Definition

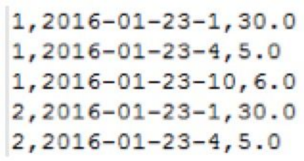
Project Overview

For my Capstone project I have decided to join the DiDi Research competition. DiDi ChuXing is a Chinese ride sharing service similar to Uber and Lyft, and it currently processes over 11 million trips, plans over 9 billion routes, and collects over 50TB of data per day. To use these data to help supply and demand forecasting, the company has created this competition that challenge participants improve algorithms, specifically how the company ensures riders always get a car when and where they need it, and drivers know where to be even before a ride is hailed.

Problem Statement

Because this project is an entry to the competition, the problem statement and the expected solution are clearly defined by the competition organizers. The problem we're facing is that whenever someone requests a ride there isn't always a driver nearby to pick them up. This is measured by the variable gap which we can figure out from r - a where r is the number of ride requests and a is the number of drivers who answered the request. For this competition the dataset comes from one city, which is divided into nonoverlapping square districts, denoted by D and a 24 hour period in a day is divided into 144 time slots t with each slot being 10 minutes long. Given the data of every district and time slots, we're asked to predict the gap variable, or the difference between ride requests and ride answers, for all districts.

The solution is expected to be a CSV file with columns being district ID, time slot, prediction value. For example:



```
1,2016-01-23-1,30.0
1,2016-01-23-4,5.0
1,2016-01-23-10,6.0
2,2016-01-23-1,30.0
2,2016-01-23-4,5.0
```

The strategy for solving the problem is a fairly standard one for regression problems. We take the data given from the competition, fit algorithms to come up with models, and use those models to produce predictions. The predictions in this case are going to be a list of numbers, which are the gap values, or the difference between demand and supply. We then put those gap values in a file generator that will line them up against the district IDs and the time slots.

Because we were given lots of different data but only district ID and time slot for prediction, the features and the predictors don't match up. We can use the data that's not the predict file, i.e. weather, traffic, etc. but it would not work when we need to predict the values in the predict file because the additional data for those predictors are not there. To deal with this we can either predict the missing data or not use them. If we go with the predict route then we would have to predict everything we needed for the time slots in the predict file, which is a lot more complex and potentially decrease in accuracy because every feature is different, and the variability increases. Or we can use only the predictors that are present which is district ID and time slot. This way we have a lot less data because we won't use weather or traffic conditions but the problem becomes much simpler.

Metrics

The performance of this project is measured by the competition ranking, and the ranking is based on the calculation of mean absolute percentage error (MAPE) of the solution predictions. The MAPE is calculated with the following formula provided by the competition:

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

Where n is the number of districts, which is a constant number of 66, and q is the number of time slots presented in the predict file, which is 43. The gap is the difference between supply and demand at any given district and time slot. And s is the predicted value for that district at that time slot. This formula only takes into account when the gap value is greater than 0, so if the gap is 0 then that row is skipped. The lower the MAPE the better.

For example if the actual gaps are 1, 2, 3 and the predictions are 1, 2, 3, then MAPE would be 0. But if the gaps are 1, 2, 3 but the predictions are 1, 2, 4 then MAPE would be 1/8514 by plugging in the numbers into the formula.

Analysis

Data Exploration

The dataset that we're provided comes in training data and testing data. These two datasets are split from a month of data collection from the beginning to the end of January 2016. The training data are from the first day of the month to the 21st of the month, and from then on the data is put into the testing set. Both training data and testing data contain information from 5 aspects of the rideshare transaction. They are:

1. City district cluster map, or the mapping between district ID and its hash value.
2. Order data, which contains the order ID, driver ID, passenger ID, start and end district hash value, the price and time of the transaction.
3. Points of interest, which is in the format of district hash and POI class. The POI class is formatted as class level 1, followed by #, and class level 2 which is more specific, followed by : and the number of facilities of the same type.
4. Traffic jam data, which is the district hash, 4 sides of the intersection and their congestion levels, along with timestamp
5. Weather data of the whole city for the month, which contains timestamp, current weather classification as a number, temperature in celsius, and the air pollution level PM2.5 which describes particulate matter that is 2.5 micrometers in diameter and smaller.

The final predictions are going to be the cumulative totals of the gaps between the number of drivers and the number of ride requests, and these gap values come from order data. This makes this dataset the most relevant to what we're trying to predict, so I decided to focus on these data. The following is a sample of the data:

order_id	driver_id	passenger_id	start_district_hash	dest_district_hash	Price	Time

97ebd0c6680f7c0535dbfdead6e51b4b	dd65fa250fca2833a3a8c16d2cf0457c	ed180d7daf639d936f1aeae4f7fb482f	4725c39a5e5f4c188d382da3910b3f3f	3e12208dd0be281c92a6ab57d9a6fb32	24	2016-01-01 13:37:23
92c3ac9251cc9b5aab90b114a1e363be	c077e0297639edcb1df6189e8cda2c3d	191a180f0a262aff3267775c4fac8972	82cc4851f9e4faa4e54309f8bb73fd7c	b05379ac3f9b7d99370d443cfd5dcc28	2	2016-01-01 09:47:54
abeefc3e2aec952468e2fd42a1649640	86dbc1b68de435957c61b5a523854b69	7029e813bb3de8cc73a8615e2785070c	fff4e8465d1e12621bc361276b6217cf	fff4e8465d1e12621bc361276b6217cf	9	2016-01-01 18:24:02
cb31d0be64cda3cc66b46617bf49a05c	4fadfa6eeaa694742de036dddf02b0c4	21dc133ac68e4c07803d1c2f48988a83	4b7f6f4e2bf237b6cc58f57142bea5c0	4b7f6f4e2bf237b6cc58f57142bea5c0	11	2016-01-01 22:13:27
139d492189ae5a933122c098f63252b3	NaN	26963cc76da2d8450d8f23fc357db987	fc34648599753c9e74ab238e9a4a07ad	87285a66236346350541b8815c5fae94	4	2016-01-01 17:00:06

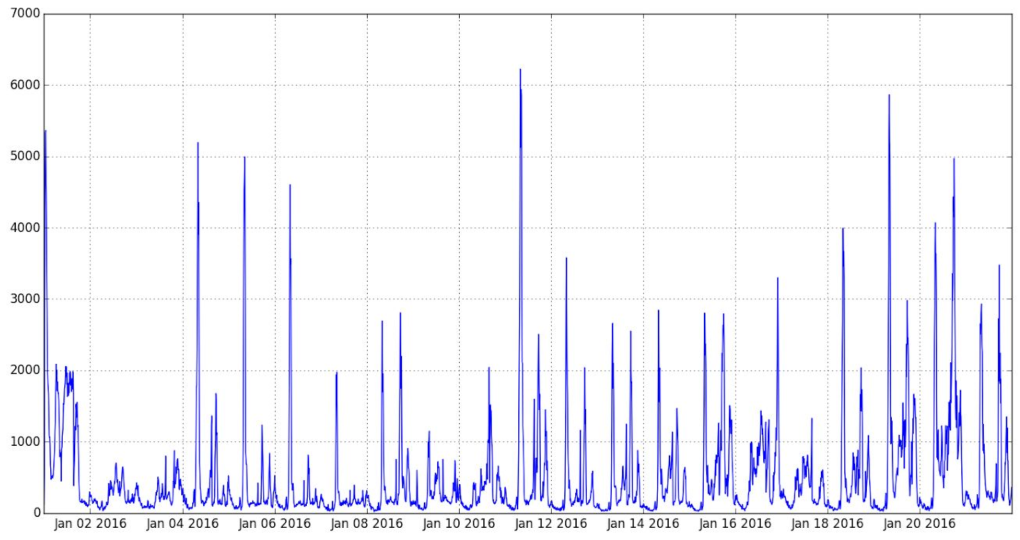
When the driver ID is null or NaN it means the ride request was not answered. We found that there's 14 districts containing null driver IDs, with district 51 being the highest amount at 1,328,539 unanswered rideshare requests. This is followed by district 7 at 423,113 and district 14 at 228,234.

When looking at the just first day of the order data, which contains 501,287 entries, 175,710 of them goes unanswered. This is 35% of the total orders just in 1 day. The average trip cost is 18.79 with a standard deviation of 16.91. The minimum is 0, 25-percentile is at 8, 50-percentile is at 14, 75-percentile is at 23 and the max being 499 for the first data file.

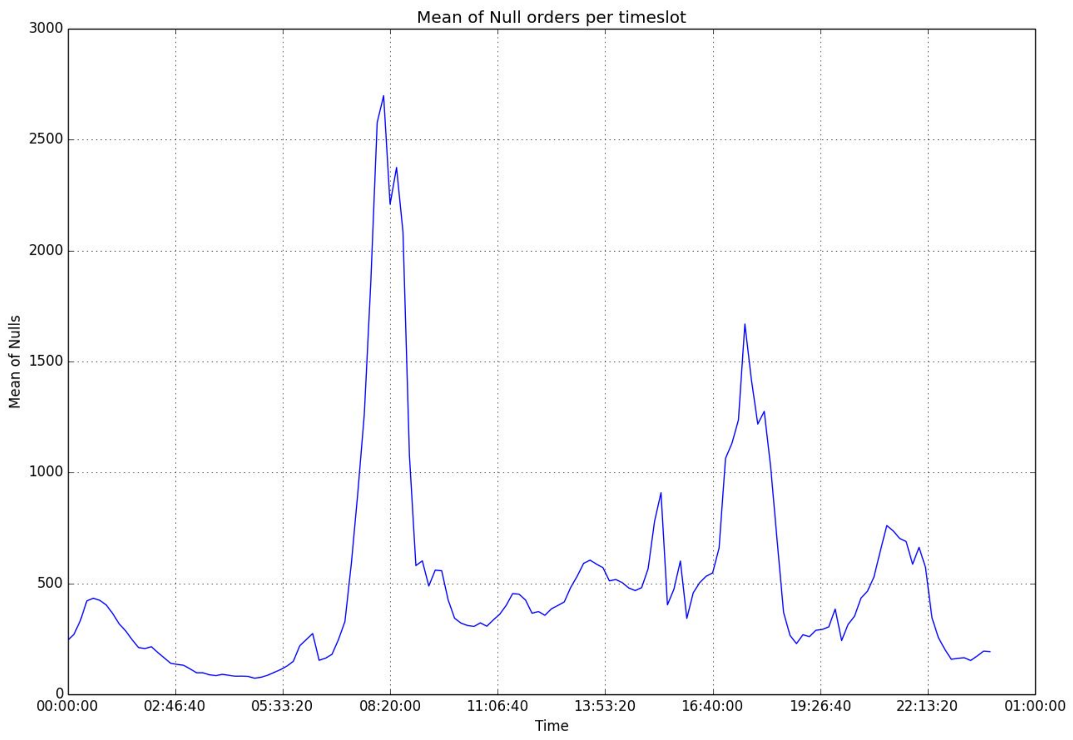
Looking at the second order data file, the general statistics are pretty much the same as the first one. The mean is 18.38 with a standard deviation of 16.6. The percentile numbers are the same except the max is 496. The number of null driver IDs is 35,626 out of 322,284 which is 11%.

Judging by these numbers, it seems that most of the trips taken are fairly short distance trips, with some outliers being very high amount in price.

Exploratory Visualization

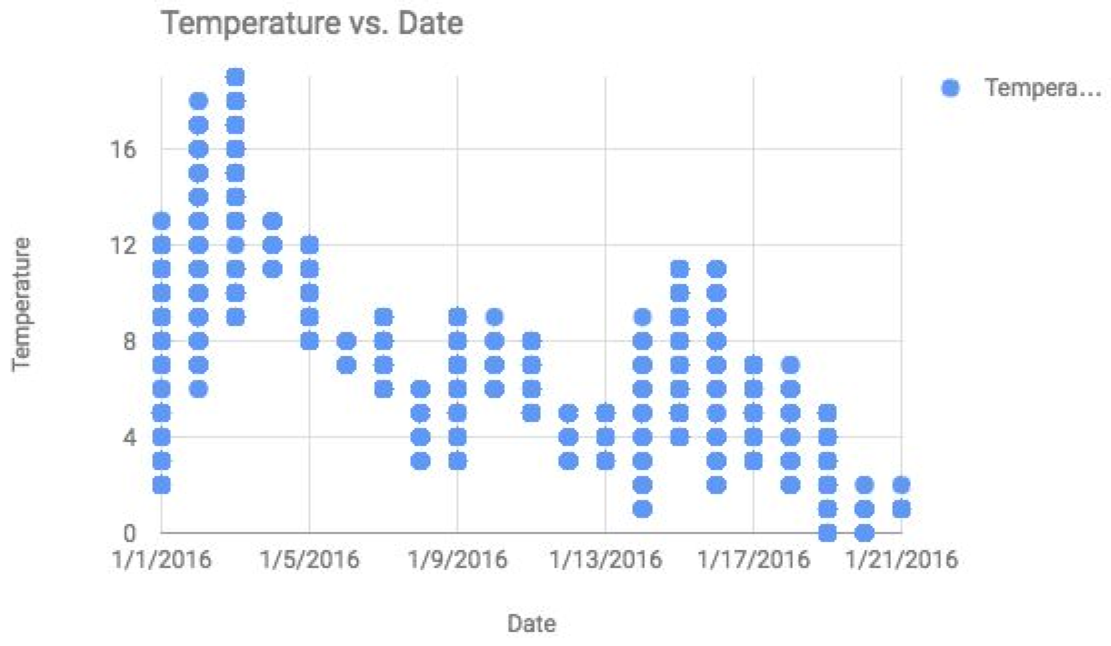


The above image shows the frequency of unfilled ride requests over the month, with each grid containing 2 days. From looking at the graph it seems that peak demand is around 8 in the morning which is when people go to work, and January 11 being the highest in demand, which was a Monday. Other Mondays have the same pattern for January 4th and 18th but morning rush hours in general creates more demand than the drivers can handle.

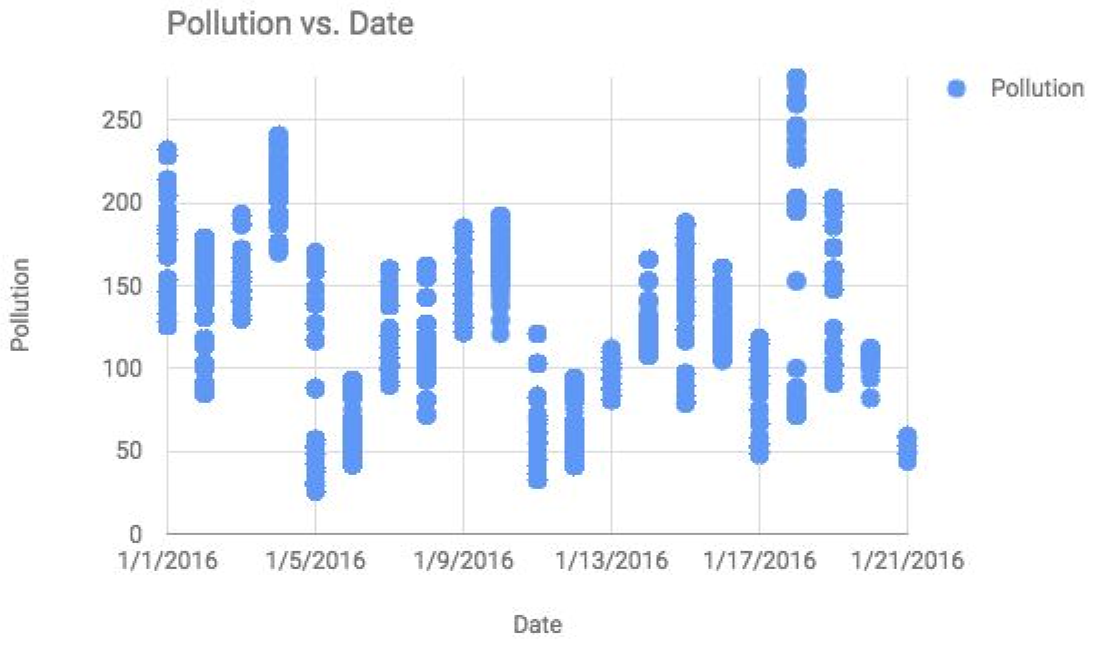


This image shows the average null order counts over a day's 24 hour period. Again we can see the number peak at morning rush hour just after 8am, and then again in the afternoon rush hour.

Having used Uber and Lyft myself, one of the biggest factors of whether or not I request a ride is if it's raining. For people who don't own car it's usually not very comfortable getting to places in the rain, even if you have rain gears like umbrella or a rain jacket. In comparison it's much easier to just wait somewhere and have a car come to you. With this in mind I looked at how the weather data goes into play for demand. The weather data for Jan 11 for example, shows that day had weather of category 2 and 4.

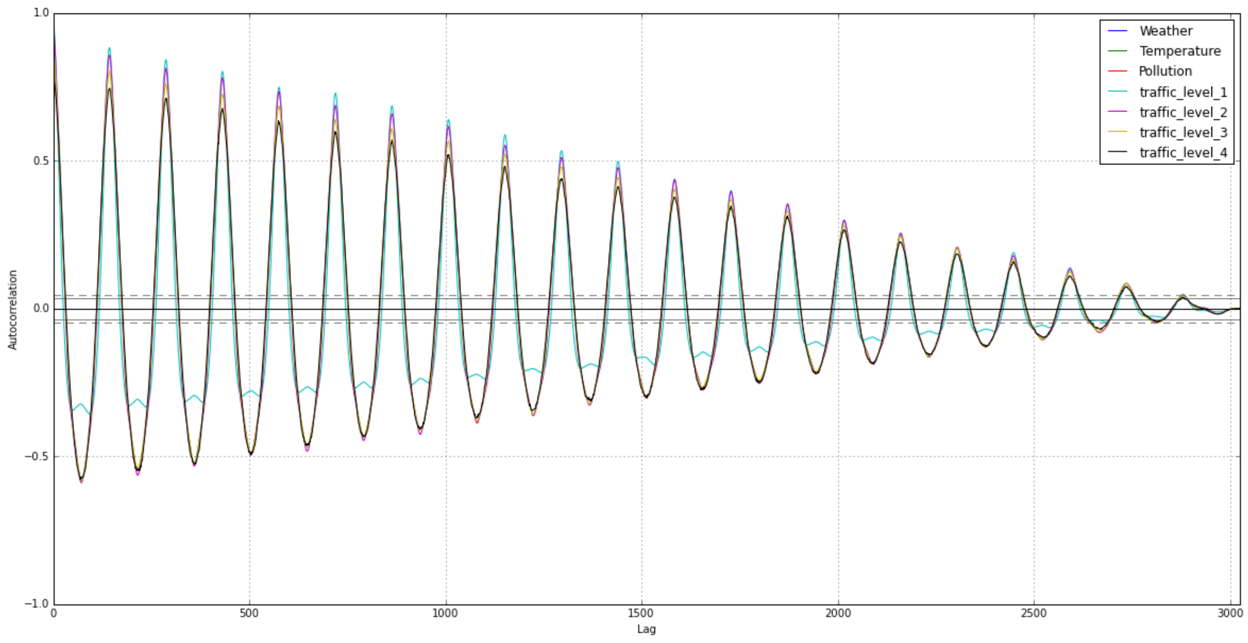


From the temperature versus date chart above it shows that the temperature dropped on the 11th and after, suggesting that precipitation occurred during that time.

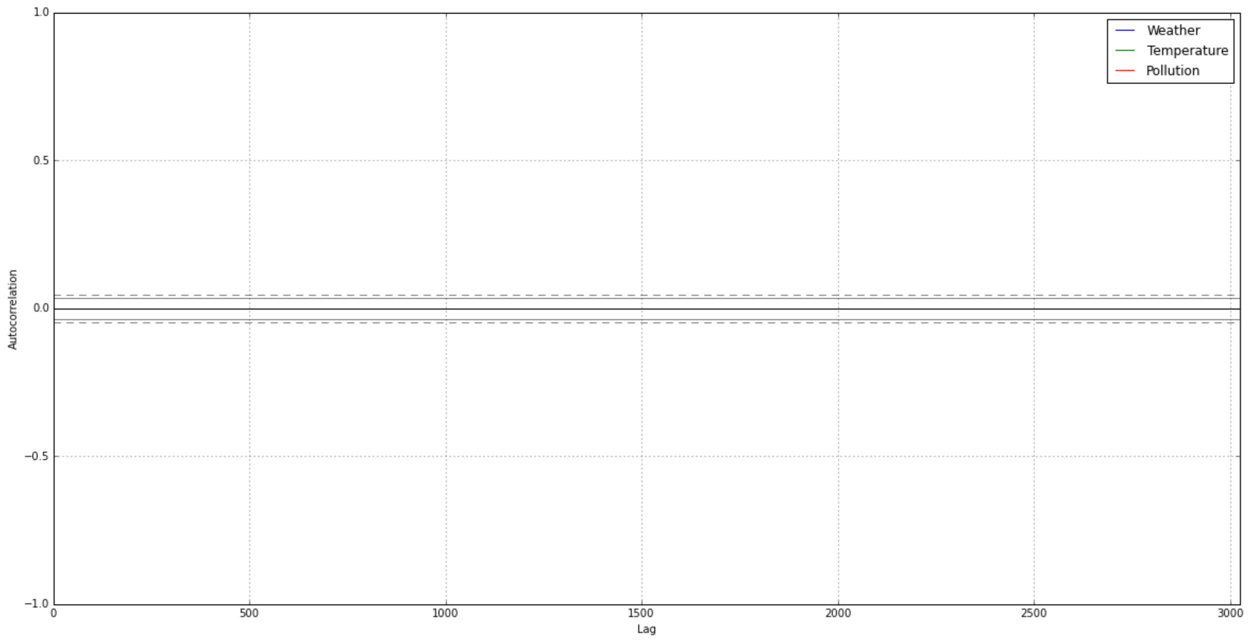


Also from the pollution chart there is a drop on the 11th, also suggesting that it rained on that day.

We also used autocorrelation plots to find out the randomness in time series. Autocorrelation plots are done by computing autocorrelations for data values at varying time lags. If time series is random, such autocorrelations should be near zero for any and all time-lag separations. If time series is non-random then one or more of the autocorrelations will be significantly non-zero.



The above autocorrelation plot shows the relationship between traffic levels over time of the month. It shows traffic in the day has a positive correlation with traffic levels in other days, and negative correlation with traffic levels at night.



As a comparison this is a autocorrelation plot of the weather data over time, which it shows is completely random.

Algorithms and Techniques

The end result of our prediction is in Double format which is a number, representing the difference between the number of ride requests and the number of driver answers. Therefore this is a regression problem. And because there are multiple independent features that affect the outcome, each one could have a deciding factor in the end result, similar to a decision tree. Hence the reason why Decision Tree Regressor (DTR) would be a good choice for this problem. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. When calling the regressor function, the parameter includes:

- Criterion, default=”mse”
- Splitter, default=”best”
- Max_features, default=None
- Max_depth, default=None
- Min_samples_split, default=2
- Min_samples_leaf, default=1
- Min_weight_fraction_leaf, default=0.
- Max_leaf_nodes, default=None
- Random_state, default=None
- Presort, default=False

All parameters are optional. The algorithm would fit the formatted data, and predict the number of null driver IDs as the number of unanswered ride requests.

In addition to using DTR I have also used Support Vector Regressor (SVR) to get a feel how the data works with different methods. Because SVR is one of the most popular methods in supervised learning for regression, it is a good choice for getting a benchmark of how the subsequent models do. The following is the parameter list, and all parameters are optional:

- C, default=1.0
- Epsilon, default=0.1
- Kernel, default=’rbf’
- Degree, default=3
- Gamma, default=’auto’
- Coef0, default=0.0
- Shrinking, default=True
- Tol, default=1e-3
- Cache_size
- Verbose, default: False
- Max_iter, default=-1

Benchmark

For the benchmark I have used SVR with all default parameters. I took all the training data and the test data provided by the competition, combined them and mixed them up using train_test_split function in cross_validation. I then fit SVR to x_train and y_train, predict on x_test, and using the MAPE implementation to calculate the score compared to y_test, and got 8.407751329 as MAPE.

Methodology

Data Preprocessing

The data provided to us comes in the format where each feature is in their own files. For order, traffic, and weather data, each day is in its own file. These files have the data in a timestamp format as opposed to time slots that we need to predict against. For this we took the minutes of each timestamp and multiplied by 10 to get the time slots.

When looking at the dataset from a whole day's perspective, there's a lot of the data missing. For example in some days there's only weather data after 7am, or there's only some data for the traffic condition during the day. When grouped by district, we found that there's no traffic data for a certain district. We ended up dropping that district altogether when reformatting the data.

When feeding the data into algorithms, there was also the issue of incompatible format. The order table information has time series data in datetime format, so we had to convert that into time slots by calculating the timestamp into time slots.

After we reformatted all the feature files into one file for each day there's some rows that that the null count column not having data, even though some other rows shows 0 when the order completed is equal to the number of total orders. Because we cannot confirm the missing null count always mean there isn't a demand gap, I had to drop those rows when training the algorithms.

When I tried to fit algorithms to the formatted data in a DataFrame initially it gave me value error, because the time format is in date time. I then wrote a function that takes the date time string, get the time component and subsequently the hour and minute components and add them up to figure out the corresponding time slot.

Implementation

To predict the gap values I have used DTR for my initial implementation. My teammate took all the data provided and basically concatenated them into a training data file and a test data file, following the format that the data was in. I then took these two csv files, load them into pandas and concatenate them to make one big dataframe. However since we don't have the data needed to predict the time slots in the predict file, I've decided to only use time, null orders, and district ID for my data columns. I then dropped all the rows that have missing values. I then took out the day of month and time of day values from the timestamp column and turned it into two columns. After this I took all the data that's left and split them up using train_test_split with default parameters from cross_validation and got x_train, x_test, y_train, y_test. The default parameter sets the test size to 0.25 which is a quarter of the whole dataset, and the rest of the data, which is 3/4th of the whole set, is the training set.

After getting the training and test data I simply used DecisionTreeRegressor to fit the training values and predicted on x_test, and then used MAPE calculator to find the score and got 8.0147406786.

Refinement

To improve upon my initial implementation I have used grid search. I made a custom scorer using the MAPE calculator with greater is better set to false. Then I used GridSearchCV with DTR and the scorer passed in. I have cross validation (CV) value set to 30 because there's roughly 30 days in a month, and since we're dealing with a month's worth of data it's good to assume each day's gap values are at least somewhat independent of each other. Also cv of 30 is much less than the data number count so there is still a significant amount of data in each fold. The max depth it searches over ranges from 10 to 30 because I was using 1 to 10 and found that it would always end up on the high side.

After the grid search produces a model, I use it to fit the training data, which is 3/4th of the whole dataset split by the train_test_split function. After the model finds the best depth for the training data I let it predict on the test dataset which is 1/4th of the original set. After the final model makes the predict on the test set I pass in the predictions into the MAPE calculator to find out the score. I got 7.50794371124 at depth 18 and no dedicated validation set was used.

Results

Model Evaluation and Validation

The final model only uses the data that's in the same format as the prediction labels. In other words all the traffic, weather, and other data were not used because we don't have the data for prediction. From the timestamps I got out day of the month and time of the day, two features as opposed to just timestamp to feed into DTR. I have also tried adding day of the week as another feature but it turns out it does not help the regressor so I took it out.

As for the final model, I have the DTR at a max depth of 18, which was produced by GridSearchCV with 30 folds of CV, along with custom MAPE scorer. Cross validation partitions the available data into multiple sets, in order to reduce the number of samples that can be used for learning the model and prediction the label at the same time. This can help prevent the model from indirectly learning about the test set and therefore reduce overfitting. CV chops up the training dataset and reuses them to make more data to train the model while keeps the data clean from being overfitted by the model. In this case the training data is split into 30 smaller sets and 29 of them are used as the actual training data, and the remaining set is used for testing.

The max_depth parameter of DTR determines the maximum number of depths, or layers of the decision tree can be if the tree is thought about graphically. It limits the amount of layers from the tree's trunk to the furthest leaves. I have it at 18 which means the DTR can only split the data 18 different times on each possible tree branch. It's kind of like it can only ask 18 questions in the 20 questions game. If this is not set then the model will keep making rules to split the data until it can't split them anymore which would cause overfitting. Having a max depth lets some data stay in a group so it can still generalize other data.

After the search settled on the final model, I took it to predict on the test set generated by the train_test_split and got 7.5 on MAPE.

Justification

The following is the comparison of models' ranking on MAPE at different stages:

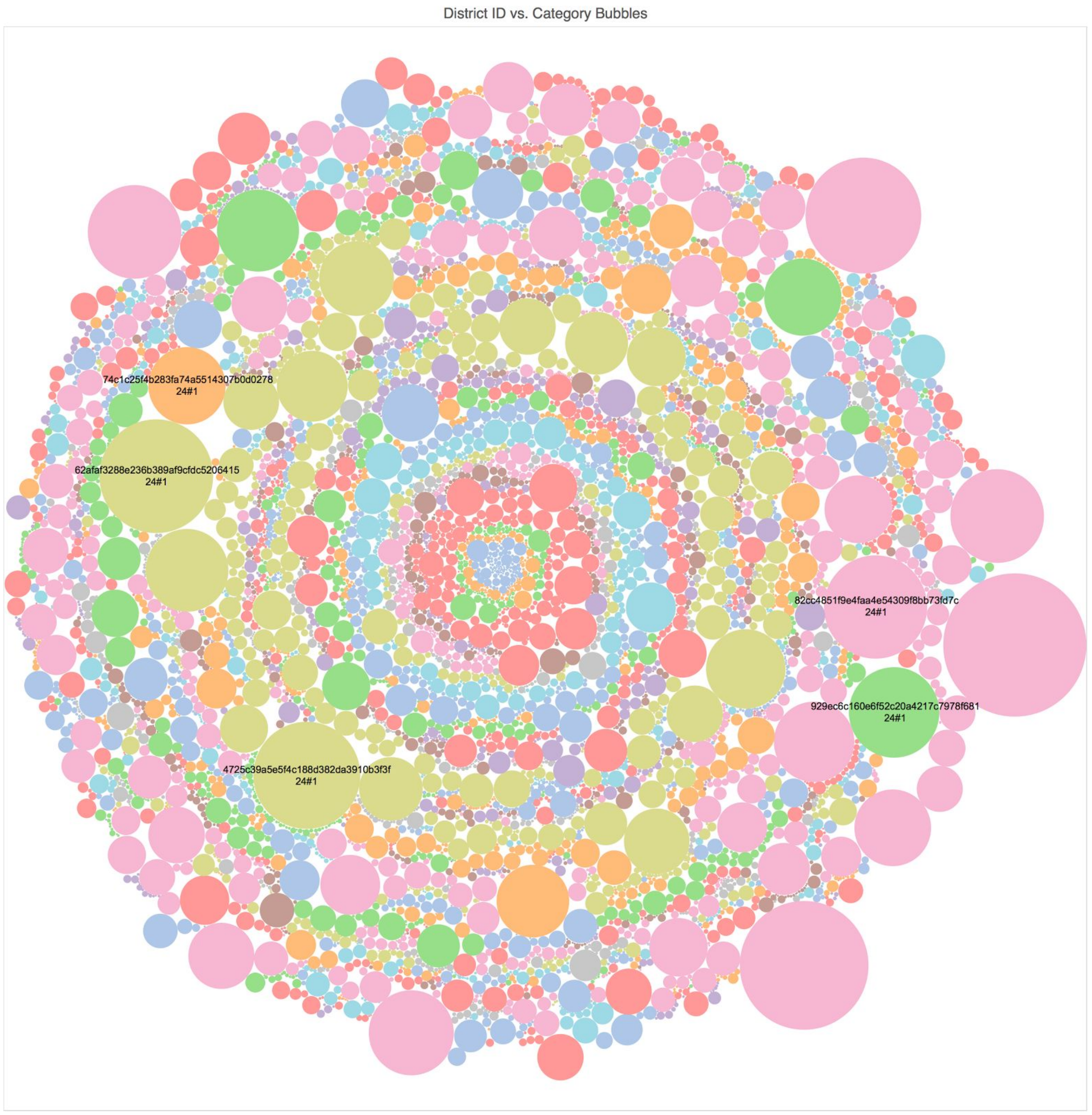
- Benchmark: SVR at 8.4
- Initial: DTR at 8.01
- Final: DTR at 7.5

As we can see, with the progression of developing the model, the MAPE score got better, so the final model is a better model than the initial or the benchmark. This model produces good result in regards to the data that's been fed into it, but it's not enough to actually solve the problem of accurately predicting the gap between supply and demand simply because the bulk of the data is not being used.

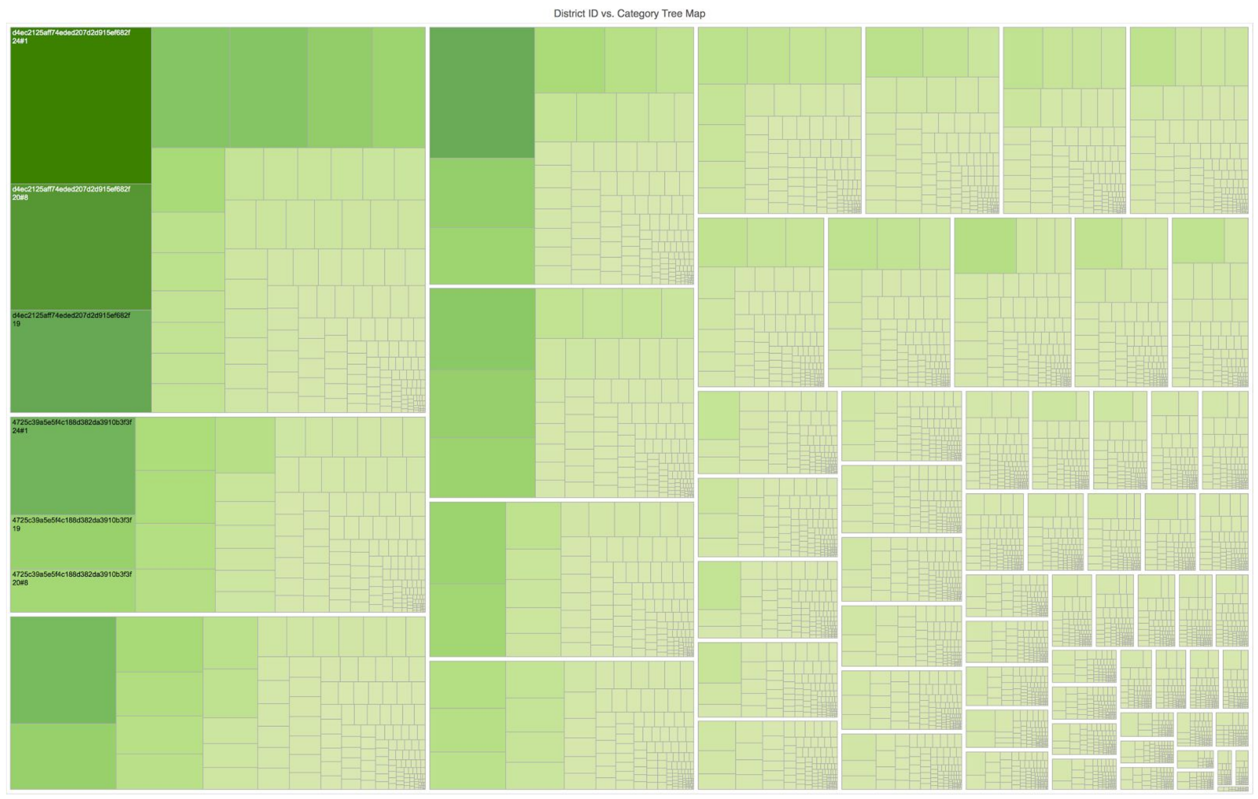
Conclusion

Free Form Visualization

Because the dataset description does not specify the details of point of interests data, many people are left wondering the relationships between the POIs, the districts, and each other. I separated the POI classifications and their number counts, and loaded the data into Tableau.



The above bubble graph is a way to look at the relationships between the districts and POIs. This is the district ID and category data plugged into the bubble graph, with the size of each bubble being the count of each POI category, and each color belongs to the same district. Here we can see much of the bigger bubbles belong to the same POI class 24#1.



This tree map is probably the best choice for getting to know the relative size or popularity of the district. This graph groups all the POI categories in each district, and ranks them by the count of POIs. Each district is in its own group, separated by the thin white lines, and each POI is its own green rectangle. Here we can see the top 2 most popular district by POIs, which corresponds to district 51 and 23. Also the most common POIs are 24#1, 19, and 20#8.

From these graphs it is easy to see that there's a definite pattern that we can use as another ingredient in our model, and that if we were able to take advantage of this then our model would do a lot better. For example the heat map tells us much of the city's population have a strong interest in just a few types of places, so if the model can distinguish that then it'll do better. This brings me to an important fact about this project, is that no matter how good the algorithm behind the model is, it will be difficult for the model to do well if there's not much data to fuel the model. Having enough data in a useable format is one of the basic requirements for an accurate model.

Reflection

Originally I was going to do the robotics project for my capstone, but when I was nearing the finish for the AI for robotics course Udacity announced their partnership with DiDi for this competition, so I switched and jumped into this. I wanted to do this competition for my capstone because 1) it is a kaggle-like competition, and it's a good experience to go along with other kaggles. 2) it is only for a limited time, and the novelty effect made it more interesting than other capstone projects. And 3) there is a prize for winners, although we were eventually found out that there's a long way to go before we can make a prize-winning machine learning product.

After I've decided to do this, I talked with other people in the MLND slack channel, and eventually got 9 other people to join my team. Unfortunately most of them were busy and preoccupied with other things so only 1 other person was contributing work towards the project.

I had heard people suggesting writing the report first then write code, so that's what I did with this project. I started writing the report and only wrote code as needed. From the beginning we followed the steps in capstone template complemented by the project rubric in the order that's listed. After i got started with writing the report, we explored the data and made visualizations, then we tried to reformat the data into a useable state, and finally built models to produce results. The problem solution can be summarized as a cross validated decision tree regressor, tuned by grid search for its max depth. Once the model is trained we used it to output a list of numbers in a file and submitted the file.

For this project I was under additional pressure because since this is done with partnership with Udacity I was able to get some sort of help from them as a MLND student, however the only way to get that is to submit my report, and I couldn't finish the report without having to come up with a good model. So I was stuck for a while because I had a mediocre model and an unfinished report and no help. But once the first round of the competition ended it was actually a relief because I was no longer worried about getting higher on the rankings, and only focused on what I was able to achieve for the project.

It was not until I was knees deep in this project that I realized that every single one of my previous projects came with just one nicely formatted data file, with every feature needed conveniently located right next to each other. This of course was a rude awakening for me, even though I have heard that from 50% to 97% of the time Data Scientists spent working is to get the data in a useable format. Having not taken the Data Analyst Nanodegree, I was no longer sheltered by the niceties of the project templates. Luckily one of my teammates was able to load all the data into MySQL and we were able to move on quickly.

Improvement

Had I known the size of the dataset beforehand I would probably invest some time looking into cloud computing and Hadoop, Spark, or some other framework that's specifically designed to work with manipulating large amounts of data efficiently. This would improve the time it takes for us to be up and running with experimenting with different models. Or some other user interface where one could simply highlight text characters and turn them into separate columns. As much of the work is on reformatting data, this kind of tools would help make machine learning much easier.