NYC Taxi demand Prediction using various prediction techniques

Capstone Project – Group 1

Alok

Anuj

Santosh

Siddhartha

Venkat

# **Table of Content**

1. **Introduction:**
2. **Exploratory Data Analysis**
3. **Data Preparation**
4. **Time Series Models**

**Holt-Winters seasonal method**

1. **Supervised Learning Models**

**Linear Regression**

**Conditional Inference Decision tree**

**Random forest**

**XG Boost**

1. **Deep Learning Model - LSTM**
2. **Results**

# **Comparison of Methods for Forecasting Taxi Demand Up to a Week Ahead**

This empirical paper compares the accuracy of six univariate methods for **short-term taxi demand** forecasting for lead times up to a week-ahead. The methods are compared using a time series of half hourly demand for New York.

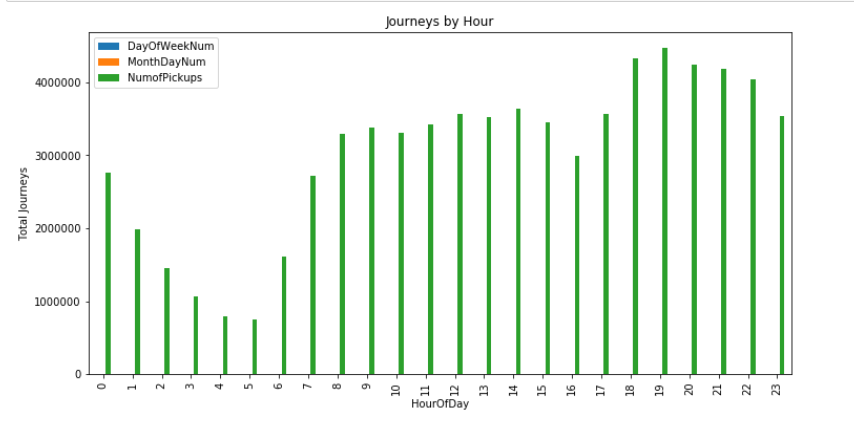
1. **Introduction:**

Taxi-supply planning requires efficient management of existing taxis and optimization of the decisions concerning additional capacity. Demand prediction is an important aspect in the development of any model for taxi planning. The form of the demand depends on the type of planning and accuracy that is required; hence it can be represented as an annual demand, a peak demand or demand like daily, weekly etc. Short-term demand forecasts are required for the control and scheduling of taxis. The focus varies from minutes to several hours ahead. The predictions can help in optimizing taxi supply at a given location and time.

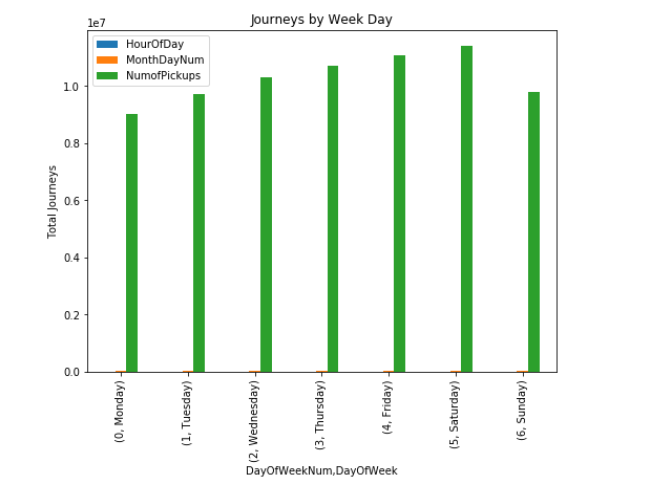
In the short run, the taxi demand is mainly influenced by seasonal effects (daily and weekly cycles, calendar holidays) and special events. Weather related variation is certainly critical in predicting taxi demand for lead times beyond a day ahead. In this paper, we compare the accuracy of simple benchmarks and four more sophisticated methods. We evaluate the methods using 24 weeks data for the New York city. We consider lead times up to a week ahead.

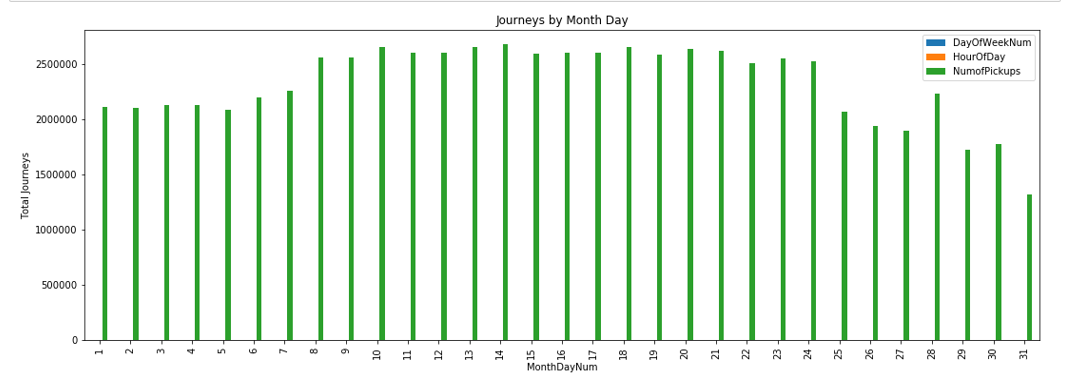
1. **Exploratory Data Analysis**

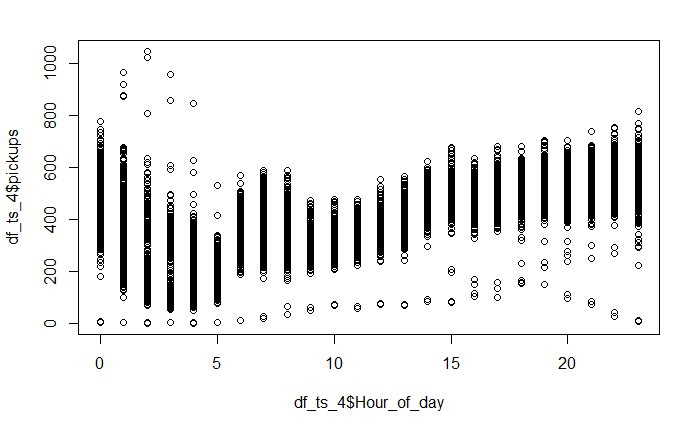
Plotting the graph for average pickups by hour of the day shows low demand in early morning hours followed by increased pickups during the day and finally maxing out in the evening.



Average pickups are highest on Saturday and lowest on Monday







1. **Data Preparation**

The raw data is available in the following format:



It has one row entry for each taxi ride ordered. As the desired input is taxi pickups every half an hour, following transformation steps are executed on data:

1. Data Imputation: Replace the missing values by average of pickups at 1 previous and 1 ahead time stamps
2. Round-up the location (latitude and longitude) to 1 decimal place
3. Group the data based on concatenated latitude and longitude. This will result in 4 datasets, one for each combination of latitude and longitude
4. Replace outliers by 5th or 95 th percentile using boxplot
5. Group the data for each time stamp in each dataset and count rows as no of pickups for that timestamp
6. Aggregate taxi pickups for each dataset on half hourly basis
7. Figure below shows the final dataset for one location



1. The dataset prepared is univariate data used for time series models and LSTM model
2. To apply Supervised learning models further data enrichment followed by feature engineering is performed
3. Figure below shows the enriched and engineered data

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 23 weeks of data has been used to train all the models and 24th week pickups have been used as test dataset. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **Time Series Models**

**Holt-Winters seasonal method**

Holt (1957) and Winters (1960) extended Holt’s method to capture seasonality.

The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the **level** ℓtℓt, one for **trend** btbt, and one for the **seasonal component** denoted by stst, with smoothing parameters αα, β∗β∗ and γγ. We use mm to denote the period of the seasonality, i.e., the number of seasons in a year.

There are two variations to this method that differ in the nature of the seasonal component. The **additive method** is preferred when the **seasonal variations are roughly constant** through the series. The **multiplicative method** is preferred when the **seasonal variations are changing proportional** to the level of the series.

* 1. To optimize alpha(level), beta(trend) and gamma(seasonality) parameters in Holt Winter model, 3 nested loops were used as below:

y = train time series

For alpha in 0.1 to 1

For beta in 0 to 1

For gamma in 0 to 1

model = HoltWinters(y, alpha,beta,gamma,start.periods = 336)

forecast = predict 336 (1 week ahead values) using computed model

result = combine forecast and test data

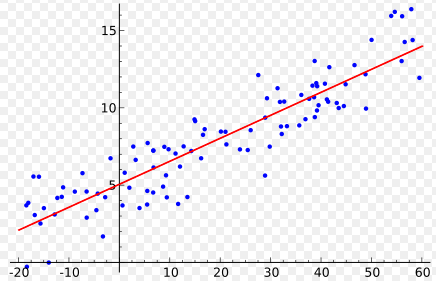
calculate MAPE

If computed MAPE < last MAPE then accept the solution

* 1. The above loop is run separately for additive and multiplicative Holt Winter model

1. **Supervised Learning Models**
2. **Linear Regression**

Linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables denoted X. The case of one explanatory variable is called simple linear regression.



Regression Analysis

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

Most commonly, the conditional mean of y given the value of X is assumed to be an affine function of X; less commonly, the median or some other quantile of the conditional distribution of y given X is expressed as a linear function of X. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of y given X, rather than on the joint probability distribution of y and X, which is the domain of multivariate analysis.

Linear regression has many practical uses with the following **two broad categories**:

* If the goal is **prediction or forecasting or error reduction**, linear regression can be used to fit a predictive model to an observed data set of y and X values. After developing such a model, if an additional value of X is then given without its accompanying value of y, the fitted model can be used to make a prediction of the value of y.
* Given a variable y and a number of variables X1, X2.., Xp that may be related to y, linear regression analysis can be applied to **quantify the strength of the relationship** between y and the Xj, to assess which Xj may have no relationship with y at all and to identify which subsets of the Xj contain redundant information about y.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression) or by minimizing a penalized version of the least squares loss function as in ridge regression (L2-norm penalty) and lasso (L1-norm penalty).

Conversely, the least squares approach can be used to fit models that are not linear models.

1. **Conditional Inference Decision tree**

Conditional inference trees estimate a regression relationship by binary recursive partitioning in a conditional inference framework. Roughly, the algorithm works as follows:

1. Test the global null hypothesis of independence between any of the input variables and the response (which may be multivariate as well). Stop if this hypothesis cannot be rejected. Otherwise select the input variable with strongest association to the response. This association is measured by a p-value corresponding to a test for the partial null hypothesis of a single input variable and the response.
2. Implement a binary split in the selected input variable.
3. Recursively repeat the above steps 1 and 2.
4. **Random Forest**

Random forest algorithm can use both for classification and the regression kind of problems.

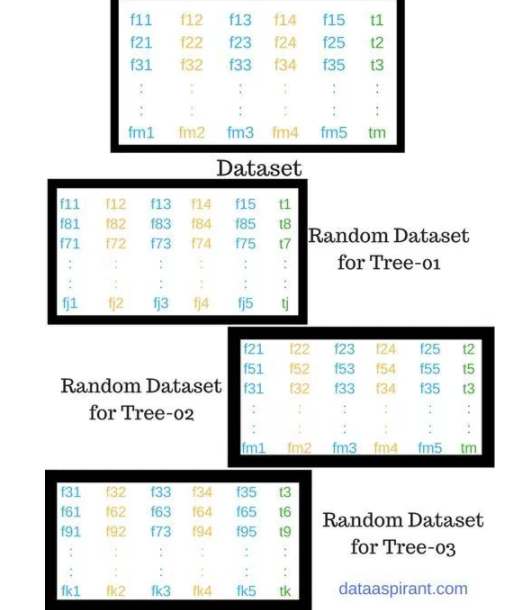
Random forest algorithm is a supervised classification algorithm. As the name suggest, this algorithm creates the forest with a number of trees.

In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

The **advantages of Random forest** algorithm are as below

* The same random forest algorithm or the random forest classifier can use for both classification and the regression task.
* Random forest classifier will handle the missing values.
* When we have more trees in the forest, random forest classifier won’t overfit the model.
* Can model the random forest classifier for categorical values also.

**How Random Forest work:**



The **pseudocode** for random forest algorithm can split into two stages.

A**. Random forest creation pseudocode**.

B. **Pseudocode to perform prediction** from the created random forest classifier.

**Random Forest pseudocode:**

1. Randomly select “k” features from total “m” features.

Where k <= m

1. Among the “k” features, calculate the node “d” using the best split point.
2. Split the node into smaller nodes using the best split.
3. Repeat 1 to 3 steps until “l” number of nodes has been reached.
4. Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

The beginning of random forest algorithm starts with randomly selecting “k” features out of total “m” features. Next, we use the randomly selected “k” features to find the root node by using the best split approach. In the next stage we calculate the child nodes using the same best split approach. The above process is repeated till the tree with a root node and target as leaf node.

The above steps are repeated to create “n” randomly created trees and thereby forms the random forest.

1. **XG Boost**

XGBoost stands for e**X**treme **G**radient **B**oosting.

XGBoost is an algorithm implementation of gradient boosted decision trees designed for speed and performance.

XGBoost is a software library that can be downloaded and installed on machine and then accessed from a variety of interfaces. Specifically, XGBoost supports the following main interfaces:

* Command Line Interface (CLI).
* C++ (the language in which the library is written).
* Python interface as well as a model in scikit-learn.
* R interface as well as a model in the caret package.
* Julia.
* Java and JVM languages like Scala and platforms like Hadoop.

**XGBoost features:**

1. **Model Features:**

The below three main forms of gradient boosting are supported:

* **Gradient Boosting** algorithm also called gradient boosting machine including the learning rate.
* **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
* **Regularized Gradient Boosting** with both L1 and L2 regularization.

### System Features:

Some of the system features provided by XGBoost are as below:

* **Parallelization** of tree construction using all of your CPU cores during training.
* **Distributed Computing** for training very large models using a cluster of machines.
* **Out-of-Core Computing** for very large datasets that don’t fit into memory.
* **Cache Optimization** of data structures and algorithm to make best use of hardware.

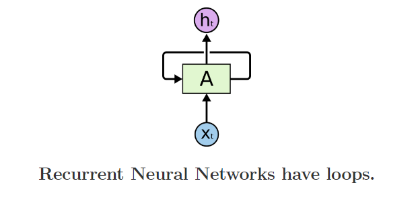
1. **Algorithm Features:**

The algorithmic features are as below:

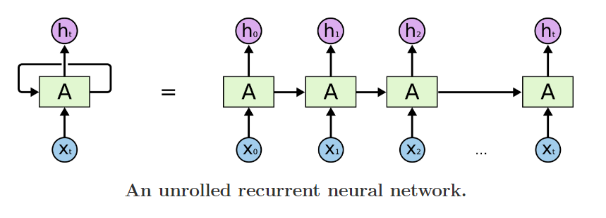
* **Sparse Aware** implementation with automatic handling of missing data values.
* **Block Structure** to support the parallelization of tree construction.
* **Continued Training** so that you can further boost an already fitted model on new data.

The **two goals** of XGBoost are:

1. Execution Speed
2. Model Performance
3. **Deep Learning Models**
4. **Long Short-Term Memory (LSTM )** – Type of RNN. Long Short-Term Memory Networks are a special kind of Recurrent Neural Networks. They were introduced by Hochreiter & Schmidhuber (1997).
5. An **Artificial Neural Network (ANN)** is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.
6. A **Recurrent Neural Network (RNN)** is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.
7. Basic RNNs are a network of neuron-like nodes, each with a directed (one-way) connection to every other node. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside the network), output nodes (yielding results), or hidden nodes (that modify the data and route from input to output).
8. Humans don’t start their thinking from scratch every second. Humans understand each word based on his/her understanding of previous words. Humans don’t throw everything away and start thinking from scratch again. Human thoughts have persistence.
9. Traditional neural networks can’t do this, and it seems like a major shortcoming. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.
10. Recurrent networks, on the other hand, take as their input not just the current input example they see, but also what they have perceived previously in time.

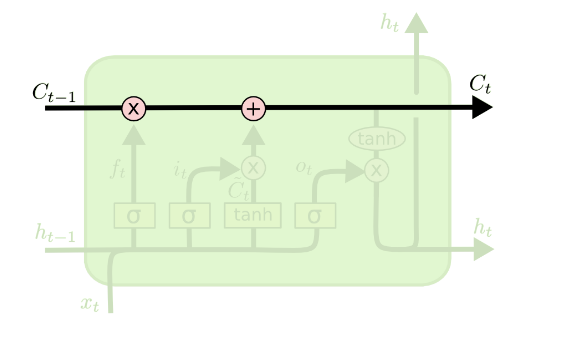


1. In the above diagram, a chunk of neural network, AA, looks at some input (xt) and outputs a value (ht). A loop allows information to be passed from one step of the network to the next.
2. These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren’t all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



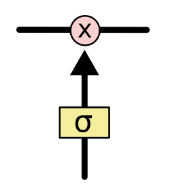
**Details about LSTM:**

* The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
* The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged

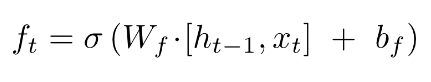


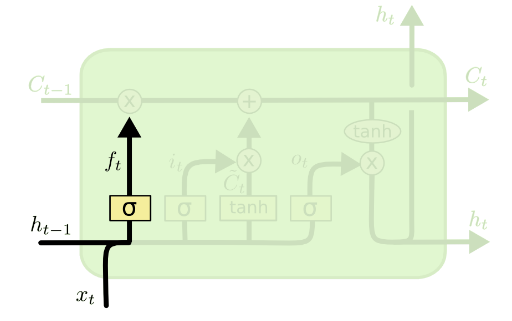
* The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

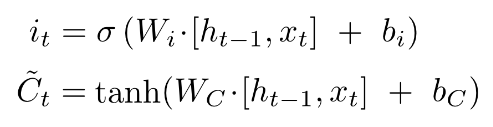


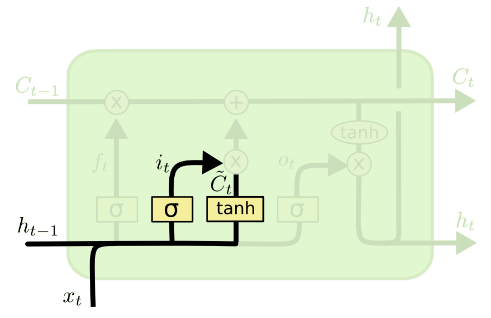
* The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”
* An LSTM has three of these gates, to protect and control the cell state.
* The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at h(t−1)and x(t), and outputs a number between 00 and 11 for each number in the cell state C(t−1) A 11 represents “completely keep this” while a 00 represents “completely get rid of this.”





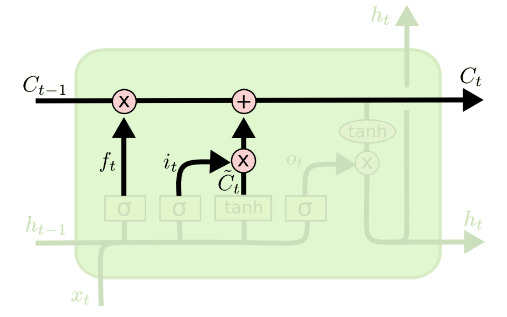
* The next step is to decide what additional information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, C~(t), that could be added to the state. In the next step, we’ll combine these two to create an update to the state.



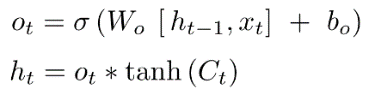


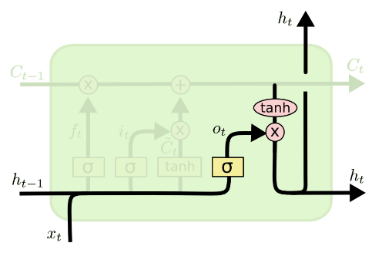
* It’s now time to update the old cell state, C(t−1), into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.
* We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it ∗C~(t). This is the new candidate values, scaled by how much we decided to update each state value.





* Finally, we need to decide what we’re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.





With this understanding on the LSTM , we have developed a model based on the data we have.

* To model LSTM data is re-arranged in the following manner:
  + Every value starting from position 337 is put into 1-D array. This is Y-variable.
  + 336 values prior to the value added to Y-variable are added to X variable
  + X variable is an array with dimensions 8046,336
  + Y variable is an array with dimensions 336,336
  + Layers used 1 input, a hidden layer with 4 LSTM blocks and an output layer that makes a single value prediction, followed by a dense layer
  + The default sigmoid activation function is used for the LSTM blocks. The network is trained for 20 epochs and a batch size of 1 is used.

1. **Graphical Results:**

