Complete R-Code

|  |
| --- |
|  |
|  | # Cab Fare Prediction |
|  |  |
|  | rm(list = ls()) |
|  | setwd("C:/Users/admin/Documents/R files") |
|  | # #loading Libraries |
|  | x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071", |
|  | "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats') |
|  | #load Packages |
|  | lapply(x, require, character.only = TRUE) |
|  | rm(x) |
|  |  |
|  | # The details of data attributes in the dataset are as follows: |
|  | # pickup\_datetime - timestamp value indicating when the cab ride started. |
|  | # pickup\_longitude - float for longitude coordinate of where the cab ride started. |
|  | # pickup\_latitude - float for latitude coordinate of where the cab ride started. |
|  | # dropoff\_longitude - float for longitude coordinate of where the cab ride ended. |
|  | # dropoff\_latitude - float for latitude coordinate of where the cab ride ended. |
|  | # passenger\_count - an integer indicating the number of passengers in the cab ride. |
|  |  |
|  |  |
|  | # loading datasets |
|  | train = read.csv("train\_cab.csv", header = T, na.strings = c(" ", "", "NA")) |
|  | test = read.csv("test.csv") |
|  | test\_pickup\_datetime = test["pickup\_datetime"] |
|  | # Structure of data |
|  | str(train) |
|  | str(test) |
|  | summary(train) |
|  | summary(test) |
|  | head(train,5) |
|  | head(test,5) |
|  |  |
|  | ######## Exploratory Data Analysis ####################### |
|  | # Changing the data types of variables |
|  | train$fare\_amount = as.numeric(as.character(train$fare\_amount)) |
|  | train$passenger\_count=round(train$passenger\_count) |
|  |  |
|  | ### Removing values which are not within desired range(outlier) depending upon basic understanding of dataset. |
|  |  |
|  | # 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields. |
|  | train[which(train$fare\_amount < 1 ),] |
|  | nrow(train[which(train$fare\_amount < 1 ),]) |
|  | train = train[-which(train$fare\_amount < 1 ),] |
|  |  |
|  | #2.Passenger\_count variable |
|  | for (i in seq(4,11,by=1)){ |
|  | print(paste('passenger\_count above ' ,i,nrow(train[which(train$passenger\_count > i ),]))) |
|  | } |
|  | # so 20 observations of passenger\_count is consistenly above from 6,7,8,9,10 passenger\_counts, let's check them. |
|  | train[which(train$passenger\_count > 6 ),] |
|  | # Also we need to see if there are any passenger\_count==0 |
|  | train[which(train$passenger\_count <1 ),] |
|  | nrow(train[which(train$passenger\_count <1 ),]) |
|  | # We will remove these 58 observations and 20 observation which are above 6 value because a cab cannot hold these number of passengers. |
|  | train = train[-which(train$passenger\_count < 1 ),] |
|  | train = train[-which(train$passenger\_count > 6),] |
|  | # 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not satisfy these ranges |
|  | print(paste('pickup\_longitude above 180=',nrow(train[which(train$pickup\_longitude >180 ),]))) |
|  | print(paste('pickup\_longitude above -180=',nrow(train[which(train$pickup\_longitude < -180 ),]))) |
|  | print(paste('pickup\_latitude above 90=',nrow(train[which(train$pickup\_latitude > 90 ),]))) |
|  | print(paste('pickup\_latitude above -90=',nrow(train[which(train$pickup\_latitude < -90 ),]))) |
|  | print(paste('dropoff\_longitude above 180=',nrow(train[which(train$dropoff\_longitude > 180 ),]))) |
|  | print(paste('dropoff\_longitude above -180=',nrow(train[which(train$dropoff\_longitude < -180 ),]))) |
|  | print(paste('dropoff\_latitude above -90=',nrow(train[which(train$dropoff\_latitude < -90 ),]))) |
|  | print(paste('dropoff\_latitude above 90=',nrow(train[which(train$dropoff\_latitude > 90 ),]))) |
|  | # There's only one outlier which is in variable pickup\_latitude.So we will remove it with nan. |
|  | # Also we will see if there are any values equal to 0. |
|  | nrow(train[which(train$pickup\_longitude == 0 ),]) |
|  | nrow(train[which(train$pickup\_latitude == 0 ),]) |
|  | nrow(train[which(train$dropoff\_longitude == 0 ),]) |
|  | nrow(train[which(train$pickup\_latitude == 0 ),]) |
|  | # there are values which are equal to 0. we will remove them. |
|  | train = train[-which(train$pickup\_latitude > 90),] |
|  | train = train[-which(train$pickup\_longitude == 0),] |
|  | train = train[-which(train$dropoff\_longitude == 0),] |
|  |  |
|  | # Make a copy |
|  | df=train |
|  | # train=df |
|  |  |
|  | ####### Missing Value Analysis ####### |
|  | missing\_val = data.frame(apply(train,2,function(x){sum(is.na(x))})) |
|  | missing\_val$Columns = row.names(missing\_val) |
|  | names(missing\_val)[1] = "Missing\_percentage" |
|  | missing\_val$Missing\_percentage = (missing\_val$Missing\_percentage/nrow(train)) \* 100 |
|  | missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),] |
|  | row.names(missing\_val) = NULL |
|  | missing\_val = missing\_val[,c(2,1)] |
|  | missing\_val |
|  |  |
|  | unique(train$passenger\_count) |
|  | unique(test$passenger\_count) |
|  | train[,'passenger\_count'] = factor(train[,'passenger\_count'], labels=(1:6)) |
|  | test[,'passenger\_count'] = factor(test[,'passenger\_count'], labels=(1:6)) |
|  | # 1.For Passenger\_count: |
|  | # Actual value = 1 |
|  | # Mode = 1 |
|  | # KNN = 1 |
|  | train$passenger\_count[1000] |
|  | train$passenger\_count[1000] = NA |
|  | getmode <- function(v) { |
|  | uniqv <- unique(v) |
|  | uniqv[which.max(tabulate(match(v, uniqv)))] |
|  | } |
|  |  |
|  | # Mode Method |
|  | getmode(train$passenger\_count) |
|  | # We can't use mode method because data will be more biased towards passenger\_count=1 |
|  |  |
|  | # 2.For fare\_amount: |
|  | # Actual value = 18.1, |
|  | # Mean = 15.117, |
|  | # Median = 8.5, |
|  | # KNN = 18.28 |
|  | sapply(train, sd, na.rm = TRUE) |
|  | # fare\_amount pickup\_datetime pickup\_longitude |
|  | # 435.968236 4635.700531 2.659050 |
|  | # pickup\_latitude dropoff\_longitude dropoff\_latitude |
|  | # 2.613305 2.710835 2.632400 |
|  | # passenger\_count |
|  | # 1.266104 |
|  | train$fare\_amount[1000] |
|  | train$fare\_amount[1000]= NA |
|  |  |
|  | # Mean Method |
|  | mean(train$fare\_amount, na.rm = T) |
|  |  |
|  | #Median Method |
|  | median(train$fare\_amount, na.rm = T) |
|  |  |
|  | # kNN Imputation |
|  | train = knnImputation(train, k = 181) |
|  | train$fare\_amount[1000] |
|  | train$passenger\_count[1000] |
|  | sapply(train, sd, na.rm = TRUE) |
|  | # fare\_amount pickup\_datetime pickup\_longitude |
|  | # 435.661952 4635.700531 2.659050 |
|  | # pickup\_latitude dropoff\_longitude dropoff\_latitude |
|  | # 2.613305 2.710835 2.632400 |
|  | # passenger\_count |
|  | # 1.263859 |
|  | sum(is.na(train)) |
|  | str(train) |
|  | summary(train) |
|  |  |
|  | df1=train |
|  | # train=df1 |
|  | ####### Outlier Analysis ####### |
|  |  |
|  | # We Will do Outlier Analysis only on Fare\_amount just for now and we will do outlier analysis after feature engineering latitudes and longitudes. |
|  | # Boxplot for fare\_amount |
|  | pl1 = ggplot(train,aes(x = factor(passenger\_count),y = fare\_amount)) |
|  | pl1 + geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100) |
|  |  |
|  | # Replace all outliers with NA and impute |
|  | vals = train[,"fare\_amount"] %in% boxplot.stats(train[,"fare\_amount"])$out |
|  | train[which(vals),"fare\_amount"] = NA |
|  |  |
|  | #lets check the NA's |
|  | sum(is.na(train$fare\_amount)) |
|  |  |
|  | #Imputing with KNN |
|  | train = knnImputation(train,k=3) |
|  |  |
|  | # lets check the missing values |
|  | sum(is.na(train$fare\_amount)) |
|  | str(train) |
|  |  |
|  | df2=train |
|  | # train=df2 |
|  | ####### Feature Engineering ####### |
|  | # 1.Feature Engineering for timestamp variable |
|  | # we will derive new features from pickup\_datetime variable |
|  | # new features will be year,month,day\_of\_week,hour |
|  | #Convert pickup\_datetime from factor to date time |
|  | train$pickup\_date = as.Date(as.character(train$pickup\_datetime)) |
|  | train$pickup\_weekday = as.factor(format(train$pickup\_date,"%u"))# Monday = 1 |
|  | train$pickup\_mnth = as.factor(format(train$pickup\_date,"%m")) |
|  | train$pickup\_yr = as.factor(format(train$pickup\_date,"%Y")) |
|  | pickup\_time = strptime(train$pickup\_datetime,"%Y-%m-%d %H:%M:%S") |
|  | train$pickup\_hour = as.factor(format(pickup\_time,"%H")) |
|  |  |
|  | #Add same features to test set |
|  | test$pickup\_date = as.Date(as.character(test$pickup\_datetime)) |
|  | test$pickup\_weekday = as.factor(format(test$pickup\_date,"%u"))# Monday = 1 |
|  | test$pickup\_mnth = as.factor(format(test$pickup\_date,"%m")) |
|  | test$pickup\_yr = as.factor(format(test$pickup\_date,"%Y")) |
|  | pickup\_time = strptime(test$pickup\_datetime,"%Y-%m-%d %H:%M:%S") |
|  | test$pickup\_hour = as.factor(format(pickup\_time,"%H")) |
|  |  |
|  | sum(is.na(train))# there was 1 'na' in pickup\_datetime which created na's in above feature engineered variables. |
|  | train = na.omit(train) # we will remove that 1 row of na's |
|  |  |
|  | train = subset(train,select = -c(pickup\_datetime,pickup\_date)) |
|  | test = subset(test,select = -c(pickup\_datetime,pickup\_date)) |
|  | # Now we will use month,weekday,hour to derive new features like sessions in a day,seasons in a year,week:weekend/weekday |
|  | # f = function(x){ |
|  | # if ((x >=5)& (x <= 11)){ |
|  | # return ('morning') |
|  | # } |
|  | # if ((x >=12) & (x <= 16)){ |
|  | # return ('afternoon') |
|  | # } |
|  | # if ((x >=17) & (x <= 20)){ |
|  | # return ('evening') |
|  | # } |
|  | # if ((x >=21) & (x <= 23)){ |
|  | # return ('night (PM)') |
|  | # } |
|  | # if ((x >=0) & (x <= 4)){ |
|  | # return ('night (AM)') |
|  | # } |
|  | # } |
|  | # 2.Calculate the distance travelled using longitude and latitude |
|  | deg\_to\_rad = function(deg){ |
|  | (deg \* pi) / 180 |
|  | } |
|  | haversine = function(long1,lat1,long2,lat2){ |
|  | #long1rad = deg\_to\_rad(long1) |
|  | phi1 = deg\_to\_rad(lat1) |
|  | #long2rad = deg\_to\_rad(long2) |
|  | phi2 = deg\_to\_rad(lat2) |
|  | delphi = deg\_to\_rad(lat2 - lat1) |
|  | dellamda = deg\_to\_rad(long2 - long1) |
|  |  |
|  | a = sin(delphi/2) \* sin(delphi/2) + cos(phi1) \* cos(phi2) \* |
|  | sin(dellamda/2) \* sin(dellamda/2) |
|  |  |
|  | c = 2 \* atan2(sqrt(a),sqrt(1-a)) |
|  | R = 6371e3 |
|  | R \* c / 1000 #1000 is used to convert to meters |
|  | } |
|  | # Using haversine formula to calculate distance fr both train and test |
|  | train$dist = haversine(train$pickup\_longitude,train$pickup\_latitude,train$dropoff\_longitude,train$dropoff\_latitude) |
|  | test$dist = haversine(test$pickup\_longitude,test$pickup\_latitude,test$dropoff\_longitude,test$dropoff\_latitude) |
|  |  |
|  | # We will remove the variables which were used to feature engineer new variables |
|  | train = subset(train,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude)) |
|  | test = subset(test,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude)) |
|  |  |
|  | str(train) |
|  | summary(train) |
|  |  |
|  | ####### Feature selection ######### |
|  | numeric\_index = sapply(train,is.numeric) #selecting only numeric |
|  |  |
|  | numeric\_data = train[,numeric\_index] |
|  |  |
|  | cnames = colnames(numeric\_data) |
|  | #Correlation analysis for numeric variables |
|  | corrgram(train[,numeric\_index],upper.panel=panel.pie, main = "Correlation Plot") |
|  |  |
|  | #ANOVA for categorical variables with target numeric variable |
|  |  |
|  | #aov\_results = aov(fare\_amount ~ passenger\_count \* pickup\_hour \* pickup\_weekday,data = train) |
|  | aov\_results = aov(fare\_amount ~ passenger\_count + pickup\_hour + pickup\_weekday + pickup\_mnth + pickup\_yr,data = train) |
|  |  |
|  | summary(aov\_results) |
|  |  |
|  | # pickup\_weekdat has p value greater than 0.05 |
|  | train = subset(train,select=-pickup\_weekday) |
|  |  |
|  | #remove from test set |
|  | test = subset(test,select=-pickup\_weekday) |
|  |  |
|  | ######## Feature Scaling ######## |
|  | #Normality check |
|  | # qqnorm(train$fare\_amount) |
|  | # histogram(train$fare\_amount) |
|  | library(car) |
|  | # dev.off() |
|  | par(mfrow=c(1,2)) |
|  | qqPlot(train$fare\_amount) # qqPlot, it has a x values derived from gaussian distribution, if data is distributed normally then the sorted data points should lie very close to the solid reference line |
|  | truehist(train$fare\_amount) # truehist() scales the counts to give an estimate of the probability density. |
|  | lines(density(train$fare\_amount)) # Right skewed # lines() and density() functions to overlay a density plot on histogram |
|  |  |
|  | #Normalisation |
|  |  |
|  | print('dist') |
|  | train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/ |
|  | (max(train[,'dist'] - min(train[,'dist']))) |
|  |  |
|  | # #check multicollearity |
|  | # library(usdm) |
|  | # vif(train[,-1]) |
|  | # |
|  | # vifcor(train[,-1], th = 0.9) |
|  |  |
|  | ######## Splitting train into train and validation subsets ######### |
|  | set.seed(1000) |
|  | tr.idx = createDataPartition(train$fare\_amount,p=0.75,list = FALSE) # 75% in trainin and 25% in Validation Datasets |
|  | train\_data = train[tr.idx,] |
|  | test\_data = train[-tr.idx,] |
|  |  |
|  | rmExcept(c("test","train","df",'df1','df2','df3','test\_data','train\_data','test\_pickup\_datetime')) |
|  | #####Model Selection################ |
|  | #Error metric used to select model is RMSE |
|  |  |
|  | ############# Linear regression ################# |
|  | lm\_model = lm(fare\_amount ~.,data=train\_data) |
|  |  |
|  | summary(lm\_model) |
|  | str(train\_data) |
|  | plot(lm\_model$fitted.values,rstandard(lm\_model),main = "Residual plot", |
|  | xlab = "Predicted values of fare\_amount", |
|  | ylab = "standardized residuals") |
|  |  |
|  |  |
|  | lm\_predictions = predict(lm\_model,test\_data[,2:6]) |
|  |  |
|  | qplot(x = test\_data[,1], y = lm\_predictions, data = test\_data, color = I("blue"), geom = "point") |
|  |  |
|  | regr.eval(test\_data[,1],lm\_predictions) |
|  | # mae mse rmse mape |
|  | # 3.5303114 19.3079726 4.3940838 0.4510407 |
|  |  |
|  |  |
|  | ###### Decision Tree ########## |
|  |  |
|  | Dt\_model = rpart(fare\_amount ~ ., data = train\_data, method = "anova") |
|  |  |
|  | summary(Dt\_model) |
|  | #Predict for new test cases |
|  | predictions\_DT = predict(Dt\_model, test\_data[,2:6]) |
|  |  |
|  | qplot(x = test\_data[,1], y = predictions\_DT, data = test\_data, color = I("blue"), geom = "point") |
|  |  |
|  | regr.eval(test\_data[,1],predictions\_DT) |
|  | # mae mse rmse mape |
|  | # 1.8981592 6.7034713 2.5891063 0.2241461 |
|  |  |
|  |  |
|  | ############# Random forest ########## |
|  | rf\_model = randomForest(fare\_amount ~.,data=train\_data) |
|  |  |
|  | summary(rf\_model) |
|  |  |
|  | rf\_predictions = predict(rf\_model,test\_data[,2:6]) |
|  |  |
|  | qplot(x = test\_data[,1], y = rf\_predictions, data = test\_data, color = I("blue"), geom = "point") |
|  |  |
|  | regr.eval(test\_data[,1],rf\_predictions) |
|  | # mae mse rmse mape |
|  | # 1.9053850 6.3682283 2.5235349 0.2335395 |
|  |  |
|  | ####### Improving Accuracy by using Ensemble technique ---- XGBOOST ###### |
|  | train\_data\_matrix = as.matrix(sapply(train\_data[-1],as.numeric)) |
|  | test\_data\_data\_matrix = as.matrix(sapply(test\_data[-1],as.numeric)) |
|  |  |
|  | xgboost\_model = xgboost(data = train\_data\_matrix,label = train\_data$fare\_amount,nrounds = 15,verbose = FALSE) |
|  |  |
|  | summary(xgboost\_model) |
|  | xgb\_predictions = predict(xgboost\_model,test\_data\_data\_matrix) |
|  |  |
|  | qplot(x = test\_data[,1], y = xgb\_predictions, data = test\_data, color = I("blue"), geom = "point") |
|  |  |
|  | regr.eval(test\_data[,1],xgb\_predictions) |
|  | # mae mse rmse mape |
|  | # 1.6183415 5.1096465 2.2604527 0.1861947 |
|  |  |
|  | ####### Finalizing and Saving Model for later use ######## |
|  | # In this step we will train our model on whole training Dataset and save that model for later use |
|  | train\_data\_matrix2 = as.matrix(sapply(train[-1],as.numeric)) |
|  | test\_data\_matrix2 = as.matrix(sapply(test,as.numeric)) |
|  |  |
|  | xgboost\_model2 = xgboost(data = train\_data\_matrix2,label = train$fare\_amount,nrounds = 15,verbose = FALSE) |
|  |  |
|  | # Saving the trained model |
|  | saveRDS(xgboost\_model2, "./final\_Xgboost\_model\_using\_R.rds") |
|  |  |
|  | # loading the saved model |
|  | super\_model <- readRDS("./final\_Xgboost\_model\_using\_R.rds") |
|  | print(super\_model) |
|  |  |
|  | # Lets now predict on test dataset |
|  | xgb = predict(super\_model,test\_data\_matrix2) |
|  |  |
|  | xgb\_pred = data.frame(test\_pickup\_datetime,"predictions" = xgb) |
|  |  |
|  | # Now lets write(save) the predicted fare\_amount in disk as .csv format |
|  | write.csv(xgb\_pred,"xgb\_predictions\_R.csv",row.names = FALSE) |