Report on efficiency dispatching taxis

Table of Contents

Objectives

The goal of this study is demand prediction around Manhattan (Low Manhattan, Midtown, Upper East Side, Upper West Side) and the airports (JFK and LaGuardia) for SUPER TAXIS. The demand has to be predicted as "low", "medium", or "high" using historical information on revenue and costs available for each region registered in 2015. The first prediction strategy has to be concentrated on the overall forecasting accuracy, the second one on the accuracy of "low" demand forecasting.

Two alternative predictive models (two different deployment strategies) have to be discussed with respect to the possibilities of revenue loss reduction.

1. To create the predictive models the following stages will be done, namely:
2. import, cleaning, and exploration of the 2015 taxi data of SUPER TAXIS;
3. aggregation of pick-ups and drop-offs into six regions;
4. summary table of pick-ups and drop-offs totals by region and hour;
5. initial analysis of summary table, structural selection of the predictive models (definition of independent variables);

6. data partitioning (datasets for training and testing);
7. models training and validation;
8. testing and evaluation of the selected models;
9. results discussion and recommendations.

Data

The initial data

In this study, we used the data registered in 2015. As a first step, we created a datastore and aggregate the data from twelve CSV-files (one file for each month of the year). The aggregated data were presented by the table TaxiData (see Datastore and Data agregation). This table contained the following variables: *Vendor, PickupTime, DropoffTime, Passengers, Distance, PickupLon, PickupLat, RateCode, HeldFlag, DropoffLon, DropoffLat, PayType, Fare, ExtraCharge, Tax, Tip, Tolls, ImpSurcharge, TotalCharge, tzPickupBorough, tzDropoffBorough, PickupZone, DropoffZone*.

With respect to the project goal, we defined four custom regions in Manhattan and two airports. Two variables were added: PickupRegion and DropOffRegion (see Regions definition)

Data cleaning

Several cleaning procedures have been completed to reduce the uncertainty related to "human-error" or "extreme-road" conditions (traffic intensity, weather conditions, etc.):

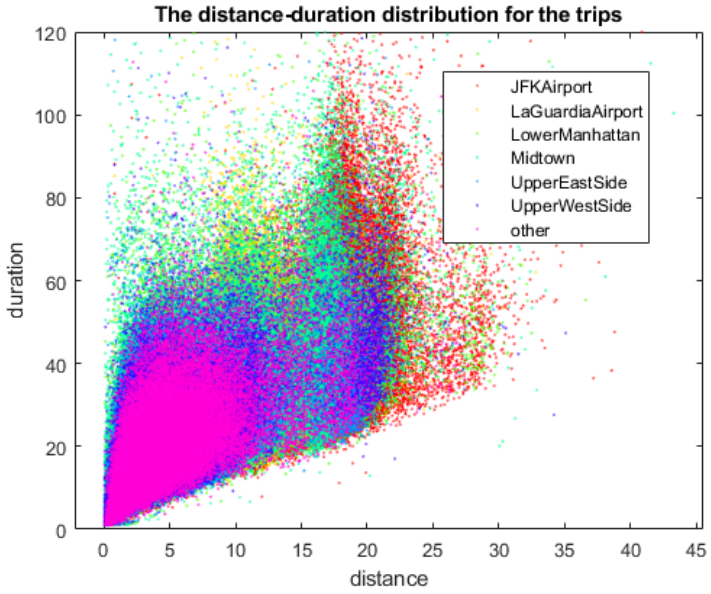
1. **the basic preprocessing (c**harges and trip information that are negative, as well as charges inconsistent with expected values are considered incorrect. In addition, trips with pickup or drop off locations outside a geographic region of interest are removed. This procedure adds new features for trip duration and average speed, it is available at ..\Predictive Modeling and Machine Learning\Code Files \Cleaning Functions\basicPreprocessing.mlx **);**
2. **the additional preprocessing**

The first part (see function AdditionalPreprocessing**)**

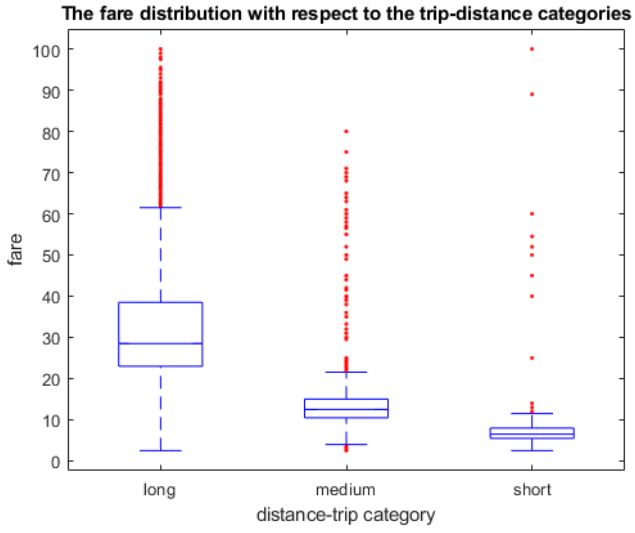
The goal of this part of preprocessing is the definition of all the trips, which had place only in Manhatten, only out of Manhatten and between Manhatten and the airports. This separation is presented by the variable "IsInCity". This allowed to separate the trips using "distance-duration" as "long", "medium", and "short" (variables idxDistance - categorical, indX - double):

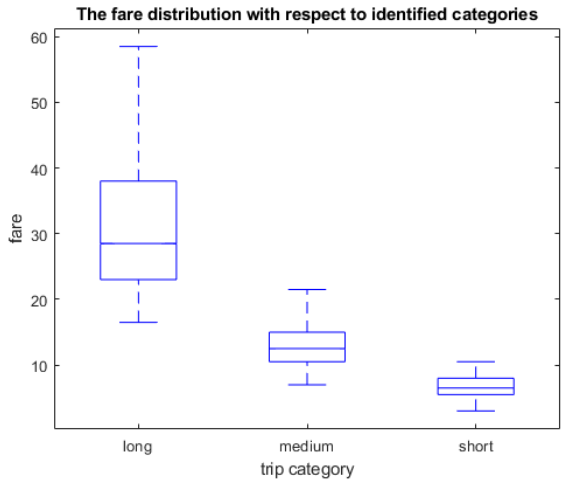
- the exclusion of all the trips where the fare for a trip was less than $2.50;
- traffic tempo - The NYC citywide maximum limit is 25mph (40 km/h) unless otherwise posted. The limit out the city is 55 mph (88 km/h). Hence, these conditions were applied to AveSpeed variable. Moreover, all the trips with a speed less than 2 km/h were also deleted from the dataset;
- we supposed that duration depended on distance. This non-linear dependence is influenced by region (the day-time conditions in the central part are different from that in the night-time, the trips to airports are longer in distance but not obligatory longer in duration, the different rates are fixed for different directions). Therefore, the distance-duration distribution can be considered as a mixture of several

distributions. To separate these distributions we applied the cluster analysis. First, we used the Gaussian mixture model for three clusters called "long-distance", "medium-distance", and "short-distance". This gave the possibility to assign probabilities to data points and exclude all the points with negligible probability. Thus, if the probability of a trip was negligible, it was not important how much it cost;

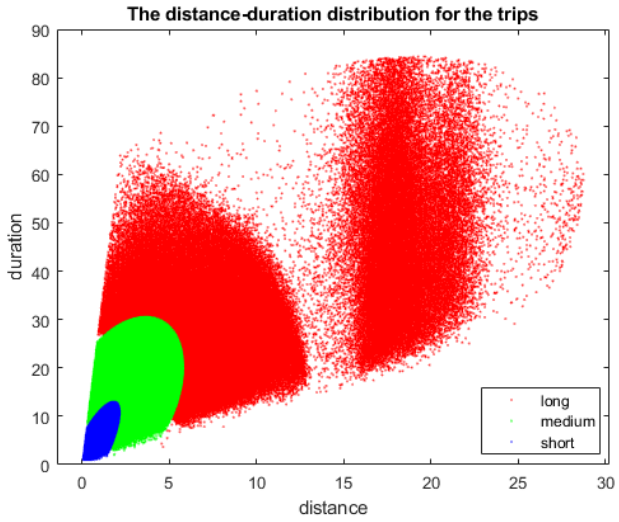


- it was noticed, that the "fare" distribution for the "long-distance" category had the asimetric distribution. It was possible to conclude that the inside-class variations can be reduced by deleting outliers with respect of the rate fixed for the trips;



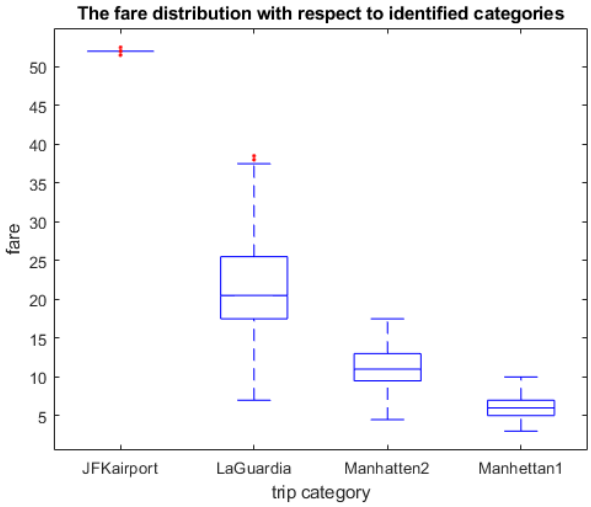


- the analysis showed that the "long-distance" class was divided into two parts (it coincides with two main directions - JFK and LaGuardia airports).

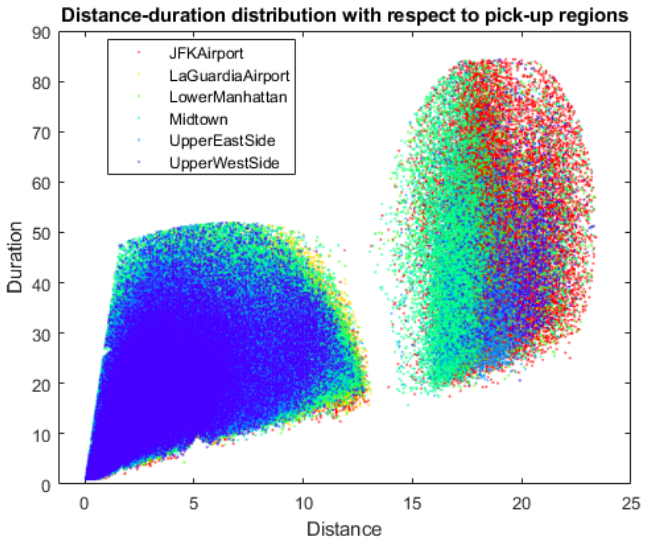


The second part (see function AdditionalPreprocessingTwo**).**

The goal of this part of preprocessing was the separation of two airports and the indication of the fares for different categories of trips. For that reason, the cluster analysis was applied for the second time with four classes - "JFK airport", "LaGuardia", "Manhatten 1", and "Manhatten 2". The trips with negligible probability were deleted. Finally, the four classes were associated with "fares". The outliers were removed. As it was possible to notice for JFK airport the SUPER TAXI has the fixed fare (variable IdxFare).



The distance-duration distribution for the regions of interest can be presented as follows



Data restructuring

Feature engineering

Let us introduce a set of features that would be used for the predictive model construction. First, to take into account "traffic" conditions we used:

- inverse distance

$$InverDistance = \frac{1}{distance},$$

- traffic intencity

$$Intencity = \frac{1}{Duration},$$

- traffic tempo

$$Tempo(i) = \frac{\frac{1}{Speed} - AveTempo(i)}{AveTempo(i)},$$

where i stands for the trip indication ($i = 0$ - inside Manhatten, $i = 1$ - between Manhatten and airports, $i = 2$ - out of Manhatten), and

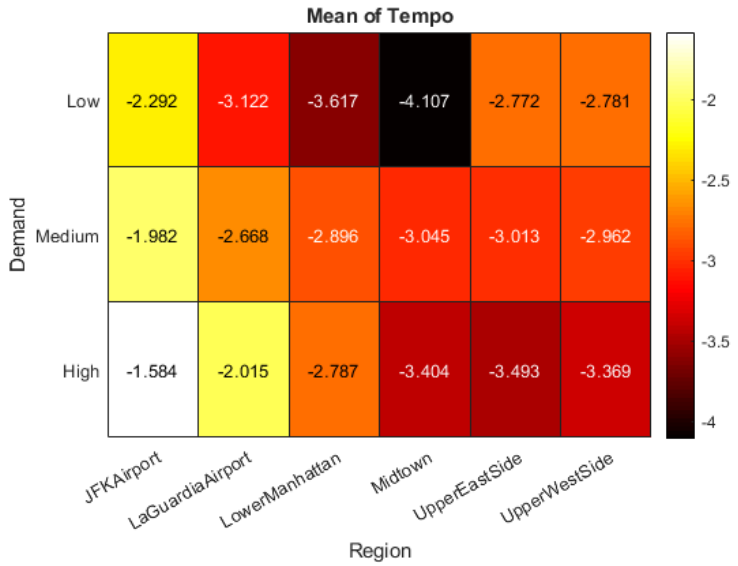
$$AveTempo(i) = \frac{1}{SpeedLimit(i)}.$$

Second, we created the summary table according to the requirements given by Tasks for Creating the Grouped Summary Table. Third, we introduced categories of "Demand". Finally, the table contained variables for analysis, namely: Region, AveDistance, AveDuration, AveFare, AveSpeed, AveTotalCharge, AveCodeDistance, AveCodeFare, Intencity, IsInCity, Tempo, InverDistance, TimeOfDay, DayOfWeek, DayOfMonth, MonthOfYear,DayOfYear, IsHoliday, NotWeekEnd, ProffCode, TD, Demand (for more details see function AddTaxiFeatures, function AddTimeFeatures, and function DependentVariable)

Relevant exploration results

Let us analyse the data. The traffic conditions are presented by variable "Tempo". The "ideal" conditions correspond to value "0". The negative values of this variable indicate "difficult" traffic conditions. As it is possible to notice "low" demand for the "airports" directions, "Midtown" and "Low Manhatten" as well as "high" demand is typical for "East Side" is related the "difficult" traffic conditions.

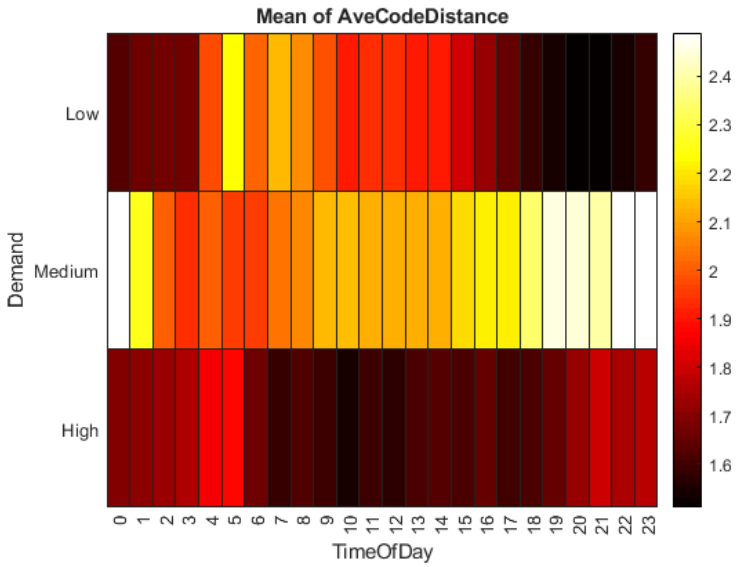
```
heatmap(Data,"Region","Demand",'ColorVariable',"Tempo","ColorMethod","mean","Colormap",hot(100))
```



Inside Mantatten taxi-demand is high during "working-hours", the "airport-directions" are mostly related to the "medium" demand.

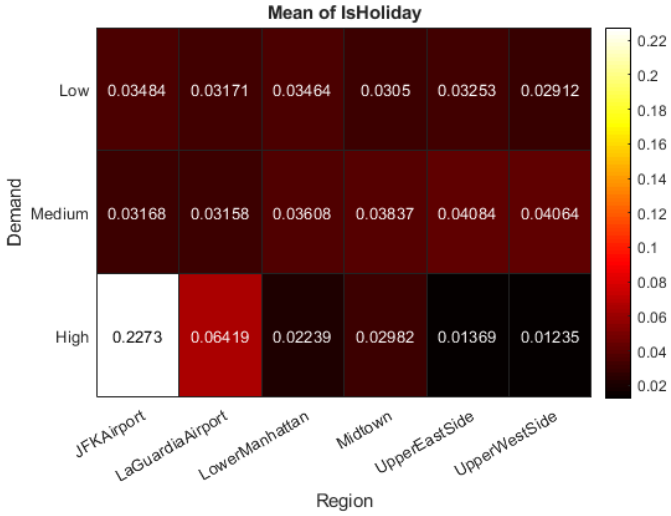
```
heatmap(Data,"TimeOfDay","Demand",'ColorVariable',"AveCodeDistance", ...
```

```
"ColorMethod","mean","Colormap",hot(100))
```



The high demand is typical for the airport-directions during holidays.

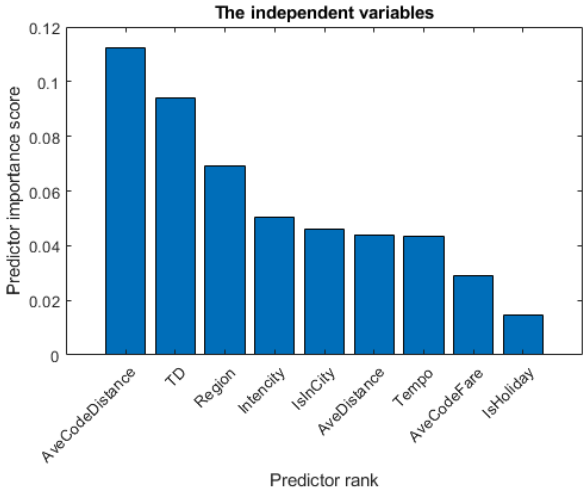
```
heatmap(Data,"Region","Demand",'ColorVariable',"IsHoliday", ...
    "ColorMethod","mean","Colormap",hot(100))
```



To select the independent variables which are important for demand prediction we used **Minimum Redundancy Maximum Relevance (MRMR) Algorithm**

```
[idx, scores] = fscmrmr(tbl,Data.Demand);
```

We included only variable with scores bigger that 0.01



Description of test data split

We split the dataset for traing and testing. To ensure a random flip of initial data we set the random number generator seed to 10. Using `cvpartition` function we separate 20% of the data set for testing later on, and create the training data.

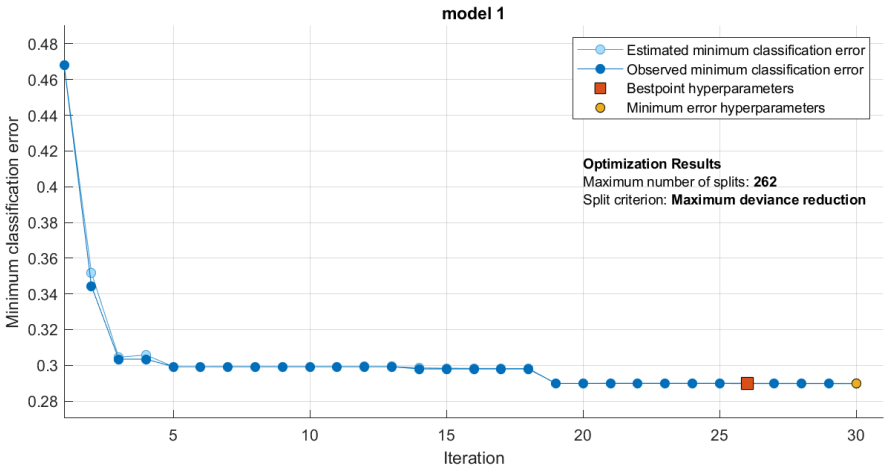
```
rng(10);
DataPartitions = cvpartition(height(Data1),"HoldOut",0.2);
%test dataset
DataTestIdx = test(DataPartitions);
DataTest = Data1(DataTestIdx,:);
%training dataset
DataTrainIdx = training(DataPartitions);
DataTrain = Data1(DataTrainIdx,:);
```

Modeling

Model 1

- Statement of the objective(s)

The primary goal was to determine the class of the predictive model with the best accuracy. Since the dependent variable is the categorical one, we used a classification methodology based on supervised machine learning. To select the model type we used two alternatives: a simple decision tree and an ensemble decision tree. In both cases, 10-fold cross-validation was used to select the optimal structure of the decision tree. In the first case, the optimizable tree gave "the best model" with an accuracy 71,9%. For the second case optimizable ensemble gave "the best" model with 70% accuracy. Since the training is faster in the first case, we use the decision tree with parameters as indicated below.



Model 1 description:

- Model type and Hyperparameters - fine decision tree (maximum split number - 262, split criterion - maximum deviance reduction),
- Required predictor features (as it was indicated in privious section - 'Region', 'AveDistance', 'AveCodeDistance', 'AveCodeFare', 'Intencity', 'IsInCity', 'Tempo', 'IsHoliday', 'TD');
- Training description (see function ModelOneClassifier):

```
[modelStruct1, validationAccuracyModelOne] = ModelOneClassifier(DataTrain);
myModelOne = modelStruct1.ClassificationTree.ModelParameters;
fprintf('Validation Accuracy is  %f   \n',validationAccuracyModelOne)
```

```
Validation Accuracy is  0.725389
```

- Validation method - 10-fold cross validation,
- Test metrics: Precision, Recall, Fallout, Specificity, F1

```
yActual=DataTest.Demand;
yPredicted=modelStruct1.predictFcn(DataTest);
t1 = cMetrics(yActual,yPredicted)
```

```
Accuracy = 72.77%
t1 = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.6179	0.5958	0.1222	0.8778	0.6067
2 Medium	0.7770	0.8091	0.3310	0.6690	0.7928
3 High	0.7021	0.6359	0.0526	0.9474	0.6674
4 Avg	0.6990	0.6803	0.1686	0.8314	0.6889
5 WgtAvg	0.7252	0.7277	0.2336	0.7664	0.7260

Second goal was the introduction of some improvement of the prediction power for "low" demand class.

Model 2

Description:

- Model type and Hyperparameters - fine decision tree (maximum split number - 262, split criterion - maximum deviance reduction),
- Required predictor features (as it was indicated in privious section);
- Training description (see function ModelTwoClassifier):

```
[modelStruct2, validationAccuracyModelTwo] = ModelTwoClassifier(DataTrain);
myModelTwo = modelStruct2.ClassificationTree.ModelParameters;
fprintf('Validation Accuracy is  %f   \n',validationAccuracyModelTwo)
```

```
Validation Accuracy is  0.717216
```

- Customizations to the cost was made to prioretize the accuracy of "low" demand prediction. Since the traffic conditions are usually very difficult, we selected to favorize the "medium" demand prediction to avoid to send taxis in "Midtown". Thus the misclassification cost matrix was as follows:

```
COST=[0 1.3 0.75; 1 0 1.2; 1 1 0]
```

- Validation method - 10-fold cross validation,
- Test metrics: Precision, Recall, Fallout, Specificity, F1

```
myModelOne = modelStruct1.ClassificationTree.ModelParameters;
yActual=DataTest.Demand;
yPredicted2=modelStruct2.predictFcn(DataTest);
t2= cMetrics(yActual,yPredicted2)
```

```
Accuracy = 70.91%
t2 = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.5292	0.6998	0.2065	0.7935	0.6027
2 Medium	0.8039	0.7565	0.2631	0.7369	0.7795
3 High	0.7672	0.5526	0.0327	0.9673	0.6424
4 Avg	0.7001	0.6696	0.1674	0.8326	0.6749
5 WgtAvg	0.7295	0.7091	0.2114	0.7886	0.7131

As it is possible to notice, recall for "Low" demand is better, however precision is very low. To improve this situation we used the class-balancing methodwith under and over factors equal "1" as follows

```
tLow = Data(Data.Demand == 'Low',:); tMedium = Data(Data.Demand == 'Medium',:); tHigh=Data(Data.Demand=="High"
underFactor=1;
numberToUnder=floor(height(tHigh)*underFactor); tMediumUnder=datasample(tMedium,numberToUnder,"Replace",false)
 overFactor=1;
numberToOver=height(tHigh)*overFactor;
tHighOver=datasample(tHigh,numberToOver,"Replace",true); newRation=height(tHighOver)/height(tMediumUnder);
tblUnderOver=[tHighOver;tMediumUnder;Data(Data.Demand=="Low",:)];
Data1=tblUnderOver;
```

The model 2 was trained once again with the same parameters. The testing gave validation accuracy 74.1%.

```
myModelTwo = modelStruct2.ClassificationTree.ModelParameters;
yActual=DataTest.Demand;
yPredicted=modelStruct2.predictFcn(DataTest);
t3 = cMetrics(yActual,yPredicted)
```

```
Accuracy = 73.26%
t3 = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.7206	0.9119	0.3825	0.6175	0.8051
2 Medium	0.7156	0.3422	0.0450	0.9550	0.4630
3 High	0.7768	0.7497	0.0649	0.9351	0.7630
4 Avg	0.7377	0.6679	0.1641	0.8359	0.6770
5 WgtAvg	0.7324	0.7326	0.2250	0.7750	0.7102

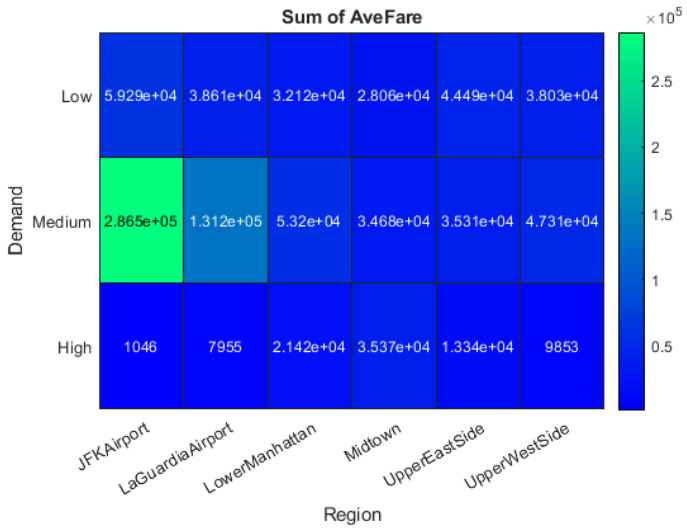
We improved recall for "Low" demand class. F1 metric is acceptable for "Low" and "High" demand classes.

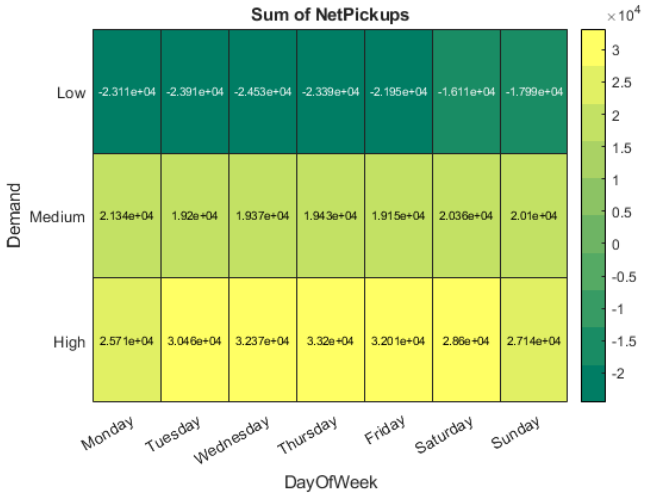
Summary of models

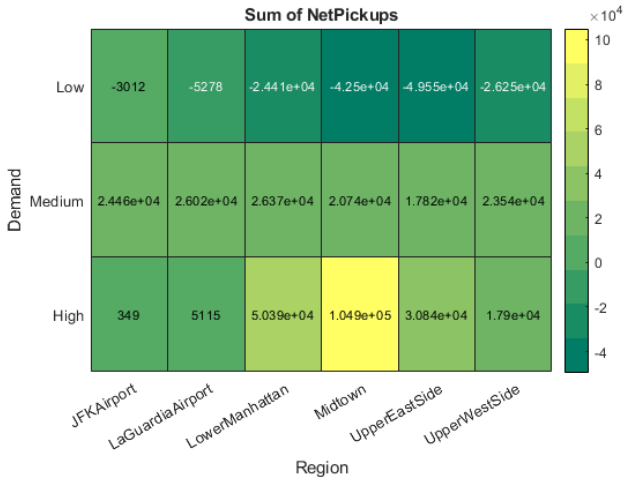
Model 2 with the misclassification cost matrix and under-over classes balancing has better performance and the prediction power for "low" and "high" demand on taxis.

Conclusions

We tested several strategies for two models. The accuracy of the second model was better than that for the first model. The recall of model 2 for "low" demand class is better than that for the first one. That is to say, the forecast is more accurate. We created the predictive model which is capable to forecast "low" and "high" demand with good accuracy. This means that SUPER TAXI can have the following strategy.







Since "medium" demand is related to bigger income (fares), the resources can be calculated with respect to this "background" demand. The precise prediction of "low" demand allows allocating the resources among regions with "medium" or "high" demand to avoid losses.

Appendix

Datastore and Data agregation

```
function TaxiData = CreateTaxiDataStore(fileName)
dataPath = fullfile(fileparts(which(fileName)));
if isempty(dataPath)
    f = msgbox(["Taxi data not found on the MATLAB path." ; ...
        "Please select the location in the following dialog."]);
    uiwait(f)
    %open folder selection dialog box
    dataPath = uigetdir([],"Select the folder containing the taxi data.");
end
ds = fileDatastore( fullfile(dataPath,"yellow*.csv"), ...
    "ReadFcn",@importTaxiDataWithoutCleaning,"UniformRead",true);
```

```
reset(ds);%reset the datastore to the state where no data has been read from it
numFiles=size(ds.Files,1);
fileNum=0;
TaxiData=table;
while hasdata(ds)
    data=read(ds);
    data=addTaxiZones(data);
    TaxiData=[TaxiData;data]; %#ok<AGROW>
    fileNum = fileNum+1;
    disp("Finished file " + fileNum + " of " + numFiles);
end
save TaxiDataStore TaxiData dataPath
clear fileName fileNum data
end
```

The details on functions *importTaxiDataWithoutCleaning and addTaxiZones* available at ..\Predictive Modeling and Machine Learning\Code Files\importTaxiDataWithoutCleaning.mlx and ..\Predictive Modeling and Machine Learning\Code Files\Feature Functions\addTaxiZones.mlx.

Regions definition

```
function TaxiData=TaxiRegionDefinition(TaxiData)
TaxiRegionsandZones= importTaxiRegZones('Taxi Regions and Zones.csv');
Zones=categorical(TaxiRegionsandZones.Properties.VariableNames');

Zone1=categorical(TaxiRegionsandZones.LowerManhattan(TaxiRegionsandZones.LowerManhattan~=""));
Zone2=categorical(TaxiRegionsandZones.Midtown(TaxiRegionsandZones.Midtown~=""));
Zone3=categorical(TaxiRegionsandZones.UpperEastSide(TaxiRegionsandZones.UpperEastSide~=""));
Zone4=categorical(TaxiRegionsandZones.UpperWestSide(TaxiRegionsandZones.UpperWestSide~=""));
Zone5=categorical(TaxiRegionsandZones.JFKAirport(TaxiRegionsandZones.JFKAirport~=""));
Zone6=categorical(TaxiRegionsandZones.LaGuardiaAirport(TaxiRegionsandZones.LaGuardiaAirport~=""));
clear TaxiRegionsandZones;

TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone1)))=Zones(1);
TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone2)))=Zones(2);
TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone3)))=Zones(3);
TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone4)))=Zones(4);
TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone5)))=Zones(5);
TaxiData.PickupRegion(find(ismember(TaxiData.PickupZone,Zone6)))=Zones(6);
%create new variable DropOffRegion
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone1)))=Zones(1);
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone2)))=Zones(2);
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone3)))=Zones(3);
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone4)))=Zones(4);
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone5)))=Zones(5);
TaxiData.DropOffRegion(find(ismember(TaxiData.DropoffZone,Zone6)))=Zones(6);
the exclusion of the "other-other" pick-up-drop-off region (it is out of the interest of this study)
%We keep "other" region for Pick-up and Dropp-off
TaxiData.PickupRegion(ismissing(TaxiData.PickupRegion),:)='other';
TaxiData.DropOffRegion(ismissing(TaxiData.DropOffRegion),:)='other';
%the exclusion of the "other-other" pick-up-drop-off region
%(it is out of the interest of this study)
TaxiData=TaxiData(~(TaxiData.DropOffRegion=="other" & TaxiData.PickupRegion=="other"),:);
%clean Workspace
clear Zone1 Zone2 Zone3 Zone4 Zone5 Zone6
```

Additional preprocessing

Part 1

```
function TaxiData=AdditionalPreprocessing(TaxiData)
TaxiData = removevars(TaxiData, {'PickupLon','PickupLat','DropoffLon','DropoffLat', ...
    'PickupZone','DropoffZone', ...
    'Vendor','HeldFlag','ExtraCharge','Tax','Tip','Tolls', ...
    'ImpSurcharge','tzPickupBorough','tzDropoffBorough'});
TaxiData = TaxiData(TaxiData.Fare >= 2.5,:);
% gscatter(TaxiData.Distance,TaxiData.Duration,TaxiData.PickupRegion,[],'.',1)
% xlabel('distance')
% ylabel('duration')
% title('The distance-duration distribution for the trips')
TaxiData.IsInCity=double(ismember(TaxiData.PickupRegion,{ ...
    'LowerManhattan',    'Midtown','UpperEastSide','UpperWestSide'}) ...
    +ismember(TaxiData.DropOffRegion,{'LowerManhattan','Midtown', ...
    'UpperEastSide','UpperWestSide'}));
TaxiData=TaxiData(~((TaxiData.AveSpeed>40 & TaxiData.IsInCity==2)| ...
    (TaxiData.AveSpeed>64 & TaxiData.IsInCity==1)| ...
    (TaxiData.AveSpeed>88 & TaxiData.IsInCity==0)),:); %speed limits
TaxiData=TaxiData(~(TaxiData.AveSpeed<2),:);

rng(10); % For reproducibility
gmmN=[TaxiData.Distance,TaxiData.Duration];
gmN=fitgmdist(gmmN,3);

TaxiData.idxDistance=cluster(gmN,gmmN);
TaxiData.idxN=categorical(TaxiData.idxDistance);
tab=groupsummary(TaxiData,"idxN","mean","Distance");
tab = sortrows(tab,'mean_Distance','descend');
D={char(tab.idxN(1));char(tab.idxN(2));char(tab.idxN(3))};
TaxiData.idxN=reordercats(TaxiData.idxN,D);
TaxiData.idxN=renamecats(TaxiData.idxN,{'long','medium','short'});
TaxiData.PDF1=pdf(gmN,gmmN);
TaxiData=TaxiData(TaxiData.PDF1>2.5e-07,:);
% boxplot(TaxiData.Fare,TaxiData.idxN,'Notch','on',"OutlierSize",5,"Colors","b","BoxStyle", ...
%     "outline","Symbol",'r.')
% xlabel('distance-trip category');
% ylabel('fare');
% title('The fare distribution with respect to the trip-distance categories')
TD1=rmoutliers(TaxiData(TaxiData.idxN=="long",:),'percentiles',[0.5, 99.25],"DataVariables","Fare");
TD2=rmoutliers(TaxiData(TaxiData.idxN=="medium",:),'percentiles',[0.25, 99.75],"DataVariables","Fare");
TD3=rmoutliers(TaxiData(TaxiData.idxN=="short",:),'percentiles',[0.25, 99.75],"DataVariables","Fare");
TaxiData=vertcat(TD1,TD2,TD3);
clear TD1 TD2 TD3
% boxplot(TaxiData.Fare,TaxiData.idxN,'Notch','on',"OutlierSize",5,"Colors","b","BoxStyle", ...
%     "outline","Symbol",'r.')
% xlabel('trip category');
% ylabel('fare');
% title('The fare distribution with respect to identified categories')
TT1=rmoutliers(TaxiData(TaxiData.RateCode=="Group",:),'mean',"DataVariables","Fare");
TT2=rmoutliers(TaxiData(TaxiData.RateCode=="JFK",:),'mean',"DataVariables","Fare");
TT3=rmoutliers(TaxiData(TaxiData.RateCode=="Nassau",:),'median',"DataVariables","Fare");
TT4=rmoutliers(TaxiData(TaxiData.RateCode=="Negotiated",:),'mean',"DataVariables","Fare");
TT5=rmoutliers(TaxiData(TaxiData.RateCode=="Newark",:),'mean',"DataVariables","Fare");
TT6=rmoutliers(TaxiData(TaxiData.RateCode=="Standard",:),'mean',"DataVariables","Fare");

TaxiData=vertcat(TT1,TT2,TT3,TT4,TT5,TT6);
clear TT1 TT2 TT3 TT4 TT5 TT6
```

```
end
```

Part 2

```
function TaxiData=AdditionalPreprocessingTwo(TaxiData)

rng(10); % For reproducibility

gmm=[TaxiData.Distance,TaxiData.Duration];
options = statset('MaxIter',150);
gm=fitgmdist(gmm,4,'Options',options);
TaxiData.idxFare=cluster(gm,gmm);
TaxiData.PDF2=pdf(gm,gmm);
TaxiData=TaxiData(TaxiData.PDF2>2.5e-06,:);
TaxiData.idx=categorical(TaxiData.idxFare);

tab1=groupsummary(TaxiData,"idx","mean","Fare");
tab1=sortrows(tab1,"mean_Fare",'descend');
D={char(tab1.idx(1));char(tab1.idx(2));char(tab1.idx(3));char(tab1.idx(4))};
TaxiData.idx=reordercats(TaxiData.idx,D);
TaxiData.idx=renamecats(TaxiData.idx,{'JFKairport','LaGuardia','Manhatten2','Manhettan1'});

TK1=rmoutliers(TaxiData(TaxiData.idx=="JFKairport",:),'mean',"DataVariables","Fare");
TK2=rmoutliers(TaxiData(TaxiData.idx=="LaGuardia",:),'mean',"DataVariables","Fare");
TK3=rmoutliers(TaxiData(TaxiData.idx=="Manhatten2",:),'mean',"DataVariables","Fare");
TK4=rmoutliers(TaxiData(TaxiData.idx=="Manhettan1",:),'mean',"DataVariables","Fare");

TaxiData=vertcat(TK1,TK2,TK3,TK4); clear  TK1 TK2 TK3 TK4;

end
```

Feature engineering

Group summary and some additional variables

```
function Data=AddTaxiFeatures(TaxiData)

AveTempo=(1/40*(TaxiData.IsInCity==2)+1/64*(TaxiData.IsInCity==1)+1/88*(TaxiData.IsInCity==0));
TaxiData.Tempo=TaxiData.AveSpeed.^(-1);
TaxiData.Tempo=(AveTempo-TaxiData.Tempo)./AveTempo;
TaxiData.Intencity=TaxiData.Duration.^(-1);
TaxiData.InverDistance=TaxiData.Distance.^(-1);

TaxiData=removevars(TaxiData, {'idx','PayType','idxN','RateCode'});
TaxiData.BinPickupTime=dateshift(TaxiData.PickupTime,"start","hour");
TaxiData.BinDropoffTime=dateshift(TaxiData.DropoffTime,"start","hour");
TaxiData=removevars(TaxiData, {'PickupTime','DropoffTime'});

pick=groupsummary(TaxiData,["PickupRegion","BinPickupTime"], ...
    "mean",["Distance","Duration","Fare","AveSpeed","TotalCharge", ...
            "idxDistance","idxFare","Intencity","IsInCity","Tempo","InverDistance"]);
drop=groupsummary(TaxiData, ["DropOffRegion","BinDropoffTime"]);

Data = outerjoin(pick,drop,'LeftKeys',{'PickupRegion','BinPickupTime'},...
    'RightKeys',{'DropOffRegion','BinDropoffTime'},'MergeKeys',true);
Data = movevars(Data, 'GroupCount_drop', 'Before', 'mean_Distance');
Data.Properties.VariableNames{1} = 'Region';
Data.Properties.VariableNames{2} = 'BinTime';
```

```
Data.Properties.VariableNames{3} = 'PickupCount';
Data.Properties.VariableNames{4} = 'DropoffCount';

Data.Region=removecats(Data.Region,'other');
Data = Data(~ismissing(Data.Region),:);

% Fill missing data
Data.DropoffCount = fillmissing(Data.DropoffCount,'constant',0);

% Remove missing data
Data=Data(~ismissing(Data.PickupCount),:);
Data.NetPickups=Data.PickupCount-Data.DropoffCount;

Data = movevars(Data, 'NetPickups', 'Before', 'mean_Distance');

Data.Properties.VariableNames{6} = 'AveDistance';
Data.Properties.VariableNames{7} = 'AveDuration';
Data.Properties.VariableNames{8} = 'AveFare';
Data.Properties.VariableNames{9} = 'AveSpeed';
Data.Properties.VariableNames{10} = 'AveTotalCharge';
Data.Properties.VariableNames{11} ='AveCodeDistance';
Data.Properties.VariableNames{12} ='AveCodeFare';
Data.Properties.VariableNames{13} ='Intencity';
Data.Properties.VariableNames{14} ='IsInCity',
Data.Properties.VariableNames{15} ='Tempo';
Data.Properties.VariableNames{16} ='InverDistance';

end
```

Date-related variables

```
function Data=AddTimeFeatures(Data)
%time of day
Data.TimeOfDay = hours(timeofday(Data.BinTime));
%day of week
Data.DayOfWeek = categorical(day(Data.BinTime,"name"));
Data.DayOfWeek=reordercats(Data.DayOfWeek,{'Monday';'Tuesday' ...
    ;'Wednesday';'Thursday';'Friday';'Saturday';'Sunday'});
%day of month
Data.DayOfMonth =day(Data.BinTime);
%month of year
Data.MonthOfYear = categorical(month(Data.BinTime,'name'));
Data.MonthOfYear=reordercats(Data.MonthOfYear, ...
    {'January','February','March','April','May','June', ...
    'July','August','September','October','November','December'});
%day of year
Data.DayOfYear =day(Data.BinTime,"dayofyear");
%holiday
BankHolidays = Holidays('2015 Bank Holidays');
BankHolidays.Date = datetime(BankHolidays.Date,'InputFormat','MM/dd/yy','Format','yyyy-MM-dd');
BankHolidays.DayOfYear=day(BankHolidays.Date,"dayofyear");
Data.IsHoliday=ismember(Data.DayOfYear,BankHolidays.DayOfYear);
Data.NotWeekEnd=ismember(Data.DayOfWeek,{'Monday','Tuesday','Wednesday','Thursday','Friday'});
%some additional variables for cost analysis
Data.ProffCode=Data.AveCodeFare.*Data.AveCodeDistance;
[indx, TD, DW]=findgroups(string(Data.TimeOfDay),string(Data.DayOfWeek));
Data.TD=strcat(TD(indx),"-",DW(indx));
```

```
end
```

Dependent variable

```
function Data=DependentVariable(Data)
Data.Demand=discretize(Data.NetPickups, ...
    [-inf,0,15,+inf],"categorical",["Low","Medium","High"]);
en
```

Modelling

Model 1

```
function [trainedClassifier, validationAccuracy] = ModelOneClassifier(trainingData)
inputTable = trainingData;
predictorNames = {'Region', 'AveDistance', 'AveCodeDistance', 'AveCodeFare', 'Intencity', 'IsInCity', 'Tempo',
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [true, false, false, false, false, false, false, true, true];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'deviance', ...
    'MaxNumSplits', 262, ...
    'Surrogate', 'off', ...
    'ClassNames', categorical({'Low'; 'Medium'; 'High'}, {'Low' 'Medium' 'High'}, 'Ordinal', true));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AveCodeDistance', 'AveCodeFare', 'AveDistance', 'Intencity', 'IsHolida
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = 'This struct is a trained model exported from Classification Learner R2020b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  yfit = c.predictFcn(

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Region', 'AveDistance', 'AveCodeDistance', 'AveCodeFare', 'Intencity', 'IsInCity', 'Tempo',
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [true, false, false, false, false, false, false, true, true];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
```

```
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

Model 2

```
function [trainedClassifier, validationAccuracy] = ModelTwoClassifier(trainingData)
inputTable = trainingData;
predictorNames = {'Region', 'AveDistance', 'AveCodeDistance', 'AveCodeFare', 'Intencity', 'IsInCity', 'Tempo',
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [true, false, false, false, false, false, false, true, true];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'deviance', ...
    'MaxNumSplits', 262, ...
    'Surrogate', 'off', ...
    'Cost', [0 1.3 0.75; 1 0 1.2; 1 1 0], ...
    'ClassNames', categorical({'Low'; 'Medium'; 'High'}, {'Low' 'Medium' 'High'}, 'Ordinal', true));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AveCodeDistance', 'AveCodeFare', 'AveDistance', 'Intencity', 'IsHolida
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = 'This struct is a trained model exported from Classification Learner R2020b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  yfit = c.predictFcn(

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Region', 'AveDistance', 'AveCodeDistance', 'AveCodeFare', 'Intencity', 'IsInCity', 'Tempo',
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [true, false, false, false, false, false, false, true, true];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```