $t_1, t_2, \cdots, t_{144}$.

For district d_i, in time slot t_j,

- the number of passengers' requests, i.e. demand, is r_{ij};
- the number of drivers' answers, i.e. supply, is a_{ij};
- the demand-supply gap, gap_{ij}, is defined as $gap_{ij} = r_{ij} - a_{ij}$.

For this challenge, given the spatial and temporal data for district d_i and time slot t_j, a participant is asked to predict gap_{ij}, $\forall d_i \in D$. The prediction results should be submitted in the following format:

Table 1. Prediction Results Example

District ID	Time Slot	Prediction Value
1	2016-01-23-46	1.0
2	2016-01-23-46	3.0
…	…	…

This is a well-defined supervised learning problem, more specifically a regression problem, with order information, traffic jam level, weather measurements and points of interest as features and gap as target values.

[1] http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016

Metrics

For district d_i and time slot t_j, the supply-demand gap is gap_{ij}; the prediction is s_{ij}. The evaluation metrics used by the leaderboard is the **m**ean **a**bsolute **p**ercentage **e**rror[2] (MAPE) which is defined as below:

$$MAPE = \frac{1}{n}\sum_{d_i}\left(\frac{1}{q}\sum_{t_j}\left|\frac{gap_{ij} - s_{ij}}{gap_{ij}}\right|\right), \qquad \forall\, gap_{ij} > 0$$

The lower the MAPE, the better the predictions.

Analysis

Data Exploration

Data files `training_set.tar.gz` and `test_set.tar.gz` can be downloaded here[3]. The data come in multiple datasets in different directories and files. Detailed data processing will be discussed in more details in section **Methodology – Data Preprocessing**. The final training set has 163491 samples, 38 features and 1 target column. The final test set has 7890 samples, 38 features and 1 target column. The target set has 10728 targets to be predicted and 38 features.

Table 2. Statistics of Gap Values

	Training Set	Test Set
Sample Sizes	163491	7890
Minimum Gap Value	0	0
Maximum Gap Value	3827	1181
Mean Gap Value	9.2	15.1
Median Gap Value	1	1
Std. of Gap Values	49.6	65.0

The great discrepancies between mean and median indicates that the gap values are highly skewed to the right which is shown in the following histograms (Fig.1, Fig.2).

[2] Update: After the 1st round of the competition, Di-tech changed the metric from MAPE to MAE (Mean Absolute Error)

[3] Update: The data files on the challenge page has been changed. The current files on the challenge page is for the final round, which has similar format to the first round data we are using for this project.

Figure 1. Training Set Gap Values Histogram

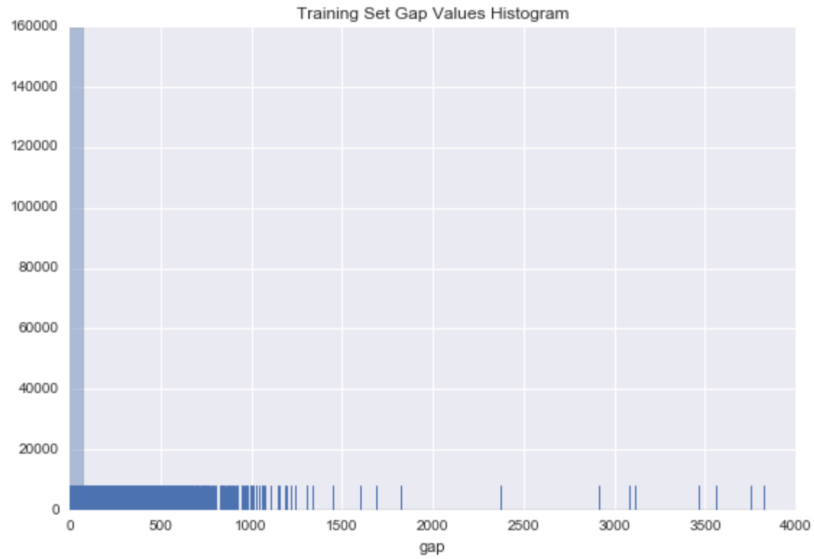
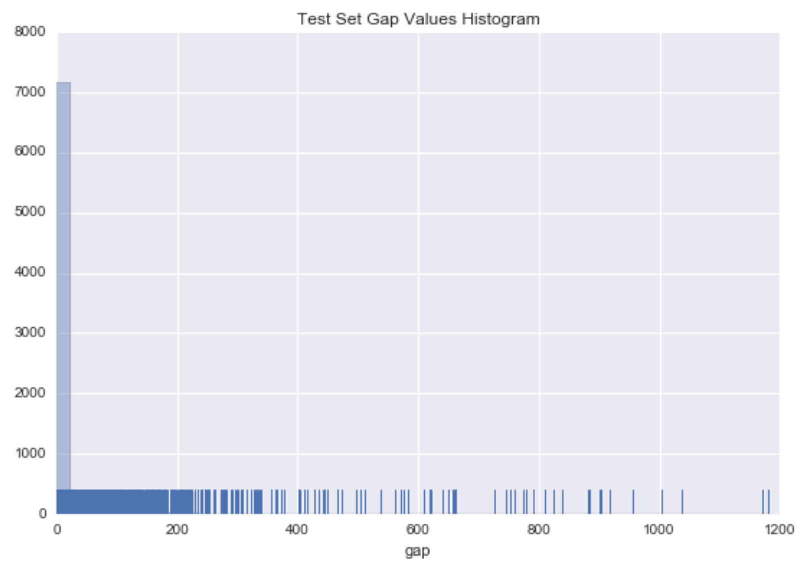


Figure 2. Test Set Gap Values Histogram



Exploratory Visualization

Gap values of training data was plotted for all `district–datetime` combinations (Fig.3). A spike was observed for some time slots at the beginning of 2016-01-01 which was the New Year's eve when the population on road surged (Fig.4). Thus gap values over 1500 were removed as outliers (Fig.5).

Figure 3. All Gap Values of Training Data

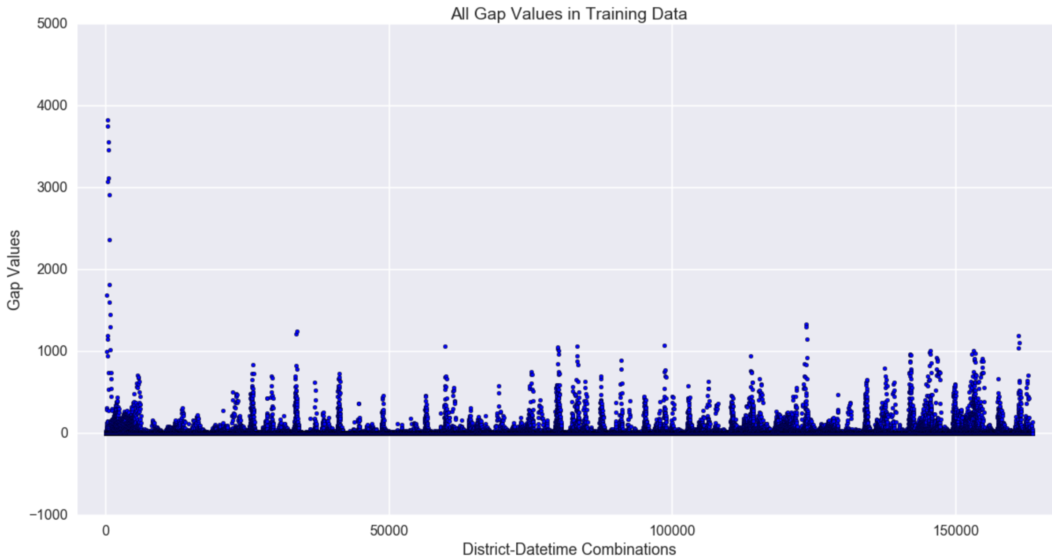


Figure 4. Zoom In on the Spike

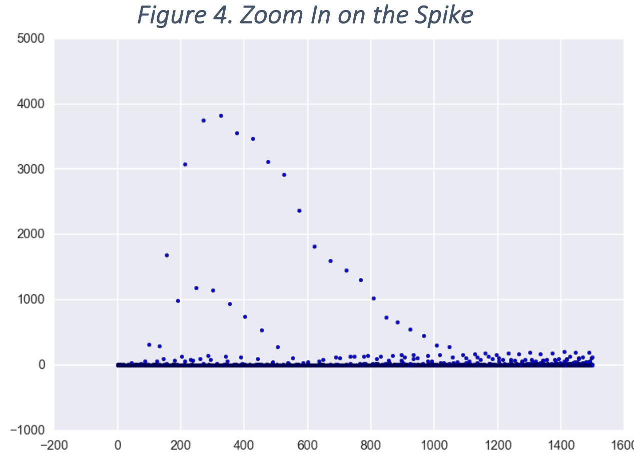
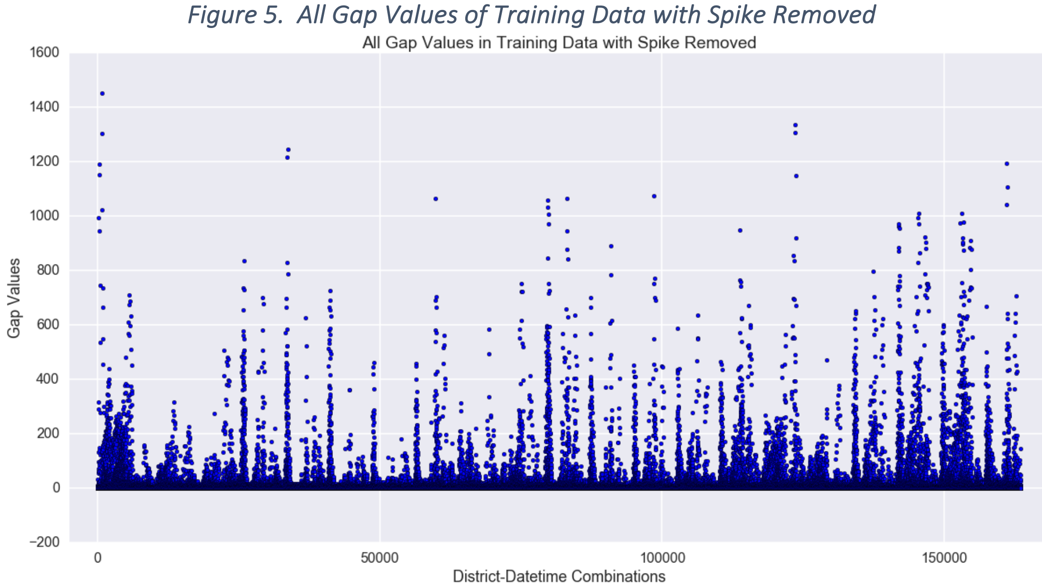
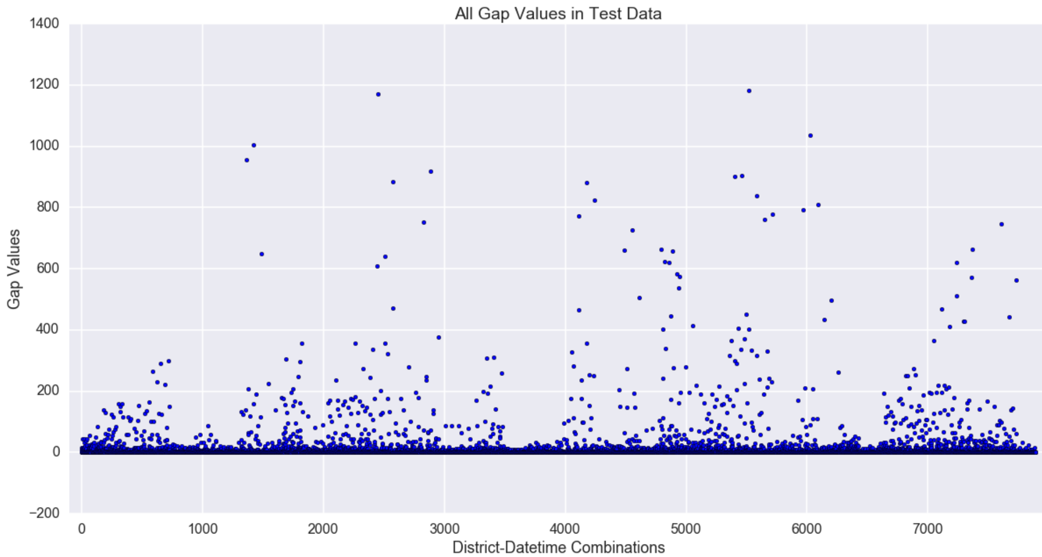


Figure 5. All Gap Values of Training Data with Spike Removed



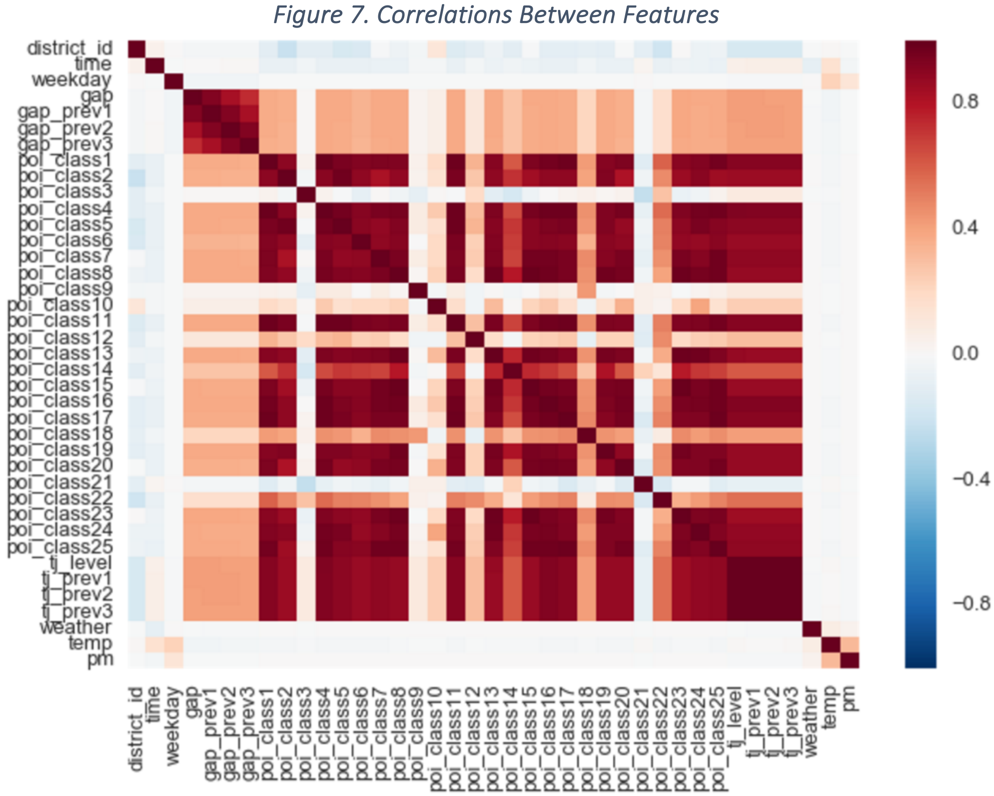
Gap values of test data were also plotted for all `district—datetime` combinations and no abnormality was observed (Fig.6).

Figure 6. All Gap Values of Test Data



The correlations between features were visualized with a heatmap (Fig.7). As we can see, weather data has weaker correlations with other features. We will first use all the features then analyze the feature importance with XGBoost built-in method.

Figure 7. Correlations Between Features

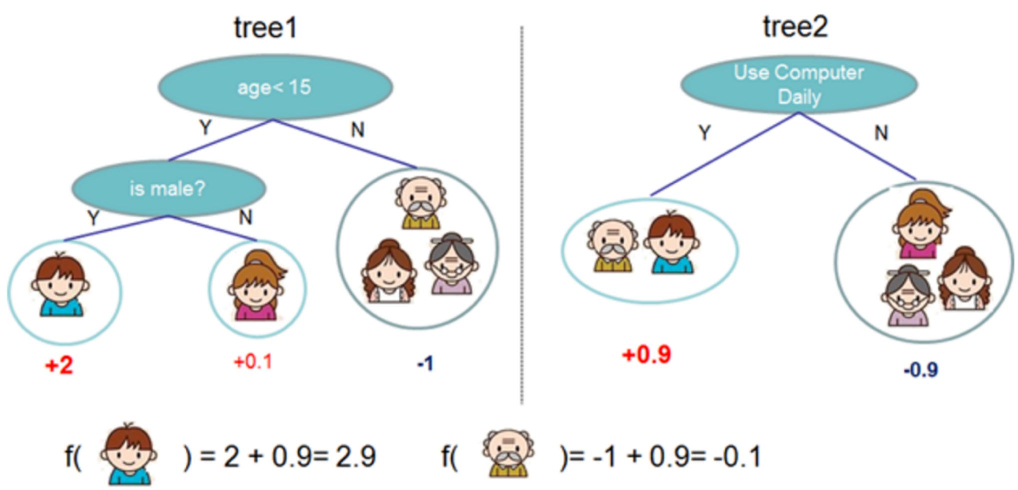


Algorithms and Techniques

For this regression problem, XGBoost (eXtreme Gradient Boosting) was used. XGBoost is an efficient and scalable implementation of gradient tree boosting. Its performance has been recognized by many winning solutions of Kaggle competitions[4].

XGBoost is a model based on tree ensemble which is a set of classification and regression trees (CART). XGBoost classifies the members of a family into different leaves and assigns scores on corresponding leaf. Usually, a single tree is a weak leaner which is not strong enough to use in practice. Boosted trees are typically shallow which are high in bias and low in variance[5]. A tree ensemble model sums prediction of multiple tree. Below is an example of tree ensemble of two trees that classifies whether someone likes play computer game (Fig.8).

Figure 8. Tree Ensemble Model.
The final prediction for a given example is the sum of predictions from each tree.[6]



XGBoost optimizes an objective function consists of training loss, l, and regularization, Ω.

$$Obj(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

XGBoost trees learn with an additive training method which is fixing what have learned and adding new tree one at a time. The prediction value at step t, $\hat{y}_i^{(t)}$ is defined as below,

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

---

[4] https://xgboost.readthedocs.io/en/latest/
[5] http://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest
[6] http://xgboost.readthedocs.io/en/latest/model.html

$$\ldots$$

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

The objective now becomes,

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i{}^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i{}^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant \end{aligned}$$

Take Taylor expansion of the loss function up to second order, the objective becomes,

$$Obj^{(t)} = [l(y_i, \hat{y}_i{}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

where gradient g_i and hessian h_i are defined as the first order and second order derivatives of the loss function respectively.

$$g_i = \partial_{\hat{y}_i{}^{(t-1)}} l(y_i, \hat{y}_i{}^{(t-1)})$$

$$\mathrm{h_i} = \partial^2_{\hat{\mathrm{y}}_\mathrm{i}{}^{(t-1)}} \mathrm{l}(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}{}^{(t-1)})$$

With constant terms removed, the specific objective at step *t* is,

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

which depends only on g_i and h_i. Thus XGBoost allows customized objective function with user-defined functions g_i and h_i.

The regularization term, i.e. the model complexity is defined as,

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

Besides the regularized objective function, to further prevent overfitting, two other techniques, shrinkage and column (feature) subsampling, are used in XGBoost. Shrinkage reduces the impact of individual tree and leaves more space to build future trees. Column subsampling means XGBoost randomly chooses certain ratio of columns to grow trees[7].

---

[7] Tianqi Chen, Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System"

An `xgboost.XGBRegressor` was implemented and scikit-learn `GridSearchCV` was used to tune the parameters of the regressor.

Benchmark

A few scikit-learn ensemble regressors and an unrefined xgboost were implemented. The performances are compared below:

Table 3. Benchmark Performances on Test Data

Algorithm	MAPE
sklearn.GradientBoostingRegressor	0.3892
sklearn.RandomForestRegressor	0.4369
sklearn.AdaBoostRegressor	1.8914
xgboost	0.3797

The low MAPE of XGBoost suggested a deeper look into this algorithm.

Methodology

Data Preprocessing

Two compressed data files, `training_set.tar.gz` and `test_set.tar.gz`, are given and can be downloaded from the competition page. The data has large size of 600 MB and comes in separate files therefore getting a well-shaped master dataset is a big challenge.

The training set contains data for City M continuously from 1/1/2016 to 1/21/2016. The test set contains data for City M for selected time points during 1/22/2016 to 1/31/2016. For targets to be predicted, only time slots were given. The test set carries the information of half an hour before the target time slots. The goal is to forecast the supply-demand gap for all districts across City M for some target time slots during 1/22/2016 to 1/31/2016. Training and test data come in multiple tables which are described in details below.

Table 4. Order Information Table Fields (Training and Test Sets)

Field	Type	Meaning	Example
order_id	string	order ID	70fc7c2bd2caf386bb50f8fd5dfef0cf
driver_id	string	driver ID	56018323b921dd2c5444f98fb45509de
passenger_id	string	user ID	238de35f44bbe8a67bdea86a5b0f4719
start_district_hash	string	departure	d4ec2125aff74eded207d2d915ef682f
dest_district_hash	string	destination	929ec6c160e6f52c20a4217c7978f681
Price	float	Price	37.5
Time	string	Timestamp	2016-01-15 00:35:11

The Order Information Table records each order history. If an order is not answered by any driver, the `driver_id` is recorded as `Null`. Thus for district d_i and time slot t_j, the value of gap gap_{ij} is the count of `Nulls`. The `start_district_hash` needs to be converted into `district_id`. The `order_id`, `passenger_id`, `dest_district_hash` and `price` are irrelevant to solving the problem hence can be dropped. The timestamps should be converted into time slots 1 - 144 within a day. The date can be converted into day of the week which is more informative than date in the format of YYYY-MM-DD.

Table 5. District Information Table Fields (Training and Test Sets)

Field	Type	Meaning	Example
district_hash	string	District hash	90c5a34f06ac86aee0fd70e2adce7d8a
district_id	string	District ID	1

The District Information maps `district_hash` to `district_id`. For City M in this challenge, there are 66 districts with ID 1 - 66.

Table 6. Points of Interest (POI) Information Table Fields (Training and Test Sets)

Field	Type	Meaning	Example
district_hash	string	District hash	74c1c25f4b283fa74a5514307b0d0278
poi_class	string	POI class and number	1#1:41 2#1:22 2#2:32

The POI Information provides the number of facilities for a class in a district. The number around "#" indicates the 1st and 2nd level of a class. For example, 2#1 represents a facility class with 1st level as 2 and 2nd level as 1. The number after ":" is the number of facilities for this specific class. For simplicity, the 2nd level was ignored and the number of facilities was summed based on the 1st level only. The actual meaning of the numbers denoting classes is not given thus is considered irrelevant.

Table 7. Traffic Jam Information Table Fields (Training and Test Sets)

Field	Type	Meaning	Example
district_hash	string	Hash value of the district	1ecbb52d73c522f184a6fc53128b1ea1
tj_level	string	Number of road sections at different congestion levels	1:231 2:33 3:13 4:10
tj_time	string	Timestamp	2016-01-15 00:35:11

The Traffic Jam Information records the number of road sections for 4 different congestion levels, with 1 being the least congested and 4 the most congested. A composite traffic level was defined as a weighted sum of the number of road sections for all 4 levels. For example, the tj_level for "1:231 2:33 3:13 4:10" is calculated as $0.1\times231 + 0.2\times33 + 0.3\times13 + 0.4\times10 = 37.6$.

Table 8. Weather Information Table Fields (Training and Test Sets)

Field	Type	Meaning	Example
Time	string	Timestamp	2016-01-15 00:35:11
Weather	integer	Weather Type	7
temperature	double	Temperature	-9
PM2.5	double	pm25	66

The Weather Information records weather conditions, temperature in ^{o}C and PM2.5 level, as a measure of air quality, every 5 minutes. The actual meaning of the numbers denoting the weather types is not given thus is considered irrelevant.

Prediction targets are provided in the following format:

Table 9. Prediction Targets (Target Set)

District ID	Date - Time Slot
1	2016-01-23-46
2	2016-01-23-46
…	…

All tables were combined by merging on mutual variables which resulted a final dataset in the following format:

Table 10. Processed Dataset Ready for Modeling

Field	Type	Meaning	Example
district_datetime	string	combined district ID, date and time slot	1,2016-01-01-4
district_id	int	district ID	1
datetime	datetime	date and timestamp for every 10 min	2016-01-01 00:30:00
time	int	time of the day	4
weekday	int	day of the week	4
gap	float	gap of current time slot	5
gap_prev1	float	gap of last time slot	10
gap_prev2	float	gap of second last time slot	7
gap_prev3	float	gap of third last time slot	9
poi_class1	int	number of POI facilities of class 1	22659
poi_class2	int	number of POI facilities of class 2	30544
poi_class3	int	number of POI facilities of class 3	1909
…	int	number of POI facilities of class 4 - 23	…
poi_class22	int	number of POI facilities of class 22	5229
poi_class23	int	number of POI facilities of class 23	1909
poi_class24	int	number of POI facilities of class 24	65653
poi_class25	int	number of POI facilities of class 25	8051
tj_level	float	traffic jam level of current time slot	267
tj_prev1	float	traffic jam level of last time slot	268
tj_prev2	float	traffic jam level of second last time slot	271
tj_prev3	float	traffic jam level of third last time slot	271
weather	float	weather type	1
temp	float	temperature	3
pm	float	PM 2.5 level	177

Targets need to be predicted have only district ID and time slot. Test set contains the data of half an hour before the target time slots. Target set and test set were combined and missing values of traffic and weather data were filled with `pandas.Dataframe.fillna` and `pandas.DataFrame.interpolate`. Please see `/code/data_processing.ipynb` for complete data processing code.

Implementation

A quick XGBoost was implemented with customized evaluation method of MAPE. The submission of predictions from benchmark XGBoost model scored **0.3585** and ranked top 47% on the leaderboard. Code snippet is shown below:

```
# split training and test data
X_train, X_test, y_train, y_test = cross_validation.train_test_split(train[features], train['gap'], test_size=0.4,
random_state=0)

dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# define customized MAPE evaluation metric for XGBoost
def mapeEval(preds, dtrain):
    gaps = dtrain.get_label()
    preds[preds<1] = 1
    err = np.abs(gaps - preds) / gaps
    err[np.where(gaps==0)[0]] = 0
    err = np.mean(err)

    return 'MAPE', err

param = {'max_depth':3, 'eta':0.2, 'silent':1}
bst = xgb.train(param, dtrain, feval=mapeEval)

# make prediction on targets
dtarget = xgb.DMatrix(target[features], missing=None)
preds = bst.predict(dtarget)
```

Refinement

With XGBoost scikit-learn wrapper, the model parameters were tuned with scikit-learn `GridSearchCV`. Parameters tuned are shown below:

Table 11. Parameters Tuned

Parameters	Meaning	Values Tested	Best Value
objective	objective function	('reg:linear', 'count:poisson')	'reg:linear'
max_depth	Maximum tree depth for base learners	(6, 8, 10)	8
min_child_weight	Minimum sum of instance weight(hessian) needed in a child	(1, 3, 5)	3
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	(0, 2, 4)	2
reg_alpha	L1 regularization term on weights	(0, 2, 4)	0
reg_lambda	L2 regularization term on weights	(1, 3, 5)	3
Subsample	Subsample ratio of the training instance	(0.5, 1)	1
colsample_bytree	Subsample ratio of columns when constructing each tree	(0.5, 1)	1
colsample_bylevel	Subsample ratio of columns for each split, in each level	(0.5, 1)	1

Complete code of exploratory analysis and model building can be found in `/code/model_building.ipynb`.

Results

Model Evaluation and Validation

A final model was generated with tuned parameters. The submission of predictions from final model scored **0.3275** and ranked top 20% on the leaderboard which was a significant improvement to the unrefined XGBoost model. Code snippet of the final model is shown below:

```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(train[features],train['gap'], test_size=0.4,
random_state=0)

# Final Model
final_xgb = xgb.XGBRegressor(max_depth=8, learning_rate=0.01, n_estimators=5000, silent=True,
objective='reg:linear', nthread=1, gamma=2, min_child_weight=3, subsample=1, colsample_bytree=1,
colsample_bylevel=1,
reg_alpha=0, reg_lambda=3, seed=0)

final_xgb.fit(X_train, y_train)
preds = final_xgb.predict(X_test)
```

MAPE of the final model on test data was **0.3629**, which aligned to MAPE of **0.3275** of the final model on unseen data in target set. Therefore, the final model is considered robust and generalized well on unseen data.

Justification

MAPE's of the predictions on target set from both benchmark model and final model are shown below:

Table 12. Comparison of Benchmark and Final Model Submissions

	Benchmark Model (Untuned XGBoost)	Final Model (Tuned XGBoost Regressor)
MAPE	0.3575	0.3275
Rank	37%	20%

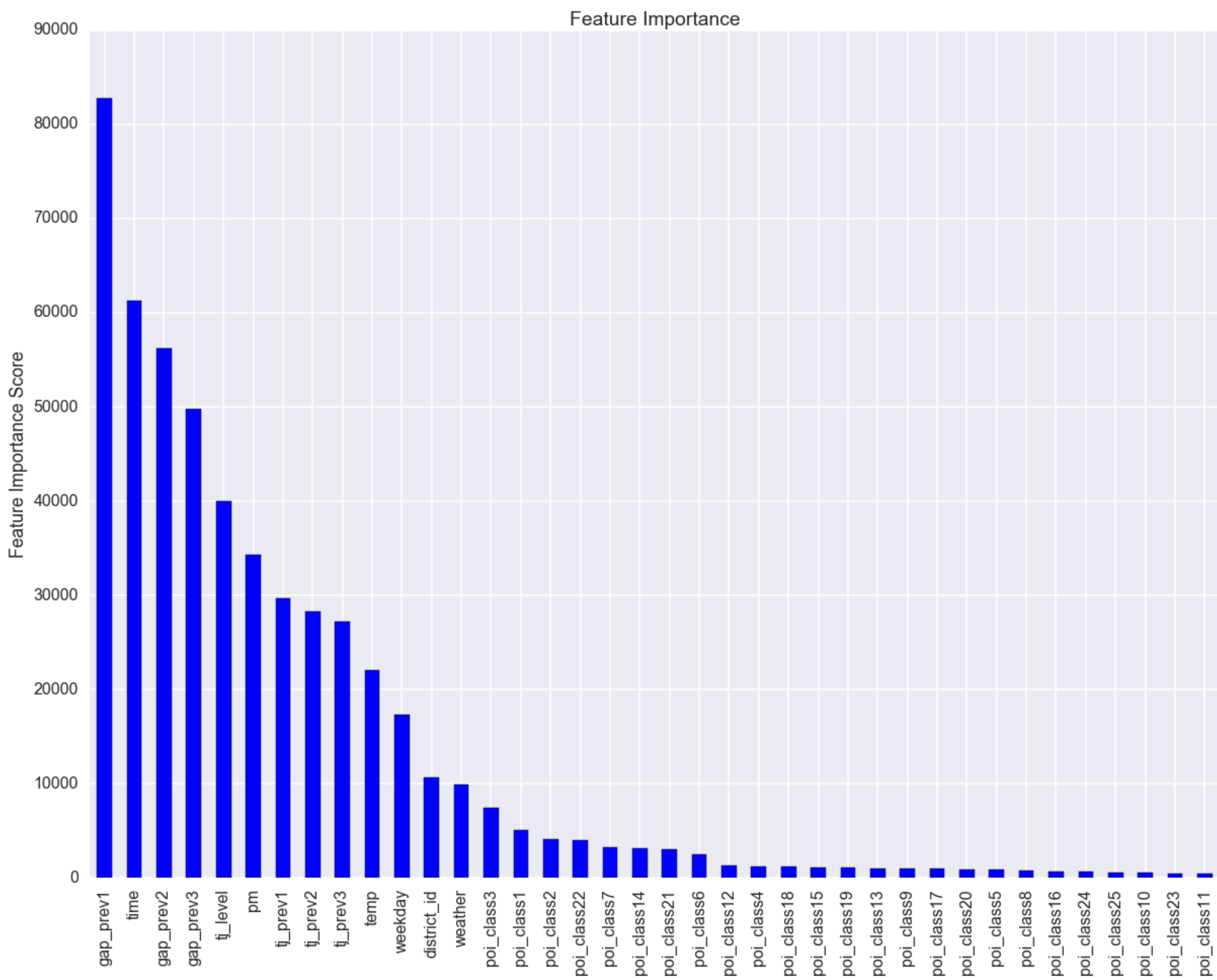
Though there is still room for improvement on the final results, clearly the tuned final model generated significantly better results for the target set. Possible movements to further improve the MAPE will be discussed in the Improvement section.

Conclusion

Free-Form Visualization

The importance of each feature was visualized with the following plot.

Figure 9. Feature Importance



To test the performance of using top important features, the final model was run with the least 10 important features dropped. The MAPE of this fewer-feature model was **0.3695** which was higher than MAPE of the all-feature model, **0.3629**. Therefore, all features were kept and used.

Reflection

The challenge is overall a well-defined regression problem. The most time-consuming part was data processing due to the large training data that is provided with multiple files located in different directories. Another challenge was the missing traffic and weather information in target set. Test set has data half an hour prior to each time slot in target set. Hence, traffic and weather data in test set was leveraged to interpolate the missing information for the target set.

After the data was processed and ready for modeling, the next step was to choose a well-performing supervised learning algorithm. With some research, XGBoost was discovered as a state-of-art, efficient and promising package. Taking advantage of XGBoost's easy implementation and its ready-to-use scikit-learn wrapper, parameters of `XGBoost.Regressor` were tuned with `GridSearchCV`.

During parameter tuning, an annoying bug on `_winreg` in module `multiprocessing` on macOS occurred when XGBoost parameter `nthread` was set to values other than 1. Therefore, `nthread=1` was used throughout the process which disabled parallel running XGBoost with multiple threads. Being unable to exploit the full computing power, parameters were tuned one at a time and parameter grids contained only 2- 3 values to search from. The most effective improvement happened when using small `learning_rate`, in this case 0.01. Another lesson learned was when decreasing `leanring_rate`, `n_estimators` should be increased[8].

Improvement

Overall, the final XGBoost Regressor model worked well on solving the challenge problem, at least better than the scikit-learn regressors implemented as benchmarks. XGBoost allows using customized objective function in running boost trees. For this challenge, build-in linear regression objective was used but customized MAPE objective could also be considered.

To further improve the current model, with the multiprocessing issue fixed, more values of parameters could be tested. According to many winning solutions on Kaggle, XGBoost is allowed to build deep trees by setting `max_depth` high which is something worth trying[9,10]. Another new concept learned during research is meta-modeling which combines results from different models. Also mentioned in many winning Kaggle solutions, this idea is worth experimenting in future problem solving. For example, neural network could be implemented and combined with XGBoost.

[8] http://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
[9] http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/
[10] http://blog.kaggle.com/2015/11/30/flavour-of-physics-technical-write-up-1st-place-go-polar-bears/