Cab Fare Prediction

Vikas Soni

SEP. 15 , 2019

CONTENTS

CHAPTER - 1

INTRODUCTION

1.1 PROBLEM STATEMENT

A cab rental start-up company have successfully run the pilot project and now want to launch your cab service across the country. They have collected the historical data from the pilot project and now want to apply analytics for fare prediction. Aim is to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data

Our data is a time series type data with given variables are:

1. Pickup_datetime
2. Pickup_latitude
3. Pickup_longitude
4. Dropoff_latittude
5. Dropoff_longitude
6. Passenger_count
7. Fare_amount

These were variables that were given to us where fare_amount is our dependent variable and to be predicted. Others are predictors .
It was told that data may contain some missing values.Total number of rows very close to 16000.

CHAPTER - 2

METHODOLOGY

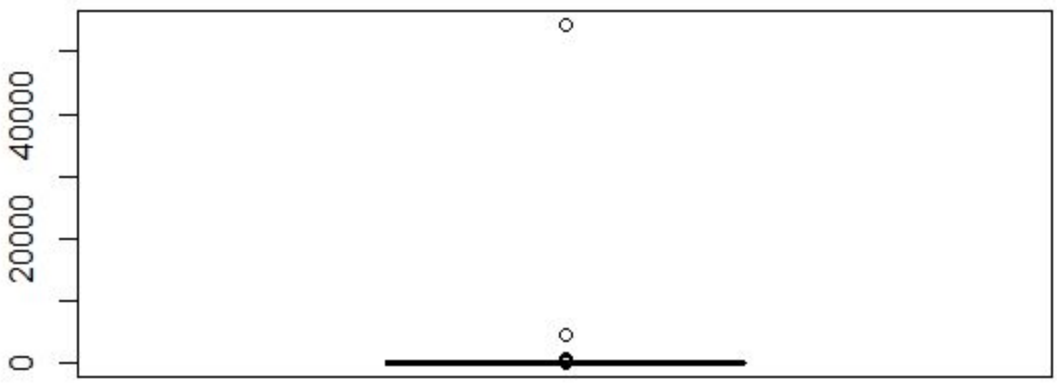
2.1 Preprocessing

Data that is collected by different means by a company is almost always messy and wrong . Apart from that , data collected is always a lot and a lot of stuff is not even useful. So, the first step that is needed to be done is clean the data and process it in a form that works well with our machine learning model. While normally the steps to clean the data are not always fixed and different methods need to be applied with different data, there are certain things that are always done and are considered as a standard in industry. We will discuss some of the techniques here and how they prove to be useful for our case. So, lets start with missing value analysis.

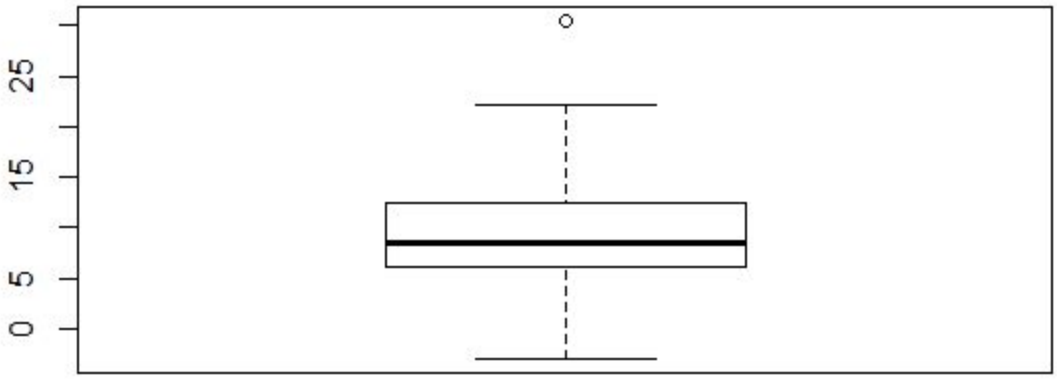
2.1.1 Outlier Analysis

There are almost always the case that outliers are found in numerical variables of a data. These outliers can affect the performance of our model and need to be removed. In our case, we had only one numerical variable and that was our fare_amount variable. To get

rid of these outliers, we used the most common boxplot method.



As we can see that data is recorded very poorly. After removing the outliers , the boxplot turned out like this.



That looks much better and our dependent variable seems to be outlier free now. That was all needed to be done for outlier analysis.

2.1.2 Missing Value Analysis

In the data provided to us, we were told that there were some missing values and that was indeed true. But when counted, there were found to be only 55 missing values in all. Now, our data has close to 16000 rows in total and missing value rows were a lot smaller than that number. So, rather than imputing them , they were simply dropped . That was all we had to do for missing values.

2.1.3 Feature selection and engineering

This was the longest step in our preprocessing for our given data. For selection , we had too few variables(7 in total) and all were important and nothing could be dropped beforehand. But there were some values which were not making any sense in our data and need to be improved before going further. Passenger_count is a variable that had the most wrongly recorded values . Now, as expected passenger_count stores the number of passengers in one ride and can only be natural numbers. But as it turns out we had a lot of floats and lots of very high number in that field and all of them were removed and when inspected it was found from the data that only possible values are 1 to 6. So, all other values had to be removed and it causes our data to shrink by some 1000 values. Also , there were fare_amount values which were zero and negative and some very unusually high and they all need to be taken care of as well. After all that cleaning, its time for some feature engineering. First thing that was done was to extract year, month, date, weekday and hour from our pickup_datetime. After doing that , now we had 12 variables out of which datetime was now useless and hence removed. Other than this, we have been given latitudes and longitudes of the pickup and dropoff coordinates .While these are important in themselves as they us at what part of town a cab is taken and fare varies with it, there can also be an important feature derived for them. We can easily find the distance and that no doubt will have a linear relationship with our dependent variable and can be very useful. For finding distance from latitude and longitude , haversine distance is used and stored in our dataset. Thats it for feature selction and engineering. There was other sort of inspection that was done with our data and explained more in appendix.

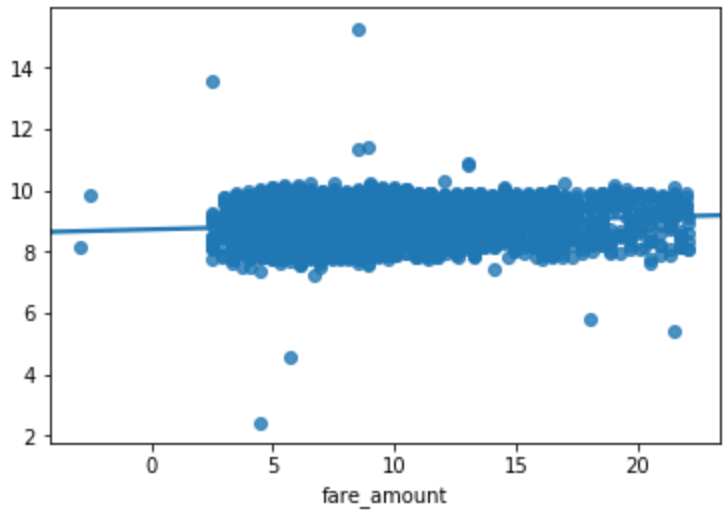
2.2 Modelling

2.2.1 Model Selection

This is a basic regression problem as we have to predict the fare_amount over a given interval of time which is a number. As we saw in the early EDA stage that our dependent variable has linear relationship with distance, we can make an inference that linear regression might be a model that can work well. So, there is no harm in trying linear regression of different kinds like ridge and lasso for the problem . Although, it is normally the case that when we have a lot of categorical variables as is in our case, decision tree based methods tends to work better than normal regression methods.So, random forest might be a good choice here. Apart from that , standard gradient boosting is a very powerful model for regression problems and we will give it a shot here.

2.2.2 Multiple Linear Regression

Linear regression is a simple mathematical method which will try to fit a straight line to a given set of points so that it has least distance from all the points.When this method is applied to our problem, graph comes out like this.



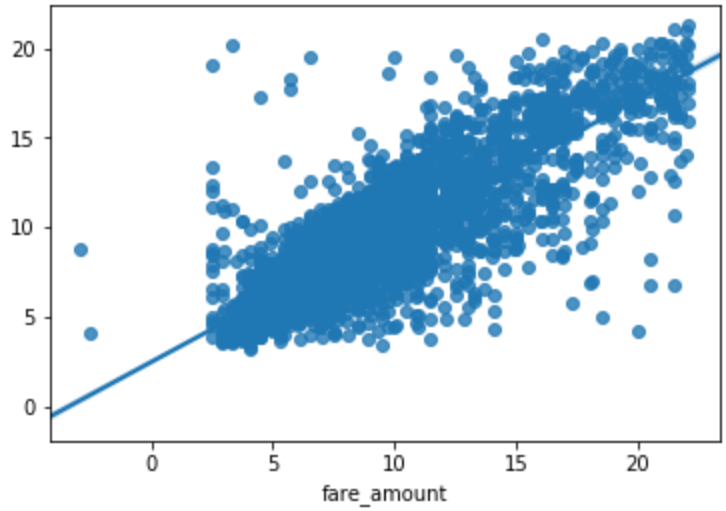
As we can see that this line is not giving a good prediction as some points are a bit far away. This is usually the case and reason behind it is that one straight line can not be drawn to get good results and we can rather use a curve instead of a line. Whatever be the case, the value for the MAE and RMSE we got in this case were not good . So, it turns out that this model can not be used. Furthermore , there are some sophisticated versions of linear regression which are ridge and lasso regression. As checked , these methods were not found to be useful as well. As the accuracy was not found to be good from our metrics , this method was dropped. It can also be noted that this data is right skewed and such type of data are seen to be fitting not so nicely with multiple regression models like linear or logistic regression.

2.2.3 Random Forest Regressor

This method is based on decision trees. So, it tends to work better when there are a lot of categorical values which is our case. This method is applied and it is found out that however there is a significant improvement over linear regression but a bit worse than gradient boosting.

2.2.4 Gradient Boosting

Standard Gradient Boosting(SGD) is a method which is more complex than the linear regression and there is a good implementation of it in python scikit learn library. This method almost always gives better results in any regression problem than any other algorithm. So, it was applied to fit our model and it turns out that it works a lot better than our linear regression and accuracy is increased by a significant amount.



As we can see from the graph that it works a lot better than linear regression but this graph only can not give as clear a representation of how much the accuracy is improved and in next section error metrics are explained which will provide us with a better insight for the same.

CHAPTER -3

CONCLUSION

3.1 Model Evaluation

As we have tried different models for our problems, it is beneficial to have a mean to tell us which of the model is working better. It is a tricky thing to find the best mathematical formula or quantity that will tell us how our model will do in real world or even on our test set. But luckily there are some standard error metrics that are generally used for a type of problem. Some of the error metrics that are applied to our model and why they were selected were discussed below.

3.1.1 MAE

It is most common metrics for machine learning algorithms. It is just mean of our absolute errors between actual values and predicted values.
Values that we got after applying this metrics to our algorithms are given as:

Linear regression : 3.2804593601434506
Random Forest : 1.60375172223751
Gradient Boosting : 1.4878742834017575

3.1.2 Mean Squared Error

Mean squared error is the mean of squares of errors of all the predictions.

Linear regression : 17.53399098031526
Random Forest : 5.348069302011574

Gradient Boosting : 4.660292053020569

As we can see that gradient boosting is working better in both those cases.

3.1.3 R2_score

R2_score is another important metric that is used to check the performance of our models. First, lets see what values did we get:

Linear regression : 0.016199147683285564
Random Forest : 0.6999294032103327
Gradient Boosting : 0.7385193536968494

3.1.1 Choosing the Right Metric

Now, we can talk a little about how one of those metrics that are seen above is chosen as the effective way of our mode. So , lets start:

1) First and most common metrics is our Mae which is absolute error of our model . This model is helpful as it provides us with absolute value of error which can give a good insight of model right away and is very simple to evaluate. It does not provide us with different weights for different errors if required.This is better when there are outliers
2) Second is RMSE.
   Rmse is root mean square error, One good point about squared metrics is that they have their L2 space which is a good thing at many places(not here tho). Other than that it provides us with high penalty on high errors which we need in this case.It is useful if we have unexpected values that we should care about. Very high or low value that we should pay attention.
3) MAPE
   Mape is mean absolute percentage error. It is widely used in regression models as it give us the percentage and a quick sense of how much bad our model is. But it is not usually a good metric for time series data which we are dealing with. So, we will not be using it.
4) R2
   R2 model is a good way to quantify whether it is a good model or not. By definition,

In conclusion, R^2 is the ratio between how good our model is vs how good is the naive mean model.

Judging from all of these things, we can infer that we can use MAE or RMSE or even R2 and either of them will be fine. But as rmse is better in some way than MAE, we are inclined to use it to better penalize the model for high error. R2 can be a good metric too but as rmse is simpler to interpret , we are using this in our model.

3.2 Model Selection

Now, let us try and find out which model worked best. As seen from all the models that we trained, in each case gradient boosting works the best beating random forest only by a little. Whereas, linear regression model accuracy was not that good . From this knowledge, we can safely say that gradient boosting model can be chosen as best model and can be deployed safely.

APPENDIX -A

A.1 R CODE

```
getwd()
setwd("C:/Users/Vikas/Desktop/Assignments/Project - 2")
data = read.csv('train_cab.csv')
library("dplyr")
## So, data is simple with attributes like pickup and drop coordinates and fare amount and number of people.Pickup time and no. of people travelling.
str(data)
data <- data[-c(1120:1130),]
data$fare_amount
str(data)
data$fare_amount = as.numeric(as.character(data$fare_amount))
data$fare_amount
summary(data)
library(ggplot2)
boxplot(data$fare_amount)

boxplot(data$passenger_count)
x <- data$fare_amount
qnt <- quantile(x, probs=c(.25, .75), na.rm = T)
caps <- quantile(x, probs=c(.05, .95), na.rm = T)
H <- 1.5 * IQR(x, na.rm = T)
x[x < (qnt[1] - H)] <- caps[1]
x[x > (qnt[2] + H)] <- caps[2]
data$fare_amount <- x
boxplot(data$fare_amount)

data$pickup_datetime[1310:1320]
data <- data[-c(1310: 1329),]
data$pickup_datetime[1320:1330]
data$pickup_datetime = as.POSIXct(data$pickup_datetime)

data1  <- data %>%
  mutate(pickup_datetime = as.POSIXct(pickup_datetime)) %>%
  mutate(hour = as.numeric(format(pickup_datetime, "%H"))) %>%
  mutate(min = as.numeric(format(pickup_datetime, "%M"))) %>%
  mutate(year = as.factor(format(pickup_datetime, "%Y"))) %>%
  mutate(day = as.factor(format(pickup_datetime, "%d"))) %>%
  mutate(month = as.factor(format(pickup_datetime, "%m"))) %>%
  mutate(Wday = as.factor(weekdays(pickup_datetime))) %>%
  mutate(hour_class = as.factor(ifelse(hour < 7, "Overnight",
                                ifelse(hour < 11, "Morning",
                                     ifelse(hour < 16, "Noon",
                                          ifelse(hour < 20, "Evening",
                                               ifelse(hour < 23, "night", "overnight") ) ))))) %>%
```

```
  filter(fare_amount > 0 & fare_amount <= 500) %>%
  filter(pickup_longitude > -80 && pickup_longitude < -70) %>%
  filter(pickup_latitude > 35 && pickup_latitude < 45) %>%
  filter(dropoff_longitude > -80 && dropoff_longitude < -70) %>%
  filter(dropoff_latitude > 35 && dropoff_latitude < 45) %>%
  filter(passenger_count > 0 && passenger_count < 10) %>%
  mutate(pickup_latitude = (pickup_latitude * pi)/180) %>%
  mutate(dropoff_latitude = (dropoff_latitude * pi)/180) %>%
  mutate(dropoff_longitude = (dropoff_longitude * pi)/180) %>%
  mutate(pickup_longitude = (pickup_longitude * pi)/180 ) %>%
  mutate(dropoff_longitude = ifelse(is.na(dropoff_longitude) == TRUE, 0,dropoff_longitude)) %>%
  mutate(pickup_longitude = ifelse(is.na(pickup_longitude) == TRUE, 0,pickup_longitude)) %>%
  mutate(pickup_latitude = ifelse(is.na(pickup_latitude) == TRUE, 0,pickup_latitude)) %>%
  mutate(dropoff_latitude = ifelse(is.na(dropoff_latitude) == TRUE, 0,dropoff_latitude)) %>%
  select(-pickup_datetime,-hour_class,-min)
summary(data1)
summary(data)
str(data1)


##getting rid of missing values

row.has.na <- apply(data1, 1, function(x){any(is.na(x))})
sum(row.has.na)
data.filtered <- data1[!row.has.na,]
unique(data$passenger_count)

## Passenger values are very poorly recorded. So, we can just try and improve it a bit and keep only values which are int and less than 6

data.filtered <- data.filtered %>% filter(passenger_count < 7)
data4 <- data.filtered %>% filter( passenger_count == 1 | passenger_count == 2|passenger_count ==3 |passenger_count ==4 | passenger_count == 5 | passenger_count ==6)
str(data4)

dlat <- data4$dropoff_latitude - data4$pickup_latitude
dlon <- data4$dropoff_longitude - data4$pickup_longitude
R_earth <- 6371
#Compute haversine distance
hav = sin(dlat/2.0)**2 + cos(data4$pickup_latitude) * cos(data4$dropoff_latitude) * sin(dlon/2.0)**2
data4$haversine <- 2 * R_earth * asin(sqrt(hav))
str(data4)

## Now, its time to strasample <- sample.int(n = nrow(data3), size = floor(.75*nrow(data)), replace = F)

## we can see that this is a regression problem and multiple regression, Random forest regressor and sgd are some of the models that can be applied.
```

```
set.seed(101)# Set Seed so that same sample can be reproduced in future also
sample <- sample.int(n = nrow(data4), size = floor(.75*nrow(data)), replace = F)
train <- data4[sample, ]
test  <- data4[-sample, ]
# Now Selecting 75% of data as sample from total 'n' rows of the data


library('e1071')
regressor = lm(formula = fare_amount ~ . , data = data4)
regressor = step(regressor)
y_pred = predict(regressor, test)

## Now, trying SVR

#regressor2 = svm(formula = fare_amount ~ .,
 #             data = train,
  #            type = 'eps-regression')
#y_pred2 = predict(regressor2, test)


## Now, moving to random forest regressor
library('randomForest')

regressor3 = randomForest(x = train[,-which(names(train)=="fare_amount")],
                          y = train$fare_amount)
y_pred3 = predict(regressor3, test)
# Error metrics calculations


library('Metrics')
MAE = mean(abs(y_pred- test$fare_amount))
#MAE2 = mean(abs(y_pred2- test$fare_amount))
MAE3 = mean(abs(y_pred3 - test$fare_amount))

## WE can aslo use metrics like rmse and others to get a better estimate of our data

RMSE = rmse(test$fare_amount, y_pred)
#RMSE2 = rmse(test$fare_amount, y_pred2)
RMSE3 = rmse(test$fare_amount, y_pred3)

## So, we can see that ensemble techniques gives us better results .



## Now, we can use various models to test which work best on our real test set.

test.act = read.csv('test.csv')
test1.act  <- test.act%>%
```

```
 mutate(pickup_datetime = as.POSIXct(pickup_datetime)) %>%
 mutate(hour = as.numeric(format(pickup_datetime, "%H"))) %>%
 mutate(min = as.numeric(format(pickup_datetime, "%M"))) %>%
 mutate(year = as.factor(format(pickup_datetime, "%Y"))) %>%
 mutate(day = as.factor(format(pickup_datetime, "%d"))) %>%
 mutate(month = as.factor(format(pickup_datetime, "%m"))) %>%
 mutate(Wday = as.factor(weekdays(pickup_datetime))) %>%
 mutate(hour_class = as.factor(ifelse(hour < 7, "Overnight",
                               ifelse(hour < 11, "Morning",
                                   ifelse(hour < 16, "Noon",
                                       ifelse(hour < 20, "Evening",
                                           ifelse(hour < 23, "night", "overnight") ) ))))) %>%

 filter(pickup_longitude > -80 && pickup_longitude < -70) %>%
 filter(pickup_latitude > 35 && pickup_latitude < 45) %>%
 filter(dropoff_longitude > -80 && dropoff_longitude < -70) %>%
 filter(dropoff_latitude > 35 && dropoff_latitude < 45) %>%
 filter(passenger_count > 0 && passenger_count < 10) %>%
 mutate(pickup_latitude = (pickup_latitude * pi)/180) %>%
 mutate(dropoff_latitude = (dropoff_latitude * pi)/180) %>%
 mutate(dropoff_longitude = (dropoff_longitude * pi)/180) %>%
 mutate(pickup_longitude = (pickup_longitude * pi)/180 ) %>%
 mutate(dropoff_longitude = ifelse(is.na(dropoff_longitude) == TRUE, 0,dropoff_longitude)) %>%
 mutate(pickup_longitude = ifelse(is.na(pickup_longitude) == TRUE, 0,pickup_longitude)) %>%
 mutate(pickup_latitude = ifelse(is.na(pickup_latitude) == TRUE, 0,pickup_latitude)) %>%
 mutate(dropoff_latitude = ifelse(is.na(dropoff_latitude) == TRUE, 0,dropoff_latitude)) %>%
 select(-pickup_datetime,-hour_class,-min)
dlat <- test1.act$dropoff_latitude - test1.act$pickup_latitude
dlon <- test1.act$dropoff_longitude - test1.act$pickup_longitude
R_earth <- 6371
#Compute haversine distance
hav = sin(dlat/2.0)**2 + cos(test1.act$pickup_latitude) * cos(test1.act$dropoff_latitude) *
sin(dlon/2.0)**2
test1.act$haversine <- 2 * R_earth * asin(sqrt(hav))
str(test1.act)
y_pred_final = predict(regressor3, test1.act)
str(data4)
str(y_pred_final)
test1.act$fare_prediction = y_pred_final
str(test1.act)
write.csv(test1.act, file = 'cabfare_RF.csv', row.names = FALSE, quote = FALSE)

## data fare amount values

data$fare_amount
data4$fare_amount
```

A.2 Additional Information About Model

When preprocessing our model , we had to take care of some other details which were specific for the problems.

First thing was the longitude and latitude columns where data was recorded poorly and it was not very straightforward to remove false samples of data as you can not simply look at data and find then. First of all , the latitudes should be between -90 to 90 and longitude must be between -180 to 180. So, all other values are removed. Other than that , recordings are bound to be from one area and that area will not have all the range of latitude and longitudes. So, the values which were found most common were kept and others are removed. While, some other time when looking up our new distance column it was found that distances were huge in some locations . This found out to be the cause of wrong latitudes and longitudes again and hence, distances are made to be kept in a range (between 0 to 500 in our case.

APPENDIX - B

B.1 Steps to run the file in DOS

A) First is our ipynb file. Steps to run it in dos is as given:

1) Download the given file as .py file.
2) Then, run the file by giving following command in DOS. Python3 project-2 (revised).py
3) Or if file is downloaded as ipynb file. We can start jupyter notebook from DOS and then run it.

B) For r code, steps are :

1) Download the file.
2) Start Command line.
3) Find the path to R.exe or Rscript.exe on your computer. If you try to run R.exe from the command line, you enter an R terminal.
4) Find the path to R file.
5) Open Notepad and combine paths together (with quotation marks if needed and additional commands “CMD BATCH” if you choose to go with R.exe).
6) Open Notepad and combine paths together (with quotation marks if needed and additional commands “CMD BATCH” if you choose to go with R.exe).
7) Run that batch file to execute R script.

B.2 Deployment of Model

There are many ways when it comes to deploy our model . We will discuss here one way to do so.

1. Train the model using Jupyter notebook(which is already done)
2. Save the trained model object as a pickle file (serialization)
3. Create a flask environment that will have an API endpoint which would encapsulate our trained model and enable it to receive inputs (features) through GET requests over HTTP/HTTPS and then return the output after de-serializing the earlier serialized model
4. Upload the flask script along with the trained model on pythonanywhere
5. Make requests to the hosted flask script through a website, bot, android app or any other application capable of sending HTTP/HTTPS requests